САМОУЧИТЕЛЬ

bhv

Microsoft Word

комфортная работа с помощью макросов

Борис Клименко
Марк Розенберг

- Индивидуальная настройка меню и панелей инструментов
- Запись макросов в автоматическом режиме
- Внесение изменений в автоматически созданные макросы
- Элементы программирования на языке VBA
- Практика самостоятельной разработки простых макросов

Удобные инструменты своими руками — простые в создании, эффективные в использовании

Борис Клименко
Марк Розенберг

САМОУЧИТЕЛЬ
Microsoft Word
комфортная работа с помощью макросов

Санкт-Петербург
«БХВ-Петербург»
2006

УДК 681.3.06
ББК 32.973.26-018.2
К49

Клименко Б. И., Розенберг М. М.

К49 Microsoft Word: комфортная работа с помощью макросов. Самоучитель. — СПб.: БХВ-Петербург, 2006. — 496 с.: ил.

ISBN 5-94157-775-3

Самоучитель предназначен для всех, кто использует Microsoft Word в работе и желает существенно повысить эффективность своего труда с помощью макросов. Рассказывается об автоматическом режиме записи макросов, описываются разнообразные приемы настройки рабочей среды, облегчающие выполнение стандартных повседневных задач. Объясняется, как понимать тексты программ, созданных при записи макросов в автоматическом режиме, и вносить в эти программы небольшие изменения, значительно повышающие эффективность их применения. Излагаются основы программирования на языке VBA, что позволит читателям самостоятельно создавать простые и эффективные макросы. Обучение построено на простых и наглядных примерах, которые сразу же можно применить на практике.

Для широкого круга пользователей

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Григорий Добин*
Редактор	*Юрий Рожко*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Наталия Першакова*
Дизайн серии	*Игоря Цырульникова*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 20.02.06.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 38,7.
Тираж 2000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию № 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 5-94157-775-3

Оглавление

Введение

"Это безобразие, товарищи, использовать компьютеры для забивания гвоздей, понимаете. Если кто не знает, для этого имеются молотки. И стоят они значительно дешевле". Евгений Максимович Примаков произнес эти слова на ученом совете Института мировой экономики и международных отношений (ИМЭМО), директором которого он являлся в те уже далекие годы (а один из авторов книги служил в отделе информационных технологий). Высокое собрание, как вы понимаете, обсуждало вопросы эффективного применения компьютерной техники.

С тех пор в этом смысле мало что изменилось. Мы имеем в виду, конечно же, не политическую карьеру уважаемого Евгения Максимовича, которая сложилась, как известно, очень удачно, и не ситуацию с применением компьютерной техники в ИМЭМО РАН (мы уверены, что она существенно изменилась к лучшему). Речь идет о подавляющем большинстве рядовых пользователей, которые по-прежнему относятся к компьютеру как к устройству, пришедшему на смену пишущей машинке.

Авторам этой книги в силу специфики их теперешних занятий также приходится довольно много работать с текстами. При этом один из нас уже имел опыт в области программирования, второго заставили заняться *его* освоением соображения чисто практического характера. Объединяло нас одно — мы оба стремились максимально использовать возможности компьютера для автоматизации рутинных операций. После того как был накоплен некоторый опыт, у нас возникла идея объединить усилия и поделиться своими достижениями с коллегами по перу, точнее — клавиатуре. В итоге родилась предлагаемая вашему вниманию книга, знакомство с которой, как мы надеемся, облегчит вам работу с текстами в Microsoft Word, поможет повысить ее эффективность и производительность.

Приведем для наглядности несколько конкретных примеров.

- Читая или редактируя уже готовый документ, довольно часто приходится листать его с целью уточнить те или иные моменты. Вернуться затем на "исходную" позицию бывает не всегда просто, особенно при работе с большими по объему документами. Оказывается, сделать это можно одним щелчком мыши (нажатием сочетания "горячих" клавиш или кнопки — что удобнее пользователю).

- При работе с документом часто возникает потребность вернуться в дальнейшем к просмотру некоторых его фрагментов. Один щелчок мыши (нажатие сочетания клавиш), и вставляется закладка с таким, например, текстом "Вернуться_и_исправить" (при этом соответствующее слово или выделенное выражение подкрашивается для наглядности цветом).
- В процессе редактирования часто требуется удалить текст от позиции курсора до конца предложения (нажал сочетание клавиш).
- Если в тексте в изобилии встречаются названия или обозначения, написанные латинскими буквами, то проверка правописания встроенными в Microsoft Word средствами резко замедляется. Эффективность этого процесса существенно повысится, если в вашем распоряжении будет кнопка, помечающая все слова, написанные латинскими буквами, признаком "не проверять правописание".

Можно создать также целый ряд других полезных (и достаточно простых по своей реализации) приемов и средств, которые могут помочь в решении более сложных задач, например, при редактировании и правке литературного стиля документа.

Догадываясь, что без программирования здесь не обойтись, вы, быть может, уже задумались, стоит ли читать дальше. Наверное, вам в руки уже попадали книжки из серии "для тех, кто собирается быстро и без лишних усилий научиться программировать", и давно решили, что это занятие не для вас. Времени катастрофически не хватает, ваши интересы и полученное гуманитарное образование никак не связаны с программированием, да и вообще для этого есть специальные люди, обладающие соответствующей квалификацией (они так и называются — программисты).

Однако советуем не спешить. Мы торжественно обещаем, что, прочтя первую часть книги (в которой о программировании не будет сказано ни единого слова), вы освоите приемы *автоматического* (т. е. не требующего участия пользователя) создания программ, предназначенных для эффективной работы с текстами. В главах первой части книги будет рассказано, какие действия следует выполнить, какие клавиши нажать, чтобы, к примеру, удалить текст от позиции курсора до конца предложения. Повторяя вместе с нами эти нехитрые манипуляции, вы попутно (и незаметно для себя) создадите программку, которая в дальнейшем позволит мгновенно осуществлять указанную процедуру (удаление текста) посредством простого нажатия сочетания клавиш. Аналогичный по своей эффективности результат будет получен и при воспроизведении других описанных в книге процедур, облегчающих подготовку и редактирование текста. Кстати, часть материалов, содержащихся в этом разделе, была опубликована в журнале "Домашний компьютер".

И поможет нам во всем этом так называемый режим записи макросов. В данном режиме все действия пользователя *автоматически* записываются в виде программы на языке VBA (Visual Basic for Application, т. е., *Visual Basic для приложений*). Эти программы, называемые макросами, можно потом вызывать и исполнять посредством нажатия кнопки или заданного сочетания клавиш.

Инструментом создания макросов, реализующих указанный режим, является редактор Visual Basic, встроенный во многие приложения Microsoft Office. В этой книге мы будем учиться применять его в Microsoft Word. Почему именно Microsoft Word, видимо, понятно — мы рассчитываем на читателей, которые имеют дело с текстами, а подавляющее число пользователей в нашей стране (как, впрочем, и во всем мире) применяет для их создания и редактирования именно эту программу. Кроме того, авторы принимали участие в локализации (упрощенно говоря — переводе на русский язык) пакета Microsoft Office, куда входит Microsoft Word, и поэтому знакомы с данной программой, как говорится, не понаслышке.

Прочтя первую часть книги, вы научитесь записывать свои собственные макросы и пользоваться макросами, созданными другими. Это не потребует от вас чрезмерных усилий и в то же время позволит заметно повысить производительность своего труда. Мы уверены, что вы будете получать истинное удовольствие и удовлетворение при каждом обращении к своему макросу, ведь с его помощью ускоряется и облегчается работа, снижается количество ошибок и поэтому появляется законный повод себя похвалить. А как приятно подарить макрос своим коллегам!

Наверное, некоторые из вас, ознакомившись с первой частью книги, решат, что этого вполне достаточно: макросы писать мы научились, эффективность работы с текстами заметно возросла — чего еще надо! Однако авторы все же надеются, что большинство наших читателей "войдут во вкус" и продолжат знакомство с другими частями книги. И поступят, конечно, совершенно правильно! Кстати, в этом им поможет знакомство с материалом "промежуточной" *гл. 6* книги, где для желающих проводится "экскурсия в сборочный цех макросов" и показывается, что в текстах программ можно вполне разобраться, даже не владея навыками программирования.

Во второй части описывается, каким образом можно, внеся небольшие изменения в текст исходного макроса, существенно повысить эффективность его работы, успешно справляться с задачами, которые в принципе не могут быть решены с помощью "обычных" макросов, созданных в режиме автоматической записи действий пользователя.

Для этого мы на конкретных примерах подробно опишем некоторые часто встречающиеся в макросах средства языка VBA. Кроме того, мы постараем-

ся доступно рассказать о фундаментальных концепциях программирования. Это позволит вам действовать осознанно, а потому и более эффективно. Овладев только самыми начальными навыками программирования, можно существенно расширить свои возможности. Тут затраченные усилия дают наибольшую отдачу.

В главах третьей части книги на практических примерах излагаются основные идеи программирования. Овладев этим материалом, вы не станете профессиональным программистом, но достигнете уровня, который позволит самостоятельно создавать простые программы на VBA, обращаясь при необходимости к более подробным справочникам и пособиям. Знакомство с основными понятиями и терминологией языка VBA позволит вам пользоваться книжками по программированию, не читая их "от корки до корки".

Подчеркнем еще раз: изложение материала нашей книги построено таким образом, что можно вообще **ограничиться первой частью** и заняться языком VBA только в случае, когда это будет вам интересно или возникнет потребность в решении какой-либо конкретной задачи. Можно поступить и наоборот. Если вы уже умеете самостоятельно создавать макросы в режиме автоматической записи, можно **начать читать книгу со второй части**, где рассказывается об элементах языка VBA, используемых при автоматической записи макросов. Можно также независимо от других глав прочесть *гл. 14*, посвященную классическим методам программирования, используемым во всех приложениях Microsoft Office. Даже в том случае, если вы уже умеете программировать на VBA, в *гл. 15* и *16* вы найдете новую для себя информацию о способах решения ряда проблем, возникающих при программировании задач навигации и поиска в документах, содержащих колонтитулы, примечания, сноски и надписи.

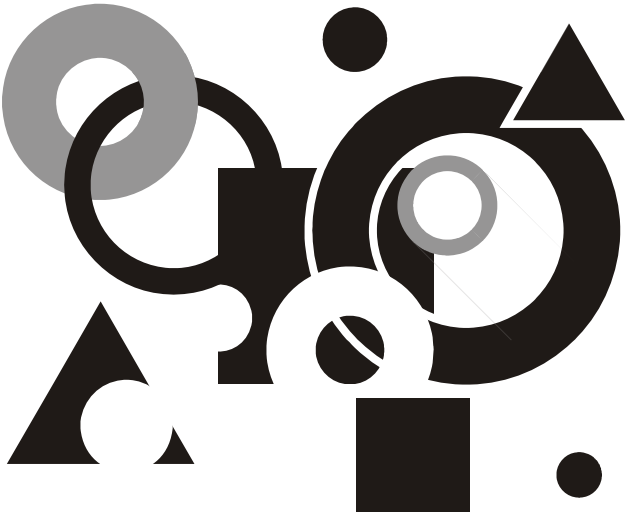
Мы стремились построить изложение материала таким образом, чтобы чтение каждой главы, начиная с самой первой, приносило читателям конкретную практическую пользу. Организация книги, включающая систему указателей и перекрестных ссылок, позволяет легко ориентироваться в прочитанном материале. Отметим также, что в тексте часто приводятся советы и рекомендации общего характера, касающиеся особенностей работы с приложением Microsoft Word и не имеющие прямого отношения к процессу создания макросов. Книга снабжена большим количеством иллюстраций (так называемых "*скриншотов*" или копий экрана) и подробными указателями.

В заключение откроем небольшой секрет. Мы подготовили для вас два шаблона: MacrosBook.dot, в котором содержатся действующие подпрограммы для всех примеров, приведенных в книге, а также MB.dot, в котором собраны макросы, имеющие *наибольшую* практическую пользу. Для их запуска преду-

смотрена инструментальная панель **MB**, содержащая кнопки для вызова макросов и получения справочной информации по их применению. Оба шаблона можно загрузить с веб-страниц **http://www.russianlocalization.com/mb/** или **http://bhv.ru/books/book.php?id=12623**. В *гл. 7* рассказано, как установить шаблон на свой компьютер, и вы сможете пользоваться представленными в нем макросами даже в случае, если не пытались их создавать вместе с нами в процессе чтения книги. Кроме того, вы сможете также самостоятельно установить на своем компьютере много других макросов, полученных из различных источников, в первую очередь, из Интернета. Иными словами, вы научитесь пользоваться *чужими* макросами. Как говорится, уже хорошо.

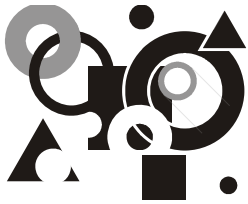
Авторы надеются также, что книга получилась не очень скучной.

Но судить об этом, конечно, будете вы, наши читатели.



ЧАСТЬ I

МАКРОСЫ БЕЗ ПРОГРАММИРОВАНИЯ



Глава 1

Макросы навигации

Возврат на исходную позицию. Создание обычных и цветных закладок; переход к нужному фрагменту документа. Дополнительные сведения.

Итак, как и обещали, мы сразу же приступаем к созданию простых и полезных макросов, с помощью которых вы сможете не только значительно облегчить ввод и редактирование текста, но и более эффективно работать с документами Microsoft Word.

При этом еще раз подчеркнем, что при чтении глав первой части книги вам не потребуется осваивать элементы какого-либо языка программирования. Нужно будет всего лишь аккуратно воспроизводить описываемые здесь элементарные действия — открывать меню, выполнять команды, нажимать кнопки. Правда, делать это придется, как правило, в "прямом эфире", то бишь в режиме записи макросов.

Отметим также, что основная цель, которую авторы поставили перед собой при изложении материала первой части книги, заключается не только и не столько в том, чтобы помочь читателям настроить интерфейс программы удобным для работы образом, продемонстрировать, каким образом осуществляется создание макросов, и даже не в том, чтобы предоставить в их распоряжение несколько полезных макросов, облегчающих ввод текста и работу с документами Microsoft Word. "Сверхзадача" этой части книги состоит в том, чтобы после ее прочтения наши читатели убедились, что главным в создании любого действительно полезного макроса являются: во-первых, формулировка ИДЕИ, позволяющей эффективным образом справиться с проблемой, для решения которой создается макрос, и, во-вторых, разработка АЛГОРИТМА (последовательности действий), позволяющего реализовать эту идею на практике.

Естественно, чтобы взять на вооружение этот подход, создатель макросов должен иметь достаточно широкое представление о возможностях Microsoft Word. Поэтому при изложении материала этой книги мы будем также при-

водить дополнительные сведения, касающиеся тех мощных функциональных возможностей этой программы, о наличии которых "рядовые" пользователи часто даже и не подозревают либо имеют довольно смутное представление. Информация о таких возможностях (которые, вообще говоря, часто не имеют непосредственного отношения к созданию макросов), будет приводиться в конце каждой главы в *разд. "Дополнительные сведения"*. Разумеется, мы будем по мере необходимости знакомить с ней наших читателей и при изложении основного материала.

Первое "занятие" мы посвятим проблемам навигации в документах Microsoft Word. В дальнейшем будут рассмотрены и другие возможности и средства, позволяющие заметно облегчить и ускорить процесс ввода и редактирования текста, поиск информации, проверку орфографии, работу с табличными данными и т. п.

Описание действий и процедур, выполняемых при записи макросов, будет основываться на версии Microsoft Word, входящей в состав пакета Microsoft Office 2000. Заметим, однако, что эти же действия могут выполняться и в предыдущей версии программы (Microsoft Word 7, входящей в пакет Microsoft Office 97), а также в Microsoft Word 2002 и Microsoft Office Word 2003. Небольшие отличия при этом касаются лишь названий отдельных команд меню и некоторых новых функций, включенных в последующие версии программы. Вышесказанное, в частности, означает, что почти все макросы, описываемые в этой книге, могут:

- создаваться в среде указанных выше программ;
- использоваться в этих программах.

О редких исключениях из этого правила мы будем сообщать вам по ходу изложения.

> ***Примечание***
>
> В версии Microsoft Word для Windows 95 фактически использовался другой язык программирования, применяемый для записи макросов. При этом макросы, созданные в среде Microsoft Word для Windows 95, практически без проблем могут работать в последующих версиях программы. Однако обратной совместимости нет, т. е. макросы, созданные в более новых версиях программы, не могут использоваться в Microsoft Word для Windows 95.

Приступая к изложению материала, заявленного в начале этой главы, мы все же начнем не с создания макроса, а — "для разгона" — с примера использования одной стандартной для Microsoft Word, но малоизвестной команды, о существовании которой многие пользователи даже и не подозревают. Такие полезные команды мы будем и в дальнейшем помогать вам включать в свой арсенал по мере изложения материала книги.

1.1. Возврат на исходную позицию

В этом разделе вы узнаете, каким образом можно легко и быстро вернуться на текущую позицию в документе, если вы случайно (или преднамеренно) переместились в какую-либо другую его часть.

Ситуация

Вы читаете документ. Затем решили вернуться на предыдущую страницу (нажав клавишу <PgUp>) или в начало документа (с помощью сочетания клавиш <Ctrl>+<Home>), чтобы уточнить некоторые моменты. Или, скажем, случайно нажали сочетание клавиш <Ctrl>+<End> и переместились в конец документа. После этого вы хотите вернуться на исходную позицию, в то место, где прервали чтение. Выполнение этой задачи с помощью привычных клавиш навигации (<PgUp> и PgDn>) займет некоторое время. Если же документ достаточно большой, блуждание по нему может затянуться надолго.

Что требуется получить

Вы нажимаете сочетание "горячих" клавиш, запускающее выполнение соответствующей команды, после чего мгновенно возвращаетесь на исходную позицию.

Полезные сведения

Сочетанием "горячих" клавиш, или просто *сочетанием клавиш*, называется определенная комбинация клавиш, приводящая к автоматическому выполнению тех или иных действий. Одной из клавиш, входящих в такое сочетание, является, как правило, клавиша <Ctrl>, <Alt> или <Shift> (или же их комбинация). Эти клавиши, называемые иногда модификаторами, являются вспомогательными, поскольку изолированное их нажатие, как правило, не приводит к выполнению каких-либо действий. Некоторым командам Microsoft Word сочетания клавиш присвоены по умолчанию, т. е. назначены разработчиками программы. Пользователь может самостоятельно задавать сочетания клавиш для тех команд, у которых они отсутствуют, а также переопределять сочетания клавиш, заданные по умолчанию. Примеры "стандартных" сочетаний клавиш: <Ctrl>+<S> (сохранение документа), <Alt>+<F4> (завершение работы с Microsoft Word), <Ctrl>+<Shift>+<F5> (открытие окна со списком закладок).

Способ решения

Упомянутая выше команда называется **GoBack** (что в переводе означает "вернуться назад"). Правда, если "вытащить" кнопку, соответствующую этой команде, на панель инструментов (как выполняется данная процедура, мы расскажем позднее), на ней будет по-русски написано "Предыдущее испра-

ление". Как говорится, не верь глазам своим. Что-то там разработчики программы немножечко напутали.

Полезные сведения

Каждая команда Microsoft Word имеет два имени: одно из них является служебным и позволяет использовать команды, так сказать, в технических целях (при настройке интерфейса программы, создании макросов и т. п.). Вторым именем является имя, под которым команда отображается в интерфейсе Microsoft Word — в меню, в названиях кнопок и т. п. Первое из указанных имен мы будем называть здесь *"внутренним"*, второе — *"понятным"*. Так, **GoBack** является внутренним именем команды, а "Предыдущее исправление" — понятным (с поправкой на приведенный выше комментарий). Внутреннее имя команды состоит из английских слов (записанных без пробелов), дающих общее представление о назначении соответствующей команды.

Примечание

Знакомство с английским языком (хотя бы на уровне знания перевода наиболее распространенных слов) существенно облегчит вам освоение приемов работы с макросами. Дело в том, что именно этот язык лежит в основе языка программирования Visual Basic for Applications (VBA), на котором осуществляется запись макросов в приложениях пакета Microsoft Office. Для тех читателей, которые не знают английского языка, в конце книги приводится словарь-указатель англоязычных терминов, упоминающихся при описании создания макросов.

По умолчанию данная команда не отображается ни в одном из меню. Однако мы ее все-таки отыщем в недрах Microsoft Word и поставим себе на службу, назначив сочетание клавиш.

Решение задачи

Основные этапы: вначале будут выполнены подготовительные действия, связанные с поиском команды **GoBack** (шаги 1—6). Затем этой команде мы присвоим сочетание клавиш, с помощью которого будет осуществляться ее вызов (шаги 7—8).

1. Откройте меню **Сервис**.
2. Выберите команду **Настройка**, после чего появится соответствующее диалоговое окно (рис. 1.1).
3. Нажмите в нем кнопку **Клавиатура**, после чего появится диалоговое окно **Настройка клавиатуры**.
4. Выберите в списке **Категории** элемент **Правка**, соответствующий одноименному меню Microsoft Word (рис. 1.2).
5. В списке **Команды** отобразится список всех команд, по умолчанию "приписанных" к меню **Правка**.

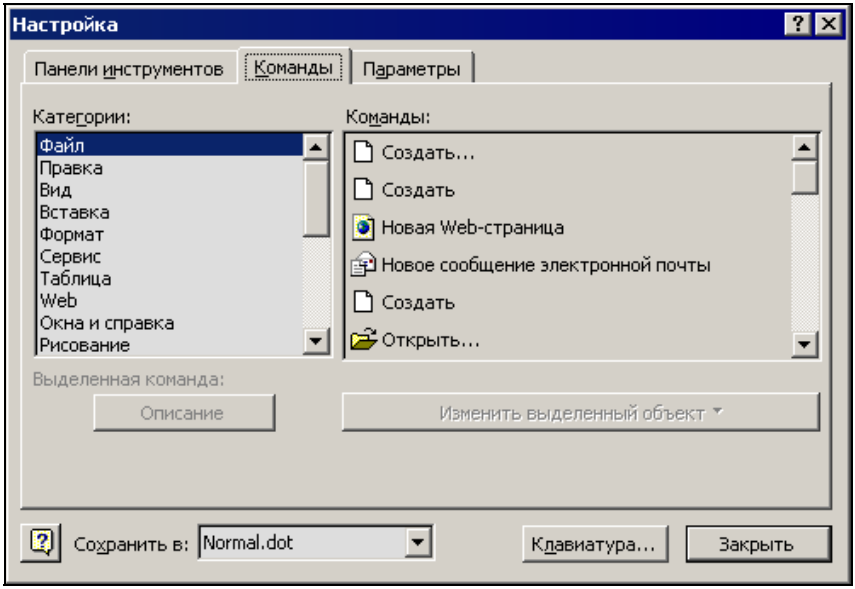


Рис. 1.1. Вкладка **Команды** диалогового окна **Настройка**

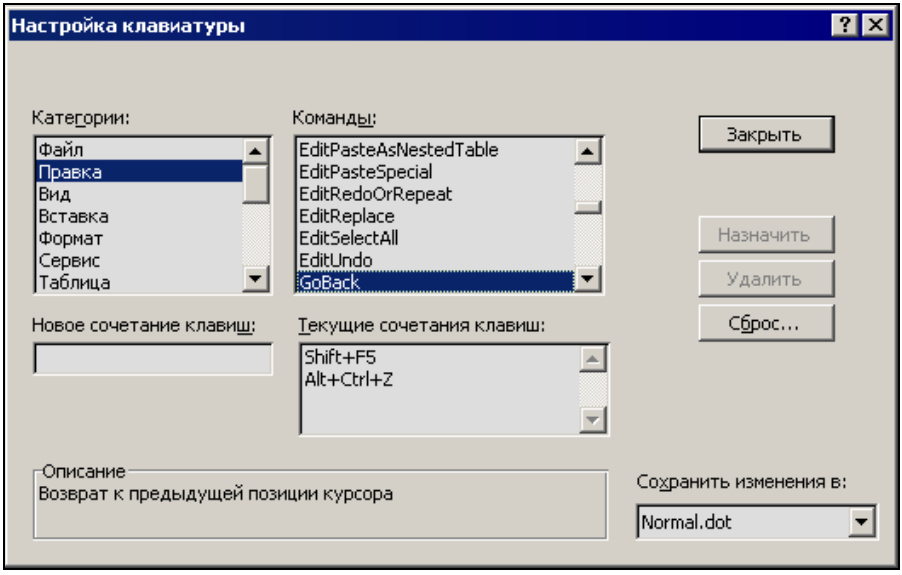


Рис. 1.2. Диалоговое окно **Настройка клавиатуры**

6. Прокрутите список команд и щелкните **GoBack**.

   В списке **Текущие сочетания клавиш** отобразятся сочетания клавиш, нажатие которых приводит к выполнению команды **GoBack**. Как показано на рис. 1.2, разработчики Microsoft Word назначили данной команде аж два таких сочетания: <Shift>+<F5> и <Alt>+<Ctrl>+<Z>. Можно запомнить эти сочетания и использовать их в дальнейшем для перехода на позицию, на которой располагался курсор (точка вставки) на предыдущем шаге.

   Совет

   На первых порах мы советуем вести список стандартных, а также назначенных вами сочетаний клавиш: это сэкономит время, когда потребуется их применить. В дальнейшем мы покажем, каким образом можно автоматически создать список всех назначенных сочетаний клавиш.

   На этом можно было бы завершить решение нашей задачи (для этого следует выполнить действие, описываемое далее на шаге 8). Однако кому-то из вас может показаться, что эти комбинации клавиш являются не очень удобными, поскольку требуют использования двух рук. Поэтому мы покажем вам, каким образом можно выполнить назначение другого, более удобного сочетания, например, <Alt>+<E> (здесь и далее в аналогичных случаях подразумеваются буквы латинского алфавита).

7. Введите сочетание клавиш <Alt>+<E> в поле **Новое сочетание клавиш** и нажмите кнопку **Назначить**, после чего данное сочетание клавиш появится в списке **Текущие сочетания клавиш**.

   Полезные сведения

   Вначале нажимается соответствующая вспомогательная клавиша (модификатор) или клавиши (в нашем случае это клавиша <Alt>). Затем, не отпуская ее (или их), следует коротко нажать "обычную" клавишу (у нас — <E>), после чего все клавиши нужно сразу же отпустить.

   Если введенное вами сочетание клавиш уже занято (т. е. используется для выполнения других команд или макросов Microsoft Word), об этом будет выведено соответствующее сообщение ниже поля **Новое сочетание клавиш.** В этом случае вы можете либо переопределить стандартный вариант, задав свое сочетание, либо удалить предложенный вами вариант и ввести новый. К переопределению стандартных сочетаний клавиш следует подходить с осторожностью, поскольку многие из них традиционно используются для выполнения соответствующих команд (например, <Ctrl>+<F> для выполнения поиска, <Ctrl>+ +<A> — для выделения всего документа и т. п.). Однако в некоторых случаях — когда стандартный вариант используется достаточно редко — на это можно все же пойти. Так, например, сочетания клавиш с использованием модификатора <Alt> используются в Microsoft Word для раскрытия меню и выбора содержащихся в них команд. Однако, как правило, эти действия гораздо удобнее выполнять с помощью мыши, а не клавиатуры.

8. Нажмите кнопку **Закрыть**, чтобы закрыть диалоговое окно **Настройка клавиатуры**.

Что же мы имеем в итоге

Ознакомившись с материалом этого раздела, вы узнали, что в Microsoft Word имеются скрытые от глаз пользователя полезные команды, которые могут применяться для повышения эффективности работы с документами. Вы научились извлекать эти команды и назначать им сочетания клавиш.

Одну из этих команд вы смогли поставить себе на службу. Теперь, чтобы вернуться к тому месту документа, где было прервано его чтение, будет достаточно просто нажать сочетание клавиш <Shift>+<F5> либо <Alt>+ +<Ctrl>+<Z> (или <Alt>+<E>, если вы назначили данной команде это сочетание клавиш). В результате вы возвратитесь в то место документа, где до этого располагался курсор (кстати, краткое пояснение результата применения команды **GoBack** можно прочесть в поле **Описание** диалогового окна **Настройка клавиатуры** — см. рис. 1.2). Если вы нажмете эти сочетания клавиш несколько раз, то переместитесь на "пред-предыдущую" позицию, а затем — и на "пред-пред-предыдущую", после чего цикл повторяется.

Если у вас было открыто несколько документов и вы переходили из одного документа в другой, то при нажатии <Alt>+<E> также переместитесь — вслед за курсором — из одного документа в другой.

> ***Предупреждение***
>
> Внимание! Все вносимые изменения в первоначальные значения параметров Microsoft Word (а также создаваемые макросы) записываются в файл Normal.dot, называемый также общим шаблоном. В зависимости от настроек программы этот файл может сохраняться при выходе из программы либо автоматически, либо по желанию пользователя. В последнем случае выводится сообщение о том, что файл Normal.dot был изменен, и предлагается его сохранить. Не забудьте положительно ответить на это предложение, иначе вся ваша работа по настройке программы и созданию макросов будет безвозвратно утеряна.

1.2. Переход к конкретному фрагменту документа

Довольно часто после первого, достаточно беглого просмотра документа возникает необходимость более внимательно ознакомиться с некоторыми его разделами или отдельными положениями. В этом разделе мы расскажем вам, каким образом можно упростить и ускорить поиск таких фрагментов с помощью созданной для этой цели пары макросов.

Ситуация

Вы читаете документ. К некоторым его фрагментам по каким-то причинам хотели бы в дальнейшем еще вернуться.

Что требуется получить

Вы нажимаете сочетание клавиш в любом месте заинтересовавшего вас фрагмента документа, после чего в него вставляется закладка с текстом стандартного (определенного вами) содержания. Слово или выделенный фрагмент текста, на котором располагался курсор при вставке закладки, подкрашивается некоторым определенным заранее цветом, чтобы его было легче заметить. В дальнейшем для перехода к этой закладке нужно будет просто нажать сочетание клавиш, назначенное соответствующему макросу.

Способ решения

Решение состоит в создании двух простейших макросов (напомним — *автоматическом* создании!), один из которых вставляет в документ простую или цветную закладку, а другой позволяет мгновенно перейти к ней в случае необходимости. Запуск макросов будет осуществляться посредством нажатия заданных вами сочетаний клавиш.

Примечание

Напомним, что закладку в документ Microsoft Word можно вставить с помощью команды **Закладка**, которая находится в меню **Вставка.**

Описываемые ниже действия состоят из достаточно большого количества элементарных шагов. Наберитесь терпения и постарайтесь аккуратно их выполнить. Уже совсем скоро вы научитесь — с помощью таких же простых макросов и команд — значительно упрощать сам процесс создания макросов.

Создание макроса, вставляющего закладку

Основные этапы: вначале будут выполнены некоторые подготовительные действия (шаг 1): размещение курсора на нужной позиции и выделение фрагмента текста. Затем мы осуществим действия, необходимые для "оформления" процедуры создания макроса (шаги 2—9): откроем соответствующее диалоговое окно, присвоим макросу имя и сочетание клавиш. Далее будет создана собственно закладка (шаги 10—11), после чего процедура создания макроса будет завершена (шаг 12).

Совет

Необходимо иметь в виду, что в процессе записи макроса все действия, выполняемые пользователем, автоматически превращаются в инструкции программы. Поэтому во время записи лучше не ошибаться. В противном случае

все ошибочные действия, а также действия по исправлению сделанных ошибок будут повторяться при выполнении записанного макроса. Поэтому все, что нужно для записи макроса, следует подготовить заранее. Кроме того, мы рекомендуем перед записью макроса потренироваться. Тогда вам потребуется меньше "дублей", и весь процесс пройдет гораздо быстрей и спокойнее.

Итак, мы начинаем.

1. Поместите курсор (точку ввода) на любую позицию в пределах нужного фрагмента текста (или для большей наглядности выделите в этом фрагменте несколько слов — рис. 1.3).

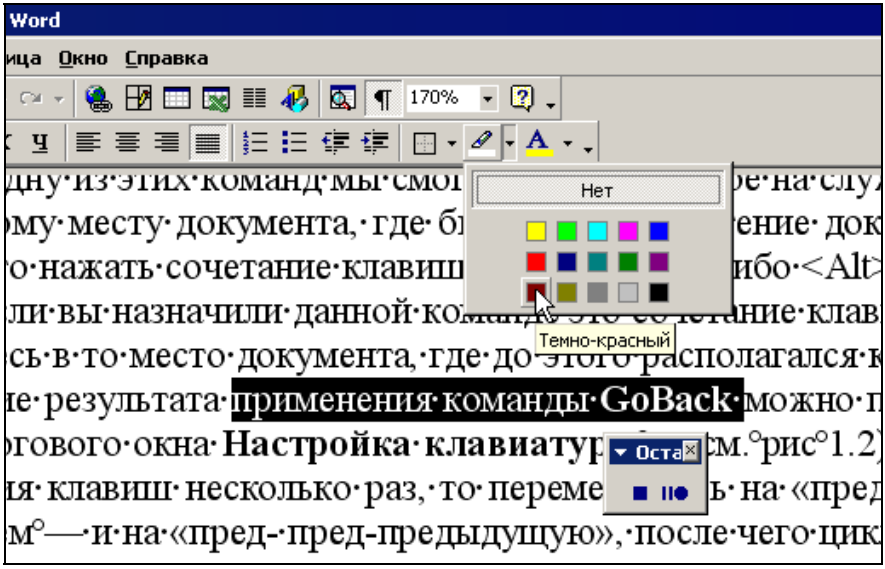


Рис. 1.3. Рабочий момент записи макроса, вставляющего цветную закладку в выделенный фрагмент текста документа

2. Откройте меню **Сервис** и выберите команду **Макрос**.
3. В открывшемся подменю щелкните команду **Начать запись**, после чего появится диалоговое окно **Запись макроса** (рис. 1.4).
4. Введите в поле **Имя макроса** вместо слов Макрос1 название создаваемого макроса, например, **Marker**, что в переводе означает "метка" (советуем имена макросов записывать латиницей во избежание возможных затруднений с отображением кириллицы).

Полезные сведения

При записи имен макросов должны соблюдаться следующие правила. Имя макроса не может начинаться с цифры, содержать пробелы или же знаки, не

являющиеся буквами или цифрами (исключение — знак подчеркивания). Если имя будет записано неправильно, то при переходе к следующему шагу будет выведено соответствующее предупреждение.

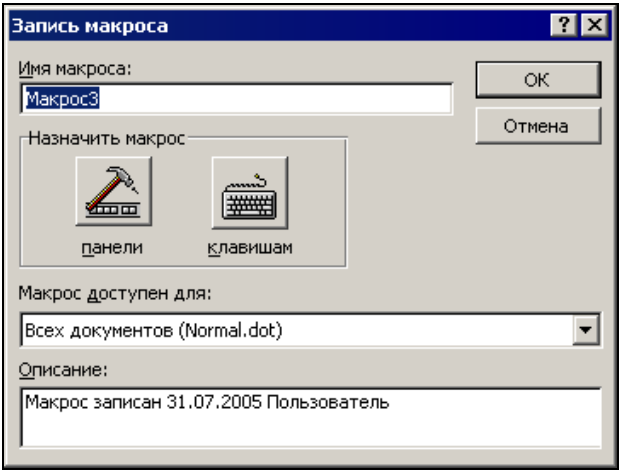


Рис. 1.4. Диалоговое окно **Запись макроса**

5. В поле **Описание** введите после сведений об авторе создаваемого макроса, например, следующий поясняющий текст "Вставка закладки Marker".
6. В поле со списком **Макрос доступен для** оставьте предлагаемое по умолчанию значение (см. рис. 1.4). Это обеспечит возможность использовать записываемый макрос во всех документах Microsoft Word.
7. В группе **Назначить макрос** нажмите кнопку **клавишам**, после чего появится диалоговое окно **Настройка клавиатуры**.

Совет

Если нажать кнопку **панели**, то в дальнейшем запуск создаваемого макроса можно будет осуществлять с помощью специальной кнопки, размещаемой на панели инструментов. Как создается такая кнопка, мы рассмотрим в следующей главе, сейчас же сформулируем следующую рекомендацию: сочетания клавиш лучше использовать в случаях, когда работать приходится в основном руками, точнее пальцами, а не мышью (чтобы не снижать темп при вводе текста), и пользоваться макросом или командой, которой назначено это сочетание, приходится достаточно часто. Кнопки же лучше применять в тех случаях, когда пользователь работает в основном мышью, а выполнение соответствующих действий имеет эпизодический характер (сочетание клавиш в этом случае запомнить трудно).

8. Введите удобное для себя сочетание клавиш, например, <Alt>+<A>, в поле **Новое сочетание клавиш** и нажмите кнопку **Назначить**. Данное сочетание клавиш появится в списке **Текущие сочетания клавиш**.

Примечание

Данное сочетание клавиш по умолчанию применяется для раскрытия меню **Файл**. Однако мы все же предлагаем переназначить его, чтобы использовать для запуска нашего макроса. Дело в том, что, как уже отмечалось ранее, доступ к командам меню с помощью клавиатуры применяется, по нашим наблюдениям, довольно редко. Впрочем, если вы не согласны с этим утверждением, то можете воспользоваться другим сочетанием клавиш, применение которого посчитаете более обоснованным.

9. Нажмите кнопку **Закрыть**, чтобы закрыть диалоговое окно **Настройка клавиатуры**.
10. На экране появится панель управления записью макроса — небольшой прямоугольник с двумя кнопками, на одной из которых нарисован маленький синий квадратик (кнопка **Остановить запись**), а на другой — две вертикальные полоски и жирная точка (кнопка **Пауза/Возобновить запись**) — см. рис. 1.3.

    Внимание! Наступил торжественный и ответственный момент. С этих пор *все* предпринимаемые вами действия — перемещение по документу, открытие меню, исполнение команд и т. п. — будут автоматически записываться в текст программы создаваемого макроса на языке Visual Basic. Поэтому старайтесь не совершать каких-либо необязательных, лишних действий.

    Впрочем, в любой момент вы можете на время прервать запись макроса, чтобы уточнить способ выполнения тех или иных действий или посмотреть результат применения тех или иных команд. Для этого нужно нажать кнопку с двумя вертикальными полосками и синей (в Microsoft Office Word 2003 — красной) точкой на панели записи макроса. Чтобы продолжить запись, нажмите данную кнопку еще раз.

Полезные сведения

В процессе создания макроса в текст соответствующей программы будут записываться все действия, выполняемые с использованием клавиатуры, однако далеко не все действия, производимые с помощью мыши. Так, допускается применение мыши для выбора команд и параметров, однако окно документа (т. е. непосредственно его текст) является для нее недоступным. Для записи действий, производимых в окне документа (перемещение курсора, выделение, копирование и перемещение текста), необходимо пользоваться только клавиатурой. Кстати, это еще одна причина, по которой стоит потренироваться перед записью макроса — "в обычной жизни" мы не пользуемся клавиатурой для реализации тех действий, которые удобнее выполнять мышью, и иногда просто не знаем, как их можно выполнить без мыши.

11. Откройте меню **Вставка** и щелкните по команде **Закладка**, после чего появится диалоговое окно **Закладка** (рис. 1.5).

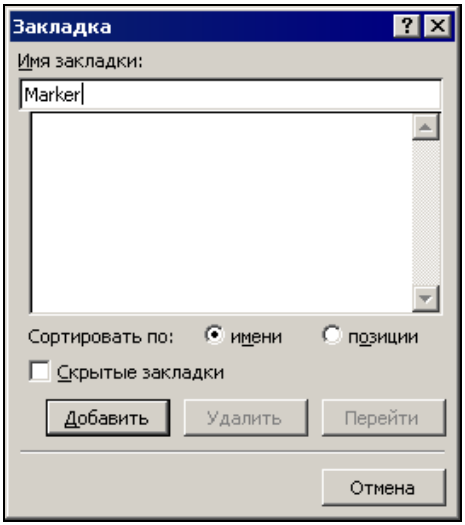


Рис. 1.5. Диалоговое окно **Закладка**

12. Введите в поле **Имя закладки** слово **Marker** (или любое другое, которое сочтете более уместным) и нажмите кнопку **Добавить**.

Полезные сведения

Ввод имени закладки осуществляется по тем же правилам, что и названия макроса. Имя не может начинаться с цифры, содержать пробелы или же знаки, не являющиеся буквами или цифрами (исключение — знак подчеркивания). Если имя будет записано неправильно, кнопка **Добавить** окажется недоступной.

13. Нажмите кнопку с синим квадратиком, расположенную на панели записи макроса, чтобы завершить процесс его создания.

Поздравляем! Ваш первый макрос создан. Как видите, ничего сложного. А если бы вы еще посмотрели на текст созданной вами (точнее, редактором Visual Basic) при этом программы, то, наверняка, на порядок повысили бы собственную самооценку. Но подробнее об этом — в следующих частях книги. Там же будет рассказано о том, как можно усовершенствовать данный макрос, "научив" его самостоятельно давать имена вставляемым закладкам *(см. гл. 15)*.

Полезные сведения

Чтобы удалить в случае необходимости закладку, нужно открыть диалоговое окно **Закладка** (см. шаг 11), щелкнуть нужную закладку и нажать кнопку **Удалить**. В *гл. 15* будет описан макрос, удаляющий все закладки (или определенный их набор).

Запуск макроса

Если при записи макроса вы назначили ему сочетание клавиш или кнопку на панели инструментов, то вопрос о том, как запустить макрос, у вас не возникнет (если только вы не забудете, какое сочетание клавиш или кнопку ему назначили). В любом случае, макрос можно запустить непосредственно из диалогового окна **Макрос**.

Чтобы открыть диалоговое окно **Макрос**, в меню **Сервис** выберите **Макрос**, а затем — **Макросы**.

Перемещаясь по списку макросов в диалоговом окне **Макрос** и просматривая их описания (тут самый момент похвалить себя за то, что вы не ленились их вводить), вы найдете нужный макрос. Чтобы запустить нужный макрос, следует нажать кнопку **Выполнить** или же просто дважды щелкнуть его имя в списке.

Создание цветной закладки

При желании вы можете сделать закладку, вставляемую с помощью макроса, более заметной, подкрасив расположенный рядом с ней фрагмент текста (такой закладке можно дать и другое имя на шаге 4, например, **ColorMarker**, т. е. "цветная метка"). Для этого необходимо дополнить описанную выше процедуру создания макроса несколькими шагами. Выполните после шага 9 этой процедуры следующие действия:

1. Нажмите на клавиатуре сочетание клавиш <Ctrl>+<Shift>+<→>, чтобы выделить расположенное справа слово (это делается на случай, если в будущем вы забудете перед выполнением макроса выделить фрагмент текста).
2. Щелкните маленький треугольничек рядом с инструментом **Выделение цветом** на панели инструментов (см. рис. 1.3). Раскроется палитра цветов, которые могут использоваться для подкрашивания выделенного текста (название цвета отображается, если навести на ту или иную ячейку указатель мыши).
3. Щелкните ячейку с нужным цветом, например, темно-красным.

Далее следует вернуться к описанной ранее процедуре создания макроса, т. е. к ее шагу 10.

> ***Полезные сведения***
>
> Если вы впоследствии захотите удалить выделение цветом, то проще всего это можно сделать так: выделите весь документ с помощью стандартного сочетания клавиш <Ctrl>+<A>, затем выберите инструмент **Выделение цветом** на панели инструментов (см. шаг 10 ранее) и щелкните кнопку **Нет** в палитре цветов. Если в документе есть такие области, как надписи, сноски, колонтитулы или примечания, то эту операцию необходимо будет повторить в каждой такой области. Подробнее об этом рассказано *в гл. 16*.

Создание макроса, выполняющего автоматический переход к закладке

1. Первые 10 шагов процедуры создания данного макроса полностью совпадают с соответствующими этапами записи макроса, вставляющего (нецветную) закладку, и поэтому мы их опускаем. Отметим только, что на шаге 4 при определении имени нового макроса следует ввести, к примеру, выражение **GoToMarker**, на шаге 5 (описание макроса) — "Переход к закладке Marker", а на шаге 8 (задание сочетания клавиш) — <Alt>+<Q>.
2. На данном шаге (находясь уже в режиме записи макроса) щелкните по закладке **Marker** в большом окне под полем **Имя закладки** диалогового окна **Закладка** (см. рис. 1.5).
3. Нажмите кнопку **Перейти**, а затем кнопку **Закрыть**, которая появится на месте кнопки **Отмена** в диалоговом окне **Закладка**.
4. Наконец, завершите создание макроса, нажав кнопку с синим квадратиком, расположенную на панели записи макроса.

> ***Примечание***
>
> Разумеется, после создания этого макроса переход к закладке может осуществляться не только в результате нажатия назначенного вами сочетания клавиш, но и с помощью стандартной для Microsoft Word процедуры. Для этого следует открыть диалоговое окно **Закладка**, выбрав команду **Закладка** в меню **Вставка** или нажав стандартное сочетание клавиш <Ctrl>+<Shift>+<F5>; затем нужно будет щелкнуть имя закладки, нажать кнопку **Перейти**, а затем — кнопку **Закрыть**. Легко можно подсчитать, что в результате создания нашего макроса вы будете затрачивать на каждый переход к закладке на 3—4 щелчка мыши (нажатия клавиш) меньше, чем при использовании стандартного способа. Неплохой результат, не правда ли?

Что же мы имеем в итоге

Нажав в любом месте документа <Alt>+<A> или какое-либо другое назначенное вами сочетание клавиш, вы вставите в это место закладку (при желании — даже цветную). Если же потребуется вернуться к заинтересовавшему вас фрагменту, вы сможете мгновенно это сделать, нажав <Alt>+<Q> (или какое-либо иное назначенное вами сочетание клавиш).

Создание других закладок

Практика показывает, что при работе с документом полезно иметь несколько стандартных закладок, например, **Marker**, **Marker_2**, **Return_and_Correct** и т. п.

Каждая из них должна выделяться своим цветом; для вставки и перехода к ним необходимо назначать свои сочетания клавиш или же создавать специальные кнопки, размещаемые на панели инструментов.

При создании новых пар макросов типа "вставка закладки/переход к закладке" можно, конечно, повторить описанные выше действия. Однако имеется и более рациональный и быстрый способ. Он также не требует умения программировать. Потребуется просто с помощью обычных средств Microsoft Word выполнить "клонирование" текста макроса-прототипа и внести в него некоторые вполне понятные и логичные изменения. Однако мы договаривались, что пока не будем иметь дела с текстами программ. Поэтому желающие узнать, как это делается, могут обратиться к "переходной" главе этой книги *(см. гл. 6)*, где приводятся некоторые общие сведения о редакторе Visual Basic и описываются простейшие приемы создания программ, не требующие от пользователя навыков программирования.

Полезные сведения

Если после создания макроса вы станете при запуске Microsoft Word получать предупреждение системы безопасности, в котором сообщается, что в загружаемых макросах могут содержаться вирусы, мы рекомендуем вам выполнить следующее. Откройте меню **Сервис**, укажите пункт **Макрос** и в открывшемся подменю выберите команду **Безопасность**, перейдите на вкладку **Надежные источники** (в Microsoft Office Word 2003 — **Надежные издатели**) открывшегося одноименного диалогового окна и установите флажок **Доверять всем установленным шаблонам и надстройкам**. Более подробно о вопросах безопасности рассказывается *в разд. 7.1.3.*

1.3. Дополнительные сведения

Как уже отмечалось, в конце большинства глав книги мы будем приводить дополнительные сведения, имеющие непосредственное отношению к материалу, изложенному в основном тексте. Эти сведения не являются "обязательными" для изучения, однако, с нашей точки зрения, они весьма полезны как справочная информация, которая может пригодиться нашим читателям не только при знакомстве с содержанием этой книги, но и в дальнейшей работе с Microsoft Word.

1.3.1. Дополнительные сведения о записи макросов

Открыть диалоговое окно **Макрос** можно также с помощью сочетания клавиш <Alt>+<F8>.

Правила записи имени макроса

Имя макроса может состоять только из букв и цифр (допустим также знак подчеркивания). Оно не должно начинаться с цифры или содержать пробелы. Длина имени макроса не должна превышать 80 знаков.

Это имя (прописными буквами) будет отображаться во всплывающей подсказке, если при записи макроса назначить кнопку панели инструментов. Кнопку макросу можно назначить и позже, и тогда во всплывающей подсказке будет учитываться регистр букв в имени макроса.

Если вы рассчитываете работать только с русифицированными продуктами, то можно использовать русские буквы, в противном случае мы советуем писать латиницей во избежание возможных затруднений с отображением кириллицы.

> ***Предупреждение***
>
> Если имя макроса совпадает с "внутренним" именем команды Microsoft Word, то при вызове соответствующей команды вместо нее будет выполняться ваш макрос. Это очень мощное, но вместе с тем весьма коварное свойство. Особенно неприятно, что при создании макроса вы не получите об этом никаких предупреждений. Чтобы оценить размеры бедствия, представьте себе, что вы присвоили своему макросу имя Italic, совпадающее с внутренним именем встроенной команды Microsoft Word. После этого данная команда становится недоступной, а при попытке воспользоваться ею будет выполняться созданный вами одноименный макрос. Сталкиваться с подобными ситуациями приходится, конечно, не так уж часто, однако если вы хотите совершенно исключить возможность их возникновения, то можете, разработать свою собственную систему именования макросов, например, записывать в начале имени каждого макроса какое-либо уникальное сочетание букв.

Перечень встроенных команд и макросов Microsoft Word

Чтобы просмотреть и при желании сохранить или распечатать список встроенных команд и макросов Microsoft Word, выберите в меню **Сервис** пункт **Макрос**, а затем — **Макросы**. В поле со списком **Макросы из** выберите **Команд Word**.

Для вывода списка всех встроенных команд и макросов выберите в окне, расположенном над полем **Макросы из**, элемент **ListCommands** (list — список, commands — команды) или непосредственно введите его с клавиатуры в поле **Имя**, а затем нажмите кнопку **Выполнить**. В открывшемся диалоговом окне **Список команд** установите переключатель **Включить в новый документ** в положение **все команды Word** и нажмите кнопку **OK**.

После этого автоматически создается новый документ, в который записывается таблица, в первом столбце которой отображаются "внутренние" имена всех встроенных команд, а также стилей и созданных пользователем макросов Microsoft Word. Во втором столбце этой таблицы указываются клавиши-модификаторы (<Alt>, <Ctrl>, <Shift>), назначенные соответствующим встроенным элементам (за исключением макросов), в третьем — клавиши, используемые в сочетании с данными модификаторами, в четвертом — меню или панель инструментов (при их наличии), в которых по

умолчанию располагаются соответствующие элементы, указанные в первом столбце таблицы.

Описание макроса

В диалоговом окне **Макрос** имеется поле **Описание**. По умолчанию в нем отображаются сведения о дате создания макроса и его авторе. Этот текст можно удалить или внести какие-либо любые изменения. Если вы при создании макроса ввели в одноименное поле диалогового окна **Создание макроса** какой-либо текст (например, поясняющий назначение макроса), то он также будет отображаться в этой области окна **Макрос**. Советуем не пренебрегать этой возможностью. Когда арсенал ваших макросов будет насчитывать несколько десятков позиций, описание макроса поможет вам вспомнить, для чего вы когда-то его создавали. Это описание вы также увидите в виде комментария, если будете работать с текстом записанного вами макроса в редакторе Visual Basic. В таком описании может содержаться до 255 знаков.

Где хранится макрос

В поле со списком **Макрос доступен для** диалогового окна **Запись макроса** можно выбрать шаблон или документ, в котором будет храниться макрос. Использование шаблонов и связанные с этим вопросы безопасности мы рассмотрим далее, в следующих частях книги.

1.3.2. Дополнительные сведения о закладках

Открыть диалоговое окно **Закладка** можно также с помощью сочетания клавиш <Ctrl>+<Shift>+<F5>.

Правила записи имени закладки

При записи имен закладок действуют те же правила, что и при записи макросов. Имя не может начинаться с цифры, содержать пробелы, а также знаки, не являющиеся буквами или цифрами (исключение — знак подчеркивания). Длина имени закладки не должна превышать 40 знаков. Если введенное имя не является допустимым, кнопка **Добавить** недоступна. Если вы рассчитываете работать только с русифицированными продуктами, то можно использовать русские буквы, в противном случае мы советуем писать латиницей во избежание возможных затруднений с отображением кириллицы.

Элементы управления диалогового окна *Закладка*

Имя закладки однозначно идентифицирует закладку. В документе не может быть две закладки с одинаковым именем (не забывайте об этом, когда тем или иным способом добавляете в документ текст с закладкой).

Если нажать кнопку **Добавить**, когда в поле **Имя закладки** указано имя уже существующей закладки, эта закладка переместится на новое место — в ту точку, в которой в данный момент располагается курсор.

Имя существующей закладки можно выбрать из приведенного под этим полем списка. Если переключатель **Сортировать по** установлен в положение **имени**, закладки перечислены в алфавитном порядке; если он установлен в положении **позиции**, они расположены в том порядке, в котором встречаются в документе.

Если установлен флажок **Скрытые закладки**, то в перечень добавляются закладки, которые устанавливаются автоматически (например, при создании оглавления, перекрестных ссылок и т. п.). Кнопки **Добавить**, **Удалить** и **Перейти** позволяют выполнить соответствующие действия с закладкой, имя которой указано в поле **Имя закладки**. Нажатие кнопки **Отмена** закрывает диалоговое окно без выполнения каких-либо действий с закладками

Как включить режим отображения закладок

Если закладки не видны на экране (в распечатке документа они не отображаются никогда), их можно сделать заметными. Для этого следует выполнить следующие действия:

1. В меню **Сервис** выберите команду **Параметры**.
2. В диалоговом окне **Параметры** перейдите на вкладку **Вид**.
3. В группе **Показывать** установите флажок **Закладки**.
4. Нажмите кнопку **ОК**. Диалоговое окно **Параметры** закроется.

В результате границы каждой закладки (кроме тех, которые являются скрытыми) будет отображаться на экране в виде соответственно открывающей и закрывающей квадратной скобки. Если же левая и правая граница закладки совпадают, то эти скобки сливаются и превращаются в значок, напоминающий букву I в шрифте с засечками.

Где еще используются закладки

Переход к закладке можно также выполнить, используя вкладку **Перейти** в диалоговом окне **Найти и заменить**. Закладки используются также при создании гиперссылок и размещении кнопок перехода с помощью полей. В дальнейшем мы будем широко применять закладки при записи макросов.

Полезные сведения

Если расставить закладки в колонтитулах документа Microsoft Word, текстовых рамках, примечаниях или сносках, то не на все закладки удается перейти из любого места этого документа. Даже не все такие закладки отображаются на вкладке **Перейти** диалогового окна **Найти и заменить**. Авторы не встречали в известной им литературе упоминание об этой особенности Microsoft Word, но

факт остается фактом — попробуйте и убедитесь сами, что некоторые области документа остаются недоступными. Чтобы обойти это неприятное ограничение, нам потребуется написать специальный макрос. Но займемся мы этим в следующих частях книги *(см. гл. 15)*.

1.4. Что нового в этой главе

Подведем некоторые итоги и коротко перечислим, с какими новыми сведениями вы познакомились в этой главе. Подобная сводка будет приводиться и в следующих главах книги. Это поможет вам не только контролировать ход изложения материала, но и позволит составить себе более четкое представление о новых возможностях, которые появились у вас после изучения материала очередной главы. Кроме того, наличие такой сводки может также оказаться полезным и при поиске нужной вам информации.

В этой главе вы узнали о том, что в Microsoft Word имеется довольно много встроенных команд, которые не отображаются в стандартных меню программы. Вы узнали также, где можно посмотреть список этих команд и каким образом их можно поставить себе на службу. Мы выяснили, что у команд Microsoft Word имеется два имени — внутреннее, которое используется при настройке интерфейса программы, создании макросов и в других вспомогательных "технических" целях, а также "понятное" — т. е. имя под которым эти команды отображаются в интерфейсе программы. Одной из этих команд, имеющей внутреннее имя **GoBack**, вы назначили сочетания клавиш и теперь можете легко и быстро вернуться на текущую позицию в документе, если по каким-то причинам переместились на другую страницу.

Вы узнали, что многие команды Microsoft Word можно выполнять, не прибегая к помощи меню или кнопок панелей инструментов, а нажимая для этого (правильным образом!) определенные сочетания клавиш. Кроме того, вы научились назначать командам новые и изменять уже имеющиеся сочетания клавиш.

Те из наших читателей, которые не были знакомы с таким эффективным инструментом работы с документом Microsoft Word, как закладка, приобрели некоторые навыки его практического использования.

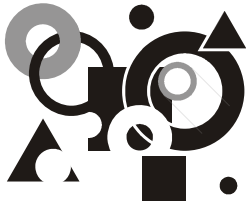
Вы начали осваивать процедуру автоматического создания макросов и создали свой первый макрос (еще раз примите наши поздравления), позволяющего вставлять закладку (по желанию — даже цветную) на любую позицию в документе. С помощью второго из созданных вами макросов вы можете теперь в любое удобное для вас время перейти в помеченный закладкой фрагмент документа и более внимательно ознакомиться с его содержанием, которое по тем или иным причинам привлекло ваше внимание.

Читатели, ознакомившиеся с материалом, представленным в разделе дополнительных сведений, имеют теперь более полное представление о правилах записи имен макросов и закладок, а также об ограничениях, накладываемых на их использование.

Эти читатели научились просматривать и выводить на печать полный список всех встроенных команд и стилей форматирования Microsoft Word (вместе с назначенными им сочетаниями клавиш), а также созданных пользователем макросов.

Здесь же приводятся сведения об элементах диалогового окна **Закладка**, видах закладок и особенностях их использования в Microsoft Word.

В следующих главах книги вы сможете закрепить приобретенные в данной главе навыки, а также познакомитесь с другими — в ряде случаев малоизвестными большинству пользователей — средствами и возможностями Microsoft Word.



Глава 2

Навигация продолжается: теперь с помощью кнопок

Команды постраничного листания документа. Размещение стандартной кнопки на панели инструментов. Настройка кнопки. Перемещение в начало и конец документа. Дополнительные сведения.

В этой главе мы продолжим знакомиться со встроенными командами Microsoft Word и создавать макросы, упрощающие и ускоряющие перемещение (навигацию) по документу. Однако главный упор будет в ней сделан на решении задач, связанных с назначением кнопок стандартным командам и создаваемым макросам.

2.1. Команды постраничного листания документа

Просмотр электронного документа обычно осуществляется посредством листания его виртуальных страниц. Обычно для этого используются клавиши <PgDn> и <PgUp> на клавиатуре компьютера. Однако для пользователей, привыкших работать с помощью мыши, этот способ является не очень удобным. Поэтому мы воспользуемся случаем и попробуем облегчить их жизнь, создав специальные кнопки, позволяющие листать документ, не прибегая к помощи клавиатуры.

Полезные сведения

В Microsoft Word имеется режим просмотра, позволяющий осуществлять быструю навигацию по разделам документа с помощью мыши, если их названия оформлены стилями, используемыми для выделения заголовков. Чтобы перейти в этот режим, необходимо нажать кнопку **Схема документа**  , которая по умолчанию располагается на панели инструментов **Стандартная**. После этого в правой части документа появляется область, в которой отображается оглав-

ление документа. Пункты этого оглавления являются гиперссылками, щелчок по которым приводит к мгновенному перемещению к заголовку соответствующего раздела документа.

Ситуация

Вы просматриваете документ с целью оценить внешний вид публикации, не вчитываясь особенно в ее содержание. Для этого вы перешли в режим разметки страницы (меню **Вид**, команда **Разметка страницы**) и уменьшили масштаб отображения таким образом, чтобы страница полностью умещалась на экране. (Впрочем, вы можете в этом режиме даже читать документ, если являетесь счастливым обладателем широкоформатного монитора.) Обычно такого рода задачи решаются с помощью режима предварительного просмотра, однако этот режим не всегда удобен, поскольку возможности работы с документом в нем ограничены.

Полезные сведения

Переход в режим предварительного просмотра может осуществляться с помощью команды **Предварительный просмотр** меню **Файл** или одноименной кнопки  , расположенной на панели инструментов **Стандартная**. По умолчанию данный режим активируется с включенной функцией **Увеличение**. Благодаря этому пользователь, щелкая кнопкой мыши, может изменять масштаб отображения документа. Однако средства редактирования в этом случае остаются недоступными. Мало кому из пользователей известно, что и в данном режиме редактировать документ все-таки можно: для этого следует отключить функцию **Увеличение**, нажав одноименную кнопку  на панели инструментов **Предварительный просмотр** (тем самым фактически придется отказаться от преимуществ режима предварительного просмотра). Данная панель инструментов не входит в список стандартных панелей инструментов Microsoft Word и включается в него только при переходе в режиме предварительного просмотра.

При этом для листания документа вам приходится использовать клавиши <PgDn> и <PgUp>, хотя вы предпочли использовать для этой цели кнопку мыши. Кроме того, использование этих клавиш навигации часто приводит к ситуации, когда на экране одновременно может отображаться нижняя часть текущей страницы и верхняя — следующей. Дело здесь в том, что, при их нажатии фактически происходит переход не к следующей странице документа, а к следующему *экрану*. Поэтому если настройка отображения страницы на экране была выполнена неточно, происходит смещение представленного на нем изображения.

Примечание

Указанные выше недостатки частично были устранены в версии Microsoft Word, входящей в пакет Microsoft Office 2003. В нее был добавлен новый режим ото-

бражения документа, получивший название **Режим чтения**. Переход в него осуществляется с помощью одноименной команды меню **Вид**. Как следует из названия этого режима, он предназначен именно для чтения документа, а не для просмотра его общего вида. В этом режиме происходит увеличение масштаба отображения документа удобным для чтения образом и на экране размещается целая его страница. Благодаря этому появляется возможность "правильного" листания документа с помощью клавиш <PgDn> и <PgUp>. В этом случае не может возникнуть ситуация, когда на экране одновременно отображаются нижняя часть текущей страницы и верхняя — следующей страницы, однако следует иметь в виду, что из-за изменения масштаба отображения документа его "экранные" страницы не будут совпадать с реальными.

Что требуется получить

Вы нажимаете кнопку на панели инструментов, и на экран выводится следующая страница. Чтобы вернуться на страницу назад, вы нажимаете другую кнопку. Таким образом осуществляется постраничное листание документа.

Способ решения

Решить данную задачу можно, не создавая для этого каких-либо специальных макросов. В Microsoft Word имеются соответствующие стандартные команды, которые, однако, как и в случае, рассмотренном в предыдущей главе, изначально (по умолчанию) не отображаются ни в одном из меню программы. Мы отыщем эти команды, назначим им индивидуальные кнопки и разместим на панели инструментов.

Решение задачи

Основные этапы: вначале будут выполнены подготовительные действия, связанные с поиском команды **GoToNextPage** (шаги 1—4). Далее посредством перетаскивания соответствующего ей элемента на панель инструментов создается кнопка, с помощью которой будет осуществляться запуск данной команды (шаг 5).

На языке программного обеспечения Microsoft Word интересующие нас команды имеют вполне осмысленные имена — **GoToNextPage** и **GoToPreviousPage**, что в переводе соответственно означает "Перейти к следующей странице" и "Перейти к предыдущей странице". Поэтому мы сможем без труда их отыскать в сводном списке всех команд Microsoft Word, который отображается в диалоговом окне **Настройка**.

Для этого необходимо выполнить следующие действия:

1. Откройте меню **Сервис**.
2. Выберите команду **Настройка**, после чего на экране появится одноименное диалоговое окно.

3. Выберите на вкладке **Команды** в списке **Категории** элемент **Все команды**.
4. Прокрутите список **Команды** и выберите в нем элемент **GoToNextPage**.
5. Перетащите мышью (нажав и не отпуская ее левую кнопку) элемент **GoToNextPage** на любую позицию на любой панели инструментов (эта позиция может располагаться и между уже существующими кнопками), после чего в этом месте появится новая кнопка с надписью **Следующая страница** (рис. 2.1).

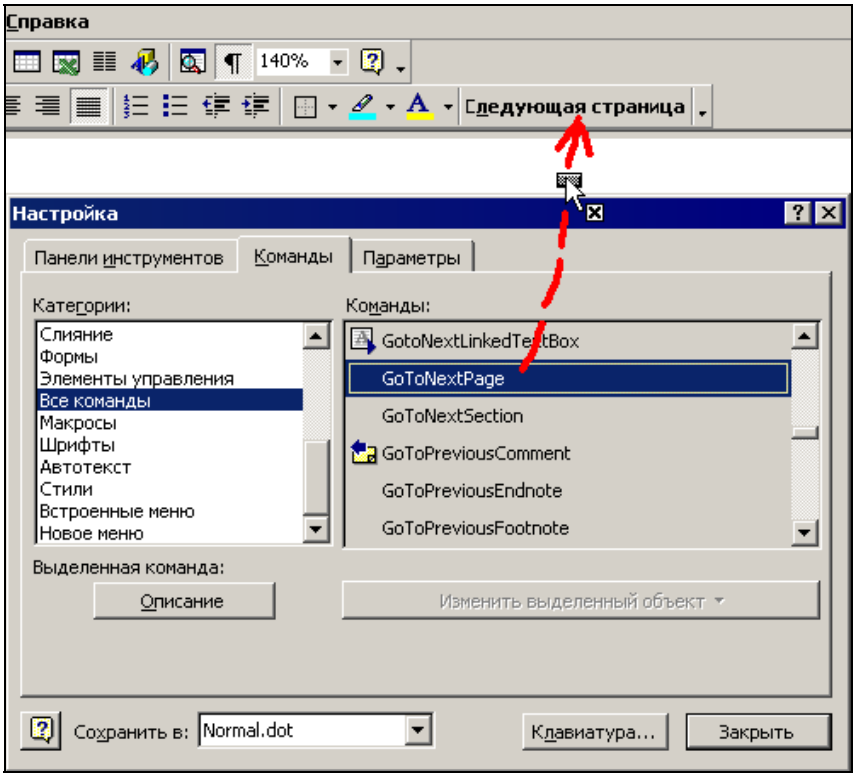


Рис. 2.1. Перетаскивание кнопки, соответствующей команде **GoToNextPage**, на панель инструментов

6. Нажмите кнопку **Закрыть** диалогового окна **Настройка**, чтобы завершить процесс создания кнопки для команды перехода к следующей странице.

Аналогичным образом осуществляется размещение на панели инструментов кнопки для команды возврата на предыдущую страницу. Разница будет состоять только в том, что в списке **Команды** нужно будет отыскать элемент **GoToPreviousPage**, а на кнопке будет написано **Предыдущая страница**.

Что же мы имеем в итоге

Благодаря созданным нами кнопкам вы получили возможность листать документ, не прибегая к помощи клавиатуры. Работая с документом, вы нажимаете кнопку **Следующая страница**, чтобы перейти на следующую страницу, а для возврата на страницу назад — кнопку **Предыдущая страница**.

2.2. Создание пользовательской кнопки и изменение ее внешнего вида

На этом можно было бы закончить процесс создания кнопки для команд постраничного листания документа. Однако, как видно на рис. 2.1, кнопочка у нас получилась немного великоватой. Пока ничего страшного в этом нет, поскольку на панелях инструментов у нас размещен пока только стандартный набор кнопок (плюс две созданные в этой главе). Однако нужно уже сейчас думать о будущем, когда кнопок у нас будет много и возникнет проблема их компактного размещения. Поэтому самое время познакомиться с имеющимися в Microsoft Word основными приемами "художественного" оформления кнопок (впрочем, эти приемы являются общими для всех приложений Microsoft Office). Вообще говоря, существуют три таких способа:

- оставить на кнопке только текст (стандартного или произвольного содержания);
- отобразить на кнопке некоторый рисунок-значок (стандартный или созданный пользователем);
- одновременно отобразить и текст и значок.

Каким из этих способов воспользоваться, каждый пользователь решает, конечно, самостоятельно. Мы же приведем краткое описание каждой из названных возможностей.

Давайте посмотрим, как с помощью указанных выше способов можно модифицировать созданную нами кнопку **Следующая страница**.

Попробуем сначала несколько сократить отображаемый на кнопке текст, сохранив его смысл, но сделав кнопку при этом более компактной. Можно, например, разместить на кнопке текст "Сл. стр." или — как на соответствующей клавише — <PgDn>. Для этого нужно (не закрывая диалогового окна **Настройка**) поместить указатель мыши над только что созданной кнопкой и нажать правую кнопку мыши. Появится контекстное меню, с помощью команд которого будет проводиться модификация кнопки (рис. 2.2). По умолчанию активной в нем является команда **Только текст (всегда)**, поскольку на кнопке указано лишь название соответствующей команды и отсутствует какой-либо значок. Чтобы изменить отображаемый на кнопке текст, следует щелкнуть поле **Имя** этого меню и внести необхо-

димые коррективы в отображаемый в данном поле текст. После завершения ввода нужного текста следует нажать клавишу <Enter>.

Полезные сведения

На рис 2.2 видно, что перед буквой "л" в слове "Следующая", расположенном в поле **Имя**, стоит символ "&". Этот символ имеет служебный характер. Его включение в имя команды или название меню при настройке элементов интерфейса программы позволяет выбирать соответствующую команду (раскрывать меню) с помощью клавиатуры, а не щелчком кнопки мыши. Наличие символа "&" в названии соответствующего элемента интерфейса проявляется в подчеркивании буквы, расположенной в названии после него. Так, в нашем случае в слове "Следующая" будет подчеркнута буква "л" (см. рис. 2.1). Чтобы раскрыть меню с помощью клавиатуры, следует сначала нажать клавишу <Alt> и, не отпуская ее, букву, подчеркнутую в названии меню. Если требуется выбрать какую-либо команду в раскрытом таким образом меню, необходимо — не отпуская клавишу <Alt> — нажать букву, подчеркнутую в имени этой команды. Следует, однако, иметь в виду следующие два момента. Во-первых, в случае, когда пользователь ранее уже присвоил соответствующее сочетание клавиш какой-либо команде или макросу, описанная здесь возможность теряет свою силу. Во-вторых, наличие подчеркнутой буквы в надписи, отображаемой на кнопке, не означает, что описанный нами способ может использоваться для выполнения команды или запуска макроса, которому эта кнопка назначена. Для этого следует просто нажать соответствующую кнопку.

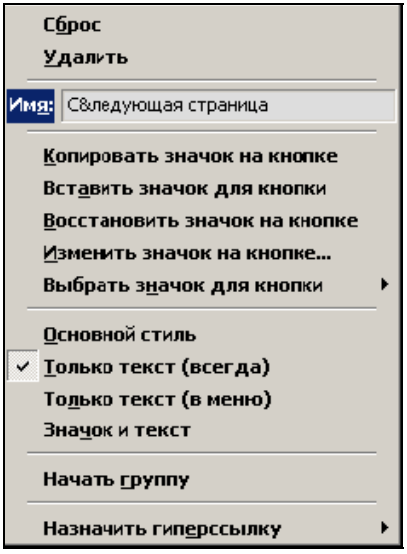


Рис. 2.2. Контекстное меню
с командами редактирования внешнего вида кнопки

Следующий из доступных в меню способов оформления кнопки соответствует третьему из указанных выше вариантов (текст и значок одновременно). Чтобы его применить, необходимо щелкнуть команду **Выбрать значок для кнопки** и щелкнуть в раскрывшейся коллекции значков (рис. 2.3) наиболее понравившийся или (что более правильно) в какой-то степени отражающий результат применения нашей команды или макроса. Последнее соображение особенно важно, если будет выбран вариант оформления кнопки, при котором на ней будет отображаться только значок (см. далее).

Рис. 2.3. Коллекция стандартных значков, которые пользователь может назначить кнопке в Microsoft Word

Руководствуясь этими соображениями, в нашем случае можно выбрать значок, на котором изображена стрелка, указывающая вправо (соответственно, когда будет назначаться значок для команды **GoToPreviousPage**, для нее следует выбрать значок со стрелкой, указывающей влево) — компактно и выразительно! Поэтому можно вполне обойтись без какой-либо поясняющей надписи. Чтобы оставить на кнопке только выбранный значок, следует щелкнуть в контекстном меню команду **Основной стиль**.

Предупреждение

При выборе значка для кнопки необходимо внимательно следить за тем, чтобы не произошло дублирования кнопок, точнее отображаемых на них значков. В противном случае при нажатии "кнопки-близнеца" могут быть выполнены совсем не те действия, которых вы ожидали.

После выбора значка можно произвести его редактирование. Для перехода в режим изменения значка необходимо выбрать в контекстном меню команду **Изменить значок на кнопке**, после чего на экране появится диалоговое окно **Редактор кнопок** (рис. 2.4).

Теперь вы можете, к примеру, раскрасить выбранный стандартный значок в другие цвета (как показано на рис. 2.5) или несколько изменить исходный рисунок. Можно вообще нарисовать свой собственный фирменный значок.

Попробуйте, поэкспериментируйте — может, откроете в себе ранее дремавшие художественные способности!

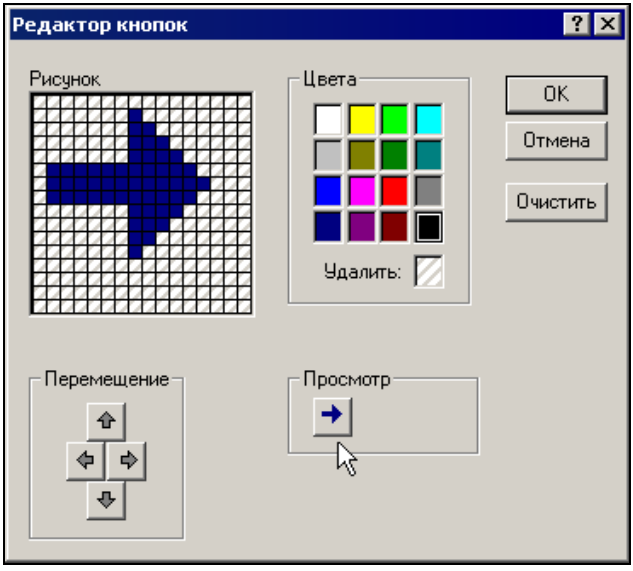


Рис. 2.4. Отображение стандартной кнопки со стрелкой в диалоговом окне **Редактор кнопок**

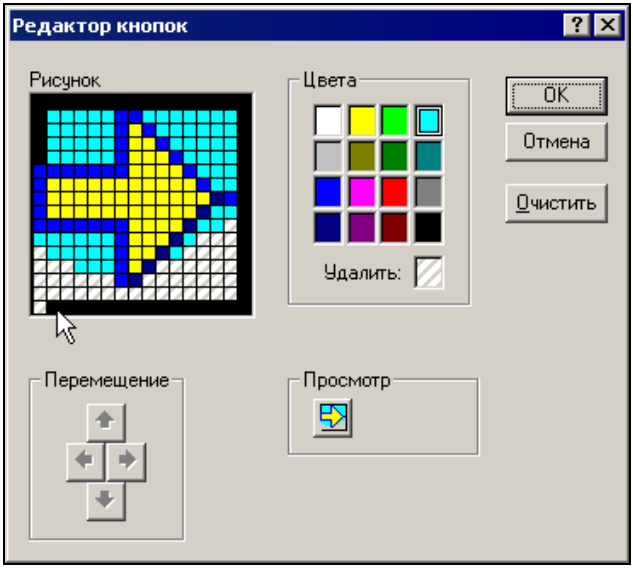


Рис. 2.5. Редактирование значка, отображаемого на кнопке

В контекстном меню имеются также две другие полезные команды: **Копировать значок на кнопке** и **Вставить значок для кнопки**. Основная идея их использования состоит в следующем. Вы можете скопировать в буфер обмена любой значок (стандартный или созданный вами ранее), отображаемый на какой-либо кнопке, а затем поместить его на вновь созданную (или любую другую кнопку). Далее, чтобы не допустить дублирования значков кнопок, необходимо внести те или иные коррективы во внешний вид вновь созданной кнопки. Напоминаем, что все операции по изменению внешнего вида кнопки проводятся в режиме настройки Microsoft Word (меню **Сервис**, команда **Настройка**). В этом режиме для применения того или иного способа модификации кнопки следует поместить указатель мыши над нужной кнопкой, нажать правую кнопку мыши и выбрать соответствующую команду.

> ***Полезные сведения***
>
> Описанная процедура копирования значка кнопки может применяться не только в Microsoft Word. С ее помощью можно также выполнять копирование значков кнопок из одного приложения Microsoft Office в другое, а также в любую другую программу, в которой настройка панели инструментов осуществляется с помощью процедуры, совместимой с использованной разработчиками данного пакета.

После окончания процедуры модификации значка следует нажать кнопку **ОК** в окне **Редактор кнопок**, а затем кнопку **Закрыть** диалогового окна **Настройка**, чтобы завершить процесс создания кнопки.

2.3. Перемещение в начало и конец документа с помощью макросов

Созданные нами только что кнопки позволяют листать документ. Вместе с тем, полученный набор кнопок навигации было бы неплохо дополнить еще двумя, позволяющими перемещаться в начало и конец документа. Подавляющему большинству пользователей, конечно, известно, что это можно легко и быстро сделать, нажав сочетание клавиш <Ctrl>+<Home> или <Ctrl>+<End>. Однако согласитесь, что если вы просматриваете документ, листая его страницы с помощью кнопок (т. е. нажимая кнопку мыши), то выполнять при этом какие-либо действия, нажимая клавиши на клавиатуре, не очень удобно. Кроме того, мы можем совместить процесс перемещения в начало или конец документа со вставкой закладки, которая позволит в случае необходимости быстро вернуться назад.

Поэтому мы предлагаем в этом разделе заняться созданием более полного комплекта кнопок навигации, включающего и кнопки перемещения в начало и конец документа. Предположим, что стандартные команды, позволяющие выполнять такое перемещение, мы не нашли в Microsoft Word. Воз-

можно, они есть, даже наверняка есть, но как-то очень уж хорошо спрятаны. Поэтому мы решили не тратить время на поиски этих команд, а просто создать аналогичные им простенькие макросы. Заодно продемонстрируем особенности процедуры назначения кнопок применительно уже не к встроенным командам Microsoft Word, а к создаваемым макросам.

Ситуация

Вы просматриваете или читаете документ, листая его с помощью кнопок, процесс создания которых был только что нами описан. Затем решили вернуться в начало документа или посмотреть, какие выводы приводятся в его заключительной части.

Что требуется получить

Вы нажимаете кнопку, с помощью которой перемещаетесь в начало или конец документа. При этом в той точке документа, где до перемещения располагался курсор, вставляется закладка, которая в дальнейшем позволит вам вернуться на исходную позицию.

Способ решения

Решение состоит в создании двух простейших макросов, один из которых позволяет мгновенно переместиться в начало документа, а другой — в его конец. Запуск макросов осуществляется с помощью нажатия созданных вами кнопок.

Решение задачи

Чтобы не загромождать изложение излишними подробностями, разобьем поставленную выше задачу на две. Первая из этих задач (вставка закладки) уже решалась нами в предыдущей главе, и поэтому мы предлагаем вам выполнить ее самостоятельно, в порядке упражнения. Здесь же мы займемся созданием макроса, позволяющего мгновенно переместиться в начало (конец) документа.

> *Основные этапы*: вначале будут выполнены подготовительные действия по запуску процедуры создания макроса (шаги 1—5). Затем мы назначим создаваемому макросу кнопку (шаги 6—8, 13) и займемся ее художественным оформлением (шаги 9—12). Далее будет выполнен переход в начало документа (шаг 14), после чего процедура создания макроса будет завершена (шаг 15).

Мы еще раз подробно воспроизведем описанную в предыдущей главе последовательность шагов по запуску процесса записи макроса с помощью команд меню. Также подробно будет описана процедура назначения кнопки макросу и оформления ее внешнего вида. В следующей главе рассказывается о более простых и быстрых способах вывода диалогового окна **Запись мак-**

роса. Что касается назначения кнопки макросу, будем считать, что после прочтения данной главы вы научитесь это делать. А если что забудете — всегда можно вернуться и уточнить детали.

1. Откройте меню **Сервис** и выберите команду **Макрос**.
2. В открывшемся подменю щелкните команду **Начать запись**, после чего появится диалоговое окно **Запись макроса**.
3. Введите в поле **Имя макроса** вместо выражения Макрос1 название создаваемого макроса, например, **GoToStart** (т. е. перейти в начало).
4. В поле **Описание** добавьте (или введите вместо имени автора и даты) поясняющий текст, например, "Переход в начало документа".
5. В поле со списком **Макрос доступен для** оставьте предлагаемое по умолчанию значение.
6. Нажмите в диалоговом окне **Запись макроса** кнопку **панели** в группе **Назначить макрос**, после чего появится диалоговое окно **Настройка**, аналогичное представленному на рис. 2.1 (отличия будут заключаться лишь в том, что в списке **Категории** будет отображаться только элемент **Макросы**, а в списке **Команды** — только элемент **Normal.NewMacros.GoToStart**).
7. Щелкните в списке **Команды** элемент **Normal.NewMacros.GoToStart**.
8. Перетащите мышью (нажав и не отпуская ее левую кнопку) данный элемент на любую удобную для вас позицию на любой панели инструментов, после чего в этом месте появится новая кнопка с надписью **Normal.NewMacros.GoToStart**. Как говорится, длинно и невнятно...

> ***Полезные сведения***
>
> Смысл приведенного выше выражения вам станет полностью ясен после прочтения *гл. 7*, в которой подробно рассказывается о некоторых особенностях "кухни" создания макросов. На всякий случай (для особо продвинутых читателей) все же сообщим, что данное выражение представляет собой "полное имя" создаваемого макроса **GoToStart**, составленное из названия шаблона **Normal**, в котором хранятся настройки и макросы Microsoft Word, имени модуля, в котором находится макрос (**NewMacros,** т. е. "новые макросы"), и собственно имени макроса.

Если в случае со стандартной командой Microsoft Word **GoToNextPage**, рассмотренной в предыдущем разделе, разработчики программы изначально присвоили кнопке, запускающей эту команду, понятное имя **Следующая страница** (большинство кнопок, автоматически назначаемых стандартным командам, также по умолчанию имеют понятное для пользователя имя, в крайнем случае на них отображается достаточно выразительный значок), то при создании макроса об этом может позаботиться только его автор. Впрочем, вы уже знаете, как это делается. Поэтому продолжим процедуру создания нашего макроса.

9. Поместите указатель мыши над только что созданной кнопкой и нажмите правую кнопку мыши. Появится контекстное меню, с помощью команд которого будет проводиться модификация кнопки (см. рис. 2.2).
10. По аналогии с командами **GoToNextPage** и **GoToPreviousPage** имеет смысл выбрать для нашей кнопки значок со стрелкой, указывающей на этот раз, естественно, вверх. Для этого выберите в контекстном меню команду **Выбрать значок для кнопки** и щелкните в раскрывшейся коллекции значков соответствующую кнопку.
11. Затем еще раз поместите указатель мыши над только что созданной кнопкой (на ней уже отображаются значок и надпись), нажмите правую кнопку мыши и выберите команду **Основной стиль**. На кнопке останется только значок со стрелкой.
12. (Необязательный этап.) Если вы хотите каким-то образом модифицировать значок на кнопке, выберите в контекстном меню команду **Изменить значок на кнопке**, после чего на экране появится диалоговое окно **Редактор кнопок** (см. рис. 2.4), в котором можно отредактировать внешний вид кнопки.
13. После завершения процедуры модификации значка кнопки нажмите кнопку **OK** в окне **Редактор кнопок**, а затем кнопку **Закрыть** диалогового окна **Настройка**, чтобы завершить процесс создания кнопки. На экране появится маленькая плавающая панель **Остановить запись** с двумя кнопками, позволяющими регулировать процесс записи макроса.

> ***Совет***
>
> Если при создании макроса (этого или любого другого) вам будет некогда заниматься оформлением "красивой" кнопки, можно спокойно пропустить шаги 9—12, оставив временно на кнопке невразумительную надпись типа **Normal.NewMacros.GoToStart** или какой-либо более короткий и более осмысленный текст. В дальнейшем, когда у вас появится для этого время, вы сможете в любой момент вернуться к оформлению внешнего вида кнопки *(см. разд. 2.2)*.

14. Наконец-то можно будет выполнить действие, ради которого, собственно, все и затевалось. Вставьте в текущую позицию курсора закладку и нажмите сочетание клавиш <Ctrl>+<Home>, после чего курсор переместится в начало документа.
15. Завершите создание макроса, нажав кнопку с синим квадратиком, расположенную на панели записи макроса **Остановить запись**.

Аналогичный макрос, позволяющий переместиться в конец документа (назовем его **GoToEnd**, т. е. перейти в конец), читатели могут в порядке упражнения создать самостоятельно. Напоминаем, что такой переход осуществляется с помощью сочетания клавиш <Ctrl>+<End>.

Что же мы имеем в итоге

Работая с документом, вы можете теперь с помощью кнопок листать его в обе стороны, а в случае необходимости — мгновенно переместиться в его начало или конец. При этом в любой момент вы можете, нажав определенное сочетание клавиш, мгновенно вернуться назад, к закладке, автоматически созданной в том месте документа, где был прерван его просмотр.

2.4. Дополнительные сведения

В этом разделе описываются функции некоторых других команд контекстного меню оформления кнопок панели инструментов. Кроме того, здесь приводится сводка команд навигации, позволяющих перемещаться по различным компонентам документа Microsoft Word.

2.4.1. Дополнительные возможности оформления кнопок и панелей инструментов

Изменить оформление кнопки панели инструментов можно в любой удобный для вас момент. Для этого следует поместить указатель мыши над любой из панелей, нажать правую кнопку мыши и выбрать в раскрывшемся контекстном меню команду **Настройка**. После появления на экране одноименного диалогового окна поместите указатель мыши над нужной кнопкой и еще раз нажмите правую кнопку мыши (чтобы отобразить кнопку на экране, возможно, потребуется перетащить диалоговое окно **Настройка** чуть в сторону). Появится контекстное меню, команды которого позволяют модифицировать внешний вид кнопок — см. рис. 2.2. О работе с этим меню рассказывалось ранее в *разд. 2.2*. Однако в этом меню имеется еще несколько полезных команд, назначение которых не было описано в основной части главы.

Прежде всего, заслуживает внимания команда **Начать группу**. С ее помощью можно добавить разделитель, позволяющий визуально выделить на панели инструментов группу тематически связанных друг с другом кнопок. Для создания левой границы такой группы необходимо выполнить данную команду, поместив указатель мыши над первой кнопкой группы. Разделитель отобразится слева от кнопки, а в контекстном меню рядом с командой **Начать группу** появится галочка. Чтобы создать правую границу группы, следует поместить указатель мыши над кнопкой, расположенной справа от последней кнопки группы, и еще раз исполнить команду **Начать группу**. Удаление разделителя группы осуществляется аналогичным образом (при этом галочка в меню исчезает).

Удаление кнопки с панели инструментов осуществляет с помощью одноименной команды контекстного меню.

Полезные сведения

Удалить кнопку с панели инструментов можно также и в обычном режиме работы с Microsoft Word, не выводя на экран диалоговое окно **Настройка**. Для этого нужно просто "сбросить" кнопку с панели инструментов — поместить над ней указатель мыши, нажать клавишу <Alt> и, не отпуская ее, перетащить кнопку в любое место рабочей области окна Microsoft Word (не занятое какой-либо панелью), после чего следует отпустить кнопку мыши. Аналогичным образом с помощью клавиши <Alt> можно удалить и разделитель группы кнопок с панели инструментов: указатель мыши помещается над кнопкой, расположенной справа от разделителя, нажимается клавиша <Alt> и кнопка перетаскивается влево "за разделитель", после чего кнопку мыши следует отпустить.

Команды **Сброс** и **Восстановить значок на кнопке** контекстного меню имеют сходные функции и используются для восстановления первоначального вида кнопки. Следует иметь в виду, что в случае, когда по умолчанию в Microsoft Word исходной кнопке не было присвоено какого-либо значка, применение данной команды приводит к появлению "пустой" кнопки  .

Для чего используется команда контекстного меню **Назначить гиперссылку**, думаем, понятно из ее названия. Вместе с тем, следует отметить следующие два момента:

1. Назначение кнопке гиперссылки отменяет команду или макрос, запускавшиеся ранее с помощью этой кнопки.
2. Гиперссылки, назначаемые кнопкам с помощью этой команды, обладают несколько ограниченными возможностями.

В частности, они позволяют осуществлять переход только к внешним документам (файлам, веб-страницам), но не внутри самого документа.

Дополнительные сведения о возможностях оформления кнопок и панелей инструментов приведены в *гл. 3*.

2.4.2. Команды навигации в Microsoft Word

В Microsoft Word имеется целый ряд встроенных стандартных команд, позволяющих осуществлять навигацию по различным структурным элементам и областями документа или нескольких документов — в пределах страницы, абзаца, слова, а также между сносками, примечаниями, колонтитулами, разделами, окнами документов и т. д.

Далее приводится подборка команд навигации, представленная в виде табл. 2.1, в первом столбце которой указано "понятное" имя встроенной

команды (чаще всего именно оно отображается на назначенной команде кнопке), во втором — ее "внутреннее" имя, под которым она фигурирует в списке **Все команды** диалогового окна **Настройка**, в третьем — сочетание клавиш, назначенных по умолчанию данной команде, в четвертом — значок, отображаемый по умолчанию на кнопке, назначенной команде (если имеется), в пятом — комментарии об особенностях использования данной команды. Некоторые из этих команд могут оказаться полезными при решении тех или иных конкретных задач, и вы сочтете нужным "вытащить" их на панель инструментов. Другие могут пригодиться при написании макросов, в которых требуется выполнять перемещение между различными структурным элементам и областями документа. Подробнее об этом будет рассказано в следующих главах книги.

Таблица 2.1. *Сводка команд навигации Microsoft Word*

"Понятное имя" команды*	Внутреннее имя	Сочетание клавиш	Значок по умолчанию	Комментарий
Следующая страница	**GoToNextPage**	—	—	
Следующая страница	**NextPage**	—	—	Работает только в режимах разметки страницы и веб-документа
Предыдущая страница	**GoToPreviousPage**	—	—	
Предыдущая страница	**PrevPage**	—	—	Работает только в режимах разметки страницы и веб-документа
На одну страницу вниз	**PageDown**	<PgDn>		Фактически перемещает на один *экран* вниз
На одну страницу вверх	**PageUp**	<PgUp>		Фактически перемещает на один *экран* вверх
В начало документа	**StartOfDocument**	<Ctrl>+ +<Home>		
В конец документа	**EndOfDocument**	<Ctrl>+<End>		
На один абзац вверх	**ParaUp**	<Ctrl>+ +<Стрелка вверх>	—	Курсор перемещается в начало *текущего* абзаца

Таблица 2.1 *(продолжение)*

"Понятное имя" команды*	Внутреннее имя	Сочетание клавиш	Значок по умолчанию	Комментарий
На один абзац вниз	**ParaDown**	<Ctrl>+ +<Стрелка вниз>	—	Курсор перемещается в начало *следующего* абзаца
На одно предложение влево	**SentLeft**	—	—	Курсор перемещается в начало *текущего* предложения
На одно предложение вправо	**SentRight**	—	—	Курсор перемещается в начало *следующего* предложения
Начало строки	**StartOfLine**	<Home>		
В конец строки	**EndOfLine**	<End>		
В начало строки	**StartOfRow**	<Alt>+ +<Home>	—	Курсор перемещается в первую ячейку текущей строки таблицы
В конец строки	**EndOfRow**	<Alt>+<End>	—	Курсор перемещается в последнюю ячейку текущей строки таблицы
В начало окна	**StartOfWindow**	<Alt>+<Ctrl>+ +<PgUp>	—	Под окном подразумевается область документа, отображаемая на экране
Конец окна	**EndOfWindow**	<Alt>+<Ctrl>+ +<PgDn>	—	То же
Сноски	**ViewFootnotes**	—	—	Позволяет указывать, какая область сносок *(обычных* или *концевых)* будет отображаться на экране. Курсор перемещается в область текста ближайшей к нему справа сноски. Повторный выбор команды возвращает курсор в тело документа

Таблица 2.1 *(продолжение)*

"Понятное имя" команды*	Внутреннее имя	Сочетание клавиш	Значок по умолчанию	Комментарий
Перейти к сноске	**ViewFootnoteArea**	—	—	Просмотр области о*бычных* сносок. Курсор перемещается в область текста ближайшей к нему справа сноски. Повторный выбор команды возвращает курсор в тело документа
Следующая сноска	**GoToNextFootnote**	—	—	Последовательный переход к следующей *обычной* сноске в теле документа
Предыдущая сноска	**GoToPreviousFootnote**	—	—	То же (в обратную сторону)
Следующая концевая сноска	**GoToNextEndnote**	—	—	Последовательный переход к следующей *концевой* сноске в теле документа
Предыдущая концевая сноска	**GoToPreviousEndnote**	—	—	То же (в обратную сторону)
Следующий раздел	**GoToNextSection**	—	—	
Предыдущий раздел	**GoToPreviousSection**	—	—	
Верхний/ нижний колонтитул	**GoToHeaderFooter**	—		
Следующее примечание	**GoToNextComment**	—		Доступна только при наличии примечаний в документе
Предыдущее примечание	**GoToPreviousComment**	—		То же
Другая область	**OtherPane**	<F6>	—	

Таблица 2.1 *(окончание)*

"Понятное имя" команды*	Внутреннее имя	Сочетание клавиш	Значок по умолчанию	Комментарий
Следующее окно	**NextWindow**	<Alt>+<F6>, <Ctrl>+<F6>	—	Переход между окнами открытых документов. Осуществляется циклически, последовательность перехода определяется порядком выбора
Предыдущее окно	**PreviousWindow**	<Alt>+ +<Shift>+<F6>, <Ctrl>+ +<Shift>+<F6>	—	То же (в обратную сторону)
Следующее поле	**NextField**	<Alt>+<F1>	—	
Предыдущее поле	**PreviousField**	<Shift>+<F11>	—	

*Понятное имя команды — имя, отображаемое на кнопке и во всплывающей подсказке, которая появляется, если поместить указатель мыши над кнопкой (должен быть установлен флажок **всплывающие подсказки** на вкладке **Вид** диалогового окна **Параметры**).

2.5. Что нового в этой главе

В этой главе мы продолжили знакомство со встроенными командами Microsoft Word и занимались созданием макросов, упрощающих и ускоряющих перемещение (навигацию) по документу. Решение связанных с этим задач позволило вам освоить практические приемы, обеспечивающие возможность запуска команд и макросов с помощью назначенных им кнопок.

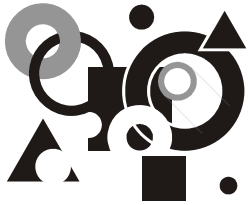
Вы узнали, какие внутренние имена имеют команды постраничного листания документа, и создали кнопки, позволяющие просматривать его, не прибегая к помощи клавиатуры.

Вы познакомились с практическими приемами создания кнопок, позволяющими оформить их внешний вид с учетом вкусов и предпочтений пользователя — оставить на кнопке только текст (предлагаемый по умолчанию или определяемый самим пользователем), отобразить на ней значок, наглядно характеризующий назначение кнопки, или же совместить эти два способа.

Вы узнали также, что пользователь может копировать значки с одной кнопки на другую, изменять их внешний вид и даже создавать свои собственные "фирменные" значки.

Освоив эти практические приемы, мы создали макросы, позволяющие мгновенно перемещаться в начало и конец документа, и назначили этим макросам кнопки. Благодаря этому в вашем распоряжении теперь имеется полный комплект средств навигации, позволяющих перемещаться по документу с помощью мыши.

В *разд. "Дополнительные сведения"* вы познакомились с рядом других средств, с помощью которых может настраиваться внешний вид кнопок и панелей инструментов. Здесь же представлен подробный список команд навигации Microsoft Word с указанием их внутреннего и "понятного" имени, назначенного по умолчанию значка и сочетания клавиш. Приводятся также сведения, касающиеся особенностей использования соответствующей команды.



Глава 3

Удобные инструменты — залог успешной работы

Назначение кнопок и сочетаний клавиш. Копирование и перемещение кнопок. Работа с панелями инструментов (вывод на экран, скрытие, создание, настройка). Создание макроса, выводящего панель инструментов на экран. Дополнительные сведения.

С удобными инструментами и работать приятнее, а главное — быстрее. Подготовительные операции по запуску процесса записи макроса, описанные в предыдущих главах, наверняка вам успели уже надоесть. В этой главе мы применим уже имеющиеся у нас знания для рационализации труда пользователя Microsoft Word вообще и создателя макросов в частности.

При выполнении процедур, описанных в предыдущих главах, нам приходилось многократно "кликать" кнопкою мыши, чтобы вывести на экран диалоговые окна **Запись макроса** (меню **Сервис | Макрос | Начать запись**) и **Настройка клавиатуры** (меню **Сервис | Настройка | Клавиатура**). К сожалению, разработчики Microsoft Word не закрепили за ними каких-либо стандартных сочетаний клавиш, хотя соответствующие кнопки имеются, и мы поможем вам их отыскать. Это позволит значительно упростить выполнение вспомогательных действий, связанных с запуском процесса записи макросов и присвоением командам и макросам сочетаний клавиш.

3.1. Рабочие инструменты должны быть всегда под рукой

Более совершенные инструменты создаются, как известно, с помощью уже имеющихся более простых и примитивных. Поэтому давайте проделаем — в последний раз — хорошо знакомую, но довольно нудную процедуру подготовки к запуску режима записи макроса.

3.1.1. Размещение дополнительных кнопок на панелях инструментов

В целях ускорения и упрощения действий, связанных с созданием и настройкой кнопок, мы займемся сейчас созданием вспомогательных кнопок, с помощью которых осуществляется вызов диалоговых окон **Настройка** и **Настройка клавиатуры**.

1. Откройте меню **Сервис**.
2. Выберите в нем команду **Настройка**, после чего на экране появится одноименное диалоговое окно.
3. Выберите на вкладке **Команды** в списке **Категории** элемент **Все команды**.
4. Прокрутите список **Команды** и выберите в нем элемент **ToolsCustomize**.
5. Перетащите мышью команду **ToolsCustomize** на любую удобную позицию на одной из панелей инструментов, после чего на ней появится новая кнопка с надписью **Настройка**.
6. Повторите теперь шаги 3 и 4 для команды **ToolsCustomizeKeyboard**, после чего на панели инструментов появится новая кнопка с надписью **Настройка клавиатуры**.

При желании вы можете изменить надписи на этих кнопках, разместить на них в дополнение к надписи еще и значок или же оставить только значок. Все эти процедуры мы уже неоднократно проделывали в предыдущих главах, и если вы не помните какие-либо детали, то можете вернуться назад, чтобы их уточнить.

Мы же, не откладывая в долгий ящик, испытаем вновь созданные инструменты в деле. Напомним, что с помощью кнопок **Настройка** и **Настройка клавиатуры** открываются одноименные диалоговые окна. Первое из них позволяет, в частности, "вытаскивать" кнопки стандартных команд Microsoft Word на панель инструментов, второе — назначать этим командам (и соответствующим им кнопкам) сочетания клавиш.

> ***Полезные сведения***
>
> Отметим, что имеется и более простой способ вывода на экран диалогового окна **Настройка**. Для этого требуется просто дважды щелкнуть кнопкой мыши свободное место между закрепленными панелями. Правда, от этой возможности придется отказаться, если пространство панелей по всей ширине экрана будет у вас заполнено кнопками.

3.1.2. Назначение сочетаний клавиш кнопкам

Чтобы назначить сочетание клавиш кнопке, открывающей диалоговое окно **Настройка**, выполните следующие действия:

1. Нажмите только что созданную нами кнопку **Настройка клавиатуры**, после чего появится одноименное диалоговое окно.

   Обратите внимание, что теперь для этого мы выполнили на два щелчка мышью меньше, чем раньше.

1. Прокрутите список **Категории** и выберите в нем элемент **Все команды**.
2. Прокрутите список **Команды** и выберите в нем элемент **ToolsCustomize**.
3. В поле **Новое сочетание клавиш** введите желаемое сочетание клавиш, например <Alt>+<N>, и затем нажмите кнопку **Назначить**, после чего данное сочетание клавиш появится в списке **Текущие сочетания клавиш**.

> ***Примечание***
>
> Еще раз напоминаем, что буквенные сочетания клавиш, в которые входит модификатор <Alt>, часто по умолчанию используются в Microsoft Word для раскрытия меню и выбора команд с помощью клавиатуры. Поэтому вы должны решить для себя, стоит ли назначать подобные сочетания клавиш создаваемым вами кнопкам и макросам (отменяя тем самым возможности их стандартного использования) или же лучше в этом случае воспользоваться какими-либо другими сочетаниями.

5. Нажмите кнопку **Закрыть**, чтобы закрыть диалоговое окно **Настройка клавиатуры**.

В качестве упражнения вы можете назначить сочетание клавиш кнопке, открывающей диалоговое окно **Настройка клавиатуры** (при этом вас не должно смущать, что кнопка **Настройка клавиатуры** в этом случае будет "обслуживать" сама себя — в начале процедуры вы нажимаете эту кнопку, а затем ей же присваиваете сочетание клавиш). Напоминаем, что данной кнопке соответствует команда **ToolsCustomizeKeyboard**.

Отметим, что Microsoft Word можно настроить таким образом, чтобы при наведении указателя мыши на кнопку "всплывала" подсказка, в которой отображается имя команды (или макроса), которой назначена эта кнопка. Для этого необходимо установить флажок **Отображать подсказки для кнопок** на вкладке **Параметры** диалогового окна **Настройка** (рис. 3.1).

> ***Примечание***
>
> Следует отметить, что задача отображения всплывающих подсказок решена в интерфейсе Microsoft Word неоднозначным образом, что иногда вызывает недоумение у пользователей. Дело в том, что помимо флажка **Отображать подсказки для кнопок**, представленного в диалоговом окне **Настройка** (вкладка

Параметры, группа **Другие**), в программе имеется также флажок **Всплывающие подсказки,** расположенный в диалоговом окне **Параметры** (вкладка **Вид**, группа **Показывать**). Однако последний из указанных флажков не имеет никакого отношения к всплывающим подсказкам, относящимся к кнопкам. С его помощью обеспечивается лишь возможность отображать во всплывающей подсказке текст имеющихся в документе примечаний (напомним, что примечание можно вставить в любое место документа, выбрав команду **Примечание** в меню **Вставка**).

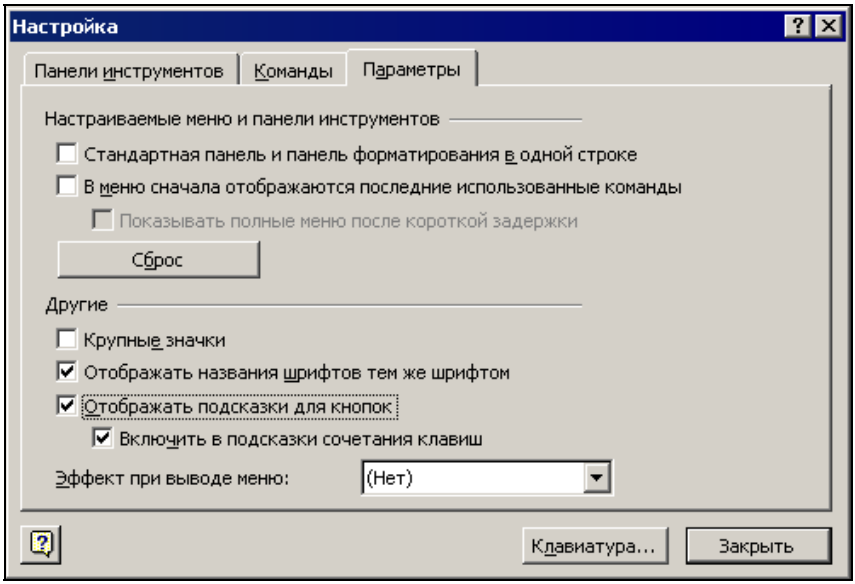


Рис. 3.1. Вкладка **Параметры** диалогового окна **Настройка**

На вкладке **Параметры** диалогового окна **Настройка** имеется также флажок **Включать в подсказки сочетания клавиш**, позволяющий отображать сочетания клавиш, назначенные кнопкам панелей инструментов. Необходимо, однако, иметь в виду, что в этом случае сочетания клавиш будут выводиться только для кнопок, относящихся к встроенным командам Microsoft Word. Таким образом, если вы присвоите сочетания клавиш кнопке, назначенной вами макросу, оно не будет отображаться во всплывающей подсказке при наведении указателя мыши на данную кнопку.

Примечание

Чтобы обойти это ограничение, в справке Microsoft Word предлагается добавить соответствующее сочетание клавиш к имени кнопки. Однако эта рекомендация является, по нашему мнению, ошибочной, поскольку во всплывающей подсказке отображается в данном случае не имя кнопки, а название макроса.

Поэтому если добавить сочетание клавиш к имени кнопки, его можно будет увидеть не во всплывающей подсказке, а непосредственно на поверхности кнопки, да и то только в том случае, когда для отображения кнопки выбран режим **Только текст (всегда)** или **Значок и текст**.

3.2. Панель Visual Basic: из семи кнопок нам пока понадобится только одна

В различных версиях Microsoft Word имеется целый ряд встроенных панелей инструментов, на которых по тематическому признаку сгруппированы инструментальные средства работы с документами — **Стандартная**, **Форматирование**, **Таблицы и границы** и т. п. (в Microsoft Office Word 2003 число таких панелей приближается уже к двадцати). Разработчики программы, конечно же, позаботились и о том, чтобы предоставить в распоряжение пользователей набор инструментов, необходимых для записи макросов. Находятся они на встроенной панели, которая называется **Visual Basic**. Правда, на данном этапе нашего знакомства с макросами реальный интерес для нас представляет лишь одна из этих кнопок, с помощью которой на экран можно быстро вывести диалоговое окно **Запись макроса**. Что касается других кнопок данной панели, то они пригодятся нам при освоении материала, представленного в следующих частях книги.

3.2.1. Вывод скрытой панели на экран

После загрузки только что установленной программы Microsoft Word на экране отображаются по умолчанию только две панели инструментов — **Стандартная** и **Форматирование**, все же остальные встроенные панели являются скрытыми. Напомним наиболее простой способ вывода на экран скрытой панели инструментов. Наведите указатель мыши на любую из панелей, отображаемых в данный момент на экране, и щелкните правой кнопкой мыши. На экране появится список панелей Microsoft Word (рис. 3.2).

Слева от названия панели инструментов, отображаемой в данный момент на экране, в списке стоит галочка. Данный список имеет динамический характер: в него будут включаться названия всех новых панелей инструментов, создаваемых пользователем.

Щелкните в списке панелей элемент **Visual Basic**, чтобы вывести на экран одноименную панель. Как видно на рис. 3.3, данная панель по умолчанию содержит семь кнопок. Однако, как уже говорилось, для наших целей понадобится пока только кнопка **Записать макрос**, выводящая на экран диалоговое окно **Запись макроса**.

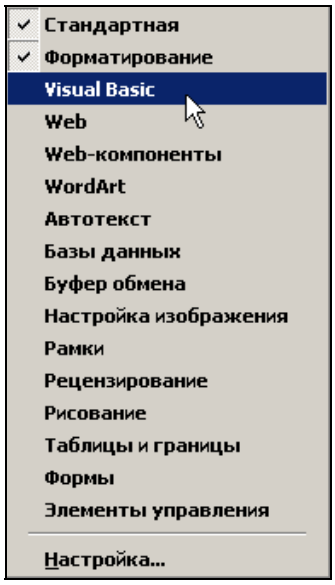


Рис. 3.2. Контекстное меню
со списком встроенных панелей инструментов Microsoft Word

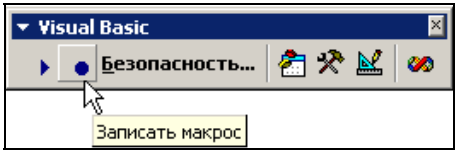


Рис. 3.3. Встроенная панель инструментов **Visual Basic**

Вообще говоря, эту кнопку мы могли бы "вытащить" на любую панель инструментов, уже отображаемую на экране, с помощью описанного в начале этой главы способа (предварительно отыскав ее в общем списке команд Microsoft Word, который отображается в диалоговом окне **Настройка**). Однако давайте воспользуемся этим случаем, чтобы более детально ознакомиться с полезными приемами работы с кнопками — их перетаскиванием и копированием.

3.2.2. Основные приемы работы с кнопками

Чтобы *переместить* кнопку с одной панели инструментов на другую (либо на другую позицию на той же самой панели), необходимо поместить на нее указатель мыши, нажать клавишу <Alt> и, не отпуская ее, перетащить кнопку в нужное место.

Чтобы *удалить* кнопку с панели инструментов, необходимо поместить на нее указатель мыши, нажать клавишу <Alt> и, не отпуская ее, перетащить кнопку в любое место рабочей области окна Microsoft Word (не занятое какой-либо панелью).

Чтобы *скопировать* кнопку с одной панели на другую (либо на другую позицию на той же самой панели), необходимо открыть диалоговое окно **Настройка**, поместить на копируемую кнопку указатель мыши, нажать клавишу <Ctrl> и, не отпуская ее, перетащить кнопку на нужное место. Затем следует отпустить клавишу <Ctrl>, после чего здесь возникнет дубликат исходной кнопки. Таким образом можно создать любое количество одинаковых по внешнему виду и функциональным возможностям кнопок. Дублировать можно как стандартные кнопки Microsoft Word, так и кнопки, создаваемые пользователем (в том числе для запуска макросов).

Примечание

Если перемещение или удаление кнопки выполняется при открытом диалоговом окне **Настройка**, клавишу <Alt> нажимать не нужно.

Для какой же цели может потребоваться подобное "клонирование" кнопок?

Как уже говорилось, сразу же после установки Microsoft Word на экране по умолчанию отображаются только две встроенных панели инструментов — **Стандартная** и **Форматирование**. Сделано это для того, чтобы предоставить в распоряжение пользователя самые необходимые (с точки зрения разработчиков программы) инструменты. Однако очень скоро может сложиться ситуация, когда для решения каких-либо конкретных задач вам понадобятся кнопки, расположенные по умолчанию на других панелях (например, как в нашем случае, — на панели **Visual Basic**). Можно, конечно, вывести на экран эту панель целиком и по мере необходимости нажимать на ней нужные вам одну либо две кнопки. Однако подобная роскошь позволительна лишь в случае, когда на экране имеется достаточно много свободного места. Практика показывает, что этот счастливый период продолжается очень и очень недолго.

Поэтому имеет смысл заранее побеспокоиться о будущем и скопировать нужные кнопки на какую-либо основную панель, после чего можно убрать с экрана "порт их постоянной приписки", т. е. панель инструментов **Visual Basic**.

Примечание

Не следует путать копирование кнопки с копированием отображаемого на кнопке значка (описывалось в предыдущей главе). В последнем случае функциональные возможности кнопки, на которую был скопирован значок с другой кнопки, остаются без изменения.

Итак, чтобы скопировать кнопку с одной панели на другую (в нашем случае это будет кнопка **Записать макрос**), необходимо проделать следующие действия:

1. Выведите на экран панель инструментов **Visual Basic** (если этого еще не сделано), для чего наведите указатель мыши на любую из панелей, которые отображаются на экране, щелкните правой кнопкой мыши и выберите в списке элемент **Visual Basic**.
2. Нажмите созданную нами в первом разделе этой главы кнопку **Настройка** (или присвоенное ей сочетание клавиш), после чего на экране появится одноименное диалоговое окно.
3. Наведите указатель мыши на кнопку **Записать макрос** (кнопка с жирной синей точкой) и нажмите клавишу <Ctrl>. Перетащите эту кнопку в удобное для вас место на одной из "основных" панелей инструментов, после чего отпустите клавишу <Ctrl> (рис. 3.4).

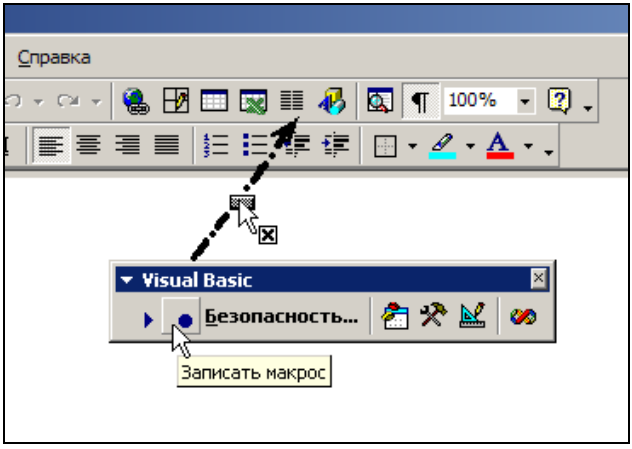


Рис. 3.4. Создание дубликата кнопки **Записать макрос** посредством ее перетаскивания с панели инструментов **Visual Basic** на одну из встроенных панелей

4. Закройте диалоговое окно **Настройка**.

Теперь можно убрать с экрана панель инструментов **Visual Basic**, поскольку дубликат нужной нам кнопки уже имеется на основной панели Microsoft Word. Проще всего это сделать, щелкнув значок с косым крестиком в правом верхнем углу данной панели.

Для закрепления полученных навыков попробуйте аналогичным образом скопировать какую-нибудь другую кнопку, например, **Выполнить макрос** (кнопка с синим треугольничком), расположенную на панели инструментов

Visual Basic. Данная кнопка позволяет открыть диалоговое окно **Макрос** (рис. 3.5). Это окно довольно часто будет использоваться нами в следующих частях книги, так что труд ваш будет ненапрасным.

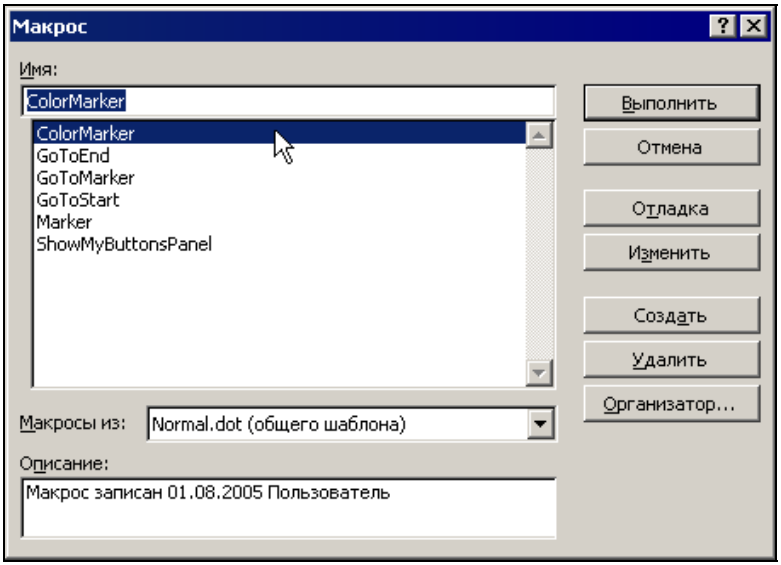


Рис. 3.5. Диалоговое окно **Макрос** со списком созданных макросов

3.3. В хозяйстве панелей инструментов требуется навести порядок

В этой и предыдущей главах мы уже успели создать несколько кнопок для запуска стандартных команд Microsoft Word и созданных нами макросов. Этот процесс будет неизбежно продолжаться и в дальнейшем. Вообще говоря, размещение вновь создаваемых кнопок на стандартных панелях инструментов не всегда является наиболее удачным решением. При его использовании может быть нарушен тематический характер инструментов, затруднится поиск нужных кнопок, сократится рабочая площадь экрана. С другой стороны, если повнимательнее присмотреться к основным панелям инструментов (**Стандартная** и **Форматирование**), то почти наверняка некоторые из размещенных на них кнопок покажутся вам "лишними". Самое время заняться наведением порядка в этом хозяйстве.

3.3.1. Создание индивидуальной панели инструментов

Первым делом мы, конечно, позаботимся о предоставлении собственной жилплощади нашим подопечным — кнопкам, ранее назначенным нескольким стандартным командам Microsoft Word (постраничного листания документа, вызова диалоговых окон **Настройка**, **Настройка клавиатуры**, **Запись макроса** и **Макрос**), а также макросам перемещения в начало и конец документа. Возможно, что вами в порядке упражнения были созданы и какие-то другие кнопки (например, для макроса, вставляющего закладку с текстом **Return_and_Correct**).

Займемся теперь созданием для этих новых кнопок отдельной панели инструментов.

1. Выведите на экран диалоговое окно **Настройка** (с помощью соответствующей кнопки или сочетания клавиш).
2. Щелкните вкладку **Панели инструментов** и нажмите кнопку **Создать** (рис. 3.6).

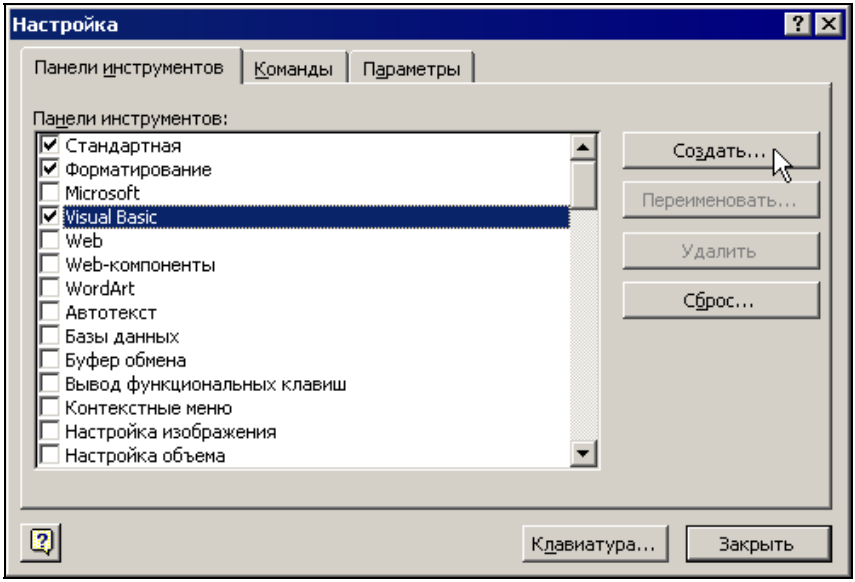


Рис. 3.6. Вкладка **Панели инструментов** диалогового окна **Настройка**

3. Введите в поле **Панель инструментов** диалогового окна **Создание панели инструментов** (рис. 3.7) название новой панели, например, **Мои кнопки,** вместо предлагаемого по умолчанию текста **Настраиваемая 1**.

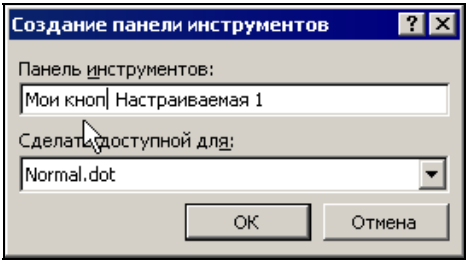


Рис. 3.7. Ввод имени новой панели в диалоговое окно **Создание панели инструментов**

4. Нажмите кнопку **OK**, после чего на экране появится пустая "плавающая" панелька с именем **Мои кнопки** (в заголовке панели видны только первые буквы ее названия): 

.
5. Нажмите кнопку **Закрыть** диалогового окна **Настройка**.

Теперь вы можете на практике реализовать накопленный в предыдущем разделе опыт перетаскивания (копирования) кнопок с одной панели инструментов на другую. В результате вновь созданная нами панель инструментов должна принять примерно такой вид, как показано на рис. 3.8.

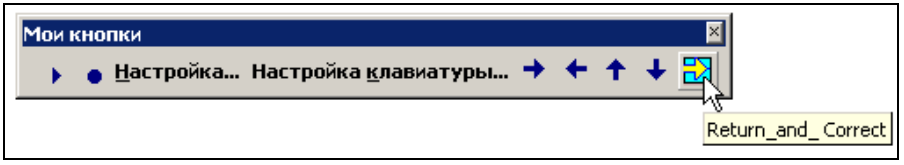


Рис. 3.8. Вновь созданная панель инструментов **Мои кнопки** с кнопками, созданными в предыдущих разделах и главах книги

3.3.2. Настройка стандартных панелей

Как уже говорилось, на основных панелях Microsoft Word изначально "прописаны" некоторые кнопки, которые вряд ли могут понадобиться для решения ваших текущих задач. Вызывает, например, большие сомнение необходимость всегда иметь под рукой такие кнопки, как **Рисование**  , **Уменьшить отступ**  , **Вырезать**  и некоторых других (у каждого пользователя будет, конечно же, свой собственный перечень подобных "излишеств"). Эти кнопки вполне можно убрать с основных панелей инструмен-

тов — либо вообще, "сбросив" их в рабочее окно Microsoft Word, либо переведя в разряд "стратегического резерва", переместив на предварительно созданную для подобных целей пользовательскую панель.

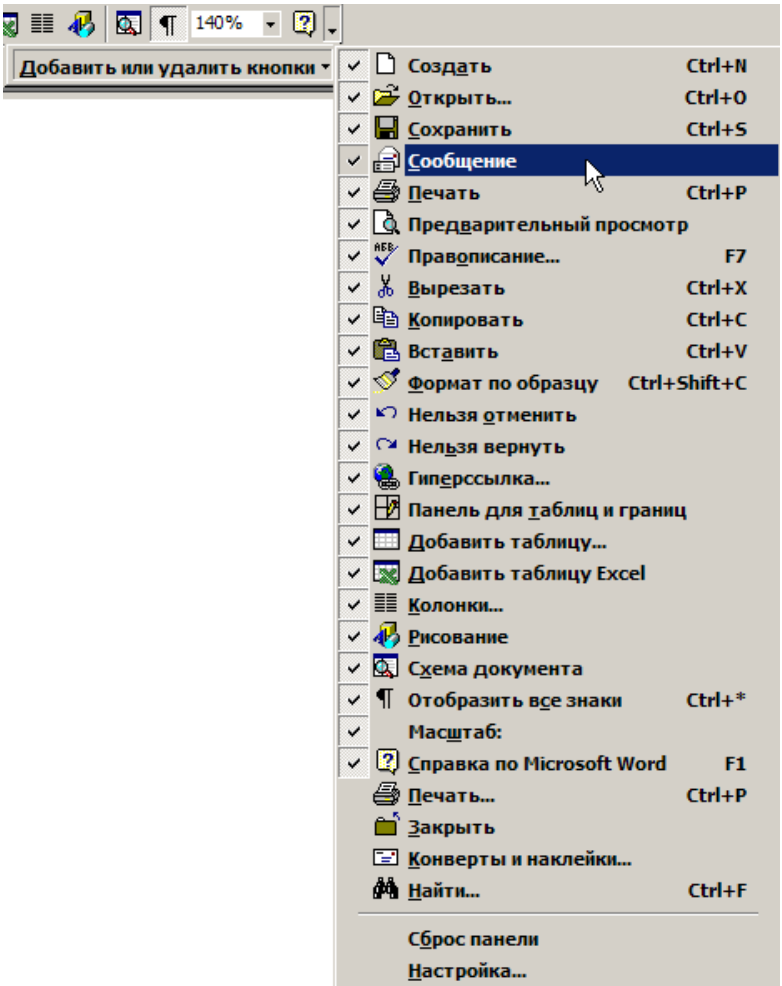


Рис. 3.9. Список кнопок, приписанных по умолчанию к панели инструментов панели **Стандартная**

С другой стороны, как мы уже убедились, в Microsoft Word имеется довольно много команд, "более достойных" находиться на первых ролях, — в том числе и на панелях **Стандартная** и **Форматирование**. Опять же, у каждого пользователя будет свой "список чистых и нечистых". Возможный перечень таких команд, которые, с нашей точки зрения, имеет смысл раз-

местить на основных панелях Microsoft Word, приводится в конце этой главы в *разд. 3.3.5*.

Добавлять кнопки на панели инструментов и убирать их с нее можно также с помощью "штатного" средства Microsoft Word, называемого **Добавить или удалить кнопки** (отсутствует в Microsoft Word 7). Доступ к этому средству осуществляется с помощью кнопки **Другие кнопки**, расположенной у правого края встроенных панелей инструментов программы. Эта кнопка имеется только у панелей, закрепленных у края экрана, и исчезает при переводе последних в режим "плавающих" панелей (и наоборот). После нажатия данной кнопки (на ней отображается значок в виде маленького черного треугольника с направленной вниз вершиной) на экране появляется кнопка большего размера с надписью **Добавить или удалить кнопки**. Если навести на нее указатель мыши, то на экран выведется список кнопок, которые по умолчанию приписаны к данной панели инструментов (рис. 3.9). Рядом с названиями кнопок, уже отображаемых на панели, установлен флажок (проставлена галочка). Чтобы отобразить или скрыть ту или кнопку, необходимо установить либо снять соответствующий флажок.

Однако возможности средства **Добавить или удалить кнопки** являются ограниченными, поскольку его действие распространяется только на те кнопки, которые, как отмечалось выше, по умолчанию приписаны к соответствующей панели.

Примечание

В версиях Microsoft Office Word 2003 и Microsoft Word 2002 при нажатии кнопки **Другие кнопки** выводится контекстное меню, включающее две команды: уже известную нам **Добавить или удалить кнопки** и команду **Отображать кнопки на одной строке**, которая после выполнения соответствующей команды превращается в кнопку **Отображать кнопки на двух строках**.

3.3.3. Управлять панелями также можно с помощью макросов

Вполне возможно, что в будущем для решения своих конкретных задач вы будете создавать отдельные "тематические" панели инструментов. Потребность в использовании инструментов, сгруппированных на этих панелях, будет возникать периодически. Однако основную часть рабочего времени они вам будут не нужны. Поэтому имеет смысл "спрятать" эти панели и вызывать их на экран лишь по мере необходимости.

Ситуация

Примерно раз в месяц вам приходится выполнять некоторую работу — например, подготавливать отчет, текст которого вы форматируете при помощи

стандартных или созданных вами стилей. У вас имеется индивидуальный набор инструментов (кнопок, назначенных стилям и командам форматирования Microsoft Word, а также созданным вами макросам). Эти инструменты находятся на отдельной панели (с именем, например, **Мои кнопки**), которая при обычных условиях работы не отображается на экране монитора и вызывается только по мере необходимости.

> ***Примечание***
>
> Вообще говоря, в этом примере можно было бы использовать и другое, более содержательное имя для данной панели инструментов, например, **Стили** или **Отчет**. Однако тогда бы нам пришлось заранее создать такую панель, поскольку она должна выводиться на экран с помощью макроса, процесс создания которого описывается далее. Чтобы ускорить изложение материала, мы не будем еще раз заниматься созданием новой панели инструментов, а используем уже имеющуюся в нашем распоряжении панель **Мои кнопки**.

Чтобы вывести панель инструментов на экран, как нам уже известно, можно навести указатель мыши на любую из уже отображаемых панелей, щелкнуть правой кнопкой мыши и выбрать нужную панель во всплывающем списке. Однако можно ускорить и упростить эту процедуру, если создать специальную кнопку, позволяющую мгновенно вызвать необходимую панель. Особенно эффективен этот прием при работе с "тематическими" панелями. Например, вы можете создать отдельную панель для средств форматирования документа, для инструментов поиска, работы с таблицами, решения какой-либо специальной задачи и т. п. При этом в целях удобства можно разместить на всех панелях дубликаты часто используемых кнопок (о том, как выполняется создание дубликата кнопки, говорилось ранее в этой главе).

Что требуется получить

Вы нажимаете кнопку на панели инструментов, и нужная панель появляется на экране.

Способ решения

Мы создадим макрос, выводящий нужную панель на экран. Запуск макроса будет осуществляться с помощью кнопки, размещенной на одной из панелей, постоянно отображаемых в окне Microsoft Word.

> ***Примечание***
>
> Выбор способа вызова панели — с помощью кнопки, а не сочетания клавиш, — обусловлен тем, что потребность в инструментах, расположенных на этой панели, возникает довольно редко.

Решение задачи

Основные этапы: подготовка к записи макроса (шаги 1—2, 5), назначение кнопки макросу и ее оформление (шаги 3—4), вывод панели стилей на экран (шаги 6—7), завершение записи макроса (шаг 8).

1. Нажмите кнопку **Начать запись** (см. рис. 3.4), с помощью которой выводится диалоговое окно **Запись макроса**.
2. Введите имя макроса, например, **ShowMyButtonsPanel**, т. е. "показать панель с моими кнопками", введите описание в соответствующее поле и в разделе **Назначить макрос** нажмите кнопку **панели**.
3. Перетащите элемент **Normal.NewMacros.ShowMyButtonsPanel** из списка **Команды** диалогового окна **Настройка** на одну из основных панелей инструментов.
4. Наведите курсор на вновь созданную кнопку и измените выражение **Normal.NewMacros.ShowMyButtonsPanel** на более понятное, например, **Мои кнопки** (вариант — назначьте кнопке значок), после чего нажмите клавишу <Enter>.
5. Закройте диалоговое окно **Настройка**.
6. Наведите курсор на любую панель инструментов и нажмите правую кнопку мыши.
7. Выберите в появившемся списке панель инструментов **Мои кнопки**.
8. Завершите создание макроса, нажав кнопку с синим квадратиком, расположенную на панели записи макроса.

Что же мы имеем в итоге

В случае необходимости вы теперь можете мгновенно вывести на экран монитора нужную панель инструментов (в нашем случае — **Мои кнопки**). Чтобы снова скрыть ее, достаточно щелкнуть кнопку с крестиком, расположенную в правом верхнем углу этой панели.

3.4. Дополнительные сведения

3.4.1. Размещение дополнительных команд и макросов в меню

Макросы Microsoft Word можно размещать не только на панелях инструментов (в виде кнопок), но и в обычных меню программы. Более того, в меню можно включать команды, которые изначально в них отсутствовали (в том числе создавая дубликаты команд, представленных в других меню), а также стили, шрифты и другие категории элементов Microsoft Word. Для этого

следует вывести на экран диалоговое окно **Настройка** и выбрать на вкладке **Команды** в списке **Категории** нужный элемент: название меню, тематическую категорию команд или **Все команды**, **Макросы**, **Стили** и т. д. После этого необходимо прокрутить список **Команды** и выбрать в нем нужный элемент (команду, макрос, стиль и т. п.). Далее этот элемент требуется перетащить с помощью мыши на название меню в верхней части экрана, в которое хотите его поместить. После того как меню откроется, разместите в нем перетаскиваемый элемент на нужной позиции и отпустите кнопку мыши, после чего закройте диалоговое окно **Настройка**.

Если же вы отпустите кнопку мыши, когда перетаскиваемый элемент будет находиться между названиями каких-либо меню (или на одном из краев в меню), то станете счастливым обладателем довольно экзотической версии строки меню, в которой наряду с названиями отдельных меню будет представлена также кнопка, соответствующая перетаскивавшемуся элементу.

Вместе с тем, мы настоятельно призываем вас не злоупотреблять описанной выше возможностью, пользоваться ею с осторожностью и применять только в случае крайней необходимости. Дело в том, что в ряде случаев включение "нестандартных" команд в меню приводит к нежелательным последствиям, которые могут проявляться в блокировке некоторых встроенных команд, зависании программы и т. п. Поэтому правильный подход должен состоять в создании для этой цели собственного пользовательского меню. Создание такого меню осуществляется точно так же, как размещение новых кнопок на панель инструментов (соответствующая процедура описана ранее в *разд. 3.1.1*). Единственное отличие будет состоять только в том, что в списке **Категории** на вкладке **Команды** диалогового окна **Настройка** следует выбрать последний элемент, **Новое меню**, а не элемент **Все команды**.

Выбор и исполнение вновь созданной команды меню (или включенного в меню макроса) осуществляется традиционным для Microsoft Word образом.

Аналогичным образом можно включить в состав меню команды и макросы, которым соответствуют кнопки, расположенные на панелях инструментов (при этом справа от значка кнопки будет отображаться имя соответствующей команды или макроса). И наоборот, команды представленные в меню, можно разместить в виде соответствующих им кнопок на панелях инструментов. В обоих случаях перетаскиваемый элемент можно как копировать, так и просто перемещать из одного места в другое (с помощью способов, описанных ранее в *разд. 3.2.2*).

Чтобы удалить из меню любую из включенных в него таким образом команд (макрос и т. п.), следует еще раз открыть диалоговое окно **Настройка**, открыть соответствующее меню и перетащить из него нужный элемент, "сбросив" в окно документа Microsoft Word. При этом если вы отпустите кнопку мыши несколько раньше, над какой-либо панелью инструментов, на ней появится кнопка, соответствующая перетаскивавшемуся элементу.

3.4.2. Назначение сочетаний клавиш командам Microsoft Word

Как уже говорилось, разработчики Microsoft Word изначально присвоили многим командам программы индивидуальные сочетания клавиш (или даже несколько таких сочетаний). Однако для целого ряда команд сочетаний клавиш предусмотрено не было. Поэтому в случае необходимости пользователи должны назначать такие сочетания самостоятельно.

Нас в первую очередь интересуют команды работы с макросами. Они, как известно, "приписаны" к меню **Сервис**. Поэтому при задании сочетания клавиш этим командам необходимо в левом списке **Категории** диалогового окна **Настройка клавиатуры** выбрать элемент, соответствующий именно этому меню. Затем в правом списке **Команды** следует выбрать элемент списка, соответствующий нужной команде.

К сожалению, процедура эта не может быть унифицирована. Дело в том, что команды Microsoft Word можно разделить (конечно, весьма условно) на две группы. К первой относятся команды, которые явным образом по умолчанию отображаются в конкретных меню программы. Ко второй — команды "приписанные" к тому или иному меню, но по умолчанию в нем не отображаемые. Чтобы просмотреть команды, относящиеся к этим двум группам, необходимо открыть диалоговое окно **Настройка**, перейти на вкладку **Команды** и выбрать в списке **Категории** элемент, соответствующий тому или иному меню. Тогда справа в списке **Команды** отобразятся команды, "приписанные" к данному меню. Обратите внимание, что названия представленных здесь команд записаны по-русски (рис. 3.10).

Вместе с тем, здесь отображаются далеко не все команды Microsoft Word. Их полный перечень можно посмотреть только в общем списке команд, который выводится при выборе элемента **Все команды** в списке **Категории**.

Проблема, однако, состоит в том, что имена, под которыми эти команды представлены в общем списке, не всегда удается правильно "расшифровать". Во-первых, они записаны здесь по-английски, что налагает ограничения, связанные с адекватностью перевода на русский язык. Во-вторых, в ряде случаев, даже зная английский язык, практически невозможно догадаться, какое конкретно имя разработчики программы присвоили той или иной команде.

Пользователям, работающим в англоязычной версии Microsoft Word, повезло в этом смысле несколько больше: внутренние имена команд, закрепленных за тем или иным меню, формируются в общем случае по правилу "Имя_менюПонятное_имя_команды". Например, ToolsCustomize соответствует команде **Customize** (Настройка), содержащейся в меню **Tools** (Сервис). Однако знание этого принципа может помочь далеко не всегда, поскольку:

- к каждому из основных меню Microsoft Word приписано гораздо больше команд, чем по умолчанию в нем отображается, и поэтому понятные

имена многих команд нельзя "подсмотреть" в интерфейсе программы (напоминаем, что сейчас речь идет об ее англоязычной версии);

- разработчики Microsoft Word по каким-то причинам иногда не включают элемент, соответствующий названию меню, во "внутреннее" имя целого ряда команд.

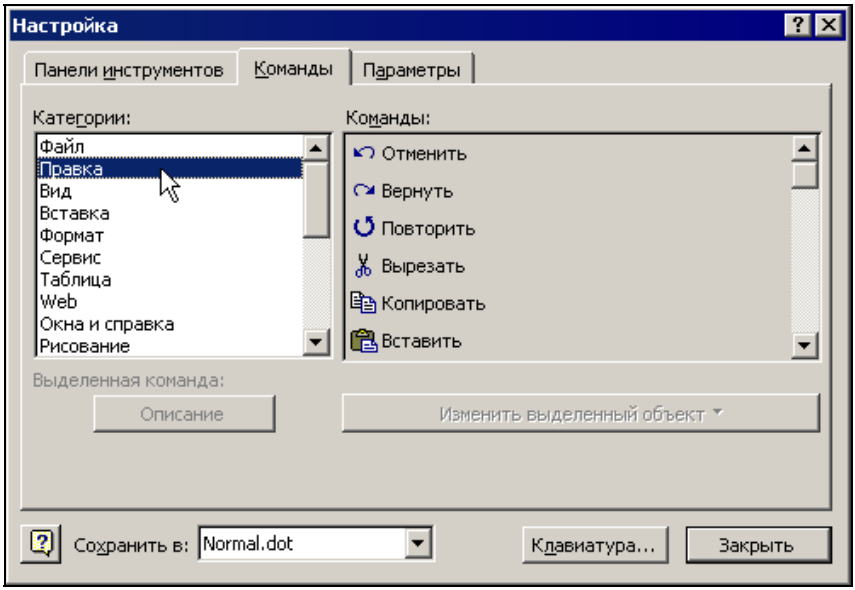


Рис. 3.10. Вкладка **Команды** диалогового окна **Настройка**. В правом списке **Команды** отображается список команд, приписанных к меню **Правка**

Так, к меню **Сервис** в диалоговом окне **Настройка** приписано более 50 команд (напоминаем, что в русской версии они отображаются в списке **Команды** по-русски), тогда как в самом меню **Сервис** по умолчанию представлено чуть более полутора десятка команд. В свою очередь, в диалоговом окне **Настройка клавиатуры** данному меню соответствует более 70 команд (в русской версии отображаются их англоязычные "внутренние" имена). При этом более чем у 30 команд "родовое" слово **Tools** в начале имени отсутствует.

Полезные сведения

Краткая содержательная характеристика команды, выбранной в списке **Команды** диалогового окна **Настройка клавиатуры**, приводится в нижней части данного окна. Правда, для некоторых команд такая характеристика не предусмотрена.

3.4.3. Дополнительные способы назначения сочетаний клавиш кнопкам Microsoft Word

В Microsoft Word имеется кнопка, позволяющая легко и быстро узнавать внутреннее имя встроенной команды, благодаря чему упрощается процедура назначения (или изменения) сочетания клавиш кнопкам и командам программы Microsoft Word. Эту кнопку можно обычным образом "вытащить" на панель инструментов из диалогового окна **Настройка**. В полном списке команд, представленном в этом диалоговом окне, ей соответствует элемент **ToolsCustomizeKeyboardShortcut**. По умолчанию этой кнопке присвоено имя **Сочетания клавиш**, которое и отображается на ее поверхности.

Процедура выяснения внутреннего имени команды Microsoft Word и назначения сочетания клавиш кнопке или команде меню Microsoft Word с помощью кнопки **Сочетания клавиш** выполняется так.

1. Нажмите кнопку **Сочетания клавиш**, после чего указатель мыши примет форму узора (⌘).

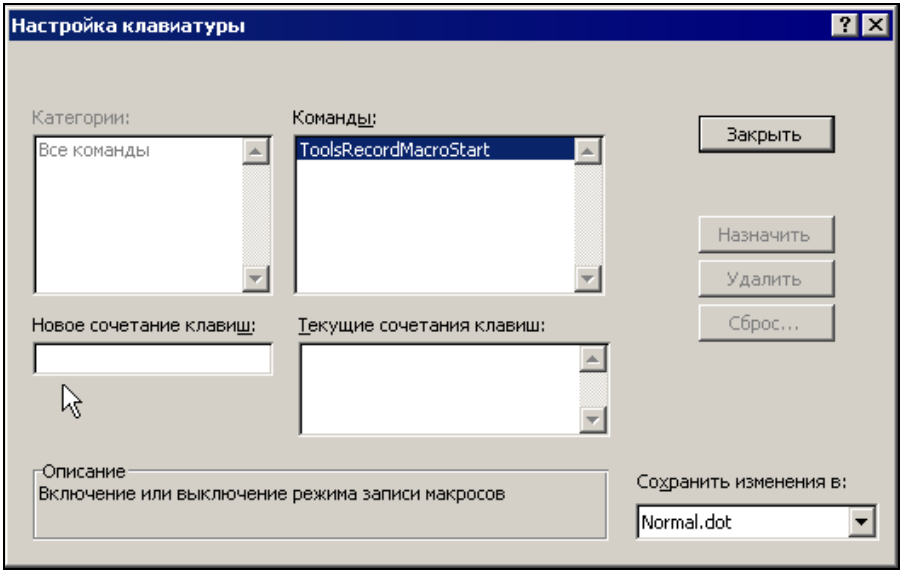


Рис. 3.11. Вид диалогового окна **Настройка клавиатуры**, которое выводится в результате применения кнопки **Сочетание клавиш** к кнопке **Записать макрос**

2. Наведите этот указатель на кнопку (или команду меню), которой хотите присвоить сочетание клавиш, и щелкните кнопкой мыши. Появится диалоговое окно **Настройка клавиатуры**, в списке **Команды** которого будет

содержаться только один элемент, представляющий собой внутреннее имя соответствующей команды (см. рис. 3.11, где показан результат использования кнопки **Сочетания клавиш** применительно к кнопке **Записать макрос**).

3. Введите желаемое сочетание клавиш в поле **Новое сочетание клавиш** и закройте диалоговое окно **Настройка клавиатуры**.

> ***Примечание***
>
> Несколько сложнее выполняется описанная только что процедура применительно к кнопкам, при нажатии которых выводится заготовка для создания таких структурных элементов документов Microsoft Word, как таблицы (кнопка **Добавить таблицу**) и колонки (кнопка **Столбцы**). В этом случае потребуется выполнить дополнительный щелчок непосредственно в появившейся на экране заготовке.

К сожалению, кнопку **Сочетание клавиш** нельзя использовать для назначения сочетания клавиш кнопкам, запускающим макросы.

3.4.4. Дополнительные сведения о некоторых полезных кнопках Microsoft Word

В этом разделе мы расскажем о некоторых полезных, но не совсем обычных кнопках Microsoft Word. Часть этих кнопок при работе с программой всегда находится у вас перед глазами, однако большинство пользователей даже не подозревают об их существовании. Другие — скрыты в недрах программы, но могут оказаться довольно полезными.

Если вы посмотрите в нижнюю часть рабочего окна Microsoft Word, то в средней части строки состояния увидите несколько малопонятных аббревиатур, записанных прописными (как правило, тусклыми) буквами (рис. 3.12). Возможно, вы не знаете, но это тоже кнопки. Непосредственное отношение к задачам, которые рассматриваются в данной главе, имеет только кнопка **ЗАП**. Если щелкнуть ее дважды, то на экран будет выведено диалоговое окно **Запись макроса** (надпись на данной кнопке станет отображаться ярким шрифтом только при переходе в режим записи макроса).

Кратко поясним назначение других кнопок, также представленных в строке состояния. Двойной щелчок кнопки **ИСПР** переводит программу в режим редактирования с отображением исправлений. Двойной щелчок по кнопке **ВДВ** включает режим выделения текста, в котором все действия пользователя, ведущие к перемещению точки вставки (курсора), приводят к выделению соответствующих фрагментов документа (более подробно об этом режиме будет сказано в *разд. 4.1.1* следующей главы, посвященной макросам, основанным на процедуре поиска). Двойной щелчок кнопки **ЗАМ** включает "забойный" режим ввода текста (старый текст не сдвигается вправо, а удаля-

ется). Чтобы выйти из указанных выше режимов, следует еще раз дважды щелкнуть соответствующую кнопку.

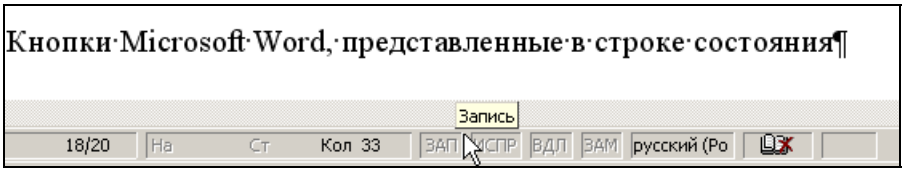


Рис. 3.12. Кнопки Microsoft Word, представленные в строке состояния

Двойной щелчок кнопки, расположенной справа от кнопки **ЗАМ**, на которой отображается текст "русский (Россия)" (см. рис. 3.10), выводит на экран диалоговое окно **Язык** (более подробно о возможностях его использования будет сказано в *разд. 4.2.1 следующей главы*).

Справа от данной кнопки расположена кнопка, на которой отображается значок с изображением раскрытой книги (отображается не всегда). Наличие этого значка свидетельствует о том, что включен режим автоматической проверки орфографии (т. е. установлен соответствующий флажок на вкладке **Правописание** диалогового окна **Параметры**). Двойной щелчок на данной кнопке в этом случае приводит к выделению первой (относительно начала документа) орфографической ошибки (если таковая имеется в документе). Одновременно на экран выводится контекстное меню с командами, связанными с проверкой орфографии. Повторный двойной щелчок приводит к выделению следующей орфографической ошибки и т. д.

> ***Примечание***
>
> Справа от данной кнопки расположена "пустая" область, в которой при сохранении файла отображается значок с изображением дискеты. При наведении на эту область указателя мыши в программе Microsoft Word 2000 появляется всплывающая подсказка с текстом "Сохранение фона", что является явной ошибкой локализации (перевода на русский язык) интерфейса программы. В последующих версиях Microsoft Word эта ошибка была исправлена, в результате чего текст всплывающей подсказки в них гласит "Фоновое сохранение".

В Microsoft Word имеется также команда, позволяющая получить справку по основным элементам интерфейса этой программы. Эта команда содержится в меню **Справка** и имеет *понятное* имя **Что это такое** (может также исполняться с помощью сочетания клавиш <Shift>+<F1>). Если разместить кнопку, соответствующую данной команде (внутренне имя которой **HelpTool**), на панели инструментов, на ней по умолчанию будет отображаться значок  . Чтобы получить краткую справку о назначении того или иного элемента интерфейса (команды меню, кнопки, элементов строки со-

стояния и т. п.), необходимо нажать эту кнопку (или выполнить указанную выше команду), навести указатель мыши, который примет вид стрелки с вопросительным знаком, на нужный элемент и щелкнуть.

3.4.5. Дополнительные сведения о некоторых полезных командах Microsoft Word

Как отмечалось в этой главе, в Microsoft Word имеется довольно много команд, которые в целях повышения эффективности работы с программой имеет смысл разместить на панели инструментов (стандартной или специально созданной для этой цели).

Создаваемые панели можно оставить "плавающими" в рабочей области окна Word (именно такое расположение панели, кстати, часто оказывается весьма удобным, поскольку максимально сокращает время, требуемое для нажатия нужной кнопки), либо прикрепить к любой из сторон окна (для этого нужно перетащить панель как можно ближе к соответствующей стороне). Панель можно также поместить — если позволяет место — "встык" с одной из стандартных панелей (также методом перетаскивания) либо вообще убрать с экрана (как — описывалось ранее).

По умолчанию новая панель отображается в виде "одномерной" горизонтальной полоски. Однако если у вас имеется плавающая панель, содержащая большое количество кнопок, работать с ней будет гораздо удобнее, если придать панели форму прямоугольника, на поверхности которого кнопки располагаются в несколько рядов. Чтобы изменить форму панели, нужно навести указатель мыши на ее нижний край и, когда указатель превратится в двустороннюю стрелку, перетащить этот край вниз. Форма панели при этом будет изменяться. Добившись нужного результата, отпустите кнопку мыши.

> ***Предупреждение***
>
> Напоминаем, что если при выходе из Microsoft Word вам будет предложено сохранить шаблон Normal.dot, обязательно сделайте это, иначе все внесенные вами изменения в настройки программы будут потеряны.

Некоторые из команд, которые мы рекомендуем дополнительно разместить на панелях инструментов, содержатся в меню программы, другие представлены в общем списке команд, к которому можно получить доступ в диалоговом окне **Настройка**.

С определенной долей условности можно выделить две основных группы таких команд: команды работы с файлами и команды работы с документами.

Далее приводится табл. 3.1 с примерным перечнем команд, которые, с нашей точки зрения, имеет смысл разместить (в виде кнопок) на панелях инструментов (в дополнение к кнопкам, уже расположенным на них по умол-

чанию). В первом столбце таблицы указано имя команды, запускаемой при нажатии кнопки, во втором — меню, в котором эта команда по умолчанию содержится (если таковое имеется), в третьем — "внутреннее" имя команды, под которым она фигурирует в списке **Все команды** диалогового окна **Настройка**, в четвертом — сочетание клавиш, по умолчанию назначенных данной команде, в пятом — значок, отображаемый по умолчанию на кнопке, назначенной команде (если имеется), в шестом — комментарии об особенностях использования данной команды.

Таблица 3.1. *Примерный список команд, которые рекомендуется дополнительно разместить на панелях инструментов*

Команда	Меню	Внутреннее имя	Сочетание клавиш	Значок	Комментарий
Команды работы с файлами					
Сохранить как...	**Файл**	**FileSaveAs**	<F12>	—	
Сохранить все	—	**FileSaveAll**	—		Сохраняет все открытые документы, в которые были внесены изменения (в том числе шаблон документа)
Закрыть все	—	**FileCloseAll**			Выводит на экран измененные документы и предлагает их сохранить
Выход	**Файл**	**FileExit**	—	—	
Команды работы с документами					
Следующее окно	—	**NextWindow**	<Ctrl>+<F6>, <Alt>+<F6>	—	Осуществляет циклический переход
Предыдущее окно	—	**PrevWindow**	<Ctrl>+<Shift>+ +<F6>, <Alt>+<Shift>+ +<F6>	—	То же (в обратную сторону)
Упорядочить все	**Окно**	**WindowArrangeAll**	—		Одновременно отображает на экране окна всех раскрытых документов Microsoft Word

Таблица 3.1 *(продолжение)*

Команда	Меню	Внутреннее имя	Сочетание клавиш	Значок	Комментарий
Команды работы с документами					
Разделить окно	**Окно**	**DocSplit**	<Alt>+<Ctrl>+ +<S>		Делит окно документа на две части, в которых отображается один и тот же документ. Навигация в этих областях осуществляется независимым образом
Другие окна	—	**WindowList**	—	—	Выводит список открытых документов, из которого можно перейти в нужное окно
Параметры страницы	**Файл**	**FilePageSetup**	—		Выводит одноименное диалоговое окно
Закладка	**Вставка**	**EditBookmark**	<Ctrl>+<Shift>+ +<F5>	—	То же
Номера страниц	**Вставка**	**InsertPageNumbers**	—		То же
Найти	**Правка**	**EditFind**	<Ctrl>+<F>		Выводит диалоговое окно **Найти и заменить** (вкладка **Найти**)
Найти далее	—	**RepeatFind**	<Shift>+<F4>		
Заменить	**Правка**	**EditReplace**	<Ctrl>+<H>		Выводит диалоговое окно **Найти и заменить** (вкладка **Заменить**)
Перейти	**Правка**	**EditGoTo**	<Ctrl>+<G>, <F5>		Выводит диалоговое окно **Найти и заменить** (вкладка **Перейти**)
Параметры	**Сервис**	**ToolsOptions**	—	—	Выводит одноименное диалоговое окно

Таблица 3.1 *(окончание)*

Команда	Меню	Внутреннее имя	Сочетание клавиш	Значок	Комментарий
Команды работы с документами					
Настройка	**Сервис**	**ToolsCustomize**	—	—	То же
Записать макрос	**Сервис**	**ToolsRecord MacroStart**	—		То же

3.5. Что нового в этой главе

В этой главе мы занимались формированием удобной рабочей среды, адаптированной к повседневной работе с Microsoft Word, а также позволяющей в случае необходимости оперативно вызывать на экран наборы инструментов, предназначенных для решения специализированных задач. Здесь получили свое дальнейшее развитие и уточнение подходы и методы, описанные в предыдущей главе.

На некоторое время — по крайней мере, до тех пор, пока вы не закончите чтение этой книжки, — в круг ваших повседневных забот и интересов будут входить и вопросы, связанные с оптимизацией рабочей среды Microsoft Word и созданием макросов. Поэтому в этой главе мы первым делом позаботились о размещении на панели инструментов вспомогательных кнопок, с помощью которых обеспечивается оперативный доступ к диалоговым окнам **Настройка** и **Настройка клавиатуры**. Благодаря этому вы теперь будете "экономить" по два-три щелчка кнопкой мыши на выполнении рутинных процедур, связанных с запуском процесса записи макросов и присвоением командам и макросам сочетаний клавиш. Это позволит вам больше внимания уделять содержательной творческой работе.

В этой главе вы узнали также, что в Microsoft Word имеются встроенные панели инструментов, которые по умолчанию не отображаются на экране, и освоили приемы их быстрого вывода на экран. Среди таких панелей — панель **Visual Basic**, на которой содержатся кнопки, используемые при решении задач, связанных с записью макросов. Большинство инструментов, запускаемых с помощью этих кнопок, предназначены, вообще говоря, для пользователей, уже владеющих навыками программирования. Поэтому мы выбрали те из них, которые нам пригодятся в текущей работе, и разместили на одной из стандартных панелей инструментов. При этом вы познакомились с полезными приемами работы с кнопками (перетаскиванием и копированием) и узнали, каким образом можно быстро выяснить назначение кнопок и присвоенные им сочетания клавиш.

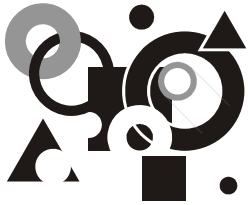
Вы научились создавать собственные панели инструментов и произвели настройку стандартных панелей, убрав с них "лишние" кнопки и добавив кнопки, которые будет полезно всегда иметь под рукой при дальнейшей работе с Microsoft Word.

Нами был создан макрос, позволяющий в случае необходимости быстро выводить на экран скрытую панель, содержащую инструменты, необходимые для решения той или иной конкретной задачи.

Те из вас, кто ознакомился с материалом раздела дополнительных сведений, узнали, что макросы и дополнительные команды можно размешать не только на панелях инструментов, но и в меню Microsoft Word (как стандартных, так и созданных пользователем). И наоборот, команды, представленные в меню, можно размещать в виде соответствующих им кнопок на панелях инструментов.

Здесь же приводятся сведения об особенностях внутренних имен команд Microsoft Word и способе, позволяющем быстро выяснить, какое имя присвоили разработчики программы той или иной команде. Использование этого способа является очень эффективным при назначении сочетания клавиш кнопке панели инструментов или команде меню.

В этом разделе содержатся также сведения о ряде полезных кнопок, о существовании которых многие пользователи даже и не догадываются. В конце раздела приводится примерный список команд, рекомендуемых нами для размещения на основных панелях инструментов в дополнение к уже имеющимся на них кнопкам.



Глава 4

Объявляется розыск

Макросы, позволяющие удалять заданный фрагмент текста. Просмотр фрагментов текста, выделенных с помощью цвета. Подкрашивание всех вхождений выделенного текста. Создание макроса, позволяющего упростить проверку правописания русскоязычных документов, содержащих большое количество знаков, записанных латинскими буквами. Макросы, предназначенные для исправления ошибок форматирования документа (удаление лишних пробелов между словами и перед знаками препинания). Дополнительные сведения.

Точнее — поиск, поскольку в этой главе мы займемся созданием макросов для работы с текстом, которые основаны на использовании возможностей процедуры поиска, реализованных в Microsoft Word. Большинство пользователей прибегают к ней лишь для того, чтобы просто отыскать в документе тот или иной *конкретный* фрагмент текста (слово, часть слова, выражение и т. п.), не подозревая даже, что с ее помощью можно выполнять не только поиск "материальных объектов" (то бишь текста), но и таких признаков, как "подкрашенность" знаков, их принадлежность к тому или иному языку, стиль форматирования, положение на странице и т. п.

"Нематериальный" характер таких признаков состоит в том, что при поиске не надо указывать, какой именно текст требуется отыскать (соответствующее поле диалога поиска остается пустым). Поиск в данном случае осуществляется по принципу "найди то — не знаю конкретно что" — будут найдены ВСЕ фрагменты текста, характеризующиеся указанными признаками, но точное написание которых нам заранее не известно.

Более того, применяя процедуру поиска с заменой, эти признаки можно при желании заменять на другие, такие же нематериальные, или же на обычный текст. Например, если в документе имеются выражения, записанные курсивом, их начертание можно мгновенно изменить на полужирное. Или же вместо каждого курсивного фрагмента подставить в документ фразу типа "Здесь раньше было что-то, написанное курсивом".

Однако мы несколько отвлеклись. Желающие более детально ознакомиться с возможностями процедур поиска и замены в Microsoft Word могут обратиться к разделу "Дополнительные сведения" в конце настоящей главы или

к соответствующим справочным материалам. У нас же сейчас другая цель — применить некоторые из названных возможностей на практике в целях повышения эффективности работы с текстом.

4.1. Это элементарно, Ватсон...

Сначала мы покажем, каким образом позволяют повысить эффективность работы с текстом возможности поиска, известные, по-видимому, подавляющему большинству пользователей Microsoft Word.

4.1.1. Приказано: найти и уничтожить

Довольно часто при редактировании текста приходится удалять целые фрагменты предложения или даже абзаца. Дело это хлопотное и отнимает относительно много времени. Однако можно существенно упростить выполнение такого рода операций с помощью специально созданных для этой цели макросов.

Ситуация

При редактировании какого-либо предложения вы решили удалить текст, начиная от текущей позиции курсора и вплоть до конца предложения.

Что требуется получить

Вы нажимаете сочетание клавиш, и весь текст, расположенный в интервале между текущей позицией курсора и точкой в конце предложения, мгновенно удаляется.

Способ решения

Прежде чем приступать к описанию процедуры создания макроса, с помощью которого решается сформулированная выше задача, мы расскажем вам еще об одной очень полезной, но малоизвестной функции Microsoft Word. Данная функция позволяет расширять зону выделения текста, не нажимая клавишу <Shift>. Так вот, обойтись без клавиши <Shift> можно, если перед началом выделения фрагмента документа будет нажата клавиша <F8>. Более того, в этом случае расширение зоны выделения может производиться с помощью процедуры поиска, чем мы и воспользуемся при создании нашего макроса.

> ***Примечание***
>
> К сожалению, возможность выделения фрагмента текста до знака, введенного с клавиатуры, с помощью клавиши <F8> не работает со знаками кириллицы.

Решение задачи

Основные этапы: запуск процедуры записи макроса (шаги 1—2); активизация функции выделения (шаг 3); поиск точки в конце предложения и выделение текста, расположенного между ею и позицией курсора (шаги 4—8); удаление выделенного фрагмента (шаг 9); завершение записи макроса (шаг 10).

Перед запуском процедуры записи макроса выведите на экран какой-либо текстовый документ Microsoft Word и поместите курсор приблизительно в середине какого-либо предложения.

1. Выведите на экран диалоговое окно **Запись макроса** и запишите имя создаваемого макроса, например, **DelToPoint** (т. е. "удалить до точки").

   #### *Примечание*

   Каким образом осуществляются все вспомогательные действия, выполняемые при записи макросов (вызов диалоговых окон **Запись макроса**, **Настройка клавиатуры** и т. п.), детально рассказывалось в предыдущих главах. То же самое можно сказать о правилах записи имен макросов, назначении сочетаний клавиш, создании кнопок и т. д. Поэтому в дальнейшем мы больше не будем подробно останавливаться на описании этих действий.

2. Присвойте этому макросу сочетание клавиш (например, <Alt>+<→>), после чего приступайте непосредственно к записи макроса.

   #### *Примечание*

   Вместо назначения сочетаний клавиш можно по желанию создать для этого макроса специальную кнопку.

3. Нажмите клавишу <F8>.
4. Нажмите стандартное сочетание клавиш <Ctrl>+<F>, после чего откроется диалоговое окно **Найти и заменить** (рис. 4.1).
5. Введите в поле **Найти** точку.

   #### *Примечание*

   Проследите, чтобы в поле **Найти** не содержалось никаких других знаков, кроме вводимых вами (в нашем случае — точки) — в том числе пробелов. Это требование относится и ко всем другим процедурам поиска, описываемым в данной главе.

6. В случае если диалоговое окно поиска открылось в компактном виде, нажмите кнопку **Больше** и убедитесь, что в поле **Направление** отображается текст **Везде** или **Вперед**. Если это не так, задайте направление поиска, выбрав его в соответствующем списке.

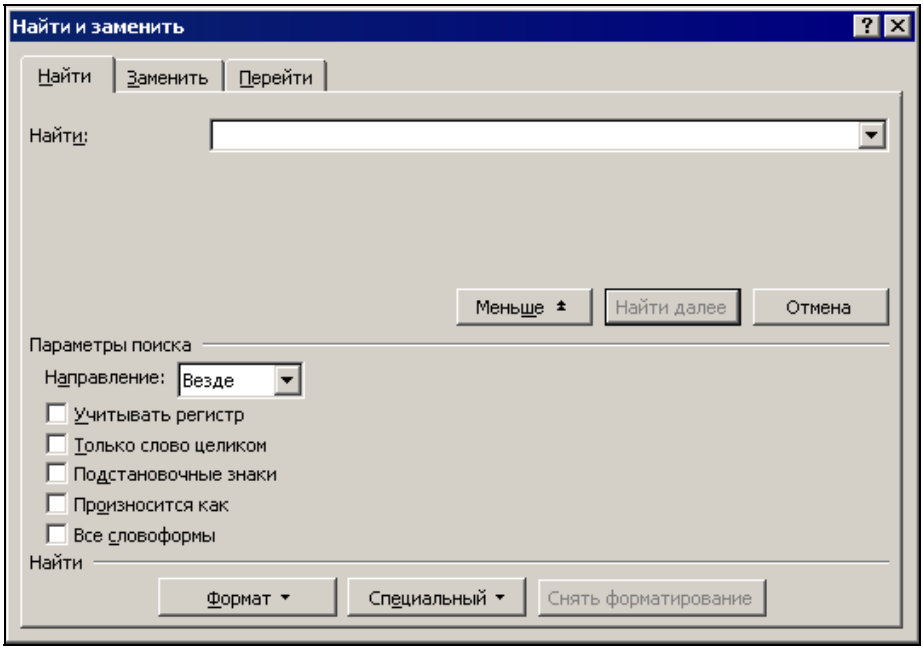


Рис. 4.1. Вкладка **Найти**
диалогового окна **Найти и заменить**

Примечание

Проследите также за тем, чтобы все остальные параметры поиска были отключены, т. е. все флажки в группе **Параметры поиска** не содержали галочек, т. е. не были установлены — см. рис. 4.1 (эти флажки могли остаться установленными после выполнявшейся ранее процедуры поиска). Чтобы снять флажок, необходимо щелкнуть соответствующий квадратик. Кроме того, необходимо убедиться, что кнопка **Снять форматирование** не активна (т. е. выглядит тусклой, свидетельствуя о том, что при поиске не будут учитываться какие-либо элементы форматирования текста). В противном случае следует нажать эту кнопку, после чего происходит "сброс" всех элементов форматирования.

7. Нажмите кнопку **Найти далее**, после чего закройте диалоговое окно поиска.
8. Нажмите клавишу <Shift>, а затем клавишу <→>, чтобы исключить из выделенного фрагмента точку.
9. Нажмите клавишу <Delete>, чтобы удалить выделенный фрагмент текста.
10. Завершите процедуру создания макроса.

Что мы имеем в итоге

С помощью только что созданного макроса **DelToPoint** вы можете, нажав сочетание клавиш <Alt>+<→>, мгновенно удалить текст от текущей позиции курсора до конца предложения.

Очевидно, что описанная выше "модель" может быть использована при создании целого семейства макросов. Например, макроса, позволяющего удалять фрагмент текста от текущей позиции курсора до ближайшей (справа) запятой, скобки, вопросительного знака и т. п.

Несколько отличается от описанных выше процедура создания макроса (назовем его **DelToParagraphMark**, т. е. "удалить до знака абзаца"), удаляющего весь текст (включая точку в конце предложения) от позиции курсора до конца абзаца. На шаге 5 описанной выше процедуры в этом случае потребуется ввести в поле **Найти** специальный символ, обозначающий конец абзаца. Для этого необходимо нажать в диалоговом окне поиска кнопку **Специальный** и выбрать в открывшемся списке элемент **Знак абзаца** (рис. 4.2).

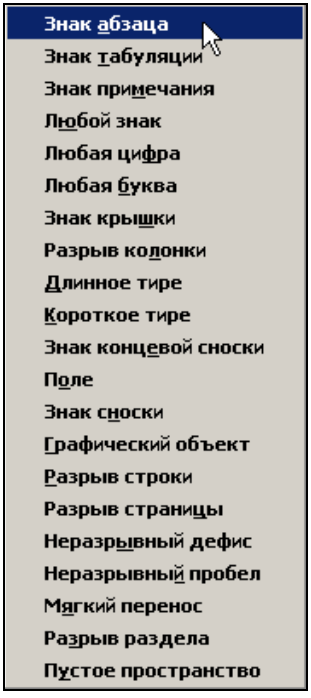


Рис. 4.2. Перечень специальных символов, используемых при поиске в диалоговом окне **Найти и заменить** (выводится при нажатии кнопки **Специальный**, флажок **Подстановочные знаки** снят)

В поле **Найти** появится выражение ^p, с помощью которого в Microsoft Word при поиске обозначается символ конца абзаца. Описание других специальных символов, позволяющих легко отыскивать самые различные структурные элементы форматирования документа Microsoft Word, приводится в разделе "Дополнительные сведения" в конце главы.

Если вы решите заняться созданием некоторого набора подобных средств уничтожения "лишнего" текста, то на данном этапе знакомства с макросами вам придется повторить всю описанную ранее процедуру для каждого знака, до которого требуется удалить текст (этот знак вводится в поле **Найти** на шаге 5). Однако в последующих главах книги мы расскажем вам, как можно значительно упростить и ускорить создание подобных "типовых" (т. е. основанных на использовании одинаковых алгоритмов) макросов. Модифицировать нужным образом текст исходного макроса смогут даже те читатели, которые твердо решили "не связываться" с программированием. Им нужно будет ознакомиться лишь с "переходной" главой 6, в которой рассказывается о внесении изменений в текст макросов с помощью традиционных средств Microsoft Word (не имеющих никакого отношения к программированию) — таких как редактирование, копирование, вставка из буфера обмена и т. п.

4.1.2. Особые приметы: они цветные, сэр

Рассмотрим теперь несколько иную задачу, связанную с использованием возможностей поиска. Довольно часто при знакомстве с текстом документа пользователь отмечает в нем некоторые фрагменты, выделяя их тем или иным цветом. Поиск этих фрагментов в тексте в дальнейшем, как правило, осуществляется при помощи обычной прокрутки, что не очень удобно и, если документ большой, может потребовать больших затрат времени. К тому же данный способ не гарантирует, что вы не пропустите какой-либо важный помеченный фрагмент. Мы предлагаем вам создать макрос, который позволит мгновенно, не допуская пропусков, переходить от одного помеченного фрагмента документа к другому.

Ситуация

При предварительном просмотре довольно объемистого документа вы пометили в нем тем или иным цветом некоторые заинтересовавшие вас фрагменты с тем, чтобы в дальнейшем ознакомиться с ними более внимательно. "Ручной" поиск этих фрагментов может занять много времени, и поэтому вы хотели бы его по возможности ускорить.

Что требуется получить

Вы нажимаете кнопку на панели инструментов и мгновенно переходите к первому из выделенных фрагментов, затем — аналогичным образом — ко второму и т. п.

Способ решения

Для решения поставленной задачи мы воспользуемся имеющейся в Microsoft Word возможностью поиска, позволяющей отыскивать любой фрагмент текста, который был выделен (подкрашен) цветом.

Если вы хотите начать просмотр выделенных фрагментов, начиная с самого первого из них, нажмите предварительно стандартное сочетание клавиш <Ctrl>+<Home>, чтобы переместиться в начало документа. Это действие нельзя включать в саму процедуру записи макроса, поскольку с помощью полученного таким образом макроса можно будет отыскать только первый фрагмент выделенного текста.

Решение задачи

Основные этапы: запуск процедуры записи макроса (шаги 1—2); поиск выделенного цветом фрагмента (шаги 3—7); завершение записи макроса (шаги 8—9).

1. Выведите на экран диалоговое окно **Запись макроса** и запишите имя создаваемого макроса, например, **FindColor** (т. е. "найти цвет").
2. Создайте для этого макроса специальную кнопку, после чего приступайте непосредственно к записи макроса.

> ***Примечание***
>
> В данном случае запуск макроса удобнее осуществлять с помощью кнопки, а не сочетания клавиш, поскольку использоваться этот макрос будет, по-видимому, не очень часто.

3. Нажмите стандартное сочетание клавиш <Ctrl>+<F>, после чего откроется диалоговое окно **Найти и заменить**.
4. Если в поле **Найти** содержатся какие-либо знаки, удалите их (в том числе все пробелы).
5. В случае если диалоговое окно поиска открылось в компактном виде, нажмите кнопку **Больше** и убедитесь, что в поле **Направление** отображается текст **Везде** или **Вперед**. В противном случае выберите в списке какой-либо из этих элементов (см. также примечание к шагу 6 процедуры, описанной при создании предыдущего макроса).
6. Нажмите кнопку **Формат** и выберите в открывшемся списке элемент **Выделение цветом** (рис. 4.3).

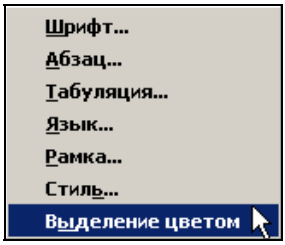


Рис. 4.3. Перечень признаков форматирования, поиск и замена которых могут осуществляться в процедуре поиска. Выводится при нажатии кнопки **Формат**

7. Нажмите кнопку **Найти далее**, после чего закройте диалоговое окно поиска.

> ***Примечание***
>
> К сожалению, задать поиск какого-либо *конкретного* цвета выделения в Microsoft Word, не прибегая к программированию, невозможно. При выполнении макроса будут последовательно отыскиваться ВСЕ подкрашенные фрагменты, независимо от цвета, использованного для выделения. Обещаем, что, прочитав эту книгу до конца, вы научитесь успешно решать подобные задачи *(см. гл. 16)*.

8. Нажмите клавишу <←>, чтобы снять выделение с найденного при поиске фрагмента текста.
9. Завершите процедуру создания макроса.

Что мы имеем в итоге

Для перехода от одного подкрашенного цветом фрагмента документа к следующему вам потребуется всего лишь периодически нажимать кнопку на панели инструментов (если вы хотите начать просмотр с самого первого из выделенных фрагментов, предварительно нажмите сочетание клавиш <Ctrl>+ +<Home>).

4.1.3. Своих героев народ должен знать в лицо

Рассмотрим теперь задачу, которая является в определенном смысле обратной по отношению к только что описанной. Там мы искали подкрашенные фрагменты текста в документе, а теперь сами займемся их выделением и подкрашиванием.

При редактировании или чтении документа иногда бывает полезно сделать более наглядным отображение в нем тех или иных слов или небольших

фрагментов текста или отдельных слов. Например, чтобы сразу же увидеть, где именно в документе располагаются ссылки на таблицы или рисунки.

Ситуация

Вы знакомитесь с документом, в котором имеются ссылки на таблицы, приводящиеся в приложении к этому документу. Все ссылки строятся по одинаковому образцу и имеют вид "см. табл. N", где N — номер таблицы. Вы бы хотели облегчить себе поиск подобных ссылок.

Что требуется получить

Вы нажимаете кнопку на панели инструментов, после чего все имеющиеся в тексте документа ссылки на таблицы почти мгновенно окрашиваются в указанный вами цвет.

Способ решения

Мы создадим макрос, который будет отыскивать вхождения в документ конкретного фрагмента текста (в нашем случае выражения "см. табл.") и выделять его с помощью процедуры поиска с заменой заранее определенным цветом, например, желтым.

Полезные сведения

Обратите внимание, что в процедурах поиска с заменой, в которых производится добавление какого-либо свойства к элементу, содержащемуся в поле **Найти,** этот элемент не требуется воспроизводить в поле **Заменить на**. В этом поле указывается только добавляемое свойство (например, курсивное начертание шрифта). Таким образом, в этом случае фактически выполняется замена "отсутствия" того или иного свойства у элемента, указанного в поле **Найти**, на "наличие" этого свойства. Если же поле замены не будет содержать ни текста, ни каких-либо свойств или специальных знаков, то в результате выполнения процедуры поиска с заменой, определенной таким образом, элемент, указанный в поле **Найти**, будет просто удаляться из документа. Иногда и такой прием может пригодиться для решения практических задач (например, с его помощью можно удалить из документа все примечания).

Решение задачи

Основные этапы: подготовительный этап (шаги 1—3); запуск процедуры записи макроса (шаги 4—5); поиск в документе вхождений выражения "см. табл." и выделение их цветом (шаги 6—11); завершение записи макроса (шаг 12).

1. Найдите и выделите интересующий вас фрагмент текста (в нашем случае это выражение "см. табл.").
2. Скопируйте в буфер обмена выделенное выражение, нажав сочетание клавиш <Ctrl>+<C> (или <Ctrl>+<Ins>).

3. Нажмите клавишу <←>, чтобы снять выделение с фрагмента текста.
4. Выведите на экран диалоговое окно **Запись макроса** и запишите имя создаваемого макроса, например, **ColorTermSeeTab**. (т. е. "подкрасить выражение см. “табл.”").
5. Создайте для этого макроса специальную кнопку, после чего приступайте непосредственно к записи макроса.
6. Откройте на панели инструментов средство **Выделение цветом** и выберите желаемый цвет (например, желтый).
7. Нажмите стандартное сочетание клавиш <Ctrl>+<H>, после чего откроется диалоговое окно **Найти и заменить**.
8. Нажмите сочетание клавиш <Ctrl>+<V> (или <Shift>+<Ins>), чтобы вставить в поле **Найти** выражение, скопированное на шаге 2 в буфер обмена.
9. Нажмите клавишу <Tab>, чтобы перейти в поле **Заменить на**. Это поле не должно содержать каких-либо знаков, в том числе пробелов. В противном случае — удалите их.
10. Нажмите кнопку **Формат** и выберите в открывшемся списке элемент **Выделение цветом**.
11. Нажмите кнопку **Заменить все** и дождитесь, когда Microsoft Word произведет процедуру замены (если при этом будет выведено сообщение с предложением продолжить поиск с начала документа, нажмите кнопку **Да**).
12. Закройте диалоговое окно после завершения процедуры поиска и завершите процедуру создания макроса.

Что мы имеем в итоге

Обращаем ваше внимание на то, что созданный только что макрос является, так сказать, "штучным", т. е. с его помощью выполняется подкрашивание некоторого *конкретного* текста — в нашем случае выражения "см. табл.". Однако что делать, если вы захотите подкрасить какое-либо другое выражение или слово? И опять мы вынуждены ответить: на данном этапе знакомства с макросами вам придется для решения этой задачи повторить описанную выше процедуру.

Как уже отмечалось, чтобы научиться легко и быстро создавать "типовые" макросы, следует ознакомиться, по крайней мере, с материалом "переходной" *гл. 6*, в которой рассказывается о внесении изменений в текст макросов с помощью традиционных средств Microsoft Word — редактирования, копирования, вставки при помощи буфера обмена и т. п. Если же вы не ограничитесь лишь знакомством с главой 6 и продолжите чтение книги, то сможете освоить основы программирования на языке Visual Basic for

Applications (VBA), что позволит вам значительно повысить эффективность как самих создаваемых вами макросов, так и процесса их создания. Например, только что рассмотренный макрос можно сделать универсальным, если внести незначительные изменения в его текст, благодаря которым он станет работать по следующей схеме: чтобы подкрасить в документе все вхождения интересующего вас выражения (любого), следует просто выделить это выражение и нажать соответствующую кнопку на панели инструментов *(см. гл. 15)*.

Тем не менее опыт, приобретенный вами при создании макроса, описанного в настоящем разделе, не окажется бесполезным. В той же главе 6 на его основе мы создадим "интеллектуальный" макрос, позволяющий выполнять первичную литературную правку документа.

4.2. Поиск продолжается...

В начале этой главы мы уже говорили о тех мощных возможностях поиска, которыми обладает Microsoft Word. В предыдущих разделах было показано, каким образом использование самых элементарных из этих средств при создании макросов позволяет упростить и ускорить редактирование текста. Пришло время ознакомиться и с некоторыми из более "продвинутых" возможностей поиска (их полное описание приводится в *разд. "Дополнительные сведения"*).

Далее в этой главе мы займемся созданием трех макросов. Первый из них позволит существенно облегчить проверку правописания "русскоязычных" документов, содержащих большое количество латинских букв. Два других макроса предназначены для исправления таких ошибок в оформлении текста документа, как наличие лишних пробелов между словами и пробелов перед знаками препинания.

4.2.1. Иностранцев просят не беспокоиться...

На этом примере будут продемонстрированы возможности использования диапазона знаков в процедуре поиска и замены. Диапазон знаков указывается в процедуре поиска с помощью квадратных скобок.

> ***Примечание***
>
> При описании приведенной далее и всех последующих задач предполагается, что в полях **Найти** и **Заменить на** диалогового окна **Найти и заменить** не содержится каких-либо знаков (в том числе пробелов) и не задано каких-либо признаков форматирования, которые могли остаться от предыдущих процедур поиска. В противном случае перед заполнением этих полей из них следует удалить все "лишние" знаки и отменить форматирование, нажав кнопку **Снять форматирование**.

Ситуация

Вам нужно выполнить проверку правописания документа, содержащего большое количество аббревиатур, фамилий, формул и т. п., записанных латинскими буквами. Правильно ли записаны эти выражения, проверять не требуется. Однако в словарях, используемых программой при проверке правописания этих выражений, как правило, нет, и поэтому скучать вам не придется — только успевай нажимать кнопку **Пропустить все**. Вообще говоря, в Microsoft Word имеется возможность пометить те или иные фрагменты текста как не требующие проверки. Однако понятно, что ее "ручная" реализация при наличии многочисленных "вкраплений" латинских знаков (наш случай) займет еще больше времени, чем сама проверка правописания. Поэтому попробуем автоматизировать соответствующую процедуру с помощью макроса.

Что требуется получить

Перед началом проверки правописания вы нажимаете нужную кнопку, после чего все записанные латинскими буквами слова и выражения помечаются как не требующие проверки.

Способ решения

При создании макроса, позволяющего решить поставленную выше задачу, мы воспользуемся имеющейся в Microsoft Word возможностью поиска знаков, содержащихся в некотором *диапазоне* (о том, как определяются диапазоны знаков, рассказывается в *разд. "Дополнительные сведения"* в заключительной части этой главы). Все знаки, которые будут обнаружены в ходе процедуры поиска, реализованной с учетом этой возможности, будут помечаться как не требующие проверки.

Отметим также, что для реализации процедуры поиска диапазона знаков необходимо установить флажок **Подстановочные знаки** в диалоговом окне **Найти и заменить**. Обращаем внимание читателей, что при создании макросов, рассматриваемых далее в этой главе, данный флажок должен быть обязательно установлен.

Решение задачи

> *Основные этапы*: запуск процедуры записи макроса (шаги 1—2); определение содержания документа как текста, записанного русскими буквами (шаги 3—4); поиск выражений, записанных латинскими буквами, и присвоение им свойства **Не проверять правописание** (шаги 5—11); завершение записи макроса (шаг 12).

1. Выведите на экран диалоговое окно **Запись макроса** и запишите имя создаваемого макроса, например, **NotSpell** (т. е. "не проверять правописание").

2. Создайте для этого макроса специальную кнопку, после чего приступайте непосредственно к записи макроса.

> ***Совет***
>
> Назначать сочетание клавиш для запуска создаваемого нами макроса не стоит, поскольку использоваться он будет, по-видимому, не так уж часто.

3. Выделите весь текст в документе, нажав сочетание клавиш <Ctrl>+<A>, а затем откройте меню **Сервис**, укажите в нем пункт **Язык** и выберите команду **Выбрать язык**, после чего на экране появится диалоговое окно **Язык**.
4. Выберите в списке **Пометить выделенный текст как** элемент **Русский (Россия)** и нажмите кнопку **ОК** (рис. 4.4).

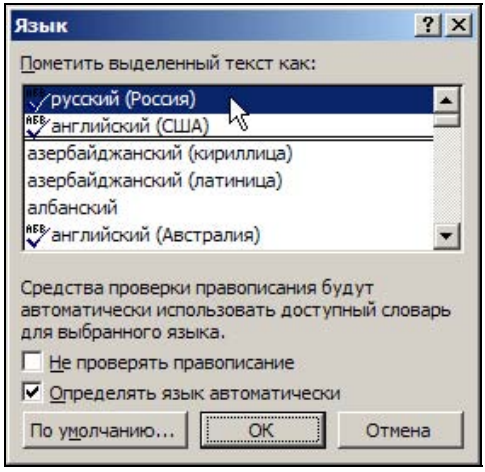


Рис. 4.4. Выбор языка для выделенного текста в диалоговом окне **Язык**

Поясним необходимость выполнения действий, указанных на шагах 3 и 4. В некоторых случаях (например, при вставке текста, скопированного в буфер обмена из других документов или программ) отдельные фрагменты текста, записанного русскими буквами, могут оказаться помеченными как не требующие проверки правописания или даже как английский (или любой другой иностранный) текст. В первом случае проверка правописания в таком документе даст неполные результаты, что чревато пропуском ошибок. Во втором — программа проверки правописания будет "спотыкаться" на каждом русском слове, выдающем себя за "иностранца". Дело в том, что при записи нашего макроса мы собираемся исключить из проверки правописания только те выражения, которые записаны *латинскими*

буквами. Поэтому все слова, записанные *кириллицей*, но имеющие "иностранный паспорт", проверяться все-таки будут. Поскольку, как мы знаем, этот "паспорт" является "фальшивым", программа проверки правописания будет каждый раз сообщать, что такого слова нет в словаре, и предлагать пользователю исправить ошибку.

5. Нажмите <Ctrl>+<H>, после чего откроется диалоговое окно **Найти и заменить**.
6. Установите флажок **Подстановочные знаки** (в случае если диалоговое окно поиска и замены открылось в компактном виде, нажмите предварительно кнопку **Больше**).
7. Введите в поле **Найти** выражение [A-z]. С его помощью будет осуществляться последовательный поиск всех встречающихся в документе латинских букв, заключенных в диапазоне между прописной латинской буквой A и строчной z. Диапазон знаков указывается в квадратных скобках.
8. Перейдите в поле **Заменить на** и нажмите кнопку **Формат**.
9. Выберите в появившемся списке элемент **Язык** (см. рис. 4.3).
10. На экране появится диалоговое окно **Заменить язык**, которое по своему внешнему виду совпадает с диалоговым окном **Язык** (см. рис. 4.4). Установите в этом окне флажок **Не проверять правописание**, а затем нажмите кнопку **ОК**.
11. Нажмите кнопку **Заменить все** диалогового окна **Найти и заменить**.
12. После окончания процедуры закройте диалоговое окно **Найти и заменить** и завершите процедуру создания макроса.

4.2.2. Знаки препинания тоже обязаны подчиняться требованиям закона

Как известно, в соответствии с правилами оформления текста знаки препинания должны записываться вплотную (без пробела) к тому слову, после которого они стоят. Однако при "непрофессиональной" печати это требование часто не соблюдается. Исправить ошибки такого рода можно в ходе стандартной проверки правописания, однако практика показывает, что на это уходит довольно много времени. Можно также попытаться ликвидировать лишний пробел с помощью стандартной процедуры поиска с заменой, но тогда придется провести такую замену по отдельности для каждого знака препинания (точки, запятой, двоеточия и т. п.). Подстановочные знаки типа "выражение" позволяют справиться с этой задачей посредством одного щелчка мыши.

Ситуация

Требуется отредактировать документ, содержащий большое количество знаков препинания, при записи которых допущена следующая типичная ошибка: перед знаком препинания имеется пробел. Попробуем справиться с этой проблемой с помощью макроса.

Что требуется получить

Вы нажимаете кнопку, после чего перед всеми знаками препинания — точкой, запятой, двоеточием, точкой с запятой, восклицательным и вопросительным знаками (кажется, никого не забыли) — удаляется лишний пробел.

Способ решения

Для решения поставленной задачи мы воспользуемся комбинацией возможностей поиска и замены, позволяющей работать как с выражениями, так и с диапазоном знаков. При этом диапазон знаков будет использоваться нами в своей упрощенной форме, при которой в квадратных скобках записывается (через дефис) не диапазон знаков, а их список (перечень знаков, записанных друг за другом без пробелов и без каких-либо разделителей). При такой форме записи будет производиться поиск всех знаков, указанных в квадратных скобках.

Решение задачи

> *Основные этапы*: запуск процедуры записи макроса (шаги 1—2); поиск выражения, включающего все знаки препинания, перед которыми имеется пробел и удаление этого пробела (шаги 5—7) завершение записи макроса (шаг 8).

1. Выведите на экран диалоговое окно **Запись макроса** и запишите имя создаваемого макроса, например, **DelSpacePunctMark** (т. е. "удалить пробел у знака препинания").
2. Создайте для этого макроса специальную кнопку, после чего приступайте непосредственно к записи макроса.
3. Выделите весь текст в документе, нажав сочетание клавиш <Ctrl>+<A>, а затем откройте меню **Сервис**, укажите в нем пункт **Язык** и выберите команду **Выбрать язык**, после чего на экране появится диалоговое окно **Язык**.
4. Выберите в списке **Пометить выделенный текст как** элемент **Русский (Россия)** и нажмите кнопку **ОК**.
5. Введите в поле **Найти** сначала пробел, а затем следующее выражение: ([.,:;\!\?]).

   Обратите внимание на особенности этого выражения. Во-первых, в нем содержится диапазон, точнее список знаков (указаны в квадратных

скобках), который, в свою очередь, заключен в круглые скобки. Об использовании круглых скобок в операциях поиска и замены будет более подробно сказано в *разд. "Дополнительные сведения"* в конце этой главы. Здесь же отметим только, что в общем случае в поле **Найти** может указываться несколько выражений (каждое — в круглых скобках), которые затем могут воспроизводиться в поле **Заменить на** под своими порядковыми номерами.

Во-вторых, знаки, входящие в список, указанный в квадратных скобках, записываются без пробелов и без каких-либо разделителей. Кроме того, перед восклицательным и вопросительным знаками стоит обратный слэш (символ \). Дело в том, что эти знаки при установленном флажке **Подстановочные знаки** приобретают специальный характер, и обратный слэш служит для отмены этого специального характера (подробнее об этом также см. в *разд. "Дополнительные сведения"*).

6. Перейдите в поле **Заменить на** и введите выражение \1. Иными словами, запишите после обратного слэша порядковый номер выражения, указанного в поле **Найти** (в этом случае обратный слэш не связан с отменой специального характера знаков, с его помощью выполняется запись выражений в поле **Найти**).
7. Нажмите кнопку **Заменить все** диалогового окна **Найти и заменить**.
8. После окончания процедуры закройте диалоговое окно **Найти и заменить** и завершите процедуру создания макроса.

> ***Примечание***
>
> Применяя этот макрос, вы должны быть уверены, что действительно следует удалить лишние пробелы перед всеми знаками препинания. В некоторых случаях, например при записи инструкций языков программирования, это не так.

В порядке закрепления материала, рассмотренного в этом разделе, попробуйте теперь расширить возможности созданного только что макроса, включив в него "забытые" нами выделительные знаки препинания — скобки и кавычки. Поскольку они, как правило, являются парными, условиям нашей задачи (наличие лишнего пробела перед знаком препинания) отвечают только их правые элементы.

У вас может возникнуть вопрос, а что делать, если перед знаком препинания будет не один пробел, а несколько. В этом случае можно просто применить созданный нами макрос несколько раз. Но лучше все-таки прочитать материал, представленный в следующем разделе, и внести в макрос необходимые коррективы, которые позволят нам за один "прогон" ликвидировать все лишние пробелы.

4.2.3. Главное в документе все-таки слова, а не то, что между ними

С некоторыми возможностями круглых и квадратных скобок при выполнении поиска мы уже ознакомились. Выясним теперь, для чего могут пригодиться фигурные скобки.

Ситуация

Требуется отредактировать подготовленный в спешке документ, в котором между словами зачастую имеется два и более пробелов. Такой документ выглядит некрасиво, его трудно читать и вообще это противоречит правилам создания печатных документов. Поэтому желательно от лишних пробелов избавиться.

Что требуется получить

Вы нажимаете кнопку, после чего в документе мгновенно удаляются все лишние пробелы.

Способ решения

Решить данную задачу при создании нашего очередного макроса нам помогут подстановочные знаки, записываемые с использованием специального оператора — фигурных скобок, — который позволяет осуществлять поиск повторений знака, стоящего перед открывающей фигурной скобкой (напоминаем, что при этом должен быть установлен флажок **Подстановочные знаки**).

> ***Примечание***
>
> В Microsoft Word, вообще говоря, не проводится четкого разделения между специальными и подстановочными знаками. Поэтому для обозначения группы подстановочных знаков, позволяющих осуществлять поиск некоторого диапазона символов, мы решили использовать условный термин *"специальный оператор"*. В определенной степени он соответствует понятию шаблона.

Данный оператор может записываться в нескольких вариантах, одним из которых мы и воспользуемся при создании нашего макроса, а именно {N;}, где N — любое целое число. Такая форма записи означает, что будет осуществляться поиск *N и более* расположенных подряд вхождений знака, стоящего перед открывающей фигурной скобкой (в нашем случае — N и более расположенных подряд пробелов).

> ***Примечание***
>
> В Интернете можно найти и другую рекомендацию по решению задачи удаления лишних пробелов (особенно часто она дается в связи с вопросами, зада-

ваемыми по поводу оформления студенческих рефератов и курсовых). Суть этой рекомендации сводится к следующему: необходимо ввести в поле **Найти** специальный знак **Пустое пространство**, обозначаемый как ^w (см. рис. 4.2, а также табл. 4.1 далее), а в поле **Заменить на** — один пробел и выполнить процедуру глобальной замены (т. е. нажать кнопку **Заменить все**). Предупреждаем легковерных студентов, что, воспользовавшись этой рекомендацией, вы иногда рискуете не столько улучшить, сколько испортить внешний вид своей публикации. Дело в том, что под *пустым пространством* в Microsoft Word понимаются не только обычные, но и неразрывные пробелы, а также знаки табуляции и некоторые другие специальные символы форматирования. Если они имеются в вашем документе, то замена любой последовательности таких знаков на один обычный пробел может привести к явно нежелательным последствиям.

Обратите внимание на знак, стоящий после числа N в фигурных скобках. Этот знак должен совпадать с так называемым *разделителем элементов списка*, вид которого (точка с запятой или запятая) определяется в зависимости от системных настроек Windows. По умолчанию в русскоязычных операционных системах в качестве разделителя списка используется точка с запятой (а запятая — в качестве десятичного разделителя), в англоязычных — запятая (в качестве десятичного разделителя — точка). Более подробно об этом, а также о принципах работы с фигурными скобками в операциях поиска говорится в *разд. "Дополнительные сведения"* в заключительной части этой главы.

Решение задачи

Основные этапы: запуск процедуры записи макроса (шаги 1—2); поиск промежутков между словами, состоящих из двух и более пробелов, и замена их на один пробел (шаги 5—7); завершение записи макроса (шаг 8).

1. Выведите на экран диалоговое окно **Запись макроса** и запишите имя создаваемого макроса, например, **DelSpaces** (т. е. "удалить пробелы").
2. Создайте для этого макроса специальную кнопку, после чего приступайте непосредственно к записи макроса.
3. Выделите весь текст в документе, нажав сочетание клавиш <Ctrl>+<A>, а затем откройте меню **Сервис**, укажите в нем пункт **Язык** и выберите команду **Выбрать язык**, после чего на экране появится диалоговое окно **Язык**.
4. Выберите в списке **Пометить выделенный текст как** элемент **Русский (Россия)** и нажмите кнопку **ОК**.
5. Введите в поле **Найти** сначала пробел, а затем {2;} (или {2,}, если в настройках вашей операционной системы в качестве разделителя элементов списка используется запятая). Это позволит нам отыскать в документе все вхождения расположенных подряд двух и более пробелов.
6. Перейдите в поле **Заменить на** и введите один пробел.

7. Нажмите кнопку **Заменить все** диалогового окна **Найти и заменить**.
8. После окончания процедуры закройте диалоговое окно **Найти и заменить** и завершите процедуру создания макроса.

> ***Примечание***
>
> Применяя этот макрос, вы должны быть уверены, что действительно следует удалить все вхождения двух и более пробелов в документе. Иногда этого не требуется.

Вернемся теперь к вопросу об удалении нескольких пробелов перед знаком препинания, поставленному в конце предыдущего раздела. Понятно, что с помощью только что созданного макроса эту проблему решить не удастся: после его применения перед "одиноко плавающим" знаком препинания все еще будет оставаться один (лишний) пробел. Поэтому мы предлагаем вам усовершенствовать уже имеющийся у нас макрос **DelSpacePunctMark**, внеся в него коррективы с учетом возможностей специального оператора {N;}. Напомним, что в этом макросе мы записали сначала в поле **Найти** пробел, а затем выражение ([.,:;\!\?]). Подумайте теперь, как с помощью оператора {N;} можно задать поиск любого количества расположенных подряд пробелов (в том числе одного пробела). Правильно, ввести сначала сам пробел, а после него записать {1;}, что означает "один и более". Думаем, теперь вы без труда внесете нужные коррективы в макрос **DelSpacePunctMark**. Для этого, правда, вам придется повторить процедуру его создания. Однако сообщаем вам, что, прочитав главу 6 книги, вы научитесь оперативно вносить подобные элементарные изменения непосредственно в текст макроса. И для этого, вообще говоря, не потребуется знания каких-либо языков программирования.

4.3. Дополнительные сведения

В этом разделе рассказывается об универсальном способе выделения текста в документе Microsoft Word, а также более подробно рассматриваются возможности процедуры поиска и замены.

4.3.1. "Универсальный" режим выделения: клавиша <F8>

Как отмечалось в основной части главы, активизация режима выделения осуществляется посредством нажатия клавиши <F8>. При создании соответствующего макроса этот режим использовался нами для выделения текста от места расположения курсора до некоторого знака, указанного в процедуре поиска (в нашем случае — точки в конце предложения). Однако следует от-

метить, что аналогичная (недокументированная) возможность была изначально заложена разработчиками Microsoft Word в режим выделения. Для ее реализации следует после нажатия клавиши <F8> просто ввести с клавиатуры знак, до которого требуется произвести выделение (вправо от места расположения курсора).

Примечание

К сожалению, этот способ выделения не работает с буквами кириллицы.

Таким образом, при создании нашего макроса можно было вполне обойтись без процедуры поиска. В порядке упражнения вы можете проделать это, воспользовавшись данным свойством режима выделения. Более того, оно может с успехом применяться в качестве альтернативы созданию макросов семейства удаления "лишнего" текста. Правда, следует признать, что макрос все же удобнее: при его использовании для удаления текста требуется выполнить всего лишь одно действие (нажать соответствующее сочетание клавиш), тогда как в режиме выделения — целых четыре (нажать клавишу <F8>, ввести знак, определяющий правую границу выделения, снять выделение с последнего знака в выделенном фрагменте, нажать клавишу <Del>).

4.3.2. Поиск с заменой в Microsoft Word

В основной части главы нами были кратко рассмотрены основные возможности, предоставляемые процедурой поиска в Microsoft Word.

Практика показывает, что, как правило, с имеющимися в Microsoft Word достаточно мощными средствами и алгоритмами поиска подавляющее большинство пользователей либо не знакомы вообще, либо считают их чересчур сложными, чтобы применять в своей повседневной работе. Однако мы надеемся, что примеры, приведенные в этой главе, достаточно убедительно продемонстрировали высокую эффективность указанных возможностей. Поэтому мы настоятельно рекомендуем потратить некоторое время на их более подробное рассмотрение. Уверяем вас, затраченные при этом усилия очень скоро окупятся сторицей.

Приведем наглядный пример из своей практики. При подготовке этой книги не всегда выдерживались принципы единообразного оформления отдельных компонентов текста. Так, названия элементов интерфейса иногда записывались в кавычках (нужно — без кавычек) и без выделения полужирным шрифтом. При этом использовались как обычные прямые кавычки ("), так и "елочки", или шевроны (« »). Чтобы исправить эту погрешность оформления, требовалось просмотреть все закавыченные элементы и в том случае, если это были элементы интерфейса, убрать кавычки и применить полу-

жирное начертание. Соответствующий шаблон поиска в этом случае может выглядеть так: ["«](<[А-Я]*)[»"]. Расшифровку этого выражения вы сможете сделать, когда ознакомитесь с материалом данного раздела.

Элементы управления диалогового окна *Найти и заменить*

Прежде всего, коротко поясним назначение элементов управления (кнопок и флажков), имеющихся в диалоговом окне **Найти и заменить** (рис. 4.5).

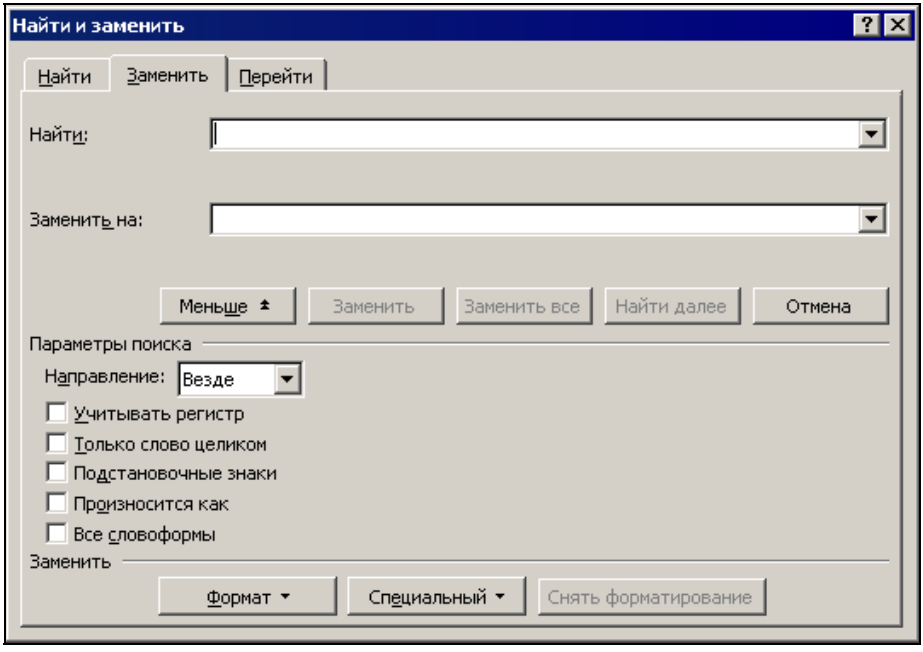


Рис. 4.5. Элементы управления диалогового окна **Найти и заменить** (вкладка **Заменить**)

С помощью кнопки **Специальный** выводится список специальных символов — своего рода шаблонов, которые могут использоваться при выполнении процедур поиска с заменой. С их помощью обозначаются, как правило, структурные элементы документа, такие как символ абзаца, поле, разрыв страницы, графический объект и т. п. Однако имеются и символы, обозначающие конкретные текстовые знаки (короткое и длинное тире), некоторые обобщенные типы текстовых знаков (любая буква, любая цифра), а также любой знак. Имеется и группа специальных символов, обычно используемых при верстке документов, — неразрывный дефис, неразрывный пробел, мягкий перенос. Чтобы автоматически вставить в поле поиска или замены

любой из указанных знаков, следует щелкнуть соответствующий ему элемент в списке, который выводится при нажатии кнопки **Специальный**. Каждый из вставляемых таким образом символов предваряется знаком крышки (^). Необходимо иметь в виду, что состав этих символов для полей **Найти** и **Заменить на** различен. Влияет на него и состояние флажка **Подстановочные знаки** (см. табл. 4.1, в которой представлен состав списка специальных символов, которые могут вводиться в поля **Найти** и **Заменить на** в зависимости от того, установлен или снят флажок **Подстановочные знаки**). Часть этих символов может использоваться (независимо от состояния флажка **Подстановочные знаки**) только в поле **Найти** (а именно, **Знак примечания**, **Графический объект** и **Разрыв раздела**), а некоторые (а именно **Содержимое буфера обмена** и **Искомый текст**) — только в поле **Заменить на**.

Таблица 4.1. *Специальные символы, которые могут вводиться в поля* ***Найти*** *и* ***Заменить на*** *в зависимости от состояния флажка* ***Подстановочные знаки***

Специальные символы	Обозначение	Флажок "Подстановочные знаки"			
		Снят		Установлен	
		Найти	Заменить на	Найти	Заменить на
Знак абзаца	^p	X	X		X
Знак табуляции	^t	X	X	X	X
Знак примечания	^a	X		X	
Любой знак	^?	X			
Любая цифра	^#	X			
Любая буква	^$	X			
Знак крышки	^^	X	X	X	X
Разрыв колонки	^n	X	X	X	X
Короткое тире	^+	X	X	X	X
Длинное тире	^=	X	X	X	X
Знак концевой сноски	^e	X			
Поле	^d	X			
Знак сноски	^f	X			
Графический объект	^g	X		X	
Разрыв строки	^l	X	X		X

Таблица 4.1 *(окончание)*

Специальные символы	Обозначение	Флажок "Подстановочные знаки"			
		Снят		Установлен	
		Найти	Заменить на	Найти	Заменить на
Разрыв страницы	^m	X	X	X	X
Неразрывный дефис	^~	X	X	X	X
Неразрывный пробел	^s	X	X	X	X
Мягкий перенос	^-	X	X	X	X
Разрыв раздела	^b	X		X	
Пустое пространство	^w	X			
Содержимое буфера обмена	^c		X		X
Искомый текст	^&		X		X

Назначение и порядок использования этих специальных символов, по нашему мнению, особых пояснений не требует. Поэтому более подробную их характеристику мы здесь не приводим. Отметим только, что, как уже отмечалось ранее в *разд. 4.2.3*, под *пустым пространством* в Microsoft Word понимаются обычные и неразрывные пробелы, а также знаки табуляции и некоторые другие специальные символы форматирования.

Нажатие кнопки **Формат** выводит раскрывающийся список команд, аналогичных командам, содержащимся в меню Microsoft Word (в основном — в меню **Формат**). С помощью этих команд вызываются диалоговые окна, в которых определяются различные характеристики форматирования текста (шрифт, параметры абзаца, стиля, языкового оформления и т. п. — см. рис. 4.3). Некоторые из них (например, **Язык** и **Выделение цветом**) уже активно использовались нами при создании макросов. Другие пока избежали этой участи.

Довольно часто при поиске текста используются флажки **Учитывать регистр** и **Только слово целиком**. Смысл этих флажков очевиден и дополнительных пояснений не требует.

Флажки **Произносится как** и **Все словоформы** могут использоваться только с англоязычными словами и поэтому здесь не рассматриваются (к тому же их смысл также интуитивно понятен).

Оставшийся флажок **Подстановочные знаки** имеет для процедур поиска и замены фундаментальное значение: от его состояния (установлен или сброшен) зависит состав набора подстановочных знаков и специальных операторов, которые могут использоваться при выполнении поиска и замены.

Специальные символы и операторы в процедуре поиска

Некоторые специальные символы и операторы (конец абзаца, знак в диапазоне, выражение, число вхождений) уже использовались нами при создании макросов. Приведем теперь пояснение для этих и других подстановочных знаков и специальных операторов и на конкретных примерах покажем возможности их практического использования.

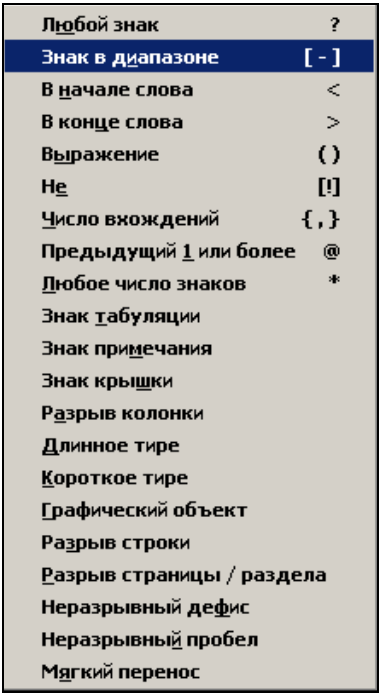


Рис. 4.6. Перечень специальных символов, используемых при поиске в диалоговом окне **Найти и заменить** (выводится по нажатию кнопки **Специальный** при установленном флажке **Подстановочные знаки**)

Примечание

Терминология, используемая при описании процедуры поиска с заменой, не является строгой. Специальные символы часто называются в справочных ма-

териалах по Microsoft Word также подстановочными знаками. По-видимому, это связано с наличием одноименного флажка. Его состояние (установлен, снят), как уже отмечалось, влияет на состав специальных знаков, список которых отображается при нажатии кнопки **Специальный**. Однако часть этих символов входит в этот список в обоих случаях. Поэтому трудно их отнести к какой-то одной категории, и в дальнейшем изложении мы будем использовать оба термина в качестве синонимов.

Рассмотрим сначала специальные операторы и подстановочные знаки, которые могут записываться в поле поиска (рис. 4.6). В конце этого раздела приводится сводная таблица с краткой характеристикой их назначения и поясняющими примерами. Напомним, что доступ к этим подстановочным знакам и специальным операторам возможен только при установленном флажке **Подстановочные знаки**.

- **Любой знак** — при выборе этого подстановочного знака в списке, открывающемся при нажатии кнопки **Специальный**, в поле поиска заносится вопросительный знак. Следует иметь в виду, что с помощью этого подстановочного знака осуществляется поиск действительно *любого* знака (знака конца абзаца, примечания и т. п.), а не только текстовых знаков (см. примеры в табл. 4.2).

Таблица 4.2. *Сводка подстановочных знаков и специальных операторов, используемых в процедурах поиска при установленном флажке* ***Подстановочные знаки****

Подстановочный знак (оператор)	Что обозначает	Пример строки поиска	Примеры результатов поиска
?	Любой один знак	б**?**к	б**а**к, б**о**к, б**у**к, б**5**к, б¶к и т. п.
[-]	Любой один знак из диапазона знаков	[а-яё]	Любая строчная буква кириллицы
		[A-Z]	Любая прописная буква латиницы
		[0-9]	Любая цифра
[ ]	Любой один из указанных знаков	б[**аоу**]к	б**а**к, б**о**к, б**у**к
<	Знаки, расположенные в начале слова	<бок	**бок**, **бок**сер и т. п., но не коло**бок**, с**бок**у и т. п.
>	Знаки, расположенные в конце слова	бок>	колобок, в**бок** и т. п., но не **бок**сер, с**бок**у и т. п.
[!]	Любой один знак, не указанный после восклицательного знака	б[!ы]к	бак, бок и т. п., но не б**ы**к

Таблица 4.2 *(окончание)*

Подстановочный знак (оператор)	Что обозначает	Пример строки поиска	Примеры результатов поиска
[!x-z]	Один любой знак, не входящий в диапазон, указанный после восклицательного знака	[!а-яё]ок	**Б**ок, **Д**ок и т. п., но не **б**ок, **д**ок и т. п.
		[!0-9]	Любой знак кроме цифр
{N}	Строго N расположенных подряд вхождений предыдущего знака, знака из диапазона или выражения, выражения, расположенного перед открывающей фигурной скобкой	10{2}	100, но не 10, 1000, 10000 и т. д.
		10[20]{2}	10**00**, 10**02**, 10**20**, 10**22**, но не 1010, 1001, 1030, 1003 и т. д.
		10(20){2}	10**2020**, но не 1020, 10202020 и т. д.
{N;}	N и более расположенных подряд вхождений знака, знака из диапазона, выражения, расположенного перед открывающей фигурной скобкой	10{2;}	1**00**, 1**000**, 1**0000** и т. д., но не 1**0**
		10[20]{2;}	10**00**, 10**000** и т. д., 10**02**, 10**22**, 10**022**, 10**0220** и т. д., но не 100, 122, 1202, 1222 и т. д.
		10(20){2;}	10**2020**, 10**202020**, и т. д., но не 10**20**
{N;M}	От N до M расположенных подряд вхождений знака, знака из диапазона, выражения, расположенного перед открывающей фигурной скобкой	10{2;4}	1**00**, 1**000**, 1**0000**, но не 10, 1**000**0 и т. д.
@	Одно или более расположенных подряд вхождений предыдущего выражения	10(20)@	10**20**, 10**2020**, 10**202020** и т. д.
*	Любое число любых знаков	б*к	б**ы**к, б**алы**к, б**елый. ¶Пол**ковник и т. п.

*Часть примеров взяты из справки к отличной программе верстки документов Microsoft Word, известной под именем "Перестройка", автором которой является С. М. Хозяинов (**http://junecalends.50free.net**). При этом как в самих примерах, так и в объяснении к ним исправлены некоторые неточности.

- **Знак в диапазоне** — с помощью этого оператора указывается, что будет производиться поиск всех знаков, принадлежащих данному диапазону. Границы этого диапазона определяются в соответствии с так называемыми ANSI-кодами указанных знаков. Слева от дефиса должен записываться знак с меньшим кодом, справа — с большим. Если это требование не соблюдается, будет выведено сообение об ошибке "Поле “Найти” содержит неверный диапазон". Например, диапазон, включающий все буквы русского алфавита (за исключением "Ё" и "ё"), следует записать как [А-я], поскольку наименьший ANSI-код (192) соответствует прописной букве "А", а наибольший (255) — строчной букве "я". Если вы при поиске с использованием диапазона получите такое сообщение, попробуйте поменять местами знаки, указанные в квадратных скобках. Краткие сведения о стандартах кодировки символов приводятся в следующем подразделе.

  Если при записи знаков в квадратных скобках дефис не используется, то диапазон знаков "вырождается" в их список (перечень знаков, записанных друг за другом без пробелов и без каких-либо разделителей). При такой форме записи будет производиться поиск всех знаков, указанных в квадратных скобках. Этим вариантом специального оператора **Знак в диапазоне** мы воспользовались при создании макроса **DelSpacePunctMark**, в котором использовали диапазон [.,:;\!\?] для поиска любого из знаков препинания.

  Вместе с тем, в одних и тех же квадратных скобках может быть указан не один, а несколько диапазонов. Так, при создании макроса **NotSpell**, с помощью которого из проверки правописания исключались знаки латиницы, нами был указан диапазон [A-z], включающий все буквы латинского алфавита. Однако если внимательно посмотреть таблицу ANSI-кодов, то можно увидеть, что внутри этого диапазона имеются также знаки, не относящиеся, строго говоря, к латинице — квадратные скобки, косая обратная черта и т. п. (всего шесть знаков, имеющих коды с 091 по 096). Для решения стоявшей перед нами задачи это было несущественно, поскольку в указанный диапазон не попадали буквы кириллицы. Однако более правильно было бы использовать в создаваемом макросе "сдвоенный" диапазон знаков, имеющий вид [A-Za-z].

  Отметим также, что в любой процедуре поиска могут использоваться не только конкретные знаки, но и их ANSI-коды — в этом случае перед кодом должен записываться символ крышки ^ и цифра 0. ANSI-код латинской буквы "A" равен 065, буквы "Z" — 090, буквы "a" — 097, а буквы "z" — 122. Таким образом, альтернативная форма записи нашего диапазона, указанного в макросе **NotSpell**, будет выглядеть так: [^0065-^0090^0097-^0122]. Более подробно об ANSI-кодах и других системах кодирования символов рассказывается в следующем подразделе.

- **В начале слова** — этот специальный оператор имеет вид левой угловой скобки. Он позволяет отыскать все слова, начинающиеся со знаков, записанных после этого символа (см. примеры в табл. 4.2).
- **В конце слова** — этот специальный оператор имеет вид правой угловой скобки. Он позволяет отыскать все слова, оканчивающиеся на знаки, записанные после этого символа (см. примеры в табл. 4.2).
- **Выражение** — представляет собой произвольную последовательность знаков, записанную в круглых скобках. Как и диапазон, выражение задается только при установленном флажке **Подстановочные знаки**. В поле **Найти** может указываться несколько выражений. Выражения в поле **Заменить на** воспроизводятся под своими порядковыми номерами, записанными после обратного слэша. Например, если вы ищете в документе фразу, состоящую из трех выражений (Мила) (мыла) (раму), то в поле **Заменить на** можно записать следующее: "\3 \2 не \1, а мама". В результате произведенной таким образом замены получим: "раму мыла не Мила, а мама".
- **Не** — этот специальный оператор имеет вид восклицательного знака, записанного в квадратных скобках. С его помощью осуществляется поиск любого (одного) знака, не указанного после восклицательного знака (в том числе — диапазона знаков). Чтобы более наглядно представить возможности этого оператора, рассмотрим следующий пример: требуется найти любой знак, за исключением букв русского алфавита, цифр и пробела (знак с ANSI-кодом 0032). Если вы посмотрите в таблицу ANSI-кодов (английский вариант таблицы символов можно найти в справке Visual Basic по ключевому слову "Character Set", а русский вариант — в многочисленных публикациях по компьютерной литературе, в специализированных справочниках, а также в Интернете), то увидите, что все буквы русского алфавита располагаются в ней подряд — с позиции 0192 по 0255 (т. е. до конца таблицы). И только букве "ё" не нашлось места в общем ряду — прописная Ё имеет ANSI-код 0168, а строчная ё — 0184. Таким образом, для "запретных" знаков в нашем примере мы должны будем указать в поле поиска два диапазона, а также перечень из трех знаков, которые располагаются в таблице ANSI-кодов на отдельных позициях: [!0-9A-яЁё^0032]. Кстати, вместо ANSI-кода пробела можно просто ввести непосредственно этот знак, нажав клавишу пробела.
- **Число вхождений** — этот специальный оператор записывается с использованием фигурных скобок. Как уже отмечалось в основном тексте главы, данный оператор позволяет осуществлять поиск расположенных подряд повторений знака, стоящего перед открывающей фигурной скобкой. Необходимо, однако, уточнить, что это может быть не только какой-либо конкретный знак (например, пробел), но и диапазон знаков (например, все цифры) или выражение (например, некоторое конкретное число).

Данный специальный оператор может записываться в следующих трех вариантах: {N}, {N;} и {N;M}, где N и M — любые целые числа. Первый из этих вариантов означает, что будет осуществляться поиск *ровно N* расположенных подряд вхождений знака (диапазона знаков, выражения), записанного перед открывающей фигурной скобкой; второй — что будет осуществляться поиск *N и более* таких вхождений (в примере, рассмотренном в основном тексте главы, это были 2 и более расположенных подряд пробелов); третий — поиск *N* и более, *но не более M*, таких вхождений.

Например, 50{3} означает, что будет осуществляться поиск всех последовательностей знаков, имеющих вид 5000, т. е. три нуля после цифры 5. В свою очередь, (50){3} позволит отыскать все последовательности знаков, имеющие вид 505050, т. е. троекратное вхождение выражения (50). Согласитесь, что подобный вариант записи специального оператора не очень впечатляет — можно было бы просто указать, что ищется именно 5000 или 505050.

Более интересными представляются варианты с использованием диапазона знаков, а также так называемого *разделителя элементов списка*, который записывается после числа N в фигурных скобках. Вид разделителя — запятая или точка с запятой — определяется в зависимости от настроек операционной системы (как отмечалось в основном тексте главы, по умолчанию в русскоязычных операционных системах в качестве разделителя списка используется точка с запятой). Поэтому если при использовании подстановочного знака **Число вхождений** вы получите сообщение "Поле “Найти” содержит неверный шаблон поиска", проверьте, какой тип разделителя используется на вашем компьютере (в Windows 2000: **Пуск | Настройка | Панель управления | Язык и стандарты | Числа**; в Windows XP: **Пуск | Панель управления | Язык и региональные стандарты | Региональные параметры | Настройка**) — см. рис. 4.7.

При использовании диапазона знаков наш пример будет выглядеть так: [50]{3} — т. е. диапазон знаков здесь фактически представлен списком, включающим цифры 0 и 5. В этом случае будет осуществляться поиск всех трехзначных последовательностей цифр, состоящих из нулей и пятерок (в любых комбинациях).

Использование варианта {N;} с разделителем элементов списка (т. е. 50{3;}) позволит отыскать все последовательности знаков вида 5000, 50000, 500000 и т. д. Аналогичным образом (50){3;} позволит отыскать все последовательности знаков, имеющие вид 505050, 50505050, 5050505050 и т. д. Чтобы ограничить число повторений искомого знака или выражения, необходимо записать после запятой в фигурных скобках соответствующее целое число.

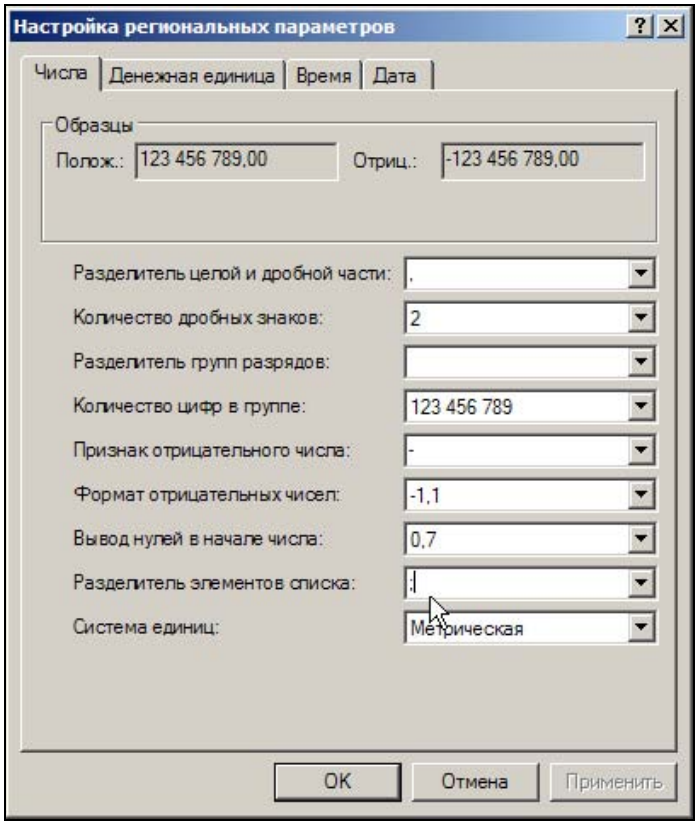


Рис. 4.7. Вкладка **Числа** диалогового окна **Настройка региональных параметров** панели управления Windows XP, на которой задается разделитель элементов списка

- **Предыдущий 1 или более** — этот специальный оператор имеет вид значка @. С обычными знаками и диапазонами знаков этот оператор работает, по нашим наблюдениям, не совсем корректно, поэтому мы рекомендуем использовать его только с выражением (последовательностью знаков, записанных в круглых скобках). В этом случае данный оператор по своему действию аналогичен специальному оператору **Число вхождений** в том его варианте, когда он используется в виде {N;} при N=1.
- **Любое число знаков** — этот подстановочный знак имеет вид звездочки. Чаще всего он используется для нахождения фрагментов текста, заключенных между какими-либо конкретными знаками (см. пример в табл. 4.2). Если использовать этот оператор без каких-либо дополнительных знаков, то с его помощью будет осуществляться поиск *любого одного* знака (в том числе скрытых символов).

Что касается специальных операторов и подстановочных знаков, которые могут использоваться в поле **Заменить на**, то их назначение, думаем, достаточно точно определяется именем соответствующего элемента списка, выводящегося (при установленном флажке **Подстановочные знаки**) при нажатии кнопки **Специальный** (рис. 4.8).

Особых пояснений мог бы требовать только специальный оператор **Искомое выражение**, однако мы думаем, вы уже хорошо уяснили его смысл, когда несколько ранее мы выяснили, что раму мыла все-таки не Мила, а ее добрая мама.

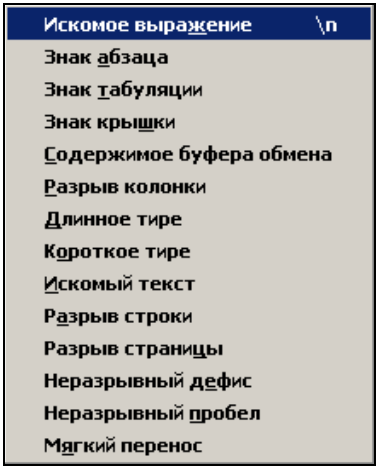


Рис. 4.8. Перечень специальных символов, используемых в поле **Заменить на** диалогового окна **Найти и заменить** (выводится по нажатию кнопки **Специальный** при установленном флажке **Подстановочные знаки**)

Отметим также, что в Microsoft Office Word 2003 к числу этих знаков были добавлены два новых — знак крышки (^) и знак конца раздела (§).

Полезные сведения

В Microsoft Word 2002 и Microsoft Office Word 2003 на вкладке **Найти** диалогового окна **Найти и заменить** имеется флажок **Выделить все элементы, найденные в**. Если его установить, кнопка **Найти далее** заменяется кнопкой **Найти все**. При ее нажатии одновременно выделяются все вхождения выражения, указанного в поле **Найти**. При этом поиск может осуществляться в указанной пользователем области документа — в выделенном фрагменте, основной части документа, колонтитулах, примечаниях (при их наличии) и т. п. (рис. 4.9). Все выделенные фрагменты текста можно одновременно отформатировать одинаковым образом, например, выделить полужирным шрифтом или подкрасить зеленым цветом. Чтобы снять выделение, нужно щелкнуть в окне документа.

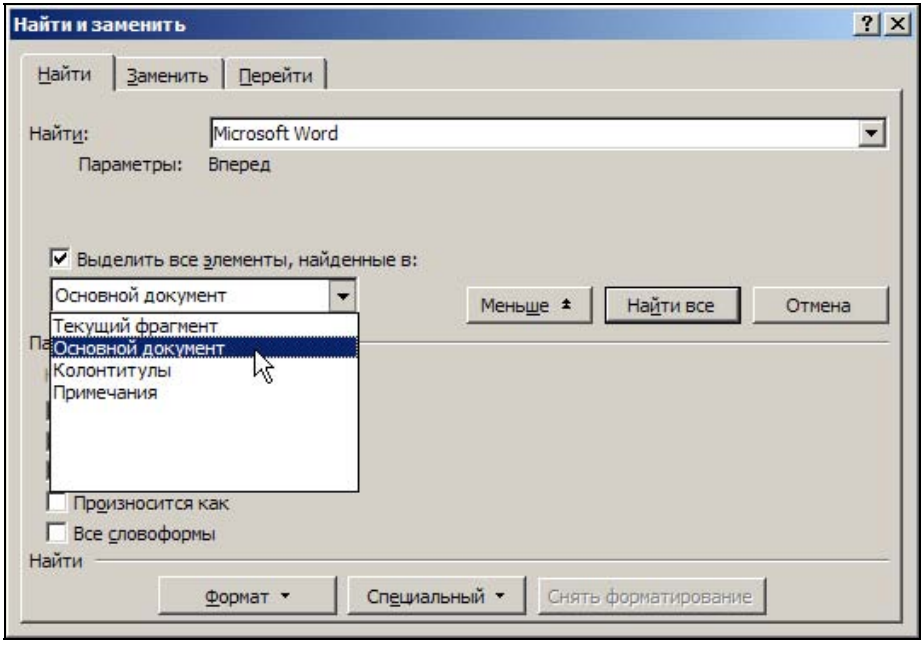


Рис. 4.9. Вкладка **Найти** диалогового окна **Найти и заменить** с установленным флажком **Выделить все элементы, найденные в** (Microsoft Word 2002 и Microsoft Office Word 2003)

Системы кодирования символов

Вопросы, связанные с использованием систем кодирования символов, являются отдельной темой, подробное освещение которой выходит далеко за рамки нашей книжки. Однако мы решили все же ее затронуть, поскольку она имеет самое непосредственное отношение как к использованию программного обеспечения вообще, так и к теме, рассматриваемой в данной главе, в частности. Мы постараемся не отвлекать надолго внимание читателя и приведем лишь самые общие сведения по этому вопросу, не усложняя изложение излишними, часто чисто техническими, деталями. Попутно мы расскажем также и о некоторых практических аспектах использования кодировки символов при работе на компьютере.

Переход от использования одних стандартов и систем кодирования символов к другим происходил, прежде всего, под влиянием усиления интернационализации современной жизни. Применительно к компьютерной сфере это проявляется, в частности, в требовании обеспечить возможность использования различных национальных алфавитов при работе с одними и теми же программами. Пользователи Microsoft Word, проживающие в России,

Чехии, Индии, Китае и других странах мира, должны иметь возможность работать с документами на своем родном языке.

Поскольку компьютеры разговаривают на языке чисел, вводимая в них текстовая информация должна преобразовываться в числовое представление. Системы кодирования как раз и решают эту задачу.

Вероятно, вам уже приходилось слышать о таких системах кодирования символов, как ASCII, ANSI и Юникод (Unicode). Первой из них в 60-х годах была разработана *система ASCII*, предложенная Национальным институтом стандартизации США (American National Standards Institute, ANSI). Она позволяет использовать 128 символов, которым были присвоены значения 0—127. При этом реальное количество символов, которые можно было выводить на печать или отображать на экране, оказалось значительно меньше, поскольку первые 32 ASCII-символа были отведены под "непечатаемые" управляющие коды. Текст в кодировке ASCII не содержит информации о форматировании, например, о полужирном или курсивном начертании или об используемых шрифтах. Кодировка ASCII и сейчас используется при работе в программе Microsoft Notepad или сохранении файла в виде обычного текста в программе Microsoft Word.

По мере распространения компьютерных технологий ограниченность возможностей системы ASCII становилась все более очевидной. Во многих странах разрабатывались собственные нестандартные модификации исходной раскладки ASCII-таблицы, что препятствовало совместимости программного обеспечения и компьютерного оборудования. Так, в нашей стране в то время получил широкое распространение вариант этой таблицы, в котором строчные английские буквы были заменены прописными русскими.

В этих условиях решением проблемы использования национальных алфавитов стал переход к *системе ANSI*, опубликованной Национальным институтом стандартизации США в 1979 г. ANSI является сегодня одним из наиболее распространенных символьных стандартов для ПК, позволяющим использовать 256 символов. Основу его составляет традиционная (основная) таблица ASCII (0—127), дополненная еще 128 символами, которые иногда называют "расширенной" таблицей ASCII (именно поэтому ANSI часто называют расширенным вариантом ASCII). Система ANSI применяется в основном для европейских языков, которые основаны на классическом латинском алфавите, но в отличие от английского языка имеют некоторые дополнительные символы, а еще чаще — варианты расширений типа "A с точкой", "I с двумя точками" и т. п.

Русский язык также смог "уместиться" в дополнительный набор символов, но с некоторыми проблемами. Дело в том, что многие коды из состава таб-

лицы 128—255 уже использовались (без соответствующей стандартизации) в целом ряде программ для некоторых специальных целей (американским разработчикам тогда не приходило в голову, что их программы будут использоваться в СССР). Пользователи со стажем, наверное, помнят о том, как популярная в то время программа работы с файлами Norton Commander 3.0 для DOS "не желала" признавать русскую букву "р". Поэтому внедрение этой системы потребовало во многих случаях адаптации программного обеспечения к использованию подобных "нестандартных" символов.

В настоящее время все большее распространение получает новая система кодирования символов, получившая название *Unicode* (в русской транскрипции "Юникод"), сопровождение которой обеспечивается организацией Unicode Consortium. Внедрение этой системы в качестве универсального единого стандарта активно поддерживается Международной организацией по стандартизации (International Organization for Standardization, ISO). Эта система позволяет отобразить практически неограниченное количество символов национальных алфавитов. Поддерживается она в Microsoft Word, начиная с версии этой программы, входящей в пакет Microsoft Office 2000.

В заключение этого раздела, как и обещали, приведем некоторые сведения практического характера, имеющие отношение к материалу данной главы. В предыдущем разделе уже отмечалось, что ANSI-коды знаков могут использоваться в Microsoft Word в операциях поиска. В дополнение к этому сообщаем, что делать это можно как при установленном, так и при снятом флажке **Подстановочные знаки**. Следует, однако, иметь в виду, что в последнем случае поиск будет осуществляться без учета регистра знака. Например, если вы введете в поле поиска выражение ^0254 (соответствующее строчной букве "ю"), будут найдены не только вхождения этой буквы, но и ее прописного варианта. При установленном флажке **Подстановочные знаки** поиск будет осуществляться в строгом соответствии с ANSI-кодом знака.

Зная ANSI-код того или иного знака, вы можете ввести его с клавиатуры. Для этого используются цифровые клавиши, расположенные в правой части клавиатуры. Эти клавиши должны работать в цифровом режиме (если он не включен, следует предварительно нажать клавишу <NumLock>). Ввод ANSI-кода знака осуществляется следующим образом: нажимается клавиша <Alt> и, не отпуская ее, набирается код в формате `0nnn` (где `0` — ноль, `nnn` — код знака). Например, для ввода знака §, ANSI-код которого 167, следует нажать клавишу <Alt> и, не отпуская ее, набрать на числовой клавиатуре (в цифровом режиме) значение 0167. Этот прием можно использовать, когда приходится достаточно часто вставлять в документ какие-либо нестандартные знаки, например, §, ±, µ , ®, ¶ и т. п. Они могут, конечно, вводиться и с помощью диалогового окна **Символы** (меню **Вставка | Символы)** или назначенных им сочетаний клавиш. Однако, возможно, кому-то придется по душе и этот способ.

Совет

Мы рекомендуем вам освоить и взять на вооружение описанный выше способ ввода нестандартных символов даже в том случае, если вы предпочитаете делать это, не прибегая к помощи клавиатуры. Дело в том, что в некоторых программах, не обладающих такой развитой поддержкой национальных и специальных знаков, как Microsoft Word, их запись может осуществляться только посредством данного метода.

Теперь несколько слов об использовании символов Юникода. Как и ANSI-коды, они могут использоваться при поиске соответствующих им знаков. При этом в поле поиска следует ввести выражение вида `^unnnnn`, где `nnnnn` — код соответствующего символа в кодировке Юникод. Например, чтобы отыскать символ $^7/_8$, которому соответствует код 8542 в кодировке Юникод, необходимо в поле **Найти** ввести выражение ^u08542.

Примечание

Обратите внимание, что буква "u", стоящая перед выражением "nnnnn" является строчной. В справке по Microsoft Office Word 2003 в данном шаблоне поиска указана прописная буква U, что является ошибкой (вы можете в этом убедиться самостоятельно, опробовав оба варианта). К тому же, в примере, приведенном в справке, после знака крышки "^" имеется пробел, которого в действительности быть не должно.

Ввод знаков в кодировке Юникод с помощью клавиатуры осуществляется несколько иначе, чем в случае использования ANSI-кодов. Сначала в нужном месте документа следует ввести шестнадцатеричный код соответствующего символа, а затем нажать сочетание клавиш <Alt>+<x>. Узнать шестнадцатеричный код знака можно в диалоговом окне **Символ** (при выборе символа его шестнадцатеричный код отображается в строке состояния). Можно также поместить курсор справа от знака и нажать сочетание клавиш <Alt>+<x>, после чего вместо этого знака в документе отобразится его шестнадцатеричный код. Например, для знака $^7/_8$ он равен 215E.

Примечание

Данная возможность поддерживается только в Microsoft Word 2002 и Microsoft Office Word 2003.

Поиск знаков, используемых для обозначения специальных операторов

В основном тексте главы при описании процедуры создания макроса, удаляющего пробелы перед знаками пунктуации, уже говорилось, что для отмены особого характера подстановочных знаков используется обратный слэш (\). Напомним, что для поиска знаков пунктуации нами использовалось выражение вида ([.,:;\!\?]). Это же правило относится и к специальным

операторам. Таким образом, чтобы при установленном флажке **Подстановочные знаки** можно было найти любой из следующих знаков:

?! ([{ }]) < > @ * \

в поле поиска следует ввести перед ним обратный слэш. Например, для нахождения самого знака обратного слэша используется шаблон поиска вида "\\" (без кавычек).

При этом чтобы отыскать другие знаки, используемые при записи специальных операторов, — дефис, запятую, а также любые другие знаки (в том числе слэш, знаки $, & и т. п.), — использовать обратный слэш не требуется. Правда, если его все же записать, поиск по-прежнему будет осуществляться правильно. Поэтому в сомнительных ситуациях лучше действовать по принципу "кашу маслом не испортишь".

В заключение вернемся к расшифровке примера, приведенного в начале данного раздела. Напомним, что нам требовалось унифицировать стиль записи элементов интерфейса, которые при подготовке рукописи этой книги иногда записывались в кавычках различного вида (нужно — вообще без кавычек) и без выделения полужирным шрифтом. Чтобы исправить эту погрешность оформления, необходимо было просмотреть все закавыченные элементы и в том случае, если это были элементы интерфейса, убрать кавычки и применить полужирное начертание. Учитывая, что кавычки используются нами довольно часто (может быть, даже чересчур часто), работа эта могла затянуться надолго. Поэтому для ускорения процедуры поиска был использован шаблон вида: ["«](<[А-Я]*)[»"]. Думаем, теперь вы сможете достаточно легко расшифровать его содержание. На всякий случай даем подсказку: при создании шаблона учитывалось, что каждый элемент интерфейса начинается с прописной буквы и записывается знаками кириллицы. В поле **Заменить на** для реализации стоявшей перед нами задачи было записано выражение \1, а также было указано, что шрифт должен иметь полужирное начертание. Таким образом с помощью данного шаблона поиска авторам книги успешно и в короткие сроки удалось решить стоявшую перед ними задачу — удалить кавычки и воспроизвести в полужирном начертании выражение, записанное в круглых скобках в поле **Найти**. Процедура поиска, конечно же, осуществлялась при установленном флажке **Подстановочные знаки.**

4.4. Что нового в этой главе

В этой главе вы познакомились с мощными средствами и возможностями процедуры поиска, реализованной в Microsoft Word. Были рассмотрены как традиционные приемы использования этой процедуры (связанные с поиском "обычного" текста), так и более сложные, позволяющие осуществлять

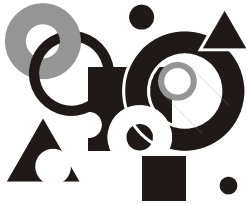
поиск и замену "нематериальных" признаков — выделения фрагментов текста цветом, диапазона знаков, букв, принадлежащих к тому или иному алфавиту (например, любых латинских букв), и т. п.

Полученные теоретические знания сразу же были реализованы нами на практике при создании макросов, позволяющих:

- удалять при редактировании документа текст от текущей позиции курсора до любого заданного знака (в нашем случае — до точки в конце предложения);
- отыскивать в документе все фрагменты текста, выделенные (подкрашенные) каким-либо цветом;
- раскрашивать заданным цветом все вхождения в документ конкретного фрагмента текста (в нашем случае выражения "см. табл.");
- исключать из проверки орфографии в документе, содержащем большое количество "иноязычных" аббревиатур, фамилий, формул и т. п., все слова и знаки, записанные латинским шрифтом;
- удалять ошибочно проставленные пробелы перед знаками препинания;
- удалять лишние пробелы между словами.

Те из вас, кто ознакомился с материалом раздела дополнительных сведений, узнали о возможностях универсального режима выделения текста, включение которого осуществляется с помощью клавиши <F8>.

Здесь же приводится описание элементов диалогового окна **Найти и заменить**, а также дается подробная характеристика (в том числе на практических примерах) возможностей и особенностей применения в процедуре поиска специальных символов и операторов — знак в диапазоне, конец абзаца, выражение, число вхождений и т. п. Рассказывается также об использовании флажка **Подстановочные знаки** и его влиянии на состав средств и модификацию возможностей поиска и замены в Microsoft Word.



Глава 5

Табличных дел мастера

Выделение элементов таблицы (строк, столбцов, ячеек) с помощью цвета. Создание макроса, позволяющего отобразить на экране шапку или боковик таблицы, расположенной на нескольких страницах документа. Дополнительные сведения.

Таблицы, как известно, не являются "профильной" областью использования Microsoft Word. При работе с ними гораздо более эффективными оказываются такие специализированные программы, как, например, Microsoft Excel. Однако таблицы довольно часто включаются в "вордовские" документы, и поэтому мы не можем обойти стороной это важное направление работы с текстом. Решаемые здесь задачи обычно отнимают много времени, требуют аккуратности, знания специализированных возможностей и команд Microsoft Word. Автоматизация операций, связанных с созданием и обработкой таблиц, может существенно облегчить и ускорить подготовку документов, повысить эффективность работы с таблицами.

В этой главе мы займемся созданием макросов, позволяющих подкрашивать отдельные элементы таблиц и облегчающих работу с таблицами большого размера. В *разд. "Дополнительные сведения"* приведены общие сведения об имеющихся в Microsoft Word командах форматирования таблиц. Кроме того, в этот раздел перенесена из основного текста главы процедура создания одного из макросов, проделать которую мы предлагаем нашим читателям в качестве упражнения. Здесь же приводятся общие сведения о некоторых средствах и возможностях Microsoft Word, используемых при создании данного макроса.

5.1. Займемся малярными работами

Таблицы, элементы которых (заголовки, строки, столбцы или ячейки) выделены с помощью того или иного цвета, выглядят, несомненно, более привлекательными и аккуратными. Однако главное преимущество этого приема форматирования состоит в том, что он позволяет повысить информатив-

ность табличных данных, поскольку с его помощью автор документа может привлечь внимание читателя к тем или иным табличным материалам.

5.1.1. Аккуратная цветная полоска: простенько и со вкусом

Наиболее часто с помощью цвета выделяется *шапка (головка) таблицы* — строка или несколько строк, в которых содержатся заголовки столбцов таблицы. Однако если вы хотите облегчить читателю (и себе) знакомство с табличными данными, то можете также выделить цветом и те строки таблицы, к которым хотите привлечь внимание.

Ситуация

Вы подготовили таблицу, содержащую довольно много разделов, в каждом из которых имеется строка итоговых показателей. Понятно, что читателя будут интересовать в первую очередь именно эти строки таблицы, поэтому вы хотите их выделить с помощью цвета, например, голубого. Кроме того, чтобы придать таблице более привлекательный вид, вы хотели бы выделить желтым цветом строку (строки) в ее шапке.

Что требуется получить

Вы помещаете курсор в нужную строку таблицы, нажимаете соответствующую кнопку на панели инструментов, и строка окрашивается заранее определенным цветом.

Способ решения

В процессе создания макроса, позволяющего решить поставленную выше задачу, сначала выделяется нужная строка таблицы, а затем определяется цвет заливки этой строки. Заниматься покраской табличных строк вы будете, по-видимому, не так уж часто, поэтому имеет смысл создать для этого специальную кнопку, а не назначать макросу сочетание клавиш.

Решение задачи

> *Основные этапы*: размещение курсора в нужной строке таблицы (предварительный шаг); запуск процедуры записи макроса и создание для него кнопки (шаги 1—2); выделение строки таблицы (шаг 3); задание цвета заливки (шаги 4—7); завершение записи макроса (шаг 8).

Поместите курсор в нужную строку таблицы.

1. Откройте диалоговое окно **Запись макроса** и введите имя создаваемого макроса, например, **FillYellow** (т. е. "залить желтым цветом").

2. Создайте для макроса кнопку и оформите ее внешний вид. Например, для наглядности можно на кнопке нарисовать желтую горизонтальную полоску.
3. Откройте меню **Таблица** и последовательно выберите команды **Выделить** и **Строка**.
4. Откройте меню **Формат** и выберите команду **Границы и заливка**, после чего на экране появится одноименное диалоговое окно.
5. Перейдите в этом диалоговом окне на вкладку **Заливка** (рис. 5.1).

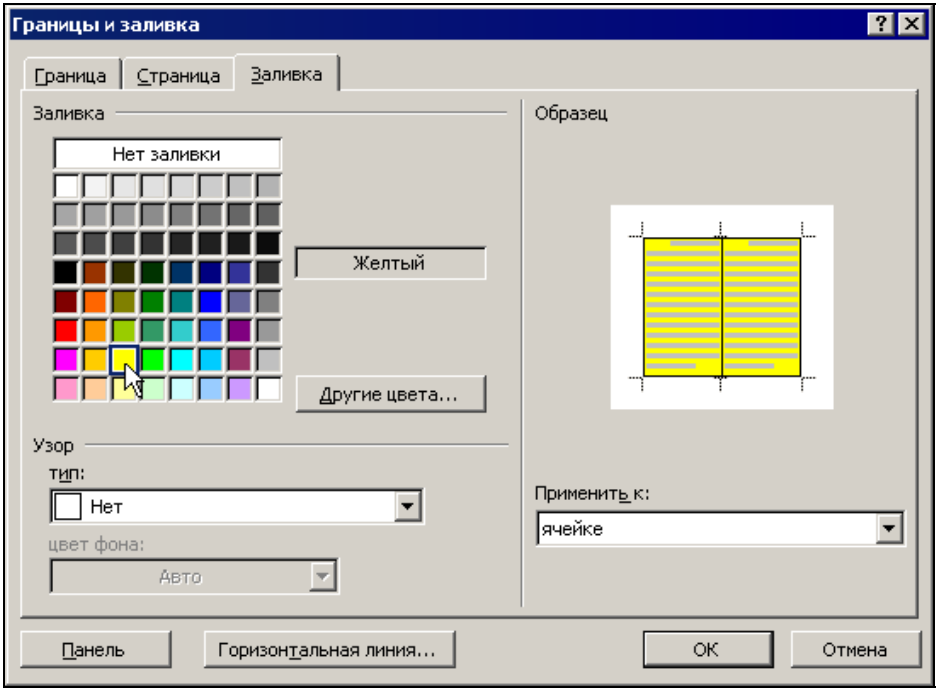


Рис. 5.1. Вкладка **Заливка** диалогового окна **Границы и заливка**

6. Щелкните в палитре цветов третью слева ячейку во втором снизу ряду (она имеет желтый цвет). После этого в поле, расположенном справа от палитры, отобразится слово "Желтый", а представленный справа образец окрасится в желтый цвет.
7. Нажмите кнопку **OK**, после чего диалоговое окно **Границы и заливка** закроется.
8. Завершите процедуру создания макроса.

Что мы имеем в итоге

С помощью только что созданного макроса **FillYellow** может быть решена только одна из сформулированных выше задач — окрашивание строки заголовка таблицы в желтый цвет. Чтобы решить вторую задачу (окрашивание строки итоговых показателей разделов таблицы), необходимо создать второй аналогичный макрос, задав голубой цвет заливки на шестом шаге соответствующей процедуры. Думаем, вы не забудете дать соответствующее имя новому макросу (для справки, голубой цвет по-английски "Blue") и изменить нужным образом цвет полоски на кнопке, с помощью которой он запускается.

> ***Примечание***
>
> Думаем, понятно, что эти, и другие аналогичные макросы могут, конечно же, использоваться и для выделения цветом любой строки таблицы, а не только ее шапки и итоговых показателей. При этом если шапка таблицы состоит из нескольких строк, макрос нужно будет применить соответствующее число раз.

5.1.2. Состав комплексной бригады маляров

Как и в ряде других случаев, рассматривавшихся ранее в этой книге, описанный только что подход может быть использован при создании целого семейства макросов. Во-первых, при желании вы можете создать еще парочку макросов, окрашивающих строки таблицы в другие, возможно, более приятные вам цвета. Во-вторых, вы можете точно таким же образом заняться раскрашиванием столбцов таблицы, а не ее строк — для этого следует на третьем шаге процедуры создания макроса выбрать команду **Столбец** (а не **Строка**, как в предыдущем случае). В-третьих (вы уже догадались), — можно будет создать макросы, раскрашивающие отдельные ячейки таблицы, а также таблицу в целом (соответственно на третьем шаге должна быть выбрана команда **Ячейка** или **Таблица**). Ну и, конечно, не забывайте давать своим макросам запоминающиеся и понятные имена.

Наконец, кто-то должен заниматься и подсобными работами. Поэтому нам не обойтись без макросов, позволяющих "смыть" следы неудачной или устаревшей покраски и вернуть соответствующим элементам таблицы их первозданный вид. Для этого потребуется создать один такой макрос для цветных строк и один — для цветных столбцов. В процессе создания этих макросов необходимо будет на шестом шаге щелкнуть в палитре цветов ячейку, в которой написано "Нет заливки".

Еще раз напоминаем, что макросы, создание которых описывается в первой части этой книги, создаются в автоматическом режиме. Это, в частности, означает, что в макрос будут записаны значения некоторых настроек среды Microsoft Word, в которой осуществлялось создание макроса. На-

пример, создание макроса, выполняющего подкрашивание строки таблицы, может происходить при отключенном отображении границ ячеек таблицы. Если в дальнейшем этот макрос будет использован для подкрашивания строки таблицы, в которой отображение границ ячеек включено, то после запуска макроса границы ячеек в этой строке станут невидимыми. Подобные побочные эффекты применения макросов могут оказаться в ряде случаев весьма нежелательными. Как можно избежать их проявления, описывается в *гл. 9*.

Совет

Если вы предполагаете в дальнейшем выводить свою таблицу на печать, то мы рекомендуем вам выбирать для окрашивания элементов таблицы цвета заливки светлого оттенка. Дело в том, что темные насыщенные цвета могут иногда весьма эффектно выглядеть на экране, однако при распечатке документа текст, содержащийся в этих строках, будет практически сливаться с фоном, что затруднит его чтение.

5.2. Покрасили, теперь можно что-нибудь и построить

Повысить информативность содержимого таблицы можно не только с помощью раскрашивания отдельных ее элементов. Довольно часто эффективность использования таблицы определяется способом ее компоновки, удобством восприятия показателей.

5.2.1. Таблицы-небоскребы: главное не фундамент, а крыша

В некоторых случаях таблицы, включаемые в документы Microsoft Word, являются очень громоздкими, их содержимое может размещаться на нескольких страницах документа. К сожалению, в Microsoft Word отсутствует возможность закрепления на экране шапки таблицы, т. е. области заголовков ее столбцов (такая возможность имеется, например, в Microsoft Excel). Поэтому если, перемещаясь по таблице, вы потеряете из виду ее шапку, то будет довольно сложно определить, какие показатели находятся в том или ином столбце. Справиться с такого рода проблемами нам помогут макросы, к созданию которых мы сейчас и приступаем.

Полезные сведения

В Microsoft Word имеется команда, позволяющая отобразить первую строку таблицы в верхней части всех страниц документа, на которых размещается таблица (меню **Таблица**, команда **Заголовки**). Однако применение этой

команды не решает стоящей перед нами задачи: во-первых, при перемещении в нижнюю часть страницы шапка таблицы все равно исчезает с экрана; во-вторых, отображаться на других страницах будет только одна (первая) строка таблицы, хотя ее шапка может, вообще говоря, состоять из нескольких строк.

Ситуация

Вы просматриваете или редактируете таблицу, данные которой располагаются на нескольких страницах. Вначале особых трудностей это не вызывает, однако при переходе на следующую страницу документа шапка (головка) таблицы стала вам не видна, и вы не можете точно вспомнить, какие показатели располагаются в том или ином столбце.

Что требуется получить

Вы нажимаете кнопку на панели инструментов, и окно документа делится по горизонтали на две области. При этом в верхней области окна, имеющей относительно небольшой размер, отображается шапка таблицы с названиями ее столбцов, а в нижней области, занимающей почти весь экран, — та часть таблицы, в которой находился курсор в момент нажатия кнопки. Вы можете прокручивать и редактировать содержимое таблицы, отображаемое в нижней области окна документа, и одновременно видеть в верхней его части названия каждого из столбцов таблицы.

Способ решения

Основная идея создания макроса, обеспечивающего решение данной задачи, состоит в использовании возможности Microsoft Word, позволяющей разделить окно документа на две части с помощью горизонтальной линии, которая появляется на экране после выполнения соответствующей команды. Положение этой линии может регулироваться пользователем с помощью мыши или клавиш навигации. Фиксация ее позиции на экране происходит по щелчку мыши или нажатию клавиши <Enter>. В каждом из созданных таким образом окон отображается один и тот же документ.

Решение задачи

Основные этапы: размещение курсора в одной из строк таблицы, находящейся на второй или третьей страницах таблицы (предварительный шаг); запуск процедуры записи макроса и создание для него кнопки (шаги 1—2); выбор обычного режима отображения документа (шаг 3); вставка закладки, фиксирующей позицию, в которой находится курсор (шаг 4); переход в область шапки таблицы (шаги 5—6); разбиение окна документа на две части (шаги 6—7); возврат на позицию, в которой находился курсор перед запуском процедуры создания макроса (шаг 8); завершение записи макроса (шаг 9).

Выведите на экран таблицу, содержимое которой занимает несколько страниц. Если у вас нет под рукой такой таблицы, можно быстро создать "пустографку" большой таблицы, состоящую из нескольких сотен строк (чтобы уж наверняка...). Для этого откройте меню **Таблица** и последовательно выберите команды **Добавить** и **Таблица**. Появится диалоговое окно **Вставка таблицы**, в поле **Число строк** которого введите произвольное, но достаточно большое значение, например, 300 (рис. 5.2).

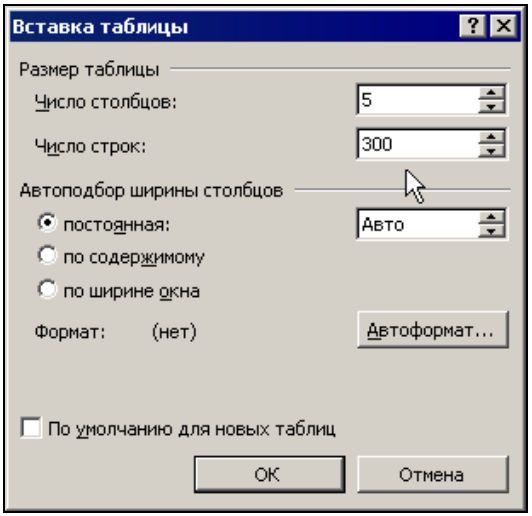


Рис. 5.2. Задание размера таблицы в диалоговом окне **Вставка таблицы**

В первой строке полученной таблицы запишите произвольный текст, "имитирующий" заголовки столбцов таблицы (рекомендуем также подкрасить эту строку — с помощью созданного нами в предыдущем разделе макроса, — чтобы она была лучше заметна на экране). Поместите курсор в произвольную ячейку одной из строк таблицы, расположенной в той части таблицы, которая находится достаточно далеко от ее шапки (например, на второй или третьей странице таблицы).

1. Откройте диалоговое окно **Запись макроса** и введите имя создаваемого макроса, например, **TableHead** (т. е. "шапка таблицы").
2. Создайте для макроса кнопку и оформите ее внешний вид, после чего приступайте непосредственно к записи макроса.
3. Откройте меню **Вид** и выберите команду **Обычный**.

   Перевод в обычный режим отображения документа следует сделать для того, чтобы в дальнейшем при записи макроса не нужно было учитывать

положение шапки таблицы на странице. Дело в том, перед началом записи макроса мы не определяли режим отображения, поэтому он может быть, вообще говоря, произвольным. Если документ, содержащий таблицу, отображается в режиме **Разметка страницы**, то после выполнения шага 3 он будет отображаться без разбиения на страницы. Выбор обычного режима отображения документа следует сделать и в том случае, когда он был включен у вас еще до начала записи макроса (чтобы гарантировать переход в этот режим независимо от исходного режима отображения документа).

4. Откройте меню **Вставка**, выберите команду **Закладка** и введите имя закладки, вставляемой в ячейку, в которой находится сейчас курсор, например, **CurrentRow** (т. е. "текущая строка").
5. Откройте меню **Таблица** и последовательно выберите команды **Выделить** и **Столбец**.
6. Нажмите клавишу <↑>, после чего курсор переместится в первую строку таблицы или в область ее заголовка (если таковой имеется; не путать с шапкой).
7. Откройте меню **Окно** и выберите команду **Разделить** (или нажмите кнопку **Разделить окно**  на панели инструментов), после чего в средней части экрана появится горизонтальная линия, делящая окно документа пополам.

Примечание

Чтобы отменить разделение окна документа, необходимо выбрать команду **Снять разделение** в меню **Окно** или повторно нажать кнопку **Разделить окно**. Отметим, что функция разделения окна не работает в режиме **Схема документа** (переход в этот режим выполняется с помощью одноименной кнопки  панели инструментов).

Перетащите с помощью мыши эту линию вверх таким образом, чтобы в верхней части окна (над линией) отображались заголовки столбцов таблицы. Определив оптимальное положение линии, щелкните кнопкой мыши.

Полезные сведения

Как в верхней, так и в нижней областях отображается один и тот же документ. Навигация в этих областях осуществляется независимым образом, благодаря чему на экране могут одновременно просматриваться различные части этого документа. Причем изменения, вносимые в документ в одной из областей, мгновенно воспроизводятся и в другой области. Разбить окно документа на большее число подобных областей в Microsoft Word нельзя.

8. Откройте меню **Вставка**, выберите команду **Закладка**, щелкните в окне **Имя закладки** закладку с именем **CurrentRow** и нажмите кнопку **Перейти**, а затем кнопку **Закрыть**.

   Курсор в нижней области окна документа переместится на позицию, в которой он находился перед запуском процедуры создания макроса. В верхней области окна по-прежнему будет отображаться шапка таблицы.

9. Завершите процедуру создания макроса.

Что мы имеем в итоге

При создании этого макроса мы воспользовались как уже известными нам приемами (создание закладки, выделение одного из элементов таблицы), так и новыми возможностями, связанными с разбиением окна документа на две области. Это позволило нам эффективно и достаточно просто решить довольно сложную задачу. Теперь, работая с большими таблицами, вы можете в любой момент уточнить, какие показатели находятся в том или ином ее столбце. Для этого нужно всего лишь нажать кнопку, запускающую только что созданный нами макрос.

> ***Совет***
>
> Если шапка таблицы занимает не одну, а несколько строк, то она может не уместиться в верхней области окна документа, создаваемой в результате применения нашего макроса. Чтобы сделать шапку таблицы полностью видимой, можно "вручную" перетащить с помощью мыши горизонтальную разделительную линию немного вниз.

5.2.2. Большие проблемы малоэтажного строительства

Таблицы большого формата могут состоять, как известно, не только из большого количества строк, но и из большого числа столбцов. К тому же, в некоторых таких таблицах имеется так называемый *боковик таблицы*. Иными словами, в первом столбце таблицы могут содержаться заголовки ее соответствующих строк. При работе с такими таблицами возникают те же проблемы, как и в случае, который рассматривался нами в предыдущем разделе. Единственное отличие состоит лишь в том, что при работе с таблицей можно потерять из виду уже ее боковик, а не шапку.

К сожалению, для решения данной проблемы мы не сможем воспользоваться тем простым приемом, который применялся нами для закрепления на экране области документа, содержащей шапку таблицы. Дело в том, что в Microsoft Word разбить окно документа на две части с помощью команды **Разделить** можно только по горизонтали, но не по вертикали. Однако с этой

целью могут быть использованы и некоторые другие средства. Макрос, созданный на их основе, позволяет получить формально такой же, как и в случае применения макроса **TableHead**, результат (разбиение экрана на две области, отображение в этих областях одного и того же исходного содержимого и т. п.). Однако "механизм", лежащий в основе этого макроса, является качественно иным. Достаточно сказать, что работать нам придется не с самим документом, а с некоторым его представлением, о существовании которого многие пользователи Microsoft Word, возможно, даже и не подозревают.

Предупреждаем также, что процедура создания макроса включает довольно много шагов и требует повышенного терпения и аккуратности. Учитывая вышесказанное, мы решили отнести описание процедуры создания макроса, закрепляющего на экране боковик таблицы, к дополнительному материалу. Поэтому читатели, желающие ознакомиться с новыми средствами и возможностями Microsoft Word, а также лишний раз потренироваться в написании макросов и получить в свое распоряжение полезное средство работы с таблицами, могут сделать это, прочитав следующий раздел главы.

5.3. Дополнительные сведения

В этом разделе содержится описание процедуры создания макроса, позволяющего закрепить на экране боковик горизонтальной таблицы большого формата. Кроме того, здесь приводятся некоторые общие сведения о веб-страницах и использовании рамок в документах Microsoft Word, а также описываются некоторые полезные команды работы с таблицами.

Примечание

Перевод на русский язык английского слова "Web" в настоящее время еще не устоялся. Довольно часто в отечественной литературе оно записывается в своей исходной англоязычной форме. Однако в этой книге мы будем руководствоваться рекомендациями корпорации Microsoft, которая при локализации (переводе на русский язык) своих программных продуктов придерживается следующих правил: слово "Web" в случаях, когда оно используется в качестве самостоятельного понятия, должно записываться кириллицей в своей транскрибированной форме, а именно как "веб". Если же оно лишь указывает на технологические особенности соответствующих инструментальных средств или технологий (например, Web technology, Web page, Web server и т. п.), то соответствующие выражения при переводе на русский язык должны записываться через дефис, т. е.: веб-технология, веб-страница, веб-сервер. Справедливости ради, следует отметить, что это правило и самой корпорацией Microsoft воплощается в жизнь недостаточно последовательно, в чем мы сможем убедиться уже в следующем разделе.

В дополнение к сказанному коснемся еще особенностей перевода выражения "Web site". В соответствии с теми же рекомендациями корпорации Microsoft оно должно переводиться на русский язык как "веб-узел". Однако подавляющее большинство пользователей, по нашим наблюдением, продолжают пользоваться буквальной калькой этого выражения — "веб-сайт". Поэтому чтобы не потерять взаимопонимания с нашими читателями, мы также возьмем его на вооружение.

5.3.1. Панорама стройки из окошка веб-сайта

В этом разделе мы займемся созданием макроса, с помощью которого вы сможете при работе с большой, многостраничной таблицей постоянно отображать на экране ее шапку.

Ситуация

Вы просматриваете или редактируете таблицу с большим количеством столбцов, имеющую "альбомную" ориентацию. При этом в первом ее столбце (боковике) содержатся заголовки соответствующих граф таблицы. При перемещении в правую ее часть боковик исчез с экрана, и вы не можете точно вспомнить, какие показатели относятся к той или иной графе.

Что требуется получить

Вы нажимаете кнопку на панели инструментов, и окно документа делится по вертикали на две области. При этом в левой области окна, имеющей относительно небольшой размер, отображается боковик таблицы с названиями ее граф, а в правой области, занимающей почти весь экран, — та часть таблицы, в которой находился курсор в момент нажатия кнопки. Вы можете прокручивать по горизонтали и редактировать содержимое таблицы, отображаемое в правой области окна документа, и одновременно видеть в левой части экрана заголовки соответствующих граф.

Способ решения

Основная идея создания макроса, обеспечивающего решение данной задачи, состоит в использовании возможностей Microsoft Word, связанных с применением так называемых *рамок*. Это средство принципиально отличается от использованного нами при создании предыдущего макроса (разбиение окна документа на две области по горизонтали). Рамки, как правило, применяются для создания страниц веб-сайтов. Оказывается, Microsoft Word может использоваться и для этой цели. Однако задачи, связанные с построением веб-сайтов, — это особая тема, выходящая за рамки данной книги.

Примечание

Следует иметь в виду, что термин "рамка" (frame) имеет в Microsoft Word неоднозначное толкование. В этом разделе речь идет о рамках, обычно используемых на веб-страницах, и их не следует путать с рамками, которые широко использовались в более ранних версиях Microsoft Word для врезки текста и рисунков. Сейчас они применяются редко, их роль в значительной степени (но не полностью) взяли на себя так называемые надписи.

Читателей, желающих ознакомиться с возможностями создания веб-документов в Microsoft Word, мы отсылаем к справке программы и соответствующей литературе (некоторые общие сведения о веб-документах приводятся далее в этом разделе). Мы же попробуем применить это "нестандартное" средство для решения стоящей перед нами конкретной проблемы.

Решение задачи

Основные этапы: размещение курсора в одной из клеток таблицы в правой части таблицы, из которой не виден ее боковик (предварительный шаг); запуск процедуры записи макроса и создание для него кнопки (шаги 1—2); выбор обычного режима отображения документа (шаг 3); вставка закладки, фиксирующей позицию, в которой находится курсор (шаг 4); переход в область боковика таблицы и вставка закладки в соответствующую ячейку первого столбца (шаги 5—7); вставка рамки в левую часть окна документа (шаги 8—9); возврат на позицию, в которой находился курсор перед запуском процедуры создания макроса (шаги 10—11); выделение таблицы и ее копирование в буфер обмена (шаги 12—13); вставка таблицы из буфера обмена в левую рамку (шаг 14); переход в область боковика таблицы, скопированной в левую область экрана, и определение размера левой рамки, в которой отображается боковик таблицы (шаги 15—16); завершение записи макроса (шаг 17).

Выведите на экран таблицу, в которой имеется достаточно большое количество столбцов. Если у вас нет под рукой такой таблицы, можно быстро создать ее "пустографку". Для этого откройте меню **Таблица** и последовательно выберите команды **Добавить** и **Таблица**. Появится диалоговое окно **Вставка таблицы**, в поле **Число столбцов** которого введите произвольное, но достаточно большое значение, например, 50. Такие большие таблицы обычно имеют альбомную (горизонтальную) ориентацию, поэтому следует на вкладке **Размер бумаги** диалогового окна **Параметры страницы** (меню **Файл**, команда **Параметры страницы**) в разделе **Ориентация** выбрать переключатель **альбомная** (рис. 5.3). В одной из ячеек первого столбца полученной таблицы запишите произвольный текст, "имитирующий" заголовок одной из строк боковика таблицы (настоятельно рекомендуем подкрасить эту строку, а также первый столбец таблицы — с помощью созданных нами ранее макросов, — чтобы они были лучше заметны на экране).

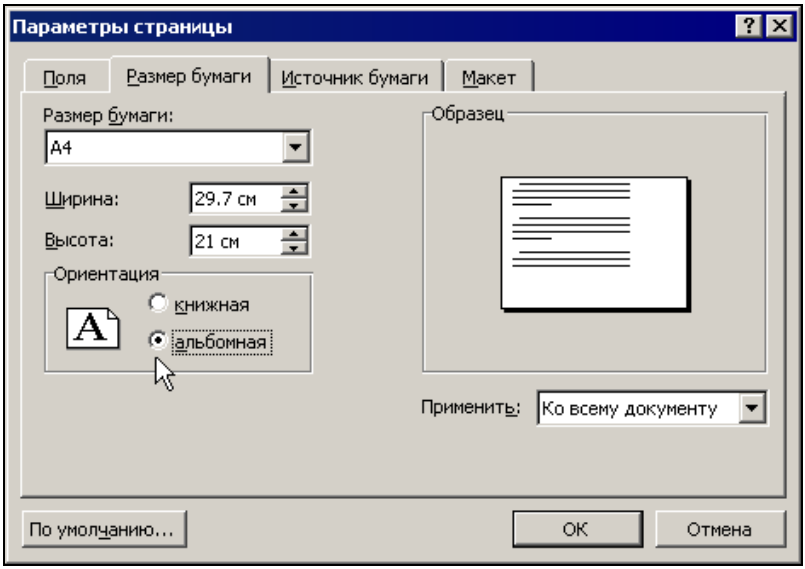


Рис. 5.3. Задание альбомной ориентации страницы на вкладке **Размер бумаги** диалогового окна **Параметры страницы**

1. Откройте диалоговое окно **Запись макроса** и введите имя создаваемого макроса, например, **TableStub** (т. е. "боковик таблицы").
2. Создайте для макроса кнопку и оформите ее внешний вид, после чего приступайте непосредственно к записи макроса.
3. Откройте меню **Вид** и выберите команду **Обычный**.
4. Откройте меню **Вставка**, выберите команду **Закладка** и введите имя закладки, вставляемой в ячейку, в которой находится сейчас курсор, например, **CurrentCell** (т. е. "текущая ячейка").
5. Откройте меню **Таблица** и последовательно выберите команды **Выделить** и **Строка**.
6. Нажмите клавишу <←>, после чего курсор переместится в первую ячейку выделенной строки таблицы.
7. Откройте меню **Вставка**, выберите команду **Закладка** и введите имя закладки, вставляемой в данную ячейку, например, **CurrentFirstCell** (т. е. "текущая первая ячейка").
8. Откройте меню **Вид** и последовательно выберите команды **Панели инструментов** и **Рамки**, после чего на экране появится панель инструментов **Рамки** (рис. 5.4).

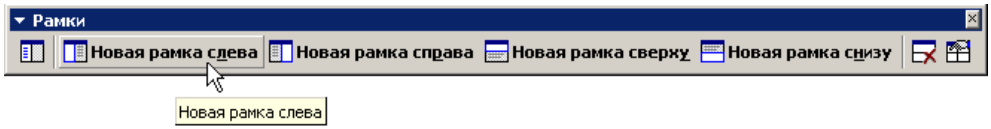


Рис. 5.4. Панель инструментов **Рамки**

Панель инструментов **Рамки** является встроенной стандартной панелью Microsoft Word. Напоминаем, что ее можно также вывести на экран, наведя курсор мыши на любую из панелей и нажав правую кнопку мыши. После чего следует выбрать в появившемся списке панель **Рамки**.

9. Нажмите кнопку **Новая рамка слева**, после чего на экране появится вертикальная линия, делящая экран на две равные области. В правой из этих областей будет отображаться содержимое исходного документа с таблицей, а левая окажется пустой. Обратите внимание, что в результате операции, произведенной на данном шаге, создается **новый** документ, у которого пока еще нет имени.
10. Щелкните внутри правой рамки, откройте меню **Вставка** и выберите команду **Закладка**.
11. Выберите в появившемся диалоговом окне **Закладка** закладку **CurrentCell** и нажмите кнопку **Перейти**.
12. Откройте меню **Таблица** и последовательно выберите команды **Выделить** и **Таблица**.
13. Откройте меню **Правка** и выберите команду **Копировать**.
14. Щелкните левую (пустую) рамку, откройте меню **Правка** и выберите команду **Вставить**. В результате в левую рамку будет скопирована таблица, выделенная на предыдущем шаге.
15. Откройте меню **Вставка** и выберите команду **Закладка**.
16. Выберите в появившемся диалоговом окне **Закладка** закладку **CurrentFirstCell** и нажмите кнопку **Перейти**.

    С помощью мыши перетащите влево вертикальную линию, разделяющую экран на две части, таким образом, чтобы в левой рамке отображался только боковик таблицы.
17. Завершите процедуру создания макроса.

Макрос создан, и мы поздравляем читателей, у которых хватило терпения выполнить вместе с нами все описанные выше действия.

Что мы имеем в итоге

После ознакомления с приведенным ранее материалом у наиболее внимательных читателей мог возникнуть следующий резонный вопрос: "Хорошо, мы решили задачу отображения боковика, что упрощает просмотр таблиц, содержащих большое количество столбцов. Однако просмотр этот осуществляется в **новом** документе, который создается при запуске нашего макроса. А как следует поступать в случае, когда потребуется внести изменения в исходную таблицу, — сначала редактировать созданную нами копию, а затем переносить изменения в исходный документ? Не очень-то это удобно...". Справедливое замечание, однако не спешите делать выводы.

Мы уже говорили, что при создании данного макроса мы воспользовались рамками — средством Microsoft Word, которое обычно применяется для создания страниц веб-сайтов. Мы не будем здесь останавливаться на особенностях веб-документов, отметим только, что документ с рамками обладает следующим замечательным свойством. Каждую его рамку можно связать с другим конкретным документом, содержимое которого будет отображаться в данной области. При этом любые изменения, вносимые в заданный таким образом документ, будут мгновенно отображаться в соответствующей области рамок. Более того, оказывается, справедливо и обратное утверждение: любые изменения, вносимые в содержимое той или иной рамки, мгновенно воспроизводятся и в соответствующем ей документе.

Это свойство документа с рамками позволяет нам решить проблему редактирования документа, для просмотра которого используется макрос **TableStub**. Чтобы связать документ, создаваемый при запуске этого макроса, с исходным документом, выполните следующие действия:

1. Щелкните внутри правой рамки, а затем нажмите на панели **Рамки** кнопку **Свойства рамки**  , после чего откроется одноименное диалоговое окно (рис. 5.5).
2. Выберите в этом окне вкладку **Рамка** и установите на ней флажок **связать с файлом**.
3. Нажмите кнопку **Обзор** и найдите на диске файл, с которым требуется связать данную рамку (например, MyBigTable.doc).
4. Нажмите кнопку **OK**.
5. Щелкните внутри левой рамки документа и повторите описанные выше шаги (1—4).

При этом связать эту рамку следует с тем же самым документом, что и правую рамку. В результате будет создано средство для просмотра и редактирования конкретной таблицы, имеющей большое количество столбцов. Сохраните документ, полученный в результате описанных ранее действий, в обычном формате Microsoft Word, присвоив ему желаемое имя.

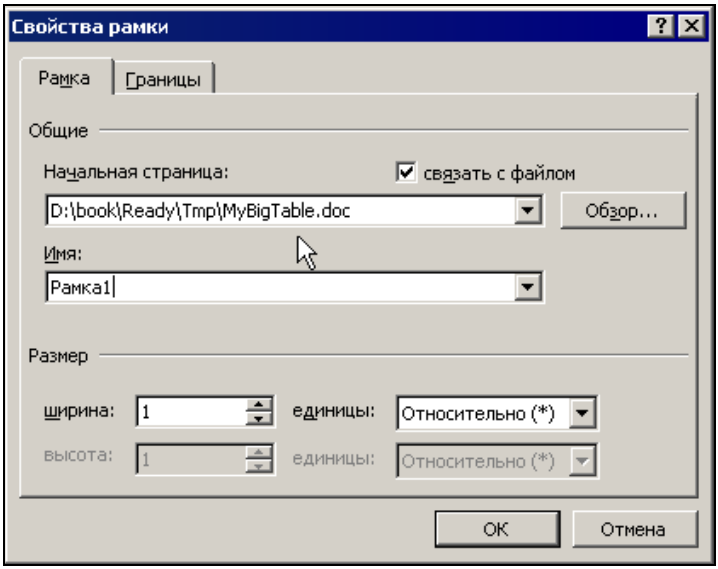


Рис. 5.5. Диалоговое окно **Свойства рамки**, вкладка **Рамка**

5.3.2. Общие сведения о рамках и создании веб-документов в Microsoft Word

Напомним, что рамки — это средство Microsoft Word, которое использовалось нами при создании макроса, позволяющего отобразить боковик таблицы. С помощью рамок может осуществляться упорядочивание отображения информации на экране компьютера.

В одном документе Microsoft Word может быть создано практически неограниченное количество рамок. В свою очередь, рамки можно создавать в левой части документа (как в случае с нашим макросом), в правой (с помощью кнопки **Новая рамка справа** на панели инструментов **Рамки**), а также в верхней (кнопка **Новая рамка сверху**) и нижней (кнопка **Новая рамка снизу**). В свою очередь, удалить рамку можно с помощью кнопки  , расположенной на панели инструментов **Рамки** (см. рис. 5.4).

Как уже отмечалось при создании макроса **TableStub**, каждую рамку можно связать с различными файлами Microsoft Word. Более того, с помощью гиперссылок (меню **Вставка**, команда **Гиперссылка**) их можно связать с теми или иными веб-сайтами в Интернете. Таким образом, рамки как бы становятся окнами, через которые можно взглянуть на содержимое тех или иных документов.

Страницы с рамками можно сохранять в формате обычного документа Microsoft Word. Однако их можно также сохранить в формате веб-страницы. Для этого следует выбрать в меню **Файл** команду **Сохранить как Web-страницу** (как видим, рекомендации корпорации Microsoft в области локализации интерфейса программ выполняются не всегда).

Говоря упрощенно, веб-страницей является файл, имеющий специальный формат (его наличие определяется расширением htm в имени файла). Содержимое файлов данного формата может просматриваться с помощью специальных средств, таких как обозреватель (браузер) Internet Explorer. Именно такие файлы, а точнее наборы таких файлов и используются при создании страниц веб-сайтов. Рамки упрощают создание стандартных элементов страниц веб-сайтов, таких как заголовки и оглавления, позволяют "листать" на экране веб-страницы.

Чтобы создать оглавление документа, отображаемого в рамке, необходимо поместить курсор в эту рамку и нажать кнопку **Оглавление в рамке** () на панели инструментов **Рамки** (или выбрать одноименную команду в меню **Формат**). После этого в левой части рамки будет создана новая рамка, в которую в качестве пунктов оглавления будут автоматически включены фрагменты документа, отформатированные с помощью стилей (стандартных или пользовательских), применяемых для выделения заголовков. Каждый пункт полученного таким образом оглавления представляет собой гиперссылку, щелкнув которую можно мгновенно переместиться в соответствующий раздел исходного документа.

На странице рамок можно также создать общее оглавление, пункты которого являются гиперссылками, указывающими на другие рамки данной страницы.

Формат границ рамок (их ширина и цвет) может определяться самим пользователем (рис. 5.6). При желании можно также скрыть отображение границ рамки. Для этого следует выбрать переключатель **Границы отсутствуют** в диалоговом окне **Свойства рамки**.

Как уже отмечалось, рамки могут использоваться при создании веб-сайта, который, говоря упрощенно, представляет собой набор взаимосвязанных веб-страниц. Общий принцип создания такого набора страниц с помощью рамок описан нами ранее. Данный набор может быть размещен на компьютере (сервере), подключенном к локальной сети организации. В результате будет получен корпоративный веб-сайт (т. е. веб-сайт отдельной организации). Пользователи сети организации, зная сетевой адрес начальной (домашней) страницы данного узла, могут загружать ее в свои веб-обозреватели (браузеры). Щелкая гиперссылки, имеющиеся на этой странице, они могут переходить на другие страницы веб-сайта (в действительности, как мы уже знаем, в этом случае в рамку загружается связанный с ней документ). На

этих же принципах основано создание и использование веб-сайтов Интернета. Разумеется, мы изложили сильно упрощенную общую схему, подробности которой наши читатели могут при желании узнать в соответствующей литературе.

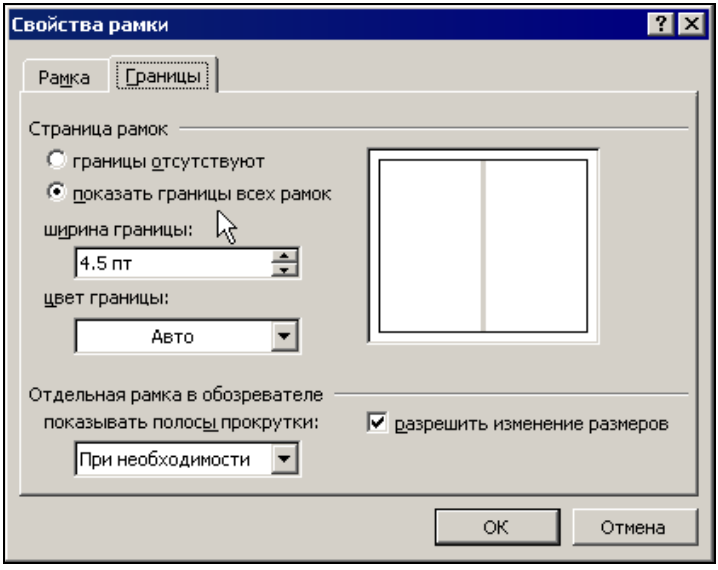


Рис. 5.6. Диалоговое окно **Свойства рамки**, вкладка **Границы**

Примечание

С помощью Microsoft Word могут создаваться только достаточно простые по своей структуре и функциональным возможностям веб-сайты. Для построения веб-сайта профессионального уровня используются более мощные специализированные программы, такие, например, как Microsoft FrontPage.

5.3.3. Общие сведения о командах работы с таблицами Microsoft Word

В Microsoft Word имеется довольно развитый аппарат работы с таблицами. Большинство этих команд доступно в меню **Таблица** (рис. 5.7). С их помощью может осуществляться добавление, удаление и выделение элементов таблицы (столбцов, строк, ячеек и таблицы в целом); объединение и разбиение ячеек; автоподбор ширины столбцов, строк и таблицы в целом; отображение и скрытие границ таблицы, преобразование текста в таблицу и табли-

цы в текст, сортировка элементов столбцов, запись простых формул, по которым могут рассчитываться показатели таблицы, и т. п.

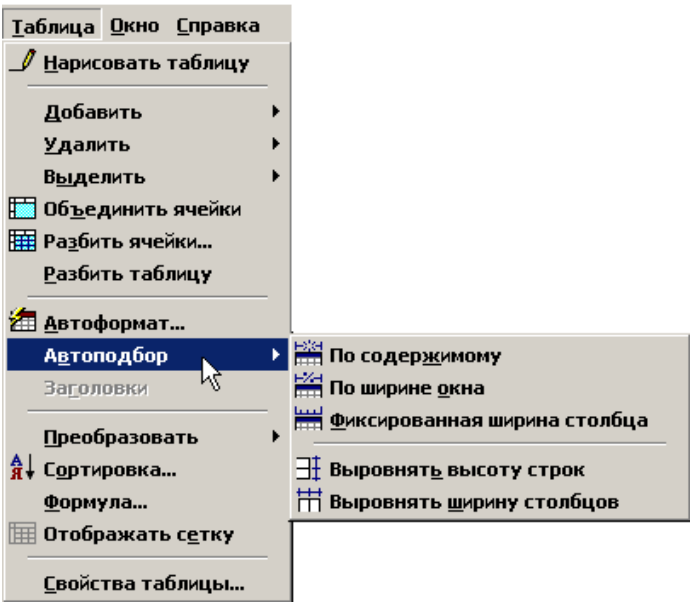


Рис. 5.7. Команды меню **Таблица**
(отображается также вложенное меню **Автоподбор**)

Команда **Свойства таблицы** позволяет уточнять характеристики элементов таблицы, а также определять место размещения таблицы в документе, задавать способ обтекания таблицы текстом и т. д.

В основном тексте главы упоминалась команда **Заголовки**, позволяющая закрепить первую строку таблицы на всех страницах, на которых отображается эта таблица. Отметим, что данная команда доступна в меню **Таблица** только в том случае, когда курсор находится в первой строке таблицы.

С помощью команды **Нарисовать таблицу** пользователь может создать таблицу нестандартного вида, содержащую ячейки различной высоты, ячейки, разделенные по диагонали, таблицу с различным числом столбцов в строке и т. п. При выборе этой команды на экране появляется панель инструментов **Таблицы и границы** (рис. 5.8), а указатель принимает вид карандаша, с помощью которого могут быть "прорисованы" внешние границы таблицы, внутри которых затем можно нарисовать линии столбцов, строк и отдельных ячеек.

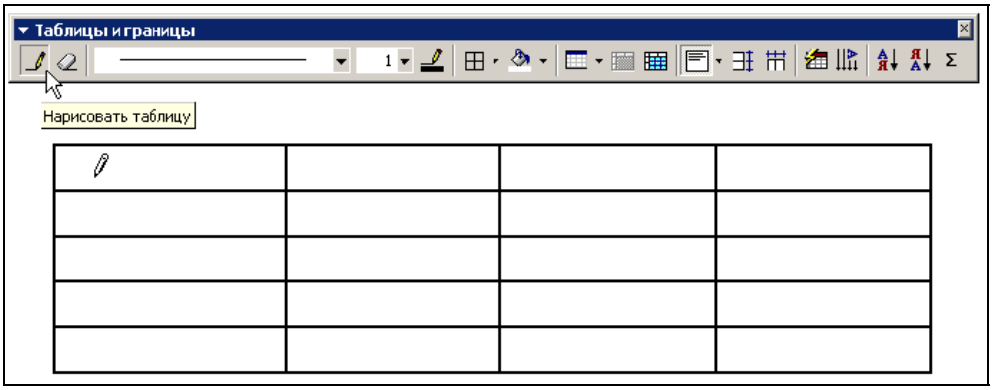


Рис. 5.8. Панель инструментов **Таблицы и границы** (активизирована кнопка **Нарисовать таблицу**)

Некоторые из кнопок, находящихся на панели **Таблицы и границы**, дублируют соответствующие команды меню **Таблица** (напоминаем, что название кнопки можно посмотреть, наведя на нее указатель мыши, — в случае если установлен флажок **Отображать подсказки для кнопок** на вкладке **Параметры** диалогового окна **Настройка**). Однако имеются также кнопки и инструментальные средства, с помощью которых осуществляется дополнительная настройка таблицы: удаление границ между ячейками (средство **Ластик**), определение формата, цвета и типа рисуемых в таблице линий, способа отображения границ ячеек (все границы, только внешние границы, только верхняя граница и т. п.), цвета заливки таблицы и ее отдельных элементов, направления текста, отображаемого в ячейке, и т. д. Здесь же имеется кнопка, позволяющая суммировать (по столбцу или строке) числа, представленные в таблице.

Как видим, команд и средств работы с таблицами в Microsoft Word довольно много. А это означает, что с их помощью вы можете создать еще много полезных макросов!

5.4. Что нового в этой главе

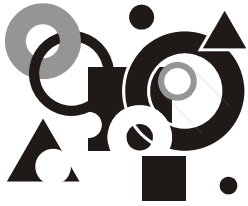
Эту главу мы посвятили вопросам повышения эффективности работы с таблицами Microsoft Word. Нами был создан макрос, позволяющий раскрашивать заданным цветом строки таблицы. Те же из вас, кто решил применить на практике прием, описанный при создании данного макроса, могли получить в свое распоряжение целый набор макросов, раскрашивающих в ваши любимые цвета строки и столбцы таблицы, отдельные ячейки и таблицу в целом (а также макрос, позволяющий убирать следы неудачной покраски).

Был создан также макрос, незаменимый при работе с многостраничными таблицами формата, содержащими большое количество строк. Этот макрос позволяет постоянно отображать на экране шапку (головку) таблицы, благодаря чему вы всегда будете точно знать, какие показатели располагаются в том или ином ее столбце.

Те из вас, кто ознакомился с материалом раздела дополнительных сведений, смогли закрепить полученные знания на практике, создав макрос, решающий описанную выше задачу применительно к многостраничным таблицам с большим количеством столбцов. Если в таких таблицах имеется боковик с заголовками строк, то применение нашего макроса позволяет постоянно отображать его на экране, даже переместившись в самые "дальние" столбцы таблицы.

Здесь же приводятся общие сведения о рамках — инструменте Microsoft Word, использованном при создании данного макроса. В частности, рассказывается о возможностях применения этого средства при создании веб-страниц и веб-сайтов.

В этом разделе дается также краткое описание имеющихся в Microsoft Word команд форматирования таблиц.



Глава 6

Экскурсия в сборочный цех макросов

Первое и очень краткое знакомство с редактором Visual Basic. Основные элементы программы: ее начало и конец. "Клонирование" макросов с помощью обычных средств Microsoft Word (выделение, копирование в буфер обмена, вставка и редактирование текста) — на примере создания макроса, вставляющего закладку. Клонирование отдельной процедуры в составе макроса, задание конкретных значений параметров для инструкций программы — на примере создания макроса, позволяющего проводить первичную литературную правку документа.

Эта глава адресуется тем из наших читателей, которые, освоив изложенный ранее материал, хотели бы поближе ознакомиться непосредственно с механизмом, лежащим в основе создания макросов. В основе этого желания может лежать обычное человеческое любопытство (а что там, в черном ящике?), а может быть, кому-то из вас уже не терпится попробовать свои силы в практическом программировании (если это может делать текстовый процессор Microsoft Word, то, возможно, и я смогу?).

Мы предоставляем вам такую возможность и приглашаем отправиться на экскурсию в сборочный цех макросов Microsoft Word. До сих пор этот цех обслуживал нас, работая в автоматическом режиме. Он исправно выдавал свою продукцию — макросы, сборка которых осуществлялась из стандартных узлов и деталей. Организация процесса бесперебойной поставки компонентов, а также соблюдение требований технологического процесса обеспечивались "начальником цеха по фамилии" Редактор Visual Basic.

Подобная конвейерная схема производства вполне подходит для организации массового выпуска стандартных изделий. Однако если потребуется внести в эти изделия какие-либо усовершенствования или наладить выпуск совершенно новой продукции, сделать это без вмешательства специалиста-наладчика будет невозможно.

В этой главе вы сможете "подержать в руках" некоторые стандартные блоки, из которых осуществляется автоматическая сборка макросов. Мы покажем

вам, каким образом можно, не вникая в принципы и нюансы работы отдельных компонентов программы, использовать их для решения некоторых практических задач. При этом будут использоваться только традиционные для Microsoft Word операции — редактирование текста, его выделение, копирование в буфер обмена и вставка на нужную позицию в документе. Освоение данного подхода, как минимум, позволит вам расширить круг своих возможностей в области создания макросов. Если же вы убедитесь, что в текстах рассмотренных нами программ можно вполне разобраться хотя бы на интуитивном уровне (особенно полезным при этом оказываются знание английского языка и, конечно же, — здравый смысл), то мы настоятельно рекомендуем вам на этом не останавливаться и продолжить чтение дальше. Освоив материал, представленный в главах второй и третьей частей книги, вы сможете существенно повысить эффективность создаваемых макросов.

Если же вы посчитаете, что еще не дозрели до принятия такого решения, не спешите ставить жирный крест на себе как на программисте. Вполне возможно, что через некоторое время, закрепив на практике навыки создания простейших макросов, вы захотите повысить уровень своей квалификации и продолжите знакомство с материалом этой книги. Поэтому советуем не убирать ее слишком далеко и, на всякий случай, запомнить, на какую полку книжного шкафа поставили.

6.1. Редактор Visual Basic: Добро пожаловать на фабрику макросов

В этой главе мы займемся модификацией некоторых макросов, созданных нами в предыдущих главах. Как не раз уже отмечалось, документ, в котором осуществляется запись макроса в автоматическом режиме, создается в специальной программе, встроенной в Microsoft Word. Эта программа называется редактор Visual Basic, точнее Microsoft Visual Basic (в пакетах Microsoft Office 2000, XP и 2003, в состав которых входит Microsoft Word, используется версия 6.3 этой программы). Более подробно о редакторе Visual Basic будет рассказано в следующей части книги, здесь же мы ограничимся лишь самыми общими сведениями, необходимыми для решения сформулированной ранее задачи.

Прежде всего, о том, как попасть в редактор Visual Basic. Вообще говоря, сделать это можно несколькими способами (более подробно об этом говорится в следующей части книги — *см. гл. 8 и 10*). Для начала мы познакомим вас лишь с одним из них. Этот способ позволит нам не заблудиться во время экскурсии и сразу же попасть в нужное место — к макросу, модификацией которой мы собираемся заняться.

Предположим для определенности, что этим макросом является макрос **ShowMyButtonsPanel**, с помощью которого на экран выводится панель инструментов **MyButtons** (созданием этого макроса мы занимались в *гл. 3* книги — а точнее в *разд. 3.3.3*). Выбор именно этого макроса является не случайным. Дело в том, что программа, реализующая действия, выполняемые с помощью макроса **ShowMyButtonsPanel**, является очень простой — всего в одну строчку. Это облегчит нам первое знакомство с редактором Visual Basic.

Итак, чтобы получить доступ к тексту того или иного макроса, необходимо выполнить следующие действия:

1. Откройте меню **Сервис** и укажите команду **Макрос**.
2. В открывшемся подменю выберите команду **Макросы**, после чего появится диалоговое окно **Макрос**. (Напоминаем, что открыть диалоговое окно **Макрос** можно также с помощью сочетания клавиш <Alt>+<F8>).
3. Щелкните нужный вам макрос (в нашем случае это **ShowMyButtonsPanel**) в списке, который отображается под полем **Имя** (элементы этого списка располагаются в алфавитном порядке; в случае необходимости прокрутите его, чтобы вывести на экран нужный вам элемент). Имя макроса можно также "вручную" ввести в поле **Имя**. В этом случае произойдет автоматическая прокрутка списка макросов, и нужный макрос отобразится в верхней части списка (рис. 6.1).

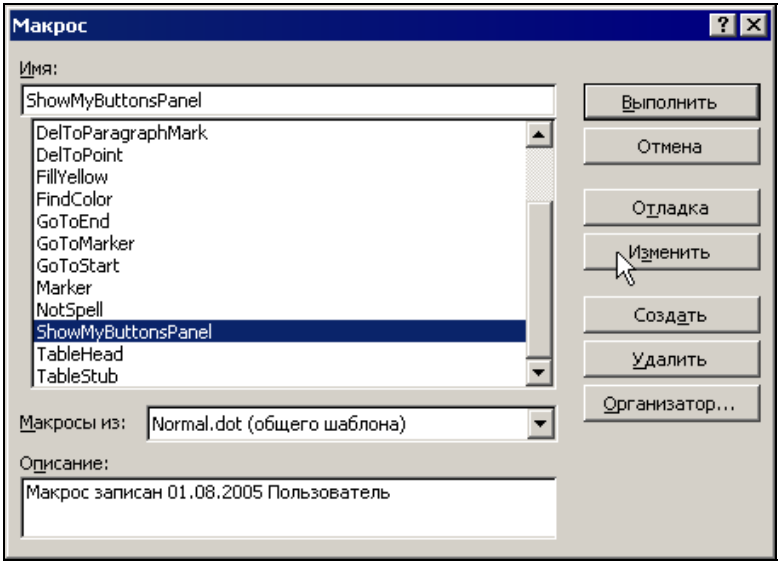


Рис. 6.1. Выбор макроса **ShowMyButtonsPanel** в диалоговом окне **Макрос**

4. Нажмите кнопку **Изменить**, после чего на экране появится редактор Visual Basic, в правом окне которого, занимающем почти всю площадь экрана, отображается текст всех созданных ранее макросов. При этом курсор будет располагаться в начале именно того макроса, который был выбран при выполнении описанного ранее действия.

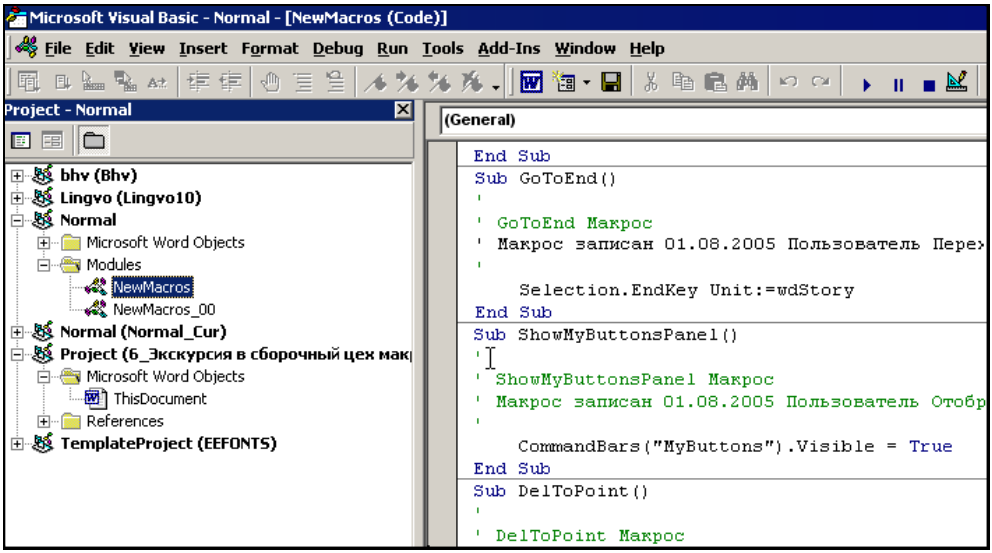


Рис. 6.2. Фрагмент окна редактора Visual Basic.
В правой части видны тексты создававшихся нами макросов.
Курсор расположен в начале макроса **ShowMyButtonsPanel**

Совет

Если окно редактора Visual Basic откроется таким образом, что будет занимать только часть экрана, щелкните кнопку **Развернуть** (), расположенную в правом верхнем углу этого окна, чтобы развернуть его на весь экран.

Описание элементов интерфейса редактора Visual Basic будет приведено в следующей части книги в *гл. 10*, здесь же мы рассмотрим тексты некоторых из созданных нами ранее макросов и на их примере познакомимся с основными компонентами программы.

6.2. Портрет макроса крупным планом

Редактор Visual Basic записывает макросы последовательным образом: каждый вновь создаваемый макрос добавляется в конец "журнала", содержащего тексты всех ранее созданных макросов.

Процесс записи каждого нового макроса начинается в редакторе Visual Basic с автоматической вставки сплошной разделительной линии. Таким образом, текст каждого макроса (за исключением первого и последнего) здесь ограничивается с двух сторон сплошными линиями (см. рис 6.2, а также фрагмент, "вырезанный" нами из основного окна редактора Visual Basic).

В начале каждого макроса располагается его заголовок, имеющий вид `Sub имя_макроса()`. Слово **Sub** является сокращением от английского "subprogram", т. е. подпрограмма (так в Visual Basic называется программа, реализующая действия, выполняемые с помощью макроса). Вместо выражения "имя_макроса" в тексте реальной программы записывается название соответствующего макроса, например, `Sub ShowMyButtonsPanel()`.

> ***Примечание***
>
> Для тех, кто не знает английского языка, сообщаем, что в нашей книге имеется словарь-указатель, в котором приводится перевод практически всех встречающихся в книге английских слов и выражений.

Заканчивается текст макроса выражением **End Sub** (т. е. "конец подпрограммы"). Руководствуясь описанными выше признаками, вы без труда сможете идентифицировать текст макроса **ShowMyButtonsPanel**, который находится в центре приводящегося ниже фрагмента. В начале этого фрагмента (над сплошной линией) располагаются несколько заключительных строк макроса, который был создан нами ранее макроса **ShowMyButtonsPanel** (напоминаем, что этим макросом является макрос **GoToEnd**, позволяющий выполнять переход в конец документа). В конце фрагмента (под сплошной линией) приводятся начальные строки макроса **DelToPoint**, который был создан нами после макроса **ShowMyButtonsPanel** *(см. разд. 4.1.1)*.

Под заголовком каждого макроса редактор Visual Basic автоматически записывает в виде комментария некоторые общие сведения. При этом в первой строке данного *комментария* приводится имя макроса, а также слово "Макрос". На следующей строке содержится информация о дате создания макроса, указывается имя его автора (в нашем случае это "Пользователь"), а также описание, введенное пользователем при создании макроса (у нас — "Отображение панели MyButtons").

> ***Примечание***
>
> Приводимое в комментарии имя автора макроса имеет, вообще говоря, условный характер. Фактически в качестве этого имени записываются сведения, отображаемые в поле **Имя и фамилия** на вкладке **Пользователь** диалогового окна **Параметры**, которое вызывается из меню **Сервис** Microsoft Word.

Напоминаем, что, как отмечалось в предыдущих главах, пользователь может удалить или изменить эти сведения в диалоговом окне **Запись макроса**, а

также добавить в нем краткое описание назначения макроса. В этом случае в комментариях, следующих за заголовком макроса, будут приводиться именно эти сведения.

Использование комментариев позволяет записать в текст макроса сведения, которые не будут учитываться в процессе его выполнения. Как отмечалось ранее, в любой макрос автоматически записываются комментарии с выходными данными. Однако комментарии могут вводиться и в "ручном" режиме. Это может сделать не только создатель макроса, но и любой пользователь, получивший доступ к тексту программы. Чаще всего в виде комментария записываются сведения, поясняющие действия, выполняемые тем или иным блоком программы. Такие пояснения могут очень пригодиться в дальнейшем, если в программу потребуется внести какие-либо изменения.

Чтобы ввести комментарий, необходимо сначала записать значок апострофа. Обращаем ваше внимание, что это "прямой" апостроф, который вводится в английском регистре клавиатуры с помощью той же клавиши, что и буква "э" в русском регистре, а не "косой" апостроф, изображенный на клавише для ввода буквы "ё"). При этом комментарий может записываться как на отдельной, "пустой" строке, так и в конце любой строки с текстом программы. В редакторе Visual Basic комментарий отображается зеленым цветом.

> ***Примечание***
>
> Вообще говоря, значок комментария может вводиться в любом месте строки программы, в том числе в ее начале. В последнем случае данная строка будет полностью рассматриваться в качестве комментария и не учитывается в ходе выполнения программы. Таким приемом часто пользуются программисты, чтобы временно "отключить" какой-либо блок программы (например, при ее отладке). Разумеется, произвольное включение части текста программы в комментарий, может привести и к сбою при ее выполнении.

Собственно текст программы, реализующей действия, выполняемые с помощью макроса, располагается ниже строк комментариев с выходными данными макроса. В случае макроса **ShowMyButtonsPanel** текст данной программы занимает, как видно из приводящейся далее распечатки, или *листинга* (именно так называют распечатку текста программы на профессиональном языке), всего одну строчку (если не считать комментарии, а также заголовок макроса и идентификатор его конца). Для увеличения наглядности мы выделили текст этого макроса в приведенном листинге 6.1 курсивом; полужирным шрифтом выделены "ключевые слова" языка VBA (таким приемом мы будем пользоваться и в дальнейшем).

Разобраться в порядке работы данной программы нам поможет, как мы сейчас убедимся, знание основ английского языка — хотя бы на уровне отдельных "ключевых" слов.

Листинг 6.1

```
    Selection.EndKey Unit:=wdStory
End Sub
```

```
Sub ShowMyButtonsPanel()
'
' ShowMyButtonsPanel Макрос
' Макрос записан 01.01.2005 Пользователь Отображение панели MyButtons
'
    CommandBars("MyButtons").Visible = True
End Sub
```

```
Sub DelToPoint()
'
' DelToPoint Макрос
' Макрос записан 01.01.2005 Пользователь Удаление текста до точки справа
'
    Selection.Extend
Selection.Find.ClearFormatting
```

Выпишем единственную "рабочую" строку макроса **ShowMyButtonsPanel** отдельно и попытаемся разобраться в смысле ее компонентов (в данном случае это будет сделать совсем нетрудно, если перевести на русский язык все представленные в этой строке слова):

```
CommandBars("MyButtons").Visible = True
```

Для тех, кто не знает английского, даем подстрочник: `CommandBars` можно перевести как "панели команд" (`Command` — команда, `Bars` — панели); `MyButtons` — имя панели инструментов, выводимой на экран с помощью нашего макроса (при создании макроса **ShowMyButtonsPanel** мы предполагали, что на этой панели размещается набор кнопок, используемых для форматирования текста. Отсюда и название панели: `MyButtons` — "Мои кнопки"); `Visible` — видимый; `True` — верный, истинный. Если попытаться объединить эти слова в предложение, то даже человек, не являющийся профессиональным переводчиком (не говоря уже — профессиональным программистом), поймет: здесь говорится о том, что панель инструментов **MyButtons** является видимой, т. е. отображается на экране.

Заметим, что использование англоязычных слов и выражений, которые часто довольно точно выражают смысл действий, выполняемых с помощью со-

ответствующих команд и операторов, является характерной особенностью языка программирования VBA, и это значительно облегчает его освоение.

6.3. Создание макросов методом клонирования

При характеристике многих макросов, описанных в предыдущих главах книги, мы неоднократно отмечали, что каждый из них может рассматриваться в качестве представителя целого семейства программ, с помощью которых выполняются сходные по своим результатам действия. Примерами таких семейств могут служить макросы, автоматически вставляющие в документ закладки с заранее определенным текстом (как это делает макрос **Marker**), макросы, выводящие на экран нужную панель инструментов (подобно макросу **ShowMyButtonsPanel**), макросы, раскрашивающие строки таблицы в разные цвета (как макрос **FillYellow**), и т. п. В этом разделе мы покажем, каким образом можно создавать сходные по своему назначению макросы посредством копирования текста их прототипов.

Основная идея этого подхода состоит в следующем. Для создания "клонов" макросов не обязательно многократно повторять практически одну и ту же процедуру их записи. Можно просто войти в редактор Visual Basic и с помощью буфера обмена создать дубликат уже имеющегося макроса-прототипа. После этого необходимо внести в текст полученного таким образом клона некоторые изменения, касающиеся только конкретных параметров, с помощью которых характеризуются индивидуальные особенности нового макроса. Кроме того, соответствующие изменения могут быть внесены в текст комментариев (это действие является желательным, но не обязательным).

> ***Примечание***
>
> Заранее оговоримся, что описанный в этом разделе прием является эффективным, когда с помощью клонируемого макроса выполняется достаточно продолжительная последовательность действий. Кроме того, его использование является оправданным при клонировании отдельных операций внутри одного и того же макроса. Рассматриваемые в этом разделе макросы, вообще говоря, не отвечают указанным условиям. Вместе с тем, соответствующие им программы являются достаточно простыми, чтобы их текст можно было понять, даже не владея навыками программирования. Это соображение являлось решающим при выборе макросов, на примере которых мы сможем познакомиться с некоторыми возможностями редактора Visual Basic.

6.3.1. Нажмите на кнопочку, панелька и откроется

Предлагаем в качестве панели инструментов, которая будет выводиться на экран с помощью клона макроса **ShowMyButtonsPanel**, выбрать стандартную

панель **Visual Basic** (напоминаем, что на ней располагаются кнопки, облегчающие выполнение действий, связанных с созданием макросов).

В обычном режиме работы панель **Visual Basic** не отображается на экране, однако если вы всерьез решите заняться созданием макросов, имеет смысл автоматизировать доступ к расположенным на ней инструментам. Для вывода этой панели на экран можно воспользоваться специальным макросом, запуск которого будет осуществляться посредством нажатия соответствующей кнопки.

Чтобы решить стоящую перед нами задачу, скопируем текст макроса **ShowMyButtonsPanel** и воспользуемся полученным "клоном" для вывода на экран панели инструментов **Visual Basic**. Нетрудно догадаться, что для придания клону индивидуальных черт нужно будет, во-первых, дать ему новое имя и, во-вторых, изменить в тексте макроса название панели, вызываемой с его помощью.

> ***Предупреждение***
>
> В Visual Basic не допускается создавать макросы, имеющие одинаковые имена. В этом случае будет выведено соответствующее сообщение об ошибке.

Для реализации этой задачи выполните следующие действия:

1. Войдите в редактор Visual Basic с помощью процедуры, описанной ранее в *разд. 6.1.*
2. Поместите курсор в начало первой строки макроса **ShowMyButtonsPanel** (напоминаем, что ею является строка `Sub ShowMyButtonsPanel()`) и выделите весь текст макроса, включая его последнюю строку, `End Sub` (см. листинг 6.1).
3. Скопируйте выделенный фрагмент в буфер обмена, нажав сочетание клавиш <Ctrl>+<C> (или <Ctrl>+<Ins>).
4. Поместите курсор в конец последней строки с текстом макроса и нажмите клавишу <Enter>, чтобы создать дополнительную пустую строку, на которую автоматически переместится курсор (обычное действие, выполняемое при работе с документами Microsoft Word).
5. Вставьте содержимое буфера обмена (нажав сочетания клавиш <Ctrl>+ +<V> или <Shift>+<Ins>), скопированное в него на шаге 3, в строчку, в которой располагается теперь курсор.

   В результате выполнения этого действия будет получено два полностью идентичных макроса, текст которых ограничивается с обеих сторон сплошными линиями (их вставка осуществляется редактором Visual Basic автоматически).

6. Измените название полученного таким образом нового макроса (оно находится в первой строке клона), записав, к примеру, `ShowVisualBasicPanel` (т. е. показать панель Visual Basic) вместо `ShowMyButtonsPanel`.
7. Замените в строке с основным текстом программы (располагается непосредственно над последней строчкой с выражением **`End Sub`**) имя панели инструментов `MyButtons` (записано в кавычках) на имя новой панели, `Visual Basic`, которая будет выводиться на экран с помощью создаваемого нами макроса.

В результате будет получен новый макрос, текст которого приведен в листинге 6.2 (без строк комментариев с описанием макроса).

Листинг 6.2

```
Sub ShowVisualBasicPanel ()
    CommandBars("Visual Basic").Visible = True
End Sub
```

8. Закройте редактор Visual Basic, выбрав в меню **File** (Файл) команду **Close and Return to Microsoft Word** (Закрыть и вернуться в Microsoft Word) или нажав сочетание клавиш <Alt>+<Q>.

> ***Примечание***
>
> Программа Microsoft Visual Basic, интегрированная в Microsoft Word, не была локализована, т. е. не переведена на русский язык. Поэтому названия всех команд и меню приводятся в ней на английском языке.

Осталось теперь назначить созданному нами макросу кнопку или сочетание клавиш — после чего его можно будет использовать для быстрого вывода на экран панели **Visual Basic**. Думаем, с этой задачей вы легко справитесь самостоятельно.

Поздравляем, одним "программистом" стало больше! Как мы и обещали, описанные выше действия по созданию нового макроса не имеют ничего общего с программированием и выполняются с помощью обычных операций Microsoft Word — выделения фрагмента текста, его копирования и вставки через буфер обмена, редактирования полученной копии.

> ***Предупреждение***
>
> Внимание! Напоминаем, что если при выходе из Microsoft Word вам будет предложено сохранить шаблон Normal.dot, обязательно сделайте это, иначе вся ваша работа будет потеряна.

6.3.2. Закладки на любой вкус

В этом разделе мы займемся клонированием макроса **Marker**, вставляющего закладку в текущую позицию курсора. Программа, соответствующая данному макросу, имеет несколько более сложный вид, чем в случае макроса **ShowMyButtonsPanel**. Это позволит нам сделать еще один шаг на пути знакомства с программами, создаваемыми редактором Visual Basic.

Занимаясь созданием макроса **Marker** в первой главе книги *(см. разд. 1.1.2)*, мы уже отмечали, что при работе с документом полезно иметь несколько стандартных закладок (например, **Marker_2**, **Return_and_Correct** и т. п.), которые можно было бы быстро вставлять с помощью макросов, нажимая соответствующие сочетания клавиш или специальные кнопки.

Попробуем создать один из таких макросов, воспользовавшись описанным выше приемом клонирования макроса-прототипа (для простоты изложения мы будем заниматься клонированием макроса, вставляющего обыкновенную, "нецветную" закладку).

Войдем для этого в редактор Visual Basic, выбрав макрос **Marker** в диалоговом окне **Макрос** и нажав кнопку **Изменить**.

Макрос **Marker** является первым из созданных нами макросов. Если в используемом вами приложении Microsoft Word макросы раньше никогда не создавались, то **Marker** будет находиться в самом начале сводного списка макросов, отображаемых в окне редактора Visual Basic. Поэтому сплошная линия, отделяющая макрос **Marker** от текста других макросов, будет располагаться только в его конце (листинг 6.3). Текст макроса **Marker** выглядит следующим образом.

Листинг 6.3

```
Sub Marker()
'
' Marker Макрос
' Макрос записан 01.08.2005 Пользователь Вставка закладки Marker
'
    With ActiveDocument.Bookmarks
        .Add Range:=Selection.Range, Name:="Marker"
        .DefaultSorting = wdSortByName
        .ShowHidden = False
    End With
End Sub
```

Создайте клон макроса **Marker** с помощью процедуры, описанной в предыдущем разделе. Как и в случае макроса **ShowMyButtonsPanel**, несложно догадаться, что для придания клону индивидуальных черт нужно будет, во-первых, дать ему новое имя и, во-вторых, изменить в тексте макроса имя закладки, вставляемой с его помощью.

Поэтому измените название "клонированного" макроса (оно находится в первой строке клона), записав вместо слова `Marker`, например, выражение `Marker_2`.

Просмотрев текст макроса, вы без труда обнаружите в нем то место, где задается имя закладки. Это особенно легко будет сделать, если вы знаете, что слову "имя" соответствует английское "name". В исходном макросе закладке присваивается имя **Marker**, и, как видно из распечатки текста программы, это имя встречается в ней только один раз (не считая заголовка) и входит в состав выражения `Name:="Marker"`, которое располагается в третьей строке основного текста макроса (если считать без учета текста комментариев с выходными данными макроса). Этих сведений нам будет вполне достаточно для создания нового макроса. Все остальные слова и выражения в тексте программы, вообще говоря, не имеют для нас особого значения — по крайней мере, на данной стадии знакомства с редактором Visual Basic. Объясняется это тем, что, как отмечалось выше, помимо названия макроса, мы должны будем изменить только имя вставляемой закладки. Весь остальной текст макроса ДОЛЖЕН остаться без изменения.

> ***Примечание***
>
> Для особо любознательных читателей приведем перевод других выражений, входящих в текст программы макроса. Здесь `ActiveDocument.Bookmarks` означает "Закладки в активном документе"; `Add` — добавить; `Range` — диапазон; `Selection` — выделение; `DefaultSorting` — сортировка по умолчанию; `wdSortByName` — сортировка по имени (`wd` — указывает, что действие выполняется в Microsoft Word); `ShowHidden` — показать скрытое; `False` — ложь. Таким образом, очевидно, что с помощью данного макроса осуществляется вставка закладки в активный документ. Правда, не очень понятно, при чем здесь диапазон, выделение, сортировка и отображение скрытого. Не вдаваясь в подробности, скажем только, что с помощью подобных выражений в большинстве случаев описываются условия, в которых протекал процесс записи макроса. (Кстати, смысл некоторых из приведенных выше выражений станет более понятным, если вы откроете диалоговое окно **Закладка** и посмотрите на представленные в нем параметры.) Исчерпывающая интерпретация этих и других подобных выражений приводится в следующих частях книги.

Введите новое имя закладки, например, `Important` (т. е. "важно"), записав его вместо слова Marker в выражении `Name:="Marker"`.

В результате будет получен новый макрос, текст которого (без строк с комментариями) приводится в листинге 6.4 (курсивом в нем выделены изменения, внесенные нами в макрос-прототип).

Листинг 6.4

```
Sub Marker_2()
    With ActiveDocument.Bookmarks
        .Add Range:=Selection.Range, Name:="Important"
        .DefaultSorting = wdSortByName
        .ShowHidden = False
    End With
End Sub
```

После завершения внесения изменений закройте редактор Visual Basic.

Для закрепления подхода к созданию однотипных макросов, который описывался в этом разделе, попробуйте самостоятельно создать с его помощью макрос, позволяющий мгновенно перейти к закладке **Important** или **Return_and_Correct** (если таковой не был ранее вами создан). Отметим, что в качестве прототипа в этом случае может использоваться макрос **GoToMarker**, создание которого описывалось в первой главе книги *(см. разд. 1.1.2)*.

6.4. Макрос, от которого, по слухам, произошли литературные редакторы

В предыдущем разделе для создания новых макросов нами был использован подход, состоящий в "клонировании" исходного макроса-прототипа. Попробуем теперь заняться клонированием *блока* макроса, описывающего конкретное отдельное действие, осуществляемое с помощью соответствующей программы. Эта задача является несколько более сложной, поскольку при получении копии целого макроса нужно было просто выделить и скопировать через буфер обмена весь его текст, заключенный между двумя сплошными линиями. При копировании же отдельного блока макроса мы должны будем самостоятельно определить его границы — в какой строке программы он начинается и где заканчивается.

В качестве прототипа мы предлагаем на этот раз взять макрос **ColorTermSeeTab**, который был описан нами в четвертой главе книги *(см. разд. 4.1.3)*. Напоминаем, что с его помощью выполнялось единовременное подкрашивание желтым цветом всех выражений "см. табл.", благо-

даря чему их можно было легко заметить при просмотре документа. Как уже отмечалось, этот макрос имеет "штучный" характер и может оказаться действительно полезным только в случае, если вам приходится достаточно часто иметь дело с документами, в которых встречается данное выражение. Понятно, что такое счастье может привалить далеко не каждому пользователю.

6.4.1. Краткий курс стилистики русского языка

В этом разделе мы покажем, каким образом макрос **ColorTermSeeTab** можно приспособить для решения проблемы, с которой сталкивается практически любой пользователь, занимающийся подготовкой и редактированием текстовых документов. Речь идет ни больше ни меньше как о литературной правке текста. Мы научим наш макрос подкрашивать "паразитные" слова и выражения, которыми большинство из нас так любит злоупотреблять при письме (имеется в виду, конечно, только нормативная лексика). У каждого "писателя", наверное, будет свой собственный список подобных выражений, однако такие слова, как "чтобы", "который", "это", "что", пользуются поистине всенародной любовью.

Как известно, в соответствии с правилами стилистики русского литературного языка не следует слишком часто употреблять в тексте одни и те же слова и выражения (и уж наверняка они не должны повторяться в пределах одного и того же предложения) — для этого существуют синонимы. Весь вопрос в том — как научить макрос отслеживать подобные нарушения стиля. Можно, конечно, создать для каждого "потенциально опасного" выражения свой отдельный макрос (например, с помощью описанного выше метода клонирования). Тогда после подготовки документа необходимо будет запускать по очереди все эти макросы для выделения в тексте всех вхождений соответствующих слов и выражений. Однако мы пойдем по другому пути: создадим один универсальный макрос, запуск которого позволит сразу же выявить указанные выше нарушения литературного стиля. Более того, этот макрос можно будет регулярно обновлять и адаптировать с учетом тематики редактируемых документов и особенностей лексикона их авторов.

Суть предлагаемого подхода состоит в следующем. При создании нашего нового макроса мы просто скопируем нужное число раз (по числу "слов-паразитов") тот фрагмент макроса-прототипа, с помощью которого реализуется соответствующая операция поиска с заменой. При этом в полученном "клоне" нужно будет поменять только объект поиска (т. е. записать вместо него одно из тех выражений, частоту употребления которых мы собираемся отслеживать). Напоминаем еще раз, что в макросе **ColorTermSeeTab** осуществлялось с помощью операций поиска с заменой подкрашивание желтым

цветом выражений "см. табл." (при этом в поле **Найти** диалогового окна **Найти и заменить** указывалось выражение "см. табл.", а в поле **Заменить на** — только атрибут **Выделение цветом**).

Ситуация

Итак, рассмотрим следующую ситуацию: по заданию начальства вы срочно подготовили некий документ. Составив его, вы хотите полученный текст немного "причесать", чтобы избавиться от наиболее очевидных стилистических ляпов. Времени, как всегда, в обрез. Через 20 минут документ должен быть представлен руководству.

Что требуется получить

Вы нажимаете кнопку на панели инструментов, запускающую макрос, подкрашивающий слова, которыми вы любите злоупотреблять. Прокручивая на экране текст, обработанный с помощью этого макроса (для скорости можно даже несколько уменьшить масштаб отображения документа), вы обращаете внимание только на те фрагменты текста, где рядом друг с другом располагаются 2—3 подкрашенные "лингвистические единицы". И сразу же с ними "разбираетесь": например, пару "этот—этим" разбавляете словом "данный", а "использование—используется" — словом "применяется" и т. п.

Способ решения

Подкрашивание "слов-паразитов" осуществляется с помощью макроса, аналогичного нашему макросу **ColorTermSeeTab**. От своего прототипа он отличается только тем, что операция поиска с подкрашиванием повторяется в нем несколько раз (по числу слов, занесенных в "черный список").

Решение задачи

Прежде чем приступать к созданию такого макроса (для определенности назовем его **Pencraft**, т. е. "литературный стиль"), необходимо иметь под рукой упомянутый выше "черный список". Как уже отмечалось, у каждого пользователя он может иметь индивидуальный характер. Поэтому в качестве примера приведем его фрагмент, составленный (для себя) одним из авторов настоящей книги. Он честно признал, что злоупотребляет на письме следующими словами и выражениями: "чтобы", "который", "что", "с помощью", "это", "данный", "можно", "предоставлять", "поддерживать", "применять", "обеспечивать", "позволять", "использовать", "осуществлять", "выполнять" (читатели этой книги смогут, наверное, без труда продолжить этот список).

Располагая таким или подобным ему списком, можно приступать непосредственно к созданию макроса. Для этого нужно с помощью процедур, описанных в предыдущих разделах, перейти в редактор Visual Basic, а именно —

в то его место, где располагается текст макроса **ColorTermSeeTab**. Затем следует выполнить клонирование данного макроса, чтобы получить "заготовку", на основе которой будет создаваться наш макрос **Pencraft**. Далее в полученном клоне нужно будет выполнить клонирование операции, с помощью которой осуществляется подкрашивание слов из "черного списка".

Приведем теперь текст макроса **ColorTermSeeTab** (опустив строки комментариев), созданный нами ранее в *гл. 4* книги (см. листинг 6.5).

Листинг 6.5

```
Sub ColorTermSeeTab()
    Options.DefaultHighlightColorIndex = wdYellow
    Selection.Find.ClearFormatting
    Selection.Find.Replacement.ClearFormatting
    Selection.Find.Replacement.Highlight = True
    With Selection.Find
        .Text = "см. табл."
        .Replacement.Text = ""
        .Forward = True
        .Wrap = wdFindContinue
        .Format = True
        .MatchCase = False
        .MatchWholeWord = False
        .MatchWildcards = False
        .MatchSoundsLike = False
        .MatchAllWordForms = False
    End With
    Selection.Find.Execute Replace:=wdReplaceAll
End Sub
```

Возможны, вообще говоря, два подхода к созданию нашего нового макроса: пожертвовать уже имеющимся, но малополезным макросом и внести изменения непосредственно в его текст или же выполнить клонирование макроса-прототипа и продолжить работу с полученной таким образом копией. Каким из них воспользоваться — решайте сами.

В первой строке макроса **ColorTermSeeTab** (если вы решили им пожертвовать) или его клона, в которой указывается имя макроса, вместо `ColorTermSeeTab` запишем `Pencraft`. Макрос **ColorTermSeeTab**, как нам известно, позволяет выполнить поиск выражения "см. табл.", которое, вообще

говоря, не имеет прямого отношения к решаемой нами задаче — правке литературного стиля. Поэтому заменим это выражение в строке `.Text = "см. табл."` каким-либо другим словом или выражением, более уместным для нашего случая (к примеру, словом "чтобы").

После этого мы переходим ко второму этапу создания макроса **Pencraft** — клонированию операции поиска с заменой.

6.4.2. Горизонты клонирования: каждый получит столько рук, ног и голов, сколько захочет

В этом разделе мы продолжим совершенствовать свое мастерство в области клонирования макросов. От клонирования макросов в целом мы перейдем к клонированию их отдельных "органов" — т. е. блоков программы, с помощью которых реализуются те или иные конкретные операции (например, поиск с заменой или задание цвета выделения).

Напомним, что с помощью макроса **ColorTermSeeTab** выполняются две основных операции. Во-первых, открывается инструмент **Выделение цветом** (кнопка  на панели инструментов), в котором задается желтый цвет для подкрашивания искомого выражения "см. табл.". Во-вторых, осуществляется операция поиска с заменой, в процессе которой, собственно, и выполняется подкрашивание данного выражения желтым цветом.

Выделить эти две операции в теле макроса нам опять же поможет знание английского языка. Ключевым словом при задании цвета для подкрашивания фрагмента текста в нашем случае является слово "желтый", по-английски — "yellow". В тексте макроса оно встречается только во второй его строке — сразу же после заголовка (без учета комментариев с выходными данными макроса):

```
Options.DefaultHighlightColorIndex = wdYellow
```

Приведем перевод и других слов, представленных в этой строке: `Options` можно перевести как "параметры", `Default` — "по умолчанию", `Highlight` — "выделение, подсвечивание, подкрашивание", `Color` — "цвет", `Index` — "признак, указатель". (Частица `wd` в выражении `wdYellow` указывает на то, что оно используется в среде приложения Microsoft Word.) Таким образом, если попытаться сложить эти слова в осмысленную фразу, то можно получить что-то вроде "Значением параметра «Признак выделения цветом» по умолчанию является желтый цвет".

В свою очередь, очевидно, что в операции поиска с заменой должны присутствовать англоязычные выражения, с помощью которых обозначаются соответствующие действия: например, "find" или "search" (найти) и "replace"

или "change" (заменить). Если просмотреть текст макроса, то можно заметить, что два из приведенных выражений, а именно "find" и "replace", встречаются практически во всех его строках, начиная с третьей и заканчивая предпоследней (при этом слово "replace" представлено также формой "replacement", т. е. "замена"). В этих строках приводятся и многие другие английские слова, перевод которых мы указывать не будем (желающие могут посмотреть их словаре-указателе в конце книги), отметим только, что, как и в случае описанного выше макроса **Marker_2**, с их помощью характеризуются в основном условия, в которых протекал процесс записи макроса, — они определяются, в частности, параметрами, задаваемыми в диалоговом окне **Найти и заменить**. Таким образом, можно предположить, что процедура поиска с заменой описывается в теле макроса, начиная с его третьей строки (без учета комментариев) и заканчивая предпоследней строкой. И это действительно так.

> ***Примечание***
>
> Конечно же, выделение отдельных блоков программы описанным способом основано в значительной степени на интуиции и не всегда может обеспечить получение желаемого результата. Чтобы избежать подобного рода неудач, советуем вам ознакомиться с материалом последующих частей книги.

Покажем теперь, как выполняется клонирование отдельного блока в тексте макроса. В нашем случае это будет операция поиска с заменой. Данная операция, как было показано ранее, реализуется блоком макроса **Pencraft**, начиная с его третьей строки (без учета комментариев) и заканчивая предпоследней строкой. Выделите данный фрагмент в тексте программы и скопируйте его в буфер обмена, нажав сочетание клавиш <Ctrl>+<C> (или <Ctrl>+<Ins>). Далее поместите курсор в конец предпоследней строки и нажмите клавишу <Enter> (ее можно нажать и два раза, чтобы создать дополнительную пустую строку, визуально отделяющую клон операции от его прототипа). После этого нужно нажать сочетание клавиш <Ctrl>+<V> (или <Shift>+<Ins>), чтобы вставить фрагмент текста макроса, скопированный ранее в буфер обмена.

Приступим теперь к третьему этапу создания макроса **Pencraft**. Для этого в полученный только что клон операции поиска с заменой мы впишем новый фрагмент текста, поиск которого будет осуществляться данным блоком макроса. Иными словами, в строке:

```
.Text = "чтобы"
```

заменим слово "чтобы" на какое-либо другое популярное выражение из черного списка — например, слово "который". При этом следует учесть, что окончание данного слова может в тексте меняться, поэтому необходимо указать только его корневую, неизменную часть, а именно — "котор".

Совет

Чтобы при просмотре текста макроса сразу было понятно, для чего именно предназначен тот или иной блок макроса, можно создать перед каждым из них пустую строку и записать в нее комментарий (после знака апострофа "'"), поясняющий назначение данного блока.

В результате описанных ранее действий и с учетом приведенного замечания текст создаваемого нами макроса примет вид, представленный в листинге 6.6.

Листинг 6.6

```
Sub Pencraft()
' Задание желтого цвета выделения
    Options.DefaultHighlightColorIndex = wdYellow

' Поиск и подкрашивание слова "чтобы"
    Selection.Find.ClearFormatting
    Selection.Find.Replacement.ClearFormatting
    Selection.Find.Replacement.Highlight = True
    With Selection.Find
        .Text = "чтобы"
        .Replacement.Text = ""
        .Forward = True
        .Wrap = wdFindContinue
        .Format = True
        .MatchCase = False
        .MatchWholeWord = False
        .MatchWildcards = False
        .MatchSoundsLike = False
        .MatchAllWordForms = False
    End With
    Selection.Find.Execute Replace:=wdReplaceAll

' Поиск и подкрашивание корня "котор"
    Selection.Find.ClearFormatting
    Selection.Find.Replacement.ClearFormatting
    Selection.Find.Replacement.Highlight = True
    With Selection.Find
```

```
        .Text = "котор"
        .Replacement.Text = ""
        .Forward = True
        .Wrap = wdFindContinue
        .Format = True
        .MatchCase = False
        .MatchWholeWord = False
        .MatchWildcards = False
        .MatchSoundsLike = False
        .MatchAllWordForms = False
    End With
    Selection.Find.Execute Replace:=wdReplaceAll
End Sub
```

Чтобы завершить создание макроса **Pencraft**, необходимо повторить описанную ранее процедуру клонирования операции поиска с заменой и внесения изменений в искомое выражение ровно столько раз, сколько слов и выражений входит в ваш черный список. После этого можно закончить процедуру создания макроса и выйти из редактора Visual Basic. Полученному макросу можно задать кнопку, с помощью которой будет осуществляться его запуск.

Подведем теперь некоторые итоги. Знакомясь с материалом данной главы, вы получили общее представление о структуре и основных компонентах программ и возможностях их редактирования. Вы научились также клонировать как макросы в целом, так и их отдельные блоки, с помощью которых реализуются те или иные конкретные операции. В принципе, на этом можно было бы и закончить нашу экскурсию на фабрику макросов. Однако давайте не будем спешить и познакомимся с некоторыми другими возможностями редактора Visual Basic. Это позволит вам закрепить прочитанный материал и, возможно, более оптимистично подойти к оценке своих перспектив, связанных с освоением основ программирования.

6.4.3. Visual Basic: "Дорогой пользователь, тебе помочь?"

В данном разделе мы займемся клонированием еще одной операции, представленной в тексте нашего макроса, — заданием цвета выделения. На этот раз нам повезло несколько больше, чем в предыдущем случае: данная операция описывается с помощью всего лишь одной строки программы:

```
Options.DefaultHighlightColorIndex = wdYellow
```

Примечание

Отметим, что клонирование двух указанных операций — задания цвета и поиска выражений из черного списка — можно было бы совместить. Однако мы этого не стали делать, чтобы более наглядно показать возможности клонирования отдельных операций в границах одного и того же макроса.

Напомним, что в нашем случае цветом выделения является желтый (yellow), с помощью которого в исходном макросе подкрашивалось выражение "см. табл.". Соответственно и в нашем макросе **Pencraft** все слова и выражения, включенные в черный список, также будут подкрашиваться желтым цветом. Согласитесь, что это не очень удобно — в тексте документа указанные выражения было бы гораздо проще отличать друг от друга, если бы они подкрашивались разными цветами. Например, все слова "чтобы" выделялись желтым цветом, "который" — зеленым, "что" — синим и т. д.

Опыт, накопленный при чтении этой главы, подсказывает: чтобы реализовать данный подход на практике, необходимо несколько раз (по числу слов и выражений, включенных в черный список) скопировать операцию задания цвета, вставив ее непосредственно перед каждой операцией поиска с заменой. Затем во все полученные клоны следует внести изменения, касающиеся цвета выделения.

Как выполняется клонирование отдельной операции, нам уже известно, однако задавать конкретные значения параметров инструкций программы (в нашем случае цвета выделения) нам еще не приходилось. Попробуем опять исходить из того, что Visual Basic "говорит" по-английски. Поэтому смело возьмем в руки русско-английский словарь и вместо слова `Yellow` в выражении `wdYellow` запишем в полученных клонах соответственно `Green` (зеленый), `Blue` (синий) и т. п. Вы не поверите, но наш авантюрный подход в случае с зеленым и синим цветом, действительно, сработает. К сожалению, с большинством других цветов дело обстоит не так просто, и нет гарантии, что при их задании нам так же повезет. Что же делать? Предлагаем обратиться за помощью к самому редактору Visual Basic. И это не шутка.

Редактор Visual Basic, действительно, очень дружелюбно относится к пользователю. Об этом более подробно будет говориться в последующих главах книги, здесь же мы воспользуемся конкретной ситуацией и покажем, каким образом Visual Basic поможет решить стоящую перед нами задачу — задание конкретного цвета выделения.

Итак, займемся вначале клонированием операции задания цвета, которым будут подкрашиваться слова и выражения, включенные в черный список. Для этого скопируем в буфер обмена вторую строку создаваемого нами макроса:

```
Options.DefaultHighlightColorIndex = wdYellow
```

и вставим ее в начало каждой операции поиска с заменой в клонированных нами ранее блоках макроса, а именно перед строкой:

```
Selection.Find.ClearFormatting
```

После этого приступим к изменению цвета выделения, задаваемого в этой строке.

Напомним, что цвет выделения определяется с помощью инструмента **Выделение цветом**. Чтобы вывести на экран соответствующую палитру цветов, необходимо нажать кнопку  с изображением маленького треугольника, расположенную правее кнопки выделения цветом на панели инструментов **Форматирование** (рис. 6.3).

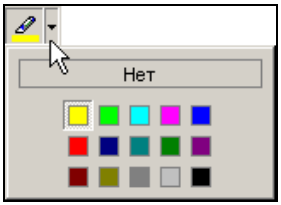


Рис. 6.3. Палитра цветов, выводимая с помощью инструмента **Выделение цветом**

Как видно из рисунка, в палитре данного инструмента имеется 16 цветов, представленных соответствующими цветными ячейками (включая "отсутствие цвета", соответствующее ячейке **Нет**).

Перечислим названия цветов палитры инструмента **Выделение цветом**, присвоенные им в Visual Basic (в порядке английского алфавита):

- `wdBlack` — черный;
- `wdBlue` — синий;
- `wdBrightGreen` — ярко-зеленый;
- `wdDarkBlue` — темно-синий;
- `wdDarkRed` — темно-красный;
- `wdDarkYellow` — коричнево-зеленый;
- `wdGray25` — серый 25%;
- `wdGray50` — серый 50%;
- `WdGreen` — зеленый;
- `wdNoHighlight` — не выделено;
- `wdPink` — лиловый;

- `wdRed` — красный;
- `wdTeal` — сине-зеленый;
- `wdTurquoise` — бирюзовый;
- `wdViolet` — фиолетовый;
- `wdWhite` — белый;
- `wdYellow` — желтый.

> ***Примечание***
>
> Если вы внимательно подсчитаете, сколько элементов содержится в приведенном списке цветов палитры, то их окажется не шестнадцать, а семнадцать. Дело в том, что "белый цвет (`wdWhite`) не может быть задан в инструменте **Выделение цветом** "вручную", т. е. посредством выбора соответствующей ячейки в палитре цветов. Чтобы подкрасить выделяемые фрагменты документа этим "дополнительным" белым цветом, необходимо задать его "принудительным" образом, внеся соответствующие изменения непосредственно в текст программы. Впрочем, и в этом случае фрагменты текста, подкрашенные белым цветом, будут заметны только в документах, цвет фона которых отличается от белого. (Напоминаем, что цвет фона окна документа задается непосредственно в системных настройках Windows: **Пуск | Настройка | Панель управления | Экран | Оформление**.)

С учетом приведенного примечания очевидно, что для подкрашивания фрагментов текста в нашем распоряжении имеется пятнадцать цветов (а если ваша система Windows настроена таким образом, что цвет фона документов в ней отличен от белого — шестнадцать). Это, конечно, налагает определенные ограничения на длину черного списка. Чтобы изменить цвет выделения слова или выражения, включенного в черный список, необходимо скорректировать каждую из только что клонированных нами строк-операций, указав в них вместо желтого (`wdYellow`) любой другой цвет, например, ярко-зеленый (`wdBrightGreen`) для подкрашивания корня слова "котор", бирюзовый (`wdTurquoise`) для подкрашивания слова "что" и т. п. Для этого нужно, во-первых, знать, что задаваемый нами цвет действительно представлен в палитре инструмента **Выделение цветом**, во-вторых, необходимо помнить, как обозначается значение соответствующего параметра в редакторе Visual Basic, и, в-третьих, не допустить при записи этого значения "орфографических" ошибок. Не слишком ли много требований для первого знакомства с Visual Basic?

Не пугайтесь, мы же обещали, что добрый редактор Visual Basic придет к нам на помощь. Дело в том, что во многих случаях в нем "автоматизирована" процедура выбора и задания значений параметров для инструкций программы. Вот и в нашем случае Visual Basic предложит нам набор, включающий все 17 значений параметра, определяющего цвет выделения, из ко-

торых можно будет сделать нужный нам выбор. Для этого необходимо выполнить следующие действия:

1. В строке, в которой задается цвет выделения:

   ```
   Options.DefaultHighlightColorIndex = wdYellow
   ```

   нужно удалить всю ее правую часть, включая знак равенства.

2. Затем следует повторно ввести знак равенства после выражения `DefaultHighlightColorIndex`.

   При этом — внимание! — редактор Visual Basic автоматически выведет на экран список, содержащий перечень всех указанных ранее цветов.

3. Прокрутив список, следует выбрать нужный цвет.

Как видите, все просто и удобно — и память не нужно напрягать, и можно больше не опасаться ошибок "ручного" ввода. Повторите описанную выше процедуру для каждой из клонированных ранее операций задания цвета выделения. В результате текст программы нашего макроса примет вид, представленный в листинге 6.7 (с точностью до выбранного вами конкретного цвета выделения).

Листинг 6.7

```
Sub Pencraft()
' Задание желтого цвета выделения
    Options.DefaultHighlightColorIndex = wdYellow

' Поиск и подкрашивание слова "чтобы"
    Selection.Find.ClearFormatting
    Selection.Find.Replacement.ClearFormatting
    Selection.Find.Replacement.Highlight = True
    With Selection.Find
        .Text = "чтобы"
        .Replacement.Text = ""
        .Forward = True
        .Wrap = wdFindContinue
        .Format = True
        .MatchCase = False
        .MatchWholeWord = False
        .MatchWildcards = False
        .MatchSoundsLike = False
        .MatchAllWordForms = False
```

```
    End With
    Selection.Find.Execute Replace:=wdReplaceAll

' Задание ярко-зеленого цвета выделения
    Options.DefaultHighlightColorIndex = wdBrightGreen

' Поиск и подкрашивание корня "котор"
    Selection.Find.ClearFormatting
    Selection.Find.Replacement.ClearFormatting
    Selection.Find.Replacement.Highlight = True
    With Selection.Find
        .Text = "котор"
        .Replacement.Text = ""
        .Forward = True
        .Wrap = wdFindContinue
        .Format = True
        .MatchCase = False
        .MatchWholeWord = False
        .MatchWildcards = False
        .MatchSoundsLike = False
        .MatchAllWordForms = False
    End With
    Selection.Find.Execute Replace:=wdReplaceAll

' Задание бирюзового цвета выделения
    Options.DefaultHighlightColorIndex = wdTurquoise

' Поиск и подкрашивание слова "что"
    Selection.Find.ClearFormatting
    Selection.Find.Replacement.ClearFormatting
    Selection.Find.Replacement.Highlight = True
    With Selection.Find
        .Text = "что"
        .Replacement.Text = ""
        .Forward = True
        .Wrap = wdFindContinue
        .Format = True
```

```
        .MatchCase = False
        .MatchWholeWord = False
        .MatchWildcards = False
        .MatchSoundsLike = False
        .MatchAllWordForms = False
    End With
    Selection.Find.Execute Replace:=wdReplaceAll

End Sub
```

Между прочим, в Visual Basic имеется довольно много средств автоматизации, подобных описанному ранее. С их помощью существенно облегчается труд программиста. Многие из них рассмотрены в последующих главах книги, мы же предлагаем вам продолжить чтение данной главы, чтобы закрепить прочитанный только что материал и еще раз убедиться в доброжелательном отношении редактора Visual Basic к пользователю.

6.4.4. Такие простые логичные истины

В приведенном ранее примере в наш черный список были включены следующие три выражения: "чтобы", "котор" и "что". При этом "чтобы" подкрашивается в процессе работы макроса желтым цветом, "котор" — ярко-зеленым, "что" — бирюзовым. Легко заметить, что в этом случае раскраска слова "чтобы" окажется пестрой — частица "что" будет выделена в нем бирюзовым цветом, а "бы" — желтым.

Дело в том, что выполнение операций в нашем макросе осуществляется последовательным образом. Поскольку слово "что" находится в конце нашего черного списка, его подкрашивание выполняется на заключительном этапе работы макроса. Однако в составе слова "чтобы", которое располагается в черном списке на первом месте, также имеется частица "что", поэтому и она окажется выделенной бирюзовым цветом. Чтобы избежать этого, необходимо было бы при поиске слова "что" задать признак **Только слово целиком**. Это можно было сделать, установив соответствующий флажок в диалоговом окне **Найти и заменить**. Однако в нашем макросе он не был установлен, так как при поиске выражения "см. табл." в макросе-прототипе этого не требовалось.

Примечание

Отметим, что необходимость поиска только целых слов не является для нашего макроса универсальным требованием. Зачастую при поиске слова из черного списка имеет смысл отбросить его окончание или приставку, которые могут принимать в тексте различные формы (например, "использовать", "использо-

вание", "пользоваться" и т. п.). Учитывая сказанное, вопрос об установке флажка **Только слово целиком** должен решаться в каждом конкретном случае индивидуально.

Сейчас мы покажем, что навыки, приобретенные нами при задании цвета выделения, позволят легко справиться и с этой задачей: мы попробуем изменить текст макроса таким образом, чтобы при поиске слова "что" во внимание принимались *только целые слова*. Еще раз подчеркнем, что для этого нам не потребуется каких-либо специальных знаний в области программирования, а только — здравый смысл и знакомство с основами английского языка (на уровне чтения со словарем).

Выпишем отдельно фрагмент, в котором реализуется операция поиска и подкрашивания слова "что" (листинг 6.8), и попробуем разобраться в смысле представленных в нем инструкций Visual Basic. Обратите внимание, что для удобства изложения строки блока нами пронумерованы.

Листинг 6.8

```
' Поиск и подкрашивание слова "что"
 1   Selection.Find.ClearFormatting
 2   Selection.Find.Replacement.ClearFormatting
 3   Selection.Find.Replacement.Highlight = True
 4   With Selection.Find
 5      .Text = "что"
 6      .Replacement.Text = ""
 7      .Forward = True
 8      .Wrap = wdFindContinue
 9      .Format = True
10     .MatchCase = False
11     .MatchWholeWord = False
12     .MatchWildcards = False
13     .MatchSoundsLike = False
14     .MatchAllWordForms = False
15  End With
16  Selection.Find.Execute Replace:=wdReplaceAll
```

Для начала напомним, что в данном блоке программы реализуются следующие действия (цвет подкрашивания задается вне этого блока): открывается диалоговое окно **Найти и заменить**, в поле **Найти** вводится искомый текст (слово "что"), в поле **Заменить на** задается признак **Выделение цветом** (с помощью кнопки **Формат**) и нажимается кнопка **Заменить все**.

Общая схема наших дальнейших действий должна включать следующие основные этапы. Сначала мы должны будем просмотреть английские слова и выражения, представленные в этом блоке программы, и попытаться определить, какие из них имеют непосредственное отношение к решению стоящей перед нами задачи. После этого нужно будет решить, каким образом следует изменить текст программы, чтобы в процессе поиска подкрашиваемого выражения во внимание принимались только целые слова. Возможность поиска целых слов, как известно, реализуется в Microsoft Word посредством установки флажка **Только слово целиком** в диалоговом окне **Найти и заменить** (рис. 6.4).

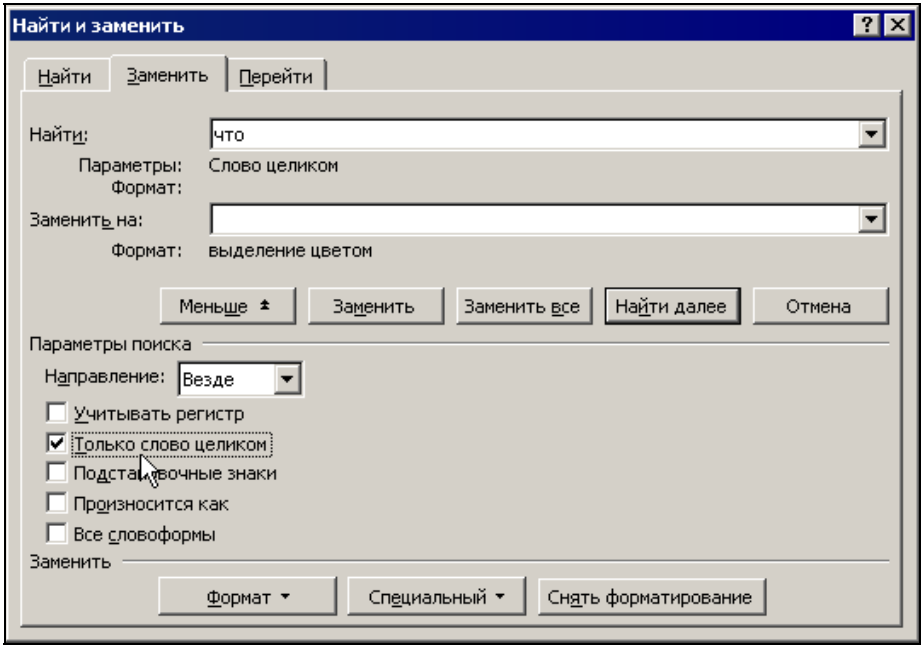


Рис. 6.4. Задание признака **Только слово целиком** в диалоговом окне **Найти и заменить**

При просмотре текста блока макроса, приведенного в листинге 6.8 (если у вас при этом возникнут затруднения, воспользуйтесь словарем или загляните в словарь-указатель в конце книги), вы легко заметите, что к решению нашей задачи имеет непосредственное отношение только одна инструкция, перевод имени которой на русский язык почти буквально совпадает с названием данного флажка. Эта инструкция, а именно `MatchWholeWord` (где `Match` можно перевести как "соответствовать", "совпадать"; `Whole` — как "це-

лое"; `Word` — "слово"), содержится в 11-й строке приведенного блока макроса:

```
.MatchWholeWord = False
```

Естественно предположить, что именно в этой строке и задается состояние интересующего нас флажка. Очевидно, что данное состояние определяется правой частью приведенного равенства. Напомним, что в рассматриваемом нами случае флажок **Только слово целиком** установлен не был. Поэтому естественно предположить, что этому состоянию флажка соответствует расположенное в данной строке слово **`False`** (ложь, ложный, ошибочный).

Читатели, знакомые с правилами записи логических выражений, наверное, уже сообразили, какие коррективы следует внести в текст программы, чтобы установить флажок **Только слово целиком** в процедуре поиска. Всем остальным мы предлагаем воспользоваться описанным в предыдущем разделе более надежным и безошибочным способом. Для этого удалите (непосредственно в тексте программы) всю правую часть приведенной 11-й строки, включая знак равенства. Затем снова введите знак равенства, после чего добрый редактор Visual Basic услужливо выведет на экран список значений, определяющих состояние флажка. Поскольку этих значений может быть только два (установлен, не установлен), то и список будет состоять только из двух элементов, а именно — **`False`** и **`True`**. Состоянию "не установлен", как нам известно, соответствовал элемент **`False`**, поэтому ошибиться в данном случае просто невозможно: чтобы в операции поиска во внимание принимались только целые слова, инструкции `MatchWholeWord` должно быть присвоено значение **`True`** (истина):

```
.MatchWholeWord = True
```

Чтобы обмениваться этими данными так, чтобы их можно было интерпретировать и использовать, эти стороны должны одинаковым образом использовать XML. Это возможно благодаря схемам, в которых язык XML используется для определения структуры документа, предназначенного для конкретного использования. Используя редактор Biztalk Editor аналитик определяет бизнес-процесс, разработчик задает реализацию этого процесса, связывая каждый шаг с приложениями и технологиями, реально запускающими этот процесс...

Рис. 6.5. Фрагмент документа, обработанный макросом **Pencraft**

Вот на этом можно, действительно, закончить создание нашего макроса **Pencraft** и закрыть редактор Visual Basic. Вновь созданному макросу можно назначить специальную кнопку, что облегчит его использование при выполнении предварительной литературной правки документов. На рис. 6.5 показан

фрагмент реального документа, обработанный макросом, аналогичным нашему **Pencraft**. Как видим, даже не читая сам документ, сразу же становится видно (в буквальном смысле слова), что писал его, мягко выражаясь, далеко не златоуст.

> ***Совет***
>
> Для удобства работы с макросом **Pencraft** советуем включить в текст программы список текстовых выражений, обрабатываемых с его помощью, а также указать, какие именно цвета используются для их выделения. Этот список можно записать в виде комментария, расположенного непосредственно после заголовка макроса. В дальнейшем по мере необходимости этот "черный список" можно корректировать, пополнять новыми словами и выражениями с учетом тематики особенностей редактируемого текста и лексикона его автора.

6.5. Что нового в этой главе

В этой главе вы познакомились с редактором Visual Basic — программой, встроенной в Microsoft Word, с помощью которой проводилась автоматическая запись всех макросов, создававшихся нами в предыдущих главах книги. Вы теперь знаете, как можно получить доступ к тексту любого конкретного макроса, научились находить его первую и последнюю строчки, записывать комментарии, не влияющие на ход выполнения программы. Более того, не владея навыками программирования, вы освоили приемы, позволяющие создавать новые макросы и модифицировать уже существующие посредством копирования ("клонирования") макросов-прототипов или их отдельных блоков и внесения в полученные копии необходимых изменений. С помощью данного метода нами были созданы следующие макросы-клоны:

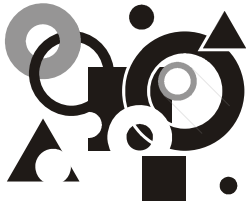
- макрос **ShowVisualBasicPanel**, выводящий на экран панель инструментов **Visual Basic** (подобно макросу **ShowMyButtonsPanel**, созданному нами ранее в *гл. 3*);
- макрос **Marker_2** (аналог макроса **Marker**, созданного нами в *гл. 1*), вставляющий в документ закладку с текстом `Important` (т. е. "важно");
- макрос **Pencraft**, с помощью которого может выполняться первичная литературная правка документа (в его основу был положен макрос **ColorTermSeeTab**, созданием которого мы занимались в *гл. 4*).

Вы узнали также, что редактор Visual Basic очень дружелюбно настроен по отношению к пользователю и может приходить к нему на помощь, подсказывая значения конкретных параметров.

Надеемся, что проведенная в этой главе ознакомительная экскурсия в сборочный цех макросов позволит вам, как уже отмечалось, более оптимистично подойти к оценке своих перспектив, связанных с освоением азов про-

граммирования. Во всяком случае, вы на практике убедились, что разобраться в общем смысле текста программы макроса при желании вполне возможно. Особенно полезным при этом оказывается знание английского языка. Данный вывод позволяет надеяться, что вам по силам окажутся и более конкретные вопросы теории и практики программирования, освоение которых позволит значительно повысить эффективность создаваемых макросов. Эти вопросы подробно рассмотрены во второй и третьей частях книги, содержание которых тесно связано с уже прочитанным вами материалом.

Авторы этой книги надеются, что освоить приемы создания макросов с использованием элементов программирования вам поможет "подарочный набор" с готовыми к применению макросами, листингами программ и т. п. Этот набор мы специально подготовили для наших читателей. О том, как можно им воспользоваться, рассказывается в следующей главе — заключительной главе первой части книги.



Глава 7

Написал макрос — поделись с друзьями

Понятие шаблона и виды шаблонов. Подключение шаблона. Шаблоны, модули и макросы. Вопросы безопасности при использовании макросов. Создание модуля, содержащего набор макросов, предназначенных для передачи другим пользователям. Создание файла-контейнера для модуля макросов. Копирование, переименование и удаление модулей и панелей. Подключение файла-контейнера к другому экземпляру Microsoft Word.

До сих пор в нашей книге речь шла о создании макросов, так сказать, индивидуального пользования. Иными словами, неявно предполагалось, что они будут применяться именно на том компьютере, на котором были созданы. Понятно, что и все преимущества от использования макросов в этом случае будут доставаться исключительно их создателю. Это конечно, хорошо, но может быть и лучше. Достоинства макроса реализуются в полной мере только в том случае, когда им могут воспользоваться и другие пользователи. Поэтому в этой, заключительной главе первой части книги, мы решили рассказать вам о том, каким образом можно поделиться результатами своего "макротворчества" с друзьями, коллегами по работе и всеми другими потенциальными пользователями (может быть, даже получив при этом и некоторую коммерческую выгоду).

Сделать это мы обязаны и по той причине, что хотим объяснить вам, как можно воспользоваться нашим "подарочным набором" с готовыми к применению макросами, о котором говорилось в конце предыдущей главы.

7.1. Шаблоны и их подключение

Как уже отмечалось, наш "подарочный набор" состоит из двух шаблонов. В первом, который называется **MB.dot**, содержатся наиболее полезные, с нашей точки зрения, макросы, созданием которых мы с вами занимались и продолжим активно заниматься в этой книге. Кстати, большинство из них

нам еще предстоит создать в следующих главах. Поэтому если вы хотите поближе познакомиться с "кухней" их разработки, мы настоятельно рекомендуем продолжить чтение, чтобы освоить приемы, с помощью которых вы сможете создать для себя еще более полезные макросы, учитывающие специфику стоящих перед вами конкретных задач.

Второй шаблон называется **MarosBook.dot**. Он является "учебным" и содержит все листинги, приведенные на страницах книги. Так что, имея его в своем распоряжении, вам не нужно будет воспроизводить вручную учебные примеры, приводящиеся в следующих главах книги.

Получить наш подарочный набор можно в Интернете по одному из следующих адресов: **http://www.russianlocalization.com/mb/** или **http://bhv.ru/books/book.php?id=12623**. Чтобы воспользоваться им, нужно будет *подключить* соответствующие шаблоны. О том, как это можно сделать, и будет рассказано в этом разделе.

7.1.1. Общие сведения о шаблонах

Как не раз уже отмечалось, создаваемые макросы по умолчанию записываются в шаблон Normal.dot, в котором хранятся практически все настройки приложения Microsoft Word, установленного на данном компьютере (часть настроек, например, сведения о расположении панелей инструментов, хранятся в реестре Windows). После установки программы на компьютер эти настройки имеют стандартные значения, однако в дальнейшем пользователь их может изменить в соответствии со своими вкусами и предпочтениями. Более того, если на одном и том же компьютере работают несколько пользователей (каждый под своей учетной записью), у каждого из них будет иметься свой индивидуальный шаблон Normal.dot. Поэтому в каждом конкретном случае шаблон Normal.dot будет иметь индивидуальный характер.

В этой связи, думаем, уместно будет остановиться несколько более подробно на понятии и особенностях файлов-шаблонов.

Говоря упрощенно, *шаблоном* является файл Microsoft Word особого типа, в котором содержатся сведения о структуре основанного на нем документа, назначенных сочетаниях клавиш, макросах, пользовательских панелях (и изменениях, внесенных в стандартные панели), настройках меню, параметрах страницы, форматировании, используемых стилях и т. п.

Существует два вида шаблонов — общие шаблоны и шаблоны конкретных документов. *Общие шаблоны*, в том числе и шаблон Normal.dot (который мы так настойчиво призывали вас сохранять в предыдущих главах книги), содержат настройки, доступные для *всех* документов Microsoft Word.

Шаблон документа содержит настройки, доступные *только для документов*, основанных на этом шаблоне. В Microsoft Word имеются *встроенные* шабло-

ны документа, например, шаблоны отчетов, записок, факсов и т. п., доступ к которым можно получить в диалоговом окне **Создание документа** (меню **Файл | Создать**) — рис. 7.1. В Microsoft Word 2002 и Microsoft Office Word 2003 имеется возможность загружать разнообразные образцы шаблонов документов непосредственно из Интернета (естественно, при наличии соответствующего подключения).

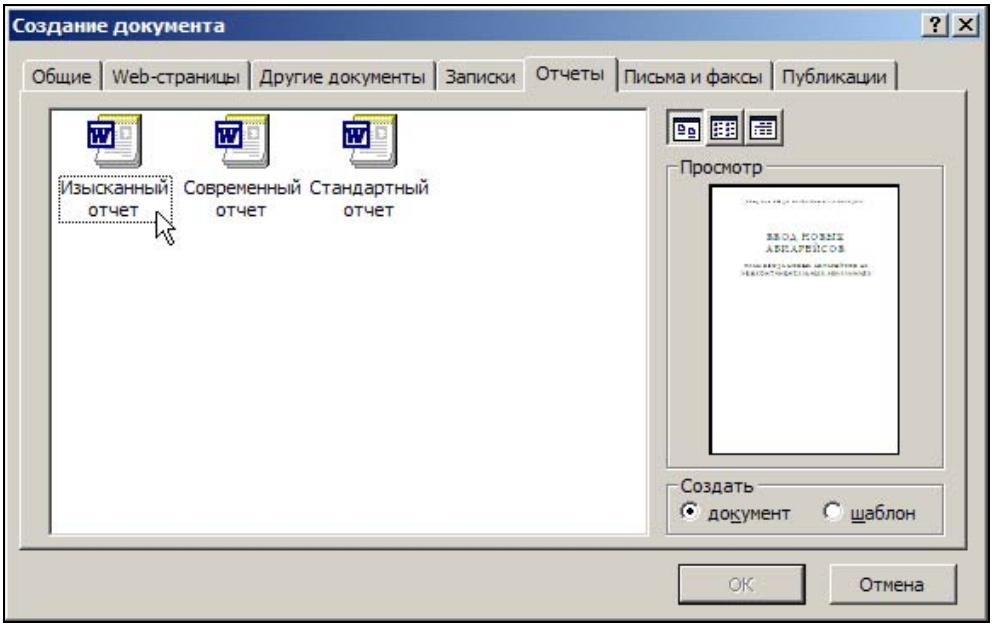


Рис. 7.1. Шаблоны документов, представленные на вкладке **Отчеты** диалогового окна **Создание документа**

Кроме того, пользователи Microsoft Word могут самостоятельно создавать шаблоны для своих документов. По умолчанию они сохраняются в папке **Шаблоны**, адрес которой можно посмотреть на вкладке **Расположение** диалогового окна **Параметры** (обычно это \Documents and Settings\Имя_Пользователя\Application Data\Microsoft\Шаблоны) — рис. 7.2. Там же находится и общий шаблон Normal.dot.

Примечание

Следует иметь в виду, что один и тот же шаблон может в зависимости от способа его подключения к документу использоваться и как общий шаблон, и как шаблон документа.

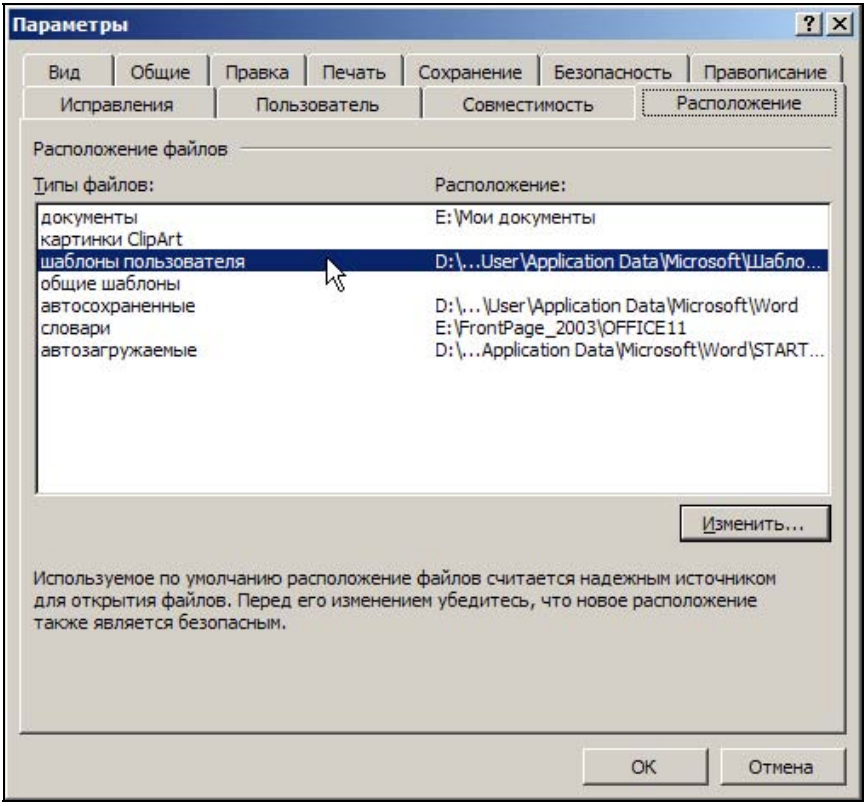


Рис. 7.2. Вкладка **Расположение** диалогового окна **Параметры** (здесь можно узнать и при желании изменить место хранения файлов различного типа)

7.1.2. Подключение шаблонов

Когда вы загружаете документ Microsoft Word, вам становятся доступны макросы, панели инструментов, сочетания клавиш и другие настройки, которые могут иметься: 1) в самом документе; 2) в шаблоне Normal.dot; 3) в загружаемых общих шаблонах (при наличии таковых); 4) в шаблоне документа (если таковой имеется).

Коротко опишем механизм, обеспечивающий реализацию этой возможности. При запуске Microsoft Word происходит автоматическая загрузка шаблона Normal.dot, а также всех шаблонов, находящихся в папке STARTUP, точнее в двух папках с именем STARTUP. Первая из этих папок имеет непосредственное отношение к программе Microsoft Word (ее местоположение можно узнать на вкладке **Расположение** диалогового окна **Параметры** —

см. рис. 7.2), вторая — к пакету Microsoft Office в целом (по умолчанию она располагается по адресу \Program Files\Microsoft Office\Office\Startup).

Все эти шаблоны являются по определению общими, и содержащиеся в них настройки становятся доступными для всех документов Microsoft Word.

Некоторые программы, например, широко известный электронный словарь Abbyy Lingvo, программа Adobe Acrobat и др., в процессе своей установки "встраивают" свои шаблоны в Microsoft Word. Эти шаблоны также автоматически загружаются в качестве общих при запуске Microsoft Word.

В Microsoft Word имеется еще один способ загрузки общих шаблонов (помимо их записи в папку STARTUP). Для этого в диалоговом окне **Шаблоны и надстройки** (доступ к которому осуществляется с помощью одноименной команды из меню **Сервис**) следует нажать кнопку **Добавить** и перейти в папку, в которой находится нужный вам шаблон, выбрать его и дважды нажать кнопку **ОК**. После этого в окне рядом с кнопкой **Добавить** появится название шаблона, слева от которого будет установлен флажок (рис. 7.3).

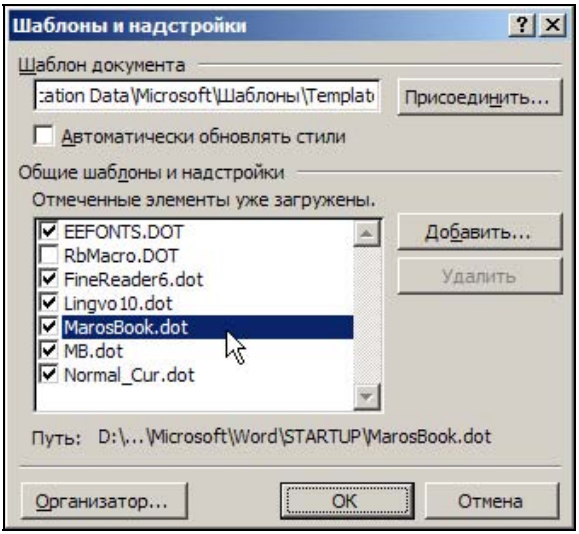


Рис. 7.3. Диалоговое окно **Шаблоны и надстройки**, в котором выполняется подключение шаблонов

Заметим, что в этом списке отображаются не только общие шаблоны, добавляемые с помощью только что описанной процедуры, но и шаблоны, автоматически загружаемые при запуске Microsoft Word. Эти шаблоны будут загружаться каждый раз при очередном запуске Microsoft Word. При этом шаблоны, находящиеся в папке STARTUP, будут загружаться автоматически и поэтому пользователю не придется устанавливать соответствующие им

флажки в диалоговом окне **Шаблоны и надстройки**. Общие шаблоны, вручную добавленные из других папок, придется активировать также вручную: щелкнуть нужный элемент в списке, чтобы установить флажок для данного шаблона.

Любой шаблон из этого списка можно отключить, сняв соответствующий ему флажок. При необходимости можно и удалить шаблон, выбрав его в списке и нажав кнопку **Удалить**. Этого нельзя, однако, сделать применительно к шаблонам, автоматически загружаемым (из папок STARTUP или других папок, в которых находятся "встраиваемые" в Microsoft Word шаблоны) — кнопка **Удалить** при выборе этих шаблонов остается недоступной.

Если вы захотите индивидуальным образом подключить к документу какой-либо конкретный шаблон, в диалоговом окне **Шаблоны и надстройки** следует нажать кнопку **Присоединить** (см. рис. 7.3), перейти в папку, в которой находится данный шаблон, выбрать его, нажать кнопку **Открыть**, а затем кнопку **ОК**.

Применительно к нашим шаблонам MB.dot и MarosBook.dot мы рекомендуем руководствоваться следующими принципами. Шаблон MB.dot лучше загрузить в качестве общего, поэтому для его подключения можно воспользоваться одним из описанных выше двух способов — либо сразу же поместить в папку STARTUP (в этом случае он будет загружаться автоматически), либо добавить его в список общих шаблонов "вручную". Каким из этих способов воспользоваться, вы должны решить самостоятельно. Шаблон MarosBook.dot предназначен для активной работы с текстами листингов. Чтобы можно было их смотреть и изменять, а не только выполнять "как есть", его следует присоединить как шаблон документа, на котором вы будете проверять работу макросов. Возможен и другой подход: открыть шаблон MarosBook.dot точно так же, как вы открываете любой другой документ, и работать с ним только при чтении этой книги.

> ***Полезные сведения***
>
> Обратите внимание: если вы открываете шаблон из диалогового окна **Открытие документа**, то у вас действительно открывается шаблон, а не обычный документ Microsoft Word. Если же вы просто дважды щелкаете соответствующий значок в проводнике Windows, то открывается новый документ, основанный на данном шаблоне. Можно открыть шаблон и из проводника, но для этого следует щелкнуть шаблон правой кнопкой мыши и в раскрывшемся контекстном меню выбрать команду **Открыть**.

7.1.3. Обеспечение безопасности при использовании макросов

В заключение раздела несколько слов об обеспечении безопасности при использовании шаблонов, полученных из внешних источников. Вообще гово-

ря, возможна ситуация, при которой после установки такого шаблона на компьютер вы будете получать при запуске Microsoft Word предупреждение системы безопасности, в котором сообщается, что в макросах, имеющихся в загружаемом шаблоне, могут содержаться вирусы, и предлагается их отключить (рис. 7.4), или же, что макросы отключены из-за того, что был выбран высокий уровень безопасности.

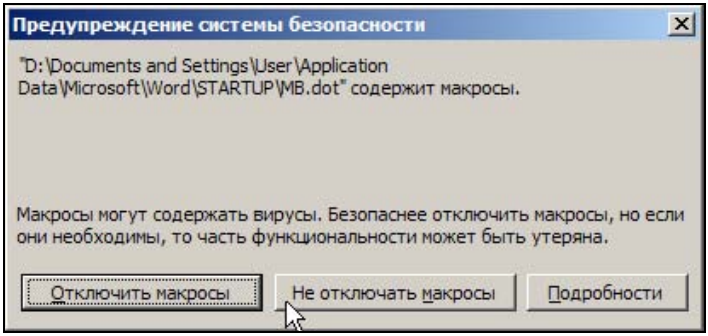


Рис. 7.4. Предупреждение системы безопасности о потенциальной опасности использования макросов, содержащихся в подключаемом шаблоне

Дело здесь в том, что в пакете Microsoft Office предусмотрены три уровня безопасности — **Высокая**, **Средняя** и **Низкая** (в Microsoft Office Word 2003 имеется еще уровень безопасности **Очень высокая**). Если в Microsoft Word установлен *средний* уровень безопасности, то решение о возможности запуска макросов (которые, вообще говоря, могут в действительности оказаться вирусами) принимается самим пользователем. В этом случае можно, конечно, каждый раз при запуске Microsoft Word нажимать кнопку **Не отключать макросы** или же установить в Microsoft Word низкий уровень безопасности, при котором защита от потенциально опасных макросов вообще отсутствует (выбор данного варианта не рекомендуется разработчиками программы, поэтому мы не будем описывать, как это можно сделать). Однако гораздо целесообразнее поступить следующим образом. Откройте меню **Сервис**, укажите пункт **Макрос** и в открывшемся подменю выберите команду **Безопасность**, перейдите на вкладку **Надежные источники** (в Microsoft Office Word 2003 — **Надежные издатели**) открывшегося одноименного диалогового окна и установите флажок **Доверять всем установленным шаблонам и надстройкам** (рис. 7.5). Логика здесь простая: раз уж вы сами установили тот или иной шаблон на свой компьютер, значит, полностью доверяете источнику, из которого его получили.

По умолчанию этот флажок в Microsoft Word установлен, однако если вы получили одно из упомянутых выше предупреждений системы безопасно-

сти, это свидетельствует о том, что он был ранее кем-то (случайно или преднамеренно) снят.

Примечание

Вообще говоря, предупреждение системы безопасности может выводиться даже в том случае, если пользователь Microsoft Word не подключал каких-либо шаблонов, а просто создал хотя бы один макрос (который, как вы уже знаете, по умолчанию сохраняется в шаблоне Normal.dot). Именно поэтому мы, собственно, и рекомендовали вам установить флажок на вкладке **Надежные источники**, если он не был ранее установлен, еще в первой главе книги *(см. разд. 1.2)*.

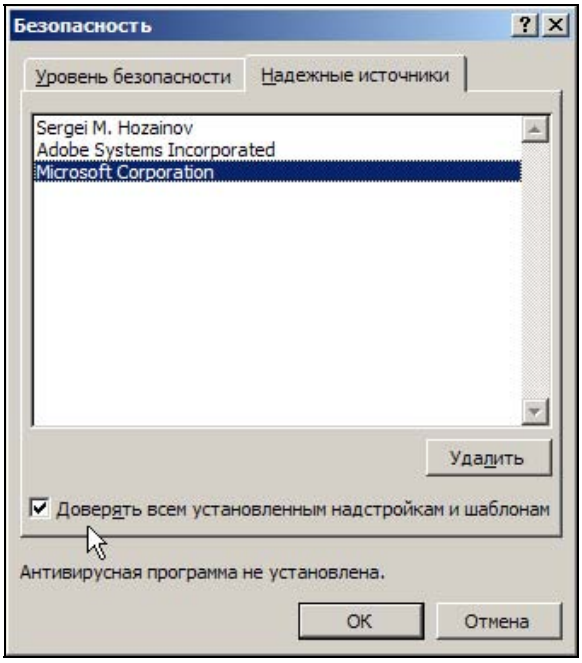


Рис. 7.5. Вкладка **Надежные источники** диалогового окна **Безопасность**, где устанавливается флажок **Доверять всем установленным шаблонам и надстройкам**

В этом случае вы сможете избавиться от надоедливых оповещений системы безопасности, одновременно сохранив приемлемый уровень защиты от потенциально опасных вирусов.

Полезные сведения

Система безопасности является службой, обеспечивающей защиту практически всех приложений Microsoft Office (за исключением Microsoft Access 2000 и

Microsoft Access 2002, а также приложений Microsoft Office 97). Поэтому в случае, если предупреждение о потенциально опасных вирусах досаждает вам и при работе с другими приложениями этого пакета, вы также можете воспользоваться описанным ранее способом для решения данной проблемы. (Попутно заметим, что мы не будем здесь касаться вопросов использования цифровых подписей для управления загрузкой макросов.)

7.2. Шаблоны, модули и макросы

Вернемся теперь к проблеме передачи созданного макроса (или набора макросов) другим пользователям. Как вы уже знаете, при автоматической записи макросов они по умолчанию сохраняются в шаблоне Normal.dot. Если говорить более конкретно, эти макросы записываются в *модуль* NewMacros, который автоматически создается в шаблоне Normal.dot, когда пользователь создает свой первый макрос.

7.2.1. Создание модуля для набора макросов

Учитывая сказанное, вы можете предположить, что решение проблемы распространения макросов состоит в передаче другим пользователям модуля NewMacros. И, с формальной точки зрения, будете правы. Однако для практического использования данный способ является малопригодным. Дело в том, что в модуле NewMacros кроме макроса (или набора макросов), который вы собираетесь передать, по умолчанию содержатся и все другие макросы, созданные в автоматическом режиме в Microsoft Word на данном компьютере. Такой подарок окажется, пожалуй, неоправданно щедрым. Кроме того, получивший его пользователь, скорее всего, не сможет им воспользоваться, поскольку не будет знать, для чего предназначены все эти макросы.

Более того, если модуль NewMacros будет передаваться в составе общего шаблона Normal.dot, сделанный вами подарок может оказаться сродни "медвежьей услуге", поскольку в Normal.dot, как вы помните, кроме макросов хранятся также все настройки рабочей среды Microsoft Word. Подключив ваш шаблон к своему экземпляру программы, другой пользователь рискует просто не узнать привычный для него интерфейс своей программы — настолько сильно он может измениться. Кроме того, в этом случае могут возникнуть многочисленные конфликты между "старыми" и "новыми" настройками (касающиеся, например, назначенных сочетаний клавиш), результат которых часто является непредсказуемым.

Таким образом, наша задача состоит в том, чтобы найти способ, с помощью которого можно было бы "изолировать" передаваемый набор макросов. Мы

предлагаем воспользоваться для этого методом, основная идея которого состоит в следующем. Необходимо, во-первых, создать новый модуль, который бы содержал только предназначенные для передачи макросы, и, во-вторых, сохранить этот модуль в таком виде, чтобы им могли воспользоваться другие пользователи.

Решить первую часть указанной ранее задачи можно следующим образом.

Если помните, описывая процедуру создания макроса в *гл. 1*, мы рекомендовали вам оставить в поле **Макрос доступен для** диалогового окна **Запись макроса** значение, предлагаемое по умолчанию, а именно **Всех документов (Normal.dot)** — *см. разд. 1.2* (в этом случае создаваемый макрос автоматически попадает в модуль NewMacros шаблона Normal.dot). Поле **Макрос доступен для** является полем со списком. Поэтому если раскрыть этот список, щелкнув кнопку с изображением маленького черного треугольника, расположенную в правой части этого поля, можно увидеть еще один элемент. Имя этого элемента совпадает с именем документа Microsoft Word, в котором осуществляется создание макроса (предположим, что это файл с именем "Макросы_для_Васи") — рис. 7.6.

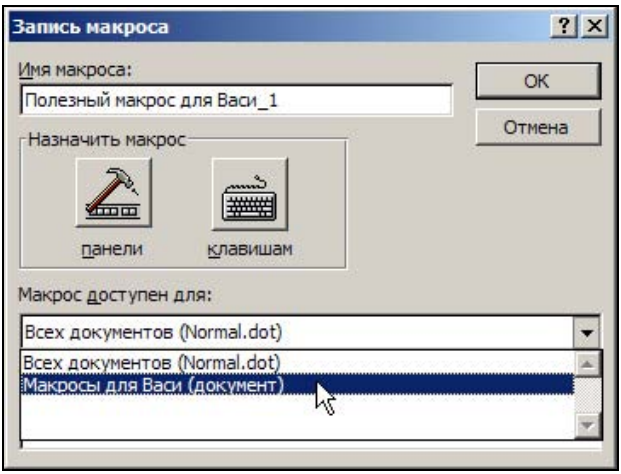


Рис. 7.6. Выбор документа, в котором будет сохранен создаваемый макрос, в диалоговом окне **Запись макроса**

Если выбрать данный элемент в списке, то в файле Макросы_для_Васи.doc (напоминаем, что он является не шаблоном, а обычным документом Microsoft Word) будет также создан модуль NewMacros, в котором и будет сохранен ваш новый макрос. Более того, все макросы, создаваемые с помощью описанного только что способа, будут записываться в новый экземпляр

модуля NewMacros. Таким образом, первая часть поставленной нами задачи решена: мы научились создавать "изолированный" набор макросов, записываемых не в общий шаблон Normal.dot, а в отдельный модуль, который находится в другом файле.

7.2.2. Создание файла-контейнера

Займемся теперь решением второй части этой задачи — сохранением вновь созданного модуля NewMacros в виде, в котором им смогли бы воспользоваться и другие пользователи. Иными словами, нам необходимо будет создать "файл-контейнер", содержащий только что полученный модуль NewMacros.

Сделать это можно, вообще говоря, двумя способами. При использовании первого из них нужно просто сохранить файл с новым модулем NewMacros (в нашем случае это Макросы_для_Васи.doc) как *обычный документ* Microsoft Word. Этот файл и может в дальнейшем использоваться в качестве контейнера для передачи набора макросов другим пользователям (как им можно будет воспользоваться, мы расскажем несколько позже).

Второй способ получения файла-контейнера состоит в сохранении документа, содержащего вновь созданный модуль NewMacros, в виде *шаблона*. Для этого необходимо перейти в данный документ и в меню **Файл** выбрать команду **Сохранить как** (или же просто нажать клавишу <F12>), а затем в появившемся диалоговом окне **Сохранение документа** раскрыть список в поле **Тип файла**, щелкнуть в нем элемент **Шаблон документа** и нажать кнопку **Сохранить**. В этом случае сохраняемая копия обычного документа Microsoft Word становится шаблоном, о чем будет свидетельствовать расширение dot, которое появится в имени файла.

> ***Примечание***
>
> При сохранении обычного документа Microsoft Word в качестве шаблона можно, разумеется, изменить и его имя (это, конечно, не является обязательным требованием). Если этого не сделать, то у вас на компьютере будут храниться два файла с одинаковыми именами, отличающимися друг от друга только своими расширениями (в нашем случае — Макросы_для_Васи.doc и Макросы_для_Васи.dot).

"Текстовая составляющая" файла-контейнера (полученного с помощью любого из описанных выше двух способов), вообще говоря, никак не связана с содержащимися в нем макросами. Более того, если в этом файле изначально имелся какой-либо текст, его можно при желании спокойно удалить. Вместе с тем, в этот документ можно записать, например, краткое описание передаваемых макросов или же инструкцию по их применению.

7.2.3. Переименование, копирование и удаление модулей и панелей

Файл-контейнер с вновь созданным модулем NewMacros можно, в принципе, передавать другим пользователям. Однако мы советуем с этим не спешить. Данный модуль по умолчанию имеет такое же имя, что и модуль, уже имеющийся в общем шаблоне Normal.dot, что может, вообще говоря, привести к целому ряду негативных последствий, связанных с так называемым конфликтом имен при установке файла-контейнера на другом компьютере. Поэтому имеет смысл заблаговременно позаботиться о переименовании данного модуля.

Для этого загрузите в Microsoft Word файл-контейнер с модулем NewMacros, который собираетесь передавать другому пользователю (в нашем случае это файл Макросы_для_Васи.dot или Макросы_для_Васи.doc), и откройте диалоговое окно **Организатор**. Это можно сделать несколькими способами, например, нажав кнопку **Организатор** в диалоговом окне **Шаблоны и надстройки** (меню **Сервис | Шаблоны и надстройки**).

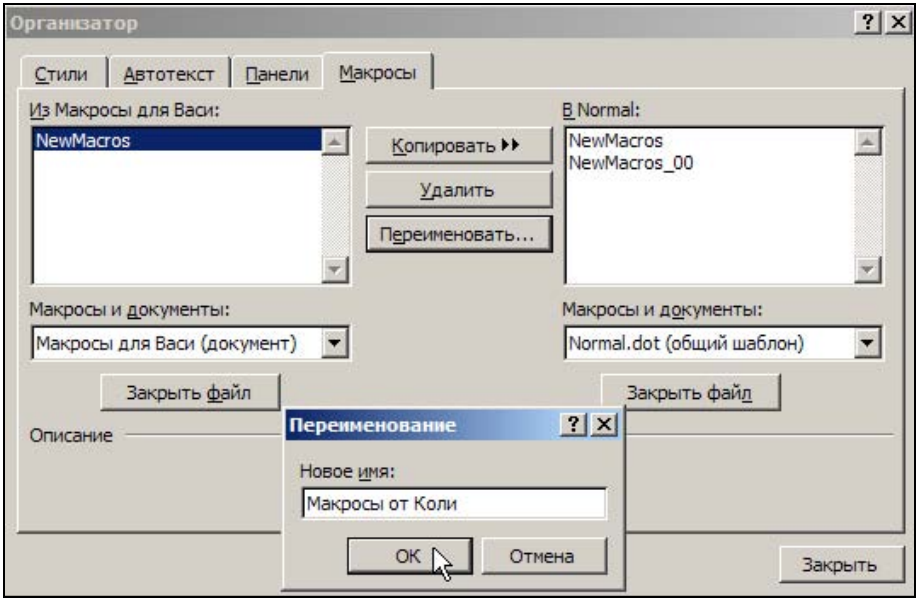


Рис. 7.7. Диалоговое окно **Организатор**, в котором может выполняться переименование, копирование и удаление модулей, панелей и других элементов Microsoft Word

Далее перейдите на вкладку **Макросы** диалогового окна **Организатор** и в левой ее части, расположенной над полем **Макросы и документы**, в котором указано имя документа **Макросы_для_Васи**, щелкните элемент **NewMacros**, нажмите кнопку **Переименовать** и введите для него новое имя, например, "Макросы_от_Коли" (рис. 7.7). После этого нажмите кнопку **OK**, а затем **Закрыть**. Сохраните открытый ранее файл Макросы_для_Васи.dot (или Макросы_для_Васи.doc), в котором теперь вместо модуля NewMacros содержится модуль с именем "Макросы_от_Коли". Именно его и следует в дальнейшем передать вашему знакомому Васе или другим пользователям.

Отметим, что если в поле **Макросы и документы** диалогового окна **Организатор** не будет указано имя нужного вам документа (в нашем случае это **Макросы_для_Васи**), то следует нажать расположенную ниже кнопку **Закрыть файл**, на месте которой отобразится кнопка **Открыть файл**. Далее нужно нажать эту кнопку и в появившемся диалоговом окне **Открытие документа** перейти в папку, где находится нужный вам файл, выбрать его и нажать кнопку **Открыть**.

Диалоговое окно **Организатор** может использоваться и для решения других задач, непосредственным или косвенным образом связанных с темой, которая рассматривается в данной главе. В частности, здесь может выполняться не только переименование модулей с макросами, но и их копирование и удаление. Например, нажав кнопку **Копировать** (см. рис. 7.7), вы можете скопировать только что переименованный модуль Макросы_от_Коли в свой шаблон Normal.dot (на рис. 7.7 элементы, относящиеся к этому шаблону, отображаются в правой части диалогового окна **Организатор**). Благодаря этому вы сможете в дальнейшем пользоваться макросами, содержащимися в модуле Макросы_от_Коли, работая с любыми документами в своем экземпляре Microsoft Word. В противном случае вам нужно будет либо подключить шаблон с этими макросами к своей программе, либо повторить их создание, записав на этот раз в модуль NewMacros своего шаблона Normal.dot.

Аналогичным образом модуль, ставший по каким-то причинам вам ненужным, можно удалить из любого шаблона или обычного документа Microsoft Word (для этого следует выбрать модуль и нажать кнопку **Удалить**).

Если вы внимательно посмотрите на рис. 7.7, то увидите, что помимо вкладки **Макросы** в диалоговом окне **Организатор** имеются еще три вкладки, из которых непосредственный интерес для нас представляет только вкладка **Панели**. С панелями инструментов мы уже неоднократно имели дело в предыдущих главах книги. Так вот, сообщаем вам, что практически все сказанное ранее об операциях с модулями макросов относится и к этому важному элементу интерфейса Microsoft Word. Иными словами, если в документе (шаблоне) имеется панель инструментов, то вы можете в любой момент переименовать, удалить или скопировать ее в другой документ с помощью элементов управления вкладки **Панели** диалогового окна **Организатор**. По-

следняя возможность (копирование панелей) является особенно важной, поскольку две первые операции могут быть выполнены, как вы знаете, и в диалоговом окне **Настройка**.

Следует отметить, что в случае с панелями инструментов пользователям Microsoft Word повезло значительно больше, чем с модулями макросов. Макросы, как вам уже известно, имеют "групповое представительство" в диалоговом окне **Организатор**, где отображаются в виде отдельных модулей (по умолчанию это NewMacros). В свою очередь, панели инструментов являются "штучным товаром" — каждая из них представлена на вкладке **Панели** диалогового окна **Организатор** индивидуально. Поэтому чтобы передать нужную панель инструментов другим пользователям, ее не требуется предварительно каким-то образом отделять от остальных панелей (как пришлось нам делать в случае модуля макросов), а можно напрямую скопировать в передаваемый файл-контейнер.

При этом необходимо иметь в виду следующий принципиальный момент. Если на панели инструментов имеются кнопки, которые были назначены вами тем или иным макросам, то копирование этой панели инструментов в другой документ *не сопровождается* одновременным копированием макросов, соответствующих этим кнопкам. Копирование макросов, как описывалось ранее в этой главе, выполняется отдельно на вкладке **Макросы**. Если же этого не сделать, то кнопки при копировании панели инструментов копироваться будут, но при переносе файла-контейнера на другой компьютер работать не станут.

> ***Примечание***
>
> Сказанное не относится к кнопкам, которые были назначены встроенным командам Microsoft Word. В силу понятных причин ими можно будет пользоваться в любом экземпляре программы.

7.2.4. Использование файла с макросами на другом компьютере

Предположим, что вы подготовили файл-контейнер с созданными вами макросами и передали его другому пользователю. Как же должен теперь поступить этот пользователь, чтобы воспользоваться содержащимися в нем макросами? (Кстати, в его положении можете, конечно, оказаться и вы сами, получив файл с макросами от другого пользователя или, скажем, скачав его из Интернета.)

Это зависит, в первую очередь, от того, какой из описанных ранее способов был использован вами при создании "файла-контейнера".

Если файл-контейнер представляет собой обычный документ, то его можно просто загрузить в Microsoft Word и использовать по мере необходимости в

своей работе. Однако следует иметь в виду, что в этом случае "имплантированные" в него макросы и панели инструментов можно будет использовать *только* непосредственно в этом документе или же в его "клонах", созданных с помощью команды **Сохранить как**. Во всех же остальных документах Microsoft Word они окажутся недоступными.

Поэтому более практичным представляется способ, состоящий в копировании модуля с макросами и панелей инструментов (при их наличии) из файла-контейнера в общий шаблон Normal.dot. Какие действия при этом следует выполнить, описывалось нами в предыдущем подразделе.

Если же файл-контейнер является шаблоном, то им можно воспользоваться в соответствии с рекомендациями, приведенными *в разд. 7.1.2* этой главы. А именно, его можно будет подключить в качестве общего шаблона, поместив в папку STARTUP или добавив к списку общих шаблонов в диалоговом окне **Шаблоны и надстройки**.

Можно также воспользоваться этим файлом как шаблоном документа, подключив к нужному документу с помощью кнопки **Присоединить**.

При подготовке шаблона с макросами, который вы собираетесь передавать другим пользователям, мы рекомендуем воздержаться от назначения этим макросам тех или иных сочетаний клавиш. Дело в том, что присвоенные вами сочетания клавиш могут быть уже закреплены в экземпляре Microsoft Word, установленном на другом компьютере, за другими макросами или командами. В этом случае назначенные вами сочетания клавиш перестанут работать.

7.3. Что нового в этой главе

В этой главе вы познакомились с особым типом документов Microsoft Word — шаблонами, их видами, назначением и возможностями использования. Вы узнали, каким образом может осуществляться подключение шаблонов к документам Microsoft Word, а также обеспечивается безопасность при использовании шаблонов, содержащих макросы.

Вы узнали также, что с помощью шаблонов может осуществляться передача созданных вами макросов другим пользователям. В главе подробно описывается, каким образом может быть подготовлен "контейнер" для передаваемого набора (модуля) макросов, рассказывается о возможностях и способах переименования, копирования и удаления модулей с макросами, а также панелей инструментов.

Наконец, вы узнали о том, как следует поступать, чтобы воспользоваться макросами и другими функциональными возможностями, содержащимися в документе Microsft Word, полученном из внешних источников.

7.4. Напутствие любознательным читателям

В начале этой книги мы обещали, что, освоив материал ее первой части, вы сможете при желании на этом остановиться и отправиться, так сказать, в "самостоятельное плавание". Однако вы должны отдавать себе отчет, что в состав команды корабля, на котором будет осуществляться ваше плавание, не входит главный механик — специалист, разбирающийся в механизмах, которые обеспечивают движение судна. Поэтому, если возникнет необходимость что-либо подправить в их функционировании или внести в работу механизмов какие-либо усовершенствования, то сделать это будет некому.

От пользователя, занимающегося созданием макросов в автоматическом режиме, требуется лишь сформулировать алгоритм решения стоящей перед ним задачи, а затем аккуратно воспроизвести соответствующие действия при включенном режиме записи макроса. Все вопросы, связанные с созданием программы, выполняемой при каждом запуске макроса, в этом случае пользователя не касаются. Данная программа создается автоматически в "фоновом" режиме средствами редактора Visual Basic, встроенного в Microsoft Word. Такой подход обладает несомненными достоинствами, поскольку позволяет заниматься созданием макросов пользователям, не имеющим какой-либо специальной подготовки. Однако мы уже неоднократно отмечали и его ограниченность, связанную, прежде всего, со следующими двумя моментами.

- Во-первых, с помощью этого подхода могут создаваться только достаточно простые макросы, буквальным образом воспроизводящие действия пользователя. Это, в частности, означает, что в макросе не могут быть реализованы какие-либо "интеллектуальные" действия, связанные, например, с выполнением условных операций (по принципу: "если произойдет некоторое событие, то будет выполнена одна цепочка действий, а если это событие не произойдет — другая цепочка"), использованием так называемых "переменных", в которых могут временно храниться полученные при выполнении макроса промежуточные результаты, и т. п.
- Во-вторых, в текст программы макроса, создаваемого в автоматическом режиме, записывается много лишней, а в некоторых случаях даже "вредной" информации, фиксирующей конкретные условия, в которых происходил процесс записи макроса (вспомните пример с отображением сетки таблицы при использовании макроса, подкрашивающего строки).

Мы надеемся, что после знакомства с содержанием первой части книги вы поняли, что главным в создании макросов является формулировка идеи, позволяющей эффективно и с помощью доступных средств решить стоящую перед вами задачу. Такая идея должна быть по возможности простой и в то же время остроумной. Для выполнения этих условий требуется достаточно глубокое знание возможностей и средств Microsoft Word. Мы предлагаем

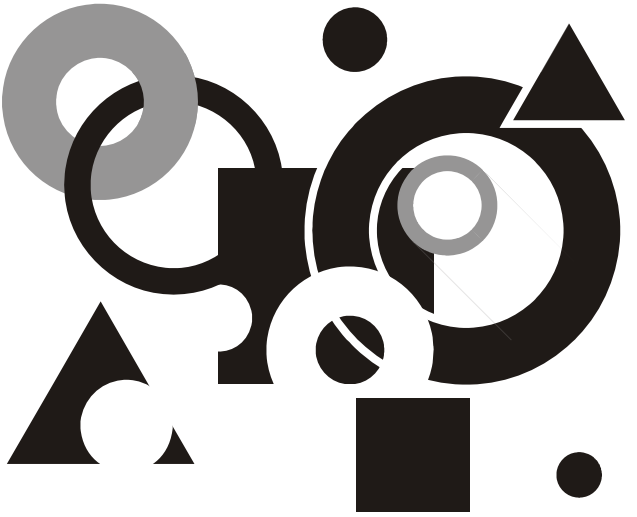
вам существенно расширить арсенал подобных средств, освоив некоторые элементы программирования на языке Visual Basic, описание которых приводится в следующих частях книги.

В этой главе, мы постарались продемонстрировать вам, что "не боги горшки обжигают": оказывается, вносить небольшие изменения в текст макросов не так уж и сложно. Возможно, вы посчитаете, что вам вполне достаточно того материала, с которым вы уже познакомились в этой и в предыдущих главах, и прервете чтение этой книги. Однако мы уверены, что если вы займетесь практическим созданием макросов на основе полученных вами знаний, то очень скоро вам захочется делать это с пониманием, а не руководствуясь только интуицией.

В этом случае мы советуем вам продолжить чтение этой книги и приступить к изучению ее второй части. В ней на конкретных примерах и гораздо более обстоятельно, чем в настоящей главе, рассказывается о структуре программ, которые создает редактор Visual Basic при автоматической записи макросов. Вы научитесь "с пониманием" читать тексты на языке Visual Basic, что поможет вам совершенствовать автоматически записанные программы, внося в них небольшую правку. На этом этапе программирование вам также практически не понадобится, решение о необходимости его изучения можно отложить до третьей части книги, где излагаются основные идеи создания "интеллектуальных программ" и демонстрируются возможности их практического применения. Однако мы еще раз честно предупреждаем, что с помощью нашей книги стать программистом нельзя. Для этого существует много других путей, и об этом мы подробнее поговорим в заключительной главе книги.

Подчеркнем еще раз, что изложение материала книги будет и далее строиться нами на рассмотрении конкретных практических примеров. Во многих случаях в них будет показано, каким образом можно усовершенствовать макросы, созданные в первой части книги, с тем, чтобы существенно повысить эффективность их использования. Надеемся, что такое построение материала упростит вам освоение практических приемов создания макросов и, в конечном счете, повысит "коэффициент полезного действия" книги.

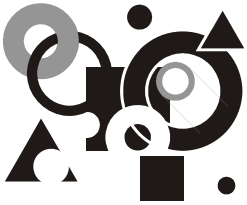
В добрый путь!



ЧАСТЬ II

Учимся читать макросы

Глава 8



Макрос изнутри

О редакторе Visual Basic. Автоматическая запись макроса (репортаж с места событий). Структура подпрограммы. Ключевые слова и комментарии. Инструкции `Sub` и `End Sub`. Объекты, свойства, методы, аргументы, константы. Объект `Selection`. Методы `MoveRight`, `MoveLeft`, `Cut`, `Paste`. Необязательные аргументы и значения по умолчанию.

Этой главой начинается новая часть книги. В первой части книги вы научились создавать макросы методом автоматической записи выполняемых действий. Теперь вы уже понимаете, как можно использовать макросы, чтобы работать в приложении Microsoft Word было эффективнее и интереснее. Вместе с тем вам известно, что возможности автоматически записанных макросов ограничены. Вы уже побывали на экскурсии в "сборочном цехе макросов" *(см. гл. 6)* и знаете, что в редакторе Visual Basic совсем несложно внести изменения в автоматически записанную подпрограмму. Вы видели, что строки в подпрограммах содержат много английских слов и, если перевести их на русский язык, то часто удается не только догадаться, что запрограммировано в той или иной строке, но даже с успехом внести в нее небольшие изменения.

Однако надо признать, что на этой экскурсии вы были гостем. Вносить изменения в программу, не понимая, что в точности означает тот или иной программный элемент, очень интересно, но, как говорят в народе, "не зная броду, не суйся в воду", — предсказать, что получится в результате, вы пока не можете. Есть много книг, из которых можно почерпнуть необходимые сведения, в вашем распоряжении имеется также хорошая справочная система по Visual Basic (на английском языке), множество журнальных статей и необъятный Интернет, но мы на своем опыте знаем, что воспользоваться этими источниками не так уж и просто. Основная причина состоит в том, что о средствах программирования, используемых редактором Visual Basic при автоматической записи макросов, во всех известных нам книгах рассказывается где-то после двухсотой страницы последовательного изложения материала, предназначенного для подготовки будущего программиста. Если же вы не собираетесь в ближайшее время освоить эту профессию, то этот

способ не для вас. Можно пойти другим путем: в редакторе Visual Basic выделить незнакомое слово и нажать клавишу <F1>. Редактор Visual Basic сразу же придет к вам на помощь и предоставит вам необходимые справочные сведения. Однако понять, о чем идет речь, удается далеко не всегда. Начнем с того, что эти сведения написаны по-английски, но не это главное, — все дело в незнакомых понятиях, о которых, возможно, рассказывается в других разделах справочной системы. Чаще всего удается, пользуясь хорошо развитой системой поиска, найти нужные сведения. Однако, затратив определенные усилия, вы лишь сможете убедиться, что справочный аппарат Visual Basic не предназначен для начального знакомства с программированием.

Мы предлагаем вам компромиссное решение. Оно основано на том, что программные средства, которые использует редактор Visual Basic при автоматической записи макросов, весьма ограничены и для того, чтобы с пониманием читать записанные им подпрограммы, не обязательно становиться программистом. Мы решили поломать сложившиеся традиции и рассказать вам об этих средствах на начальном этапе знакомства с языком Visual Basic. Во-первых, это интересно, а во-вторых, сразу принесет практическую пользу. В результате вы сможете с пониманием вносить небольшие изменения в автоматически созданные подпрограммы, что сразу расширит ваши возможности в применении макросов.

В ходе изложения мы вводим много терминов, но не ради наукообразия, а для того, чтобы помочь вам впоследствии лучше ориентироваться в справочной системе редактора Visual Basic, а также использовать учебники по программированию в качестве справочников, не читая их "от корки до корки".

Таким образом, цель, которую мы преследуем в этой части книги, — это научить вас читать автоматически сгенерированные программы и с пониманием вносить в них минимальные исправления.

Об элементах программирования, которыми не владеет редактор Visual Basic, мы рассказываем в следующей части.

8.1. Пара слов о редакторе Visual Basic

После изучения *гл. 6* вы уже знаете, что для того чтобы просматривать текст записанных макросов, вносить в них изменения или создавать свои собственные макросы "с чистого листа", используется редактор Microsoft Visual Basic. Он содержится в пакете Microsoft Office и специально предназначен для программирования на языке VBA (Visual Basic for Application — Visual Basic для приложений). Это диалект языка Visual Basic, приспособленный для приложений Microsoft Office (Word, Excel, Access, Outlook и др.). Его разработчики старались, чтобы инструкции программы, записанные на этом

языке, были похожими на предложения английского языка. Поэтому тем из вас, кто хочет научиться читать программы на языке Visual Basic, но не читает по-английски, придется выучить несколько десятков английских слов. Если английский вам хоть немного знаком, позаботьтесь о том, чтобы установить справку по Visual Basic (этот компонент Microsoft Office находится в группе "Средства Office"). В противном случае мы рекомендуем приобрести одну из многих книг с описанием средств программирования на языке VBA, чтобы пользоваться ею как справочником. О некоторых таких книгах мы расскажем в *заключении*.

Однако на данном этапе беспокоиться об этом не надо. В этой главе мы не будем сколько-нибудь подробно изучать редактор Visual Basic. Для работы с макросами в основном будет достаточно тех навыков, которые вы уже приобрели, работая в редакторе Word. Все встречающиеся в тексте английские слова мы переведем для вас на русский язык, а если перевод забудется, можно воспользоваться небольшим словарем-указателем, приведенным в конце книги.

8.2. Автоматическая запись макроса. Репортаж с места событий

Обычно редактор Visual Basic во время автоматической записи макроса не открывают, но сейчас нам будет интересно посмотреть на создание программы "в живом эфире". Для этого мы произведем запись макроса при открытом редакторе Visual Basic.

В качестве примера запишем простой макрос, переставляющий две соседние буквы, между которыми расположен курсор. Мы по себе знаем, что при печати многие не очень опытные в машинописи пользователи иногда нажимают клавиши не в том порядке, из-за чего некоторые соседние буквы в слове меняются местами. Этот макрос позволяет исправлять такие опечатки. Итак, выполните следующие действия:

1. Откройте новый документ Microsoft Word, напишите какое-нибудь слово и поместите курсор в середину этого слова.
2. Откройте редактор Visual Basic. Это просто — в меню **Сервис** выберите последовательно пункты **Макрос** и **Редактор Visual Basic** (или нажмите сочетание клавиш <Alt>+<F11>).
3. Чтобы быстро и удобно разместить окна Visual Basic и Microsoft Word на экране, минимизируйте все остальные открытые окна и, щелкнув правой кнопкой на свободном месте в панели задач, например, рядом с часами, выберите в контекстном меню команду **Окна сверху вниз** (рис. 8.1).

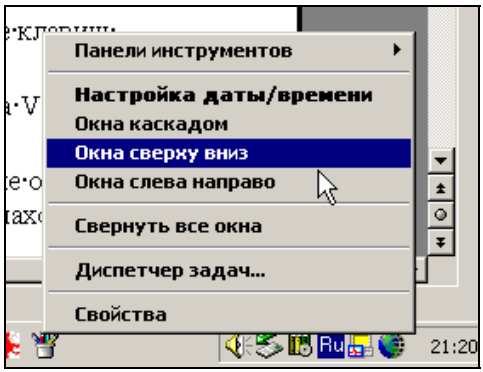


Рис. 8.1. Контекстное меню панели задач

В результате у вас на экране будут одновременно видны и окно документа, и редактор Visual Basic.

4. Перейдите в окно документа и начните записывать макрос **CharFlip_1** (переворот знаков, версия 1) с описанием "Перестановка соседних букв".
5. Когда на экране появится панель **Остановить запись**, нажмите на ней кнопку **Пауза**, чтобы приостановить запись макроса.
6. Так же, как вы это делали в *разд. 6.1*, перейдите в редактор Visual Basic к макросу **CharFlip_1** (меню **Сервис | Макрос | CharFlip_1 | Изменить**).

   Станет активным окно редактора Visual Basic, и в нем вы увидите автоматически созданную заготовку для будущего макроса:

```
Sub CharFlip_1()
'
' CharFlip_1 Макрос
' Перестановка соседних букв
'
End Sub
```

   Вы уже знаете, что первая и последняя строки — это первая и последняя инструкции будущей программы с расположенными между ними строками комментария.

7. Вернитесь в окно документа и нажмите кнопку **Возобновить запись** () на панели **Остановить запись**.
8. Выделите букву справа от курсора, нажав сочетание клавиш <Shift>+<→>.

   В окне Visual Basic ничего не изменяется.

9. Вырежьте выделенную букву в буфер обмена, нажав сочетание клавиш <Shift>+<Del> (или <Ctrl>+<X>).

   В программе макроса добавляются сразу две строчки:

```
Sub CharFlip_1()
 '
 ' CharFlip_1 Макрос
 ' Перестановка соседних букв
 '
     Selection.MoveRight Unit:=wdCharacter, Count:=1, Extend:=wdExtend
     Selection.Cut
End Sub
```

10. Переместите курсор на одну букву влево клавишей <←>.

    В окне Visual Basic ничего не изменяется.

11. Вставьте содержимое буфера обмена, нажав сочетание клавиш <Shift>+ +<Ins>. Цель достигнута, буквы переставлены.

    В программе макроса добавляются еще две строчки, и подпрограмма `CharFlip_1` приобретает окончательный вид, показанный в листинге 8.1.

12. Остановите запись макроса.

Листинг 8.1

```
 Sub CharFlip_1()
'
' CharFlip_1 Макрос
' Перестановка соседних букв
'
    Selection.MoveRight Unit:=wdCharacter, Count:=1, Extend:=wdExtend
    Selection.Cut
    Selection.MoveLeft Unit:=wdCharacter, Count:=1
    Selection.Paste
End Sub
```

Позже мы объясним, почему при записи макроса строчки добавляются не по одной, а парами. А сейчас просто обратите на это внимание.

Макрос готов. Мы надеемся, он будет вам полезен. Однако этот макрос ценен для нас в первую очередь тем, что в нашем распоряжении теперь есть

программа, написанная на языке Visual Basic, которая хорошо подходит для начального этапа изучения этого языка. Но сначала нам придется немного отвлечься, чтобы познакомиться с терминологией и некоторыми основными понятиями языка Visual Basic.

8.3. Структура программы

Результатом записи макроса является так называемая *подпрограмма* (subprogram).

Подпрограммы состоят из инструкций (мы следуем терминологии, принятой в Microsoft Word, некоторые авторы их называют командами или операторами).

На одной строке можно записать несколько инструкций, разделив их двоеточием, но в автоматически записанных макросах каждая инструкция начинается с новой строки. Мы тоже не пользуемся этим приемом и вам не советуем, поскольку считаем, он ухудшает наглядность текста программ.

Если инструкция длинная, ее можно продолжать на нескольких строках. В качестве символа переноса инструкции на следующую строку используется знак подчеркивания (_). Перед ним должен быть пробел, и он должен быть последним символом строки. Заранее предупредим, что перенос строки нельзя применять внутри текста, заключенного в кавычки. Перенос строк вы не раз встретите в примерах, приводимых в нашей книге. Его можно также увидеть и в подпрограммах, созданных при автоматической записи макросов.

Каждая подпрограмма начинается с инструкции **`Sub`** (начало слова *subprogram* — подпрограмма), а заканчивается инструкцией **`End Sub`**. (*end* — конец).

Для того чтобы текст подпрограммы был понятнее, редактор Visual Basic автоматически его раскрашивает.

Зеленым цветом в тексте подпрограмм помечены *комментарии* (comment), т. е. части строк, начинающиеся с апострофа (`'`), а темно-синим цветом — так называемые *ключевые слова* (keyword) Visual Basic, которые имеют строго определенный смысл и не могут использоваться для других целей (цвета при желании можно изменить). В нашей книге ключевые слова выделены полужирным шрифтом, а комментарии мы не выделяем.

В следующих разделах подробно рассмотрим каждую инструкцию подпрограммы `CharFlip_1`. Мы не будем пытаться дать исчерпывающее описание инструкций. На этом этапе мы расскажем только о том, какую роль выполняет каждая из них в данной подпрограмме.

8.4. Инструкции *Sub* и *End Sub*

Подпрограмма `CharFlip_1` начинается с инструкции **`Sub`** `CharFlip_1()` и заканчивается инструкцией **`End Sub`**.

Эти инструкции формируются автоматически в самом начале записи макроса.

В инструкции **`Sub`** после ключевого слова **`Sub`** задается название подпрограммы. При выполнении этой инструкции редактор Visual Basic проводит формальную проверку всех инструкций подпрограммы и, если есть ошибки, пользователь получает соответствующее сообщение и работа макроса прерывается. При автоматической записи макроса подпрограмма обычно не содержит ошибок, и после ее проверки редактор Visual Basic из понятных программистам инструкций создает понятную компьютеру программу.

> ***Полезные сведения***
>
> Если требуется переименовать макрос, измените его имя в инструкции **`Sub`**. При этом проследите, чтобы новое название удовлетворяло требованиям, предъявляемым к названиям макроса *(см. разд. 1.1.2)*. В противном случае мы получим такое же сообщение об ошибке, как при использовании недопустимого имени макроса в окне **Запись макроса**. Учтите, что после изменения имени макроса перестанут работать кнопки, команды меню и сочетания клавиш, которые были назначены данному макросу. Обратите внимание на то, что, не пользуясь редактором Visual Basic, переименовать макрос нельзя.

Инструкция **`End Sub`** отмечает конец подпрограммы. При ее выполнении никаких видимых действий не происходит, но на самом деле она "заметает следы": освобождается вся память, которая была во временном использовании при выполнении подпрограммы.

8.5. Комментарии

В подпрограмме `CharFlip_1` вы видите следующие комментарии:

```
'
' CharFlip_1 Макрос
' Перестановка соседних букв
'
```

Комментарий — это поясняющий текст. Он пишется для человека, читающего программу, а для редактора Visual Basic пишется только знак апострофа (это так называемый "прямой апостроф", изображенный на той же клавише, что и кавычки) в начале комментария. Все, что находится в инструкции после знака апострофа, Visual Basic пропускает.

Программисты часто используют апостроф для превращения в комментарий временно ненужных инструкций (например, используемых для отладки программы или написанных "про запас"). Про такие строки говорят, что они "закомментированы". Потом их можно "раскомментировать".

Как вы помните, при записи макроса вышеприведенный комментарий сформировался автоматически на основе информации, предоставленной вами при записи макроса. Первая строка включает название макроса и слово "Макрос", а затем идет строка, указанная во время записи макроса в окне **Запись макроса** в поле **Описание**. Ее потом можно будет увидеть в окне **Макрос**. Эту строку, также как и другие комментарии, можно свободно изменять. На описание макроса в окне **Макрос** это уже не повлияет (в свою очередь, это описание можно независимо изменять в том же окне).

Назначение комментариев — помочь читателю понимать программу.

Тем, кто будет программировать самостоятельно, рекомендуется в начале программы записывать, для чего нужна программа, как ей пользоваться, кто и когда ее разработал. Если из текста какого-либо фрагмента программы не ясно, зачем он написан, или как он действует, значит, нужны комментарии.

Не следует записывать в комментариях то, что сразу видно из текста программы. В идеале в программе должны использоваться такие обозначения, чтобы их смысл был понятен без комментариев, а сама программа была структурирована так, чтобы ее логика не требовала объяснений. Если же такого идеала достичь не удалось, то нужны комментарии.

8.6. Метод *MoveRight*

Перейдем теперь к описанию так называемых *исполняемых инструкций*, которые соответствуют определенным действиям, выполняемым программой. Инструкции, которые мы рассматривали до сих пор, играют служебную роль и, хотя тоже исполняются, по традиции называются *неисполняемыми*. Начнем с того, что попытаемся понять, что обозначает первая исполняемая инструкция в нашем макросе (см. листинг 8.1):

```
Selection.MoveRight Unit:=wdCharacter, Count:=1, Extend:=wdExtend
```

Вспомните, что мы делали во время записи макроса. Мы начали с того, что выделили один знак справа от курсора. Иными словами, переместили курсор на один знак вправо в режиме расширения выделения (т. е. при нажатой клавише <Shift>). Это действие и записано в данной инструкции. Перевод с языка Visual Basic, в основе словаря которого лежат слова английского языка, звучит примерно так: "Переместить курсор вправо (`Selection.MoveRight`). Единица измерения перемещения — символ (`Unit:=wdCharacter`), количество единиц перемещения — 1 (`Count:=1`), режим перемещения — расшире-

ние выделения (`Extend:=wdExtend`)". Чтобы это было понятно тем, кто не знает английского языка, приведем следующий англо-русский минисловарик:

Selection — выделение

Move — переместить

Right — вправо

Unit — единица измерения

wd — сокращение от Microsoft Word

Character — символ

Count — счет

Extend — расширять

Теперь, когда мы понимаем все слова в инструкции, на очереди знаки препинания языка Visual Basic, но сначала мы ненадолго займемся *объектно-ориентированным программированием* (Object-oriented Programming). Не пугайтесь, мы только чуть-чуть "приоткроем калитку" в этот увлекательный мир и познакомимся лишь с отдельными представителями его населения.

Центральное место в этой самой популярной в настоящее время методологии программирования занимает понятие *объекта* (object). Определение объекта мы давать не будем, а примеров будет много. Собственно, этот разговор мы затеяли потому, что в нашей процедуре используется объект `Selection` (выделение).

Для простоты будем считать, что объект `Selection` — это выделенный участок документа или (если в документе ничего не выделено) курсор, который можно рассматривать как вырожденное выделение (об этом никак нельзя догадаться из перевода).

> ***Примечание***
>
> Начиная с версии Microsoft Word 2002 предусмотрена возможность выделения несмежных участков документа. В документации по VBA мы не обнаружили упоминаний об особенностях работы с таким выделением. В этой книге мы предполагаем, что используется только традиционное выделение, охватывающее непрерывную область документа.

Сейчас нам повезло: этот объект достаточно наглядный, его можно увидеть на экране. Позже мы познакомимся с более абстрактными объектами.

Объектами можно управлять, используя их *свойства* (Property) и *методы* (method). Важно, что у всех объектов `Selection` один и тот же набор свойств и методов, а отличаются разные объекты `Selection` значениями свойств. Это так же, как все стулья имеют ножки, спинку и сиденья, на стульях можно сидеть и стулья можно перемещать. Иными словами стулья отлича-

ются от других объектов набором свойств. А между собой различаются стулья свойствами ножек, спинок и сидений. То есть имеется объект "стул" как некоторая абстракция с определенным набором свойств и возможностей, а есть конкретные стулья со своими конкретными свойствами. Точно также, когда говорят об объекте `Selection`, иногда имеют в виду конкретное выделение, а иногда некоторую абстракцию, характеризующую все выделения, которые только бывают. Впрочем, для этой абстракции есть специальное название: *класс* (class), так что вы можете встретить такое выражение, как класс `Selection`.

Со свойствами вы, наверное, уже знакомы. Если щелкнуть правой кнопкой любой значок на рабочем столе в Windows, то в раскрывшемся контекстном меню последним пунктом будет **Свойство**. Свойствами мы займемся позже, а сейчас на очереди методы. Методы интереснее, потому что они заставляют объекты действовать, делают их "почти живыми". Для каждого действия, которое можно выполнить над данным объектом, предусмотрен свой метод.

В данном случае к объекту `Selection` применяется его метод `MoveRight` (переместить вправо).

Чтобы обратиться к методу на языке Visual Basic, необходимо сначала указать конкретный объект (в данном случае для этого используется слово `Selection`), поставить точку, а затем (без пробела) записать название метода (в данном случае — `MoveRight`). Далее приводится так называемый *список аргументов* (argument list), в котором через запятую задаются так называемые *аргументы* (argument) метода, детально описывающие полученное методом задание.

Например, для того чтобы указать, что в качестве единицы измерения перемещения используется символ (это может быть также слово, предложение или ячейка таблицы), используется конструкция `Unit:=wdCharacter`.

Здесь `Unit` (единица измерения) — это имя аргумента, а `wdCharacter` — *константа*, задающая его значение (`wd` — показывает, что эта константа действует в Microsoft Word, `Character` — символ).

Константа (constant) — это просто обозначение числа. Например, `wdCharacter` обозначает единицу. На первый взгляд это выглядит странным. Зачем вводить длинные обозначения для чисел? Немного позже вы сами убедитесь, что это облегчает чтение программы, исключает необходимость запоминать, чему соответствует то или иное число.

Действительно, вместо `Unit:=wdCharacter` можно записать `Unit:=1` и это вполне допустимая конструкция. Однако чтобы ее понять, надо помнить, что значению 1 соответствует символ в качестве единицы измерения перемещения. Если же вы знаете, что слово *character* переводится как "символ", то запись `Unit:=wdCharacter` для вас существенно нагляднее.

Двоеточие с равенством (:=) — это не смайлик[1]. Такое сочетание знаков обозначает присваивание значения аргументу. Обратите внимание, что просто знак равенства здесь использовать нельзя.

Аргумент `Count` (счет) — количество единиц, на которое производится перемещение. В нашем случае `Count:=1` указывает на перемещение на один символ вправо.

Здесь мы хотим на минутку отвлечься, чтобы объяснить некоторую странность в поведении редактора Visual Basic, о которой говорилось ранее. Мы заметили, что во время записи макроса после выделения символа справа от курсора в окне Visual Basic ничего не изменилось, а после вырезания выделенного символа в этом окне появились сразу две инструкции. Почему инструкции не были записаны по одной после каждого действия? Оказывается, редактор Visual Basic ждал, чтобы узнать, не выделите ли вы еще один символ. Тогда бы он сэкономил лишнюю строчку и записал `Count:=2`. Как видите, этот редактор довольно умен, и в этом вам еще не раз предстоит убедиться.

Аргумент `Extend` (расширять) — указывает, нужно ли расширять выделение при перемещении. Если значением этого аргумента является константа `wdExtend`, то перемещение происходит с расширением выделения, т. е. в этом случае применение метода `MoveRight` эквивалентно нажатию сочетания клавиш <Shift>+<→>.

Теперь вы можете с пониманием прочесть инструкцию

```
Selection.MoveRight Unit:=wdCharacter, Count:=1, Extend:=wdExtend
```

Если какое-то слово или знак в ней вам не понятен, проглядите этот подраздел еще раз. На этом этапе наша задача состоит в том, чтобы получить лишь самое общее представление об инструкциях. Поэтому более подробно на методе `MoveRight` мы останавливаться не будем и перейдем к следующей инструкции.

8.7. Метод *Cut*

Рассмотрим теперь следующую инструкцию из текста нашего макроса:

```
Selection.Cut
```

[1] На всякий случай поясним (вдруг, кто еще не знает), что "смайлик" (от английского smile — улыбка) — это составленная из знаков препинания картинка, похожая на рожицу, позволяющая обогатить свое письмо выражением эмоций. Со смайликами хорошо знакомы те, кто ведет неформальную переписку на компьютере.

Эта инструкция была записана, когда мы вырезали в буфер обмена выделенную букву. Метод `Cut` (вырезать) объекта `Selection` удаляет выделенный участок документа из текста и помещает его в буфер обмена. Выделение превращается в курсор, который остается на месте выделенного участка документа. У этого метода аргументы не предусмотрены.

8.8. Метод *MoveLeft*

Далее в тексте нашего макроса содержится инструкция:

```
Selection.MoveLeft Unit:=wdCharacter, Count:=1
```

В этой инструкции используется метод `MoveLeft` объекта `Selection`, перемещающий курсор на один символ влево (*move* — переместить, *left* — влево). Она очень похожа на описанную ранее инструкцию с методом `MoveRight`. Пояснения требует только отсутствие аргумента `Extend` (расширять).

Оказывается, аргументы бывают *обязательными* (required) и *необязательными* (optional). Обязательные аргументы всегда надо указывать в списке аргументов, а необязательные можно пропускать. Для каждого необязательного аргумента всегда предусмотрено так называемое *значение по умолчанию* (default value). Это значение используется, когда аргумент отсутствует (иногда его называют стандартным значением).

Аргумент `Extend` (расширять) является необязательным, и для него значением по умолчанию является константа `wdMove` (*move* — переместить). Если для аргумента `Extend` задано значение `wdMove` (или аргумент `Extend` в списке аргументов отсутствует) применение метода `MoveLeft` эквивалентно нажатию клавиши <←> при отпущенной клавише <Shift>. Поскольку выделения в данный момент на экране нет, курсор перемещается влево на один символ.

8.9. Метод *Paste*

Последняя исполнимая инструкция нашего макроса:

```
Selection.Paste
```

Эта инструкция была записана, когда мы выполняли вставку вырезанной ранее буквы из буфера обмена (*paste* — вставить из буфера обмена). С помощью метода `Paste` объекта `Selection` вырезанная ранее буква помещается в место, указанное курсором (поскольку на экране нет выделения). У этого метода аргументы не предусмотрены.

8.10. Что нового в этой главе

Вы научились автоматически записывать макрос при открытом редакторе Visual Basic и увидели, как при этом создается подпрограмма для макроса **CharFlip_1**, который переставляет буквы, расположенные справа и слева от курсора. Вы познакомились со структурой программы, узнали, что программа состоит из инструкций, и научились читать с пониманием инструкции начала и конца программы, комментарии, инструкции, в которых реализованы методы перемещения курсора, вырезания в буфер и вставки из буфера. Вы получили первое представление об объектах, методах, свойствах и аргументах. Вы узнали о необязательных аргументах и их значениях по умолчанию.

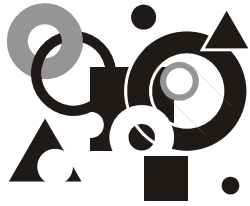
Теперь посмотрите на приведенные табл. 8.1 и 8.2. Постарайтесь вспомнить, что означают написанные в каждой ячейке слова. Может быть, некоторые из вас захотят прочитать про них еще раз.

Таблица 8.1. *Новые инструкции*

Sub <имя_подпрограммы>
End Sub

Таблица 8.2. *Новые объекты и методы*

Объекты	Методы	Аргументы	Константы
Selection	MoveRight MoveLeft	Unit	wdCharacter wdWord
		Count	
		Extend	wdExtend wdMove
	Cut		
	Past		



Глава 9

Лишние инструкции

Влияние настроек на работу макросов. Побочные эффекты при выполнении макросов. Блок **With**. Объект Options и диалоговое окно **Параметры**. Ссылка на активный документ. Как с помощью свойств устанавливать флажки в диалоговых окнах. Как в макрос попали лишние инструкции.

Чтобы читать и даже понимать прочитанное, не обязательно знать все слова, которые есть в языке. В первую очередь надо научиться распознавать структуру текста. Это в полной мере относится и к алгоритмическим языкам, на которых составляются программы для компьютеров. В этом отношении нам повезло. Структура подпрограмм, которые создаются при автоматической записи макросов, очень простая, и, прочитав эту главу, вы познакомитесь со всеми типами инструкций, с помощью которых составлены эти подпрограммы.

В этой главе рассказывается об одной важной особенности записи макросов. Если при записи макросов открывается диалоговое окно, в текст подпрограммы записывается вся информация о настройках, указанных в этом окне. Это удобно, если макрос используется как прототип будущей подпрограммы, поскольку из текста записанной подпрограммы сразу видно, как нужно задавать эти настройки. Однако во многих случаях эта информация является лишней и даже вредной. Об этом мы вам расскажем в данной главе.

Вам, наверное, еще в детстве говорили о пользе чтения. Так вот, когда вы научитесь читать автоматически записанные подпрограммы, тогда вы сможете безбоязненно удалять лишние инструкции, совершенствуя свои макросы.

9.1. Влияние настроек на работу макросов

Приложение Microsoft Word обладает широкими возможностями настройки в соответствии с конкретными потребностями пользователя. Это, разумеется, его существенное преимущество. Но за все хорошее приходится платить:

поскольку от настройки существенно зависит поведение приложения Microsoft Word, макросы, записанные на одном компьютере, на другом компьютере могут повести себя непредсказуемо.

Рассмотрим, например, как работает макрос, который переставляет местами два соседних слова. Начнем с того, что запишем этот макрос.

Ситуация

При редактировании какого-либо предложения вы решили поменять местами два соседних слова, между которыми находится курсор.

Что требуется получить

Вы нажимаете сочетание клавиш, и слова, между которыми находится курсор, мгновенно меняются местами.

Способ решения

Придумать решение этой задачи несложно. Например, можно во время записи макроса выделить слово, стоящее слева от курсора, вырезать его, поместив в буфер обмена, переместить курсор на одно слово вправо и вставить выделенное слово из буфера обмена. Прежде чем записывать макрос, попробуйте выполнить эти действия в двух случаях: когда первоначально курсор находится слева от пробела, разделяющего слова, и когда справа от этого пробела. Оказывается, чтобы получить требуемый результат, необходимо в этих двух случаях выполнять различные действия. Неужели для каждого случая нужен свой макрос? Оказывается, есть универсальное решение (при создании макросов часто возникают подобные задачки, и мы надеемся, что многим из наших читателей будет интересно их решать).

Одно из возможных решений: в качестве первого шага перейдите на одно слово влево, а уж потом выделите это слово.

Теперь представьте себе, что вы случайно выполнили макрос и переставили соседние слова. В этой ситуации хотелось бы, чтобы при повторном применении макроса слова возвращались на исходное место. Для этого в конце записи макроса сделайте еще одно действие: переместите курсор на одно слово влево, чтобы он оказался между переставленными словами.

Решение задачи

Вы уже знаете, как записывать макросы. Назовите этот макрос **WordFlip_1** (*Word* — слово, *Flip* — переворот), в качестве описания используйте следующий текст: "Перестановка соседних слов".

Прежде чем начать запись макроса, установите курсор между словами, после которых стоят пробелы (а не знаки препинания).

После перехода в режим записи макроса нажмите последовательно следующие сочетания клавиш:

1. <Ctrl>+<←> (переместить курсор на одно слово влево).
2. <Ctrl>+<Shift>+<→> (выделить слово).
3. <Ctrl>+<C> или <Shift>+<Del> (вырезать выделенное слово в буфер обмена).
4. <Ctrl>+<→> (переместить курсор на одно слово вправо).
5. <Ctrl>+<V> или <Shift>+<Ins> (вставить слово из буфера обмена).
6. <Ctrl>+<←> (переместить курсор на одно слово влево).

Макрос **WordFlip_1** готов. Прежде чем начать с ним экспериментировать, сверим настройки. В меню **Сервис** выберем пункт **Параметры** и в открывшемся одноименном окне перейдем на вкладку **Правка**. Убедимся, что установлен флажок **учитывать пробелы при копировании**.

> ***Примечание***
>
> Начиная с версии Microsoft Office Word 2002 этот флажок помещен в группу **Параметры вырезания и вставки** и называется **учитывать пробелы**. Кроме того, появились дополнительные возможности учета пробелов при копировании, но для нашего примера это не имеет принципиального значения. Все приведенные нами примеры можно использовать в любых версиях, начиная с Microsoft Word 97.

Проверим, что макрос **WordFlip_1** работает должным образом. Более того, оказывается, его можно использовать и в тех случаях, когда после переставляемых слов есть знак препинания, что, разумеется, и приятно, и полезно. Однако эта возможность исчезает, если снят флажок **учитывать пробелы при копировании**.

Что же это означает? Неужели надо всю оставшуюся жизнь помнить, что при использовании макроса **WordFlip_1** должен быть установлен этот флажок? Пожалуй, легче во время записи макроса в первую очередь установить этот флажок и больше об этом не беспокоиться. Мы, конечно, понимаем, что это компромиссное решение. Для работы с некоторыми видами текста этот флажок должен быть снят. Однако нам удобно начать именно с этого простого варианта, а дальнейшие усовершенствования оставим на будущее *(см. гл. 13)*.

9.2. Стоит только на минутку открыть окно...

Читатель уже, наверное, подумал, что слова ему переставлять приходится не часто и что ради того, чтобы макросом переставлять слова, между которыми есть знаки препинания, не стоит огород городить. С этим доводом можно

согласиться, но мы рассчитываем, что наш читатель, уже научившийся записывать макросы, будет применять свое уменье на практике, а на этом простейшем примере мы хотим рассказать о подводных камнях, которые он может встретить на своем пути.

Для того чтобы понять проблему, которую мы собираемся обсуждать в этом разделе, посмотрим, как работает макрос, который перед перестановкой слов устанавливает флажок **учитывать пробелы при копировании**.

Прежде чем начать запись такого макроса (с именем **WordFlip_2** и описанием "Перестановка соседних слов, учет пробелов при копировании"), в меню **Сервис** выберите пункт **Параметры**, в открывшемся одноименном окне перейдите на вкладку **Правка** и, если флажок **учитывать пробелы при копировании** не установлен, установите его.

После перехода в режим записи макроса в меню **Сервис** выберите пункт **Параметры**. В открывшемся одноименном окне вы увидите вкладку **Правка**. Ничего не изменяя на этой вкладке, нажмите кнопку **OK**. Затем выполните те же действия, которые выполнялись при записи макроса **WordFlip_1**.

Тем, кто записал таким образом макрос **WordFlip_2**, наверное, будет интересно узнать, что этот макрос устанавливает флажок **учитывать пробелы при копировании** несмотря на то, что при записи этого макроса состояние этого флажка не менялось. Эта особенность важна, обратите на нее внимание. Убедитесь сами, что макрос **WordFlip_2** одновременно с перестановкой слов устанавливает флажок **учитывать пробелы при копировании**.

Теперь вас уже не должно удивить, что макрос **WordFlip_2** не только устанавливает этот флажок, но и восстанавливает значение всех параметров, имеющихся на вкладке **Правка**, к тому состоянию, которое было в момент записи макроса.

Например, измените на вкладке **Правка** состояние флажка **Режим замены** (это эквивалентно нажатию клавиши <Insert>). После выполнения макроса **WordFlip_2** состояние этого флажка восстановится, что может быть неприятной неожиданностью для пользователя нашего макроса.

Эта особенность имеет общий характер. Стоит только во время записи макроса изменить один из параметров в каком-нибудь диалоговом окне (или даже ничего не менять, а только открыть диалоговое окно и закрыть его, нажав кнопку **OK**), и макрос будет с упорством, достойным лучшего применения, восстанавливать значения всех параметров, которые вы видели (не меняли, только видели!) в этом окне во время записи макроса.

Итак, можно сформулировать следующее правило.

Правило

Если при записи макроса открывается диалоговое окно, то в макрос записываются значения всех параметров, содержащихся в этом окне. При последующих

запусках макроса все эти параметры получат те же значения, которые были для них установлены во время записи макроса.

Такой "побочный эффект" снижает качество макроса; более того, он часто оказывается неожиданным даже для его автора, и тогда последствия применения данного макроса могут быть непредсказуемыми.

Решить эту проблему совсем не сложно, достаточно удалить из макроса лишние строки. Те, кто не дочитал до этого места, ограничившись изучением первой части книги, об этом не знают, и им придется смириться с таким недостатком автоматически записанных макросов, а вы скоро научитесь его исправлять. Однако сначала мы с вами разберем автоматически записанные подпрограммы `WordFlip_1` и `WordFlip_2`.

9.3. Новые элементы языка Visual Basic

Сначала посмотрим на подпрограмму, созданную при записи макроса **WordFlip_1** (листинг 9.1). О том, как перейти к тексту подпрограммы, рассказано в *разд. 6.1.*

Листинг 9.1

```
Sub WordFlip_1()
'
' WordFlip_1 Макрос
' Перестановка соседних слов
'
    Selection.MoveLeft Unit:=wdWord, Count:=1
    Selection.MoveRight Unit:=wdWord, Count:=1, Extend:=wdExtend
    Selection.Cut
    Selection.MoveRight Unit:=wdWord, Count:=1
    Selection.Paste
    Selection.MoveLeft Unit:=wdWord, Count:=1
End Sub
```

Здесь мы видим новую константу `wdWord` (*word* — слово), показывающую, что перемещение курсора измеряется в словах. Эта константа используется для того, чтобы задавать значение аргументу `Unit` методов `MoveLeft` и `MoveRight` объекта `Selection`. Все остальное вам знакомо. Если вы не можете прочесть текст этого макроса с пониманием, то следует повторить материал, изложенный в *гл. 8.*

Далее в редакторе Visual Basic записан текст подпрограммы `WordFlip_2` (листинг 9.2).

Листинг 9.2

```
Sub WordFlip_2()
'
' WordFlip_2 Макрос
' Перестановка соседних слов, учет пробелов при копировании
'
    With Options
        .ReplaceSelection = True
        .AllowDragAndDrop = True
        .AutoWordSelection = True
        .INSKeyForPaste = False
        .SmartCutPaste = True
        .AllowAccentedUppercase = False
        .PictureEditor = "Microsoft Word"
        .TabIndentKey = False
        .Overtype = False
        .AllowClickAndTypeMouse = False
        .AutoKeyboardSwitching = False
    End With
    ActiveDocument.ClickAndTypeParagraphStyle = "Обычный"
    Selection.MoveLeft Unit:=wdWord, Count:=1
    Selection.MoveRight Unit:=wdWord, Count:=1, Extend:=wdExtend
    Selection.Cut
    Selection.MoveRight Unit:=wdWord, Count:=1
    Selection.Paste
    Selection.MoveLeft Unit:=wdWord, Count:=1
End Sub
```

Примечание

Если вы работаете в версии Microsoft Office Word 2002 или более поздней, полученная при записи макроса подпрограмма будет выглядеть несколько иначе. В ней будут учтены изменения, сделанные в VBA в связи с появлением новых параметров в диалоговом окне **Параметры**. Однако мы надеемся, что после изучения этого раздела вы сможете самостоятельно разобраться во всех инст-

рукциях записанной вами подпрограммы и получить лишнее подтверждение, что вы не зря потратили время и кое-чему все-таки научились.

Как и следовало ожидать, в тексте макроса появились строки, в которых устанавливаются значения параметров, указанных в диалоговом окне **Параметры** на вкладке **Edit**. Часть этих строк желательно удалить, чтобы во время выполнения макроса не восстанавливались значения всех параметров, имевшихся в этом окне на момент записи макроса. Но, чтобы "с водой не выплеснуть ребенка", сначала надо научиться читать эти строки. Итак, посмотрим, что в них записано.

9.3.1. Блок *With*

Так называется последовательность инструкций, начинающихся с инструкции `With` (*with* — это английский предлог, по смыслу похожий на русский предлог с), и кончающихся инструкцией `End With` (конец блока `With`). Такой блок, содержащийся в подпрограмме `WordFlip_2`, выглядит следующим образом:

```
With Options
    .ReplaceSelection = True
    .AllowDragAndDrop = True
    .AutoWordSelection = True
    .INSKeyForPaste = False
    .SmartCutPaste = True
    .AllowAccentedUppercase = False
    .PictureEditor = "Microsoft Word"
    .TabIndentKey = False
    .Overtype = False
    .AllowClickAndTypeMouse = False
    .AutoKeyboardSwitching = False
End With
```

В инструкции `With` после ключевого слова `With` указывается ссылка на объект. В нашем случае указывается ссылка на объект `Options` (параметры). О нем мы расскажем чуть позже, в следующем подразделе. Сейчас важно усвоить, что внутри блока `With` инструкции, начинающие с точки, эквивалентны таким же инструкциям, в которых перед этой точкой указано имя этого объекта. Например, первая после открытия блока инструкция (`.ReplaceSelection = True`) может быть записана в эквивалентном виде:

```
Options.ReplaceSelection = True
```

Благодаря этому не надо в начале каждой инструкции повторять ссылку на объект, которая может состоять из нескольких слов. Это делает программу нагляднее и сокращает текст программы.

Примечание

Строго говоря, эти инструкции не совсем эквивалентны. Блок **With** не только сокращает текст подпрограммы. При открытии блока **With** создается временная ссылка на объект, и последующие инструкции пользуются этой ссылкой. В результате программа работает быстрее, но это не главное. В *разд. 12.2.2* приведен пример, показывающий, что при этом может измениться логика работы программы. При выполнении инструкции **End With** эта временная ссылка удаляется.

Обратите внимание на то, что внутри блока **With** инструкции написаны с дополнительным отступом от левого края. Это делается для удобства чтения программы, и хотя на выполнение программы эти отступы не влияют, мы настоятельно рекомендуем следовать методике ступенчатой записи программ.

9.3.2. Объект *Options* и его свойства

Объект `Options` (параметры) на экране увидеть нельзя. Его придумали для того, чтобы на языке Visual Basic можно было работать с различными параметрами приложения Microsoft Word. Большую их часть можно посмотреть и изменить в диалоговом окне **Параметры** (в английской версии это окно называется так же, как и объект — **Options**), которое можно открыть из меню **Сервис**.

Полезные сведения

При закрытии приложения Microsoft Word значения этих параметров сохраняются и, когда вы откроете его следующий раз, они будут такими же, даже если вы откроете другой документ. Иными словами, это параметры именно приложения Microsoft Word, а не открытого в нем документа или окна, в котором открыт этот документ.

На языке Visual Basic применительно к объектам вместо слова "параметр" употребляется термин *свойство* (property). Свойства объектов записываются так же, как и методы. Иными словами, сначала записывается ссылка на объект, затем ставится точка, а потом (без пробела) имя свойства.

Например, чтобы сослаться на свойство `SmartCutPaste` объекта `Options`, применяется конструкция `Options.SmartCutPaste`.

При записи макроса **WordFlip_2** мы открывали диалоговое окно **Параметры**, чтобы перед перестановкой слов установить флажок **учитывать пробелы при копировании**. В английской версии этот флажок называется **Use smart cut**

and paste (выполнять по-умному операцию вырезания и вставки). Когда данный флажок установлен, автоматически добавляются или удаляются лишние пробелы при операциях удаления и добавления текста, в большинстве случаев освобождая пользователей от выполнения этой скучной работы. По аналогии с английским названием флажка соответствующее свойство объекта `Options` называется `SmartCutPaste`.

> ***Полезные сведения***
>
> Установленному флажку соответствует значение свойства **`True`** (истина), а снятому флажку — значение **`False`** (ложь).
>
> Для присваивания свойству значения используется знак равенства. По этому признаку их иногда можно отличить от методов. Методам ничего присвоить нельзя, а для присваивания значений их аргументам, как мы уже упоминали, используется сочетание двоеточия и знака равенства (`:=`).

Таким образом, чтобы установить флажок **учитывать пробелы при копировании**, используется инструкция:

```
Options.SmartCutPaste = True
```

а чтобы снять этот флажок, инструкция

```
Options.SmartCutPaste = False
```

Для справки приведем названия параметров, собранных на вкладке **Правка** для остальных инструкций, устанавливающих свойства объекта `Options` в подпрограмме `WordFlip_2`.

Все инструкции, в которых устанавливаются свойства, перечисленные в табл. 9.1, попали в подпрограмму при автоматической записи макроса, когда мы открывали вкладку **Правка** в диалоговом окне **Параметры**. Эти инструкции устанавливают значения параметров, которые мы не собирались задавать. Из-за них макрос **WordFlip_2** обладает побочным эффектом, т. е. выполняет не соответствующие его назначению действия, и мы намерены их удалить.

Таблица 9.1. *Некоторые свойства объекта* `Options` *и соответствующие параметры вкладки* ***Правка*** *диалогового окна* ***Параметры***

Свойство	Название параметра (в английской версии)	Название параметра (в русской версии)
`ReplaceSelection`	**Typing replaces selection**	**заменять выделенный фрагмент**
`AllowDragAndDrop`	**Drag-and-drop text editing**	**использовать перетаскивание текста**

Таблица 9.1 *(окончание)*

Свойство	Название параметра (в английской версии)	Название параметра (в русской версии)
AutoWordSelection	**When selecting, automatically select entire word**	**автоматически выделять слова**
INSKeyForPaste	**Use the INS key for paste**	**использовать клавишу INS для вставки**
AllowAccentedUppercase	**Allow accented uppercase in French**	**прописные с надстрочными знаками**
PictureEditor	**Picture editor**	**редактор рисунков**
TabIndentKey	**Tab and backspace set left indent**	**установка отступов клавишами**
Overtype	**Overtype mode**	**режим замены**
AllowClickAndTypeMouse	**Enable click and type**	**разрешить свободный ввод**
AutoKeyboardSwitching	**Auto-Keyboard switching**	**автоматическая смена клавиатуры**

После удаления этих инструкций блок **With** примет следующий вид:

```
With Options
    .SmartCutPaste = True
End With
```

Однако блок **With** с одной инструкцией не дает никаких преимуществ ни при выполнении, ни при чтении подпрограммы. Его можно заменить одной эквивалентной инструкцией:

```
Options.SmartCutPaste = True
```

Преимущество такой замены не только в том, что без блока **With** программа будет быстрее работать и занимать меньше места в памяти. Гораздо важнее, что без этого блока программу легче читать и понимать.

9.3.3. Ссылка на активный документ

В подпрограмме WordFlip_2 встречается еще одна незнакомая нам инструкция, которая отсутствует в подпрограмме WordFlip_1:

```
ActiveDocument.ClickAndTypeParagraphStyle = "Обычный"
```

В этой инструкции устанавливается значение параметра **стиль абзаца по умолчанию** (в английской версии — **Default paragraph style**) в группе **Свободный ввод**. *Свободный ввод* (по-английски *click and type* — щелкни и печатай) это новая функция, появившаяся в Word 2000.

Мы не хотим отвлекаться на ее описание (о ней можно прочесть в электронной справке Microsoft Word). Для нас эта инструкция интересна тем, что в ней используется ссылка на активный документ, т. е. на документ, с которым мы в данный момент работаем.

Слова, из которых составлена данная инструкция, теперь понятны, а синтаксис вам уже знаком. `ActiveDocument` (активный документ) — ссылка на объект, представляющий активный документ. Через точку записано имя свойства `ClickAndTypeParagraphStyle`, которому присваивается строка с именем стиля `"Обычный"`.

Имя стиля `"Обычный"` записано в кавычках. Так обозначаются строки, используемые в качестве значений. Обратите внимание на изображение кавычек. В качестве элемента синтаксиса используются только прямые кавычки. Внутри строки (заключенной в прямые кавычки) можно использовать кавычки любого типа. Если внутри строки требуется использовать прямую кавычку, ее обозначают двумя прямыми кавычками, чтобы отличить от кавычки, закрывающей строку.

Обратите также внимание, что это свойство `ClickAndTypeParagraphStyle` не относится к объекту `Options` (параметры), несмотря на то что оно задает параметр в диалоговом окне **Параметры** (напоминаем, что в англоязычной версии у этого объекта и у диалогового окна одинаковые названия). Поскольку это свойство относится к документу, когда мы откроем другой документ, значение этого свойства (а следовательно, и значение соответствующего параметра) может измениться, в то время как свойства объекта `Options` сохранятся (если вы помните, они относятся к приложению Microsoft Word, а не к конкретному документу). Это важная особенность свойств: они всегда связаны с определенным объектом.

При чтении следующих глав вы еще не раз встретитесь со ссылкой на активный документ, а из данной подпрограммы эту инструкцию мы собираемся удалить вместе с другими лишними инструкциями, попавшими в документ без нашей санкции. В результате мы получим усовершенствованный вариант подпрограммы, приведенный в листинге 9.3.

Листинг 9.3

```
Sub WordFlip_3()
'
' Перестановка соседних слов, учет пробелов при копировании
```

```
' Лишние инструкции удалены
'
    Options.SmartCutPaste = True

    Selection.MoveLeft Unit:=wdWord, Count:=1
    Selection.MoveRight Unit:=wdWord, Count:=1, Extend:=wdExtend
    Selection.Cut
    Selection.MoveRight Unit:=wdWord, Count:=1
    Selection.Paste
    Selection.MoveLeft Unit:=wdWord, Count:=1
End Sub
```

Обратите внимание, что в тексте программы мы оставили пустую строку, разделяющую программу на логически связанные фрагменты. Это стандартный прием, облегчающий чтение программ. Если смысл какой-либо инструкции вам не очевиден, вернитесь к ее описанию, разберитесь в нем, а в текст программы добавьте комментарии, чтобы следующий раз вопросы у вас больше не возникали.

В заключение мы хотим вас порадовать. Основное из того, что нужно, чтобы понимать структуру подпрограмм, созданных при автоматической записи макросов, вы уже знаете. Вы знаете, как записываются инструкции начала и окончания подпрограммы, вы умеете распознавать блок **With**, вам знакомы инструкции, в которых применяются методы, и инструкции, в которых присваиваются значения свойствам. При автоматической записи макросов другие средства языка VBA не используются.

9.4. Что нового в этой главе

Вы узнали, что если во время автоматической записи макроса открывается диалоговое окно, все настройки, которые установлены в этом окне, сохраняются в программе макроса и при запуске макроса восстанавливаются к тому состоянию, которое было во время записи макроса. Во многих случаях это нежелательно. Оказалось, что избавиться от такого побочного эффекта можно, просто удалив часть инструкций.

Вы научились распознавать блок **With** и усвоили, как присваиваются значения свойствам объектов. Таким образом, вам уже знакомы все инструкции, используемые при автоматической записи макросов.

Вы узнали об объекте `Options` и некоторых его свойствах, связанных с параметрами вкладки **Правка** диалогового окна **Параметры**. Теперь вы знаете, как присваиваются значения свойствам объектов и как записываются инст-

рукции, при помощи которых устанавливаются и снимаются флажки. Наконец, вы узнали, как можно сослаться на активный документ, чтобы задать его свойства.

Теперь посмотрите на приведенные табл. 9.2—9.4. Постарайтесь вспомнить, что означают написанные в каждой ячейке слова. Может быть, некоторые из вас захотят прочитать про них еще раз.

Таблица 9.2. *Новые инструкции*

With <имя_объекта>
End With

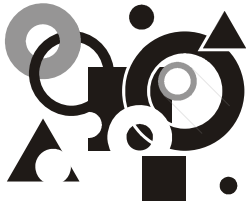
Таблица 9.3. *Новые объекты и свойства*

Объекты	Свойства	Константы
Options	SmartCutPaste	**True** **False**
ActiveDocument	ClickAndTypeParagraphStyle	

Таблица 9.4. *Новые константы*

Объекты	Методы	Аргументы	Константы
Selection	MoveRight MoveLeft	Unit	wdWord

Теперь вернитесь к текстам подпрограмм, приведенным в этой главе. Мы надеемся, что вы сможете прочесть их с пониманием, и это означает, что на данном этапе наша цель достигнута.



Глава 10

В гостях у редактора Visual Basic

Переход в редактор Visual Basic и выход из него. Переключение между окнами редактора Visual Basic и активного документа. Конфликт имен, связанный с одинаковым именованием проектов для всех шаблонов. Переименование проектов. Создание, переименование, удаление, экспорт и импорт модулей. Сохранение изменений. Применение различных режимов отображения окна модуля при копировании подпрограмм.

Вы уже знаете, что в редакторе Visual Basic редактирование подпрограмм выполняется привычным для пользователей Microsoft Windows образом и в нем предусмотрено множество средств, помогающих создавать и отлаживать программы.

Во встроенной электронной справке по Visual Basic и во всех книгах по Visual Basic, которые нам попадались в руки, приводится подробное описание редактора Visual Basic. Для тех наших читателей, которых сильно огорчает, что ни редактор Visual Basic, ни встроенная в него электронная справка не переведены на русский язык, сообщаем, что в русскоязычной версии Microsoft Office 97 меню, диалоговые окна и сообщения редактора Visual Basic, а также справка по нему были на русском языке. Если в вашем распоряжении есть эта версия, ее можно установить в той же операционной системе, в которой у вас установлена более новая версия Microsoft Office (при установке следует указать папку, отличную от той, в которой установлена новая версия). Вам будет приятно убедиться, что редактор Visual Basic в новых версиях практически не изменился.

Примечание

Обратите внимание, что та часть справки, в которой дано описание объектов Microsoft Word, в русскоязычной версии Microsoft Office 97, осталась на английском языке. На русский язык переведено только описание редактора Visual Basic и сведения по программированию, общие для всех приложений Microsoft Office. Эти сведения вам будут полезны при чтении *гл. 14*.

Исходя из сказанного, мы решили не увеличивать объем нашей книги за счет подробного описания редактора Visual Basic. В этой небольшой главе

мы рассказываем только о некоторых его возможностях, знакомство с которыми считаем необходимым читателям нашей книги на данном этапе. Некоторые дополнительные сведения об этом редакторе мы приводим в последующих главах.

10.1. Как попасть в редактор Visual Basic и как найти дорогу обратно

Вы уже знаете, что редактор Visual Basic не является самостоятельным приложением. Чтобы перейти в него, надо сначала открыть Microsoft Word. (Мы не будем рассказывать об особенностях работы с другими приложениями, в которых предусмотрена возможность программирования на языке VBA, такими как Excel, Access, Outlook, PowerPoint и т. д.) Если требуется работать с конкретным макросом, удобнее всего перейти в редактор Visual Basic с помощью диалогового окна **Макрос** (меню **Сервис | Макрос | Макросы** или сочетание клавиш <Alt>+<F8>) (рис. 10.1).

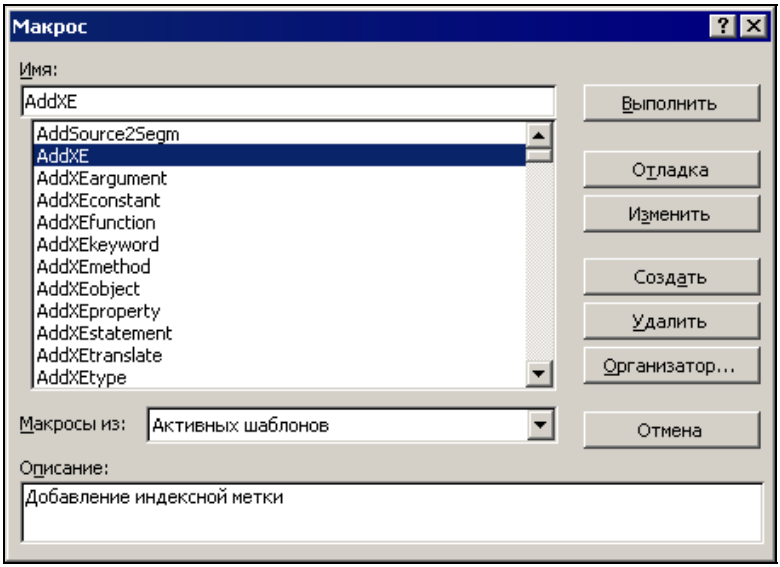


Рис. 10.1. Диалоговое окно **Макрос**

В диалоговом окне **Макрос** кнопки **Отладка** и **Изменить** открывают редактор Visual Basic и выводят на экран макрос, указанный в поле **Имя** (т. е. выбранный в списке под этим полем). В зависимости от нажатой кнопки в редакторе Visual Basic устанавливается режим отладки или редактирования.

Если к конкретному макросу переходить не требуется, открыть редактор Visual Basic можно из меню **Сервис** с помощью последовательного выбора команд **Макрос** и **Редактор Visual Basic** (сочетание клавиш <Alt>+<F11>). Можно также нажать кнопку  на панели **Visual Basic**, но по умолчанию эта панель не отображается (как сделать, чтобы эта кнопка была под рукой, рассказывалось в *гл. 3*). Этот же способ применяется для перехода в редактор Visual Basic, если он уже открыт в другом окне и курсор в нем уже находится в нужном месте. В последнем случае также можно использовать стандартные способы перехода к другому окну, предусмотренные в Microsoft Windows, например, сочетание клавиш <Alt>+<Tab>.

Чтобы закрыть редактор Visual Basic и вернуться в активный документ, можно в меню **File** (Файл) выбрать команду **Close and Return to Microsoft Word** (Закрыть и вернуться в Microsoft Word) или нажать сочетание клавиш <Alt>+<Q>. То же самое можно сделать, просто закрыв окно редактора Visual Basic стандартными способами с помощью сочетания клавиш <Alt>+<F4> или щелкнув кнопку с крестиком в правом верхнем углу окна редактора (учтите, что внутри окна редактора могут быть другие окна с таким же крестиком). Важно, что в этом случае сделанные при работе в редакторе Visual Basic изменения не сохраняются и никаких предупреждений вы не получаете. Разумеется, это не означает, что они будут потеряны, просто запрос о сохранении изменений вы получите позже, когда будете закрывать в Microsoft Word соответствующие файлы. О принудительном сохранении изменений мы расскажем чуть позже.

Чтобы перейти в активный документ, не закрывая редактор Visual Basic, в меню **View** (Вид) выберите команду **Microsoft Word** (Сочетание клавиш <Alt>+<F11>) или нажмите кнопку  на панели инструментов **Standart** (Стандартная), которая обычно расположена в левом верхнем углу окна. Можно также использовать стандартные способы перехода к другому окну, предусмотренные в Microsoft Windows, например, сочетание клавиш <Alt>+<Tab>.

10.2. Окно проектов и окно свойств

В редакторе Microsoft Word вы работаете в окнах документов, а управление и настройка осуществляются с помощью диалоговых окон. В редакторе Visual Basic вы создаете и редактируете тексты программ в окнах **Code** (Программа). Однако в редакторе Visual Basic предусмотрены и другие окна, не похожие на те, с которыми вы встречались в редакторе Microsoft Word. Мы расскажем о двух таких окнах: **Project Explorer** (Окно проектов) и **Properties Window** (Окно свойств). Если их нет на экране, их можно открыть из меню **View** (Вид) с помощью команд **Project Explorer** (Окно проектов) и **Properties**

Window (Окно свойств). Можно также воспользоваться сочетанием клавиш <Ctrl>+<R> и клавишей <F4> соответственно. По умолчанию эти окна закреплены вдоль левой границы окна Visual Basic (рис. 10.2).

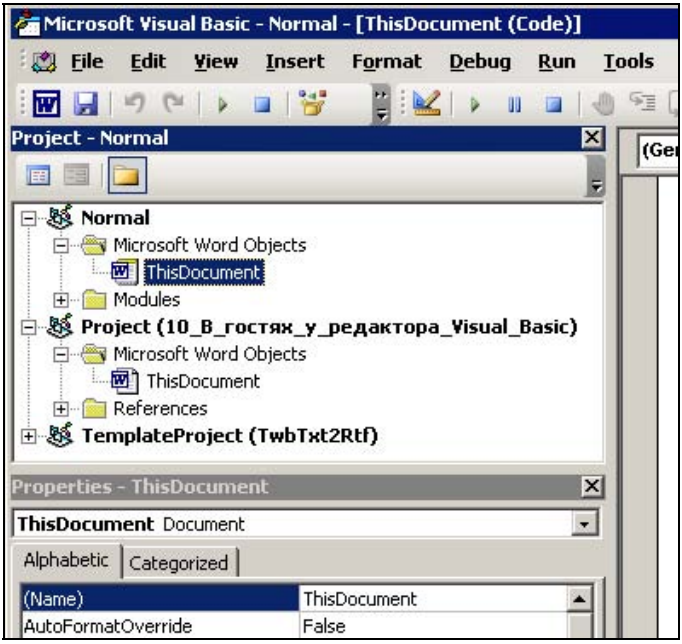


Рис. 10.2. Окна **Project Explorer** и **Properties Window** в окне редактора Visual Basic

Сейчас нам потребуется немного отвлечься, чтобы договориться о терминологии. Вы уже знаете *(см. гл. 7)*, что каждому макросу соответствует подпрограмма, а подпрограммы объединяются в модули. Когда вы создаете макросы в режиме автоматической записи, все подпрограммы записываются в модуль **NewMacros** (Новые макросы). При изучении *гл. 7* вы узнали, что модули можно переименовывать и копировать из других документов и шаблонов. Немного позже мы расскажем, как создавать модули в редакторе Visual Basic. Таким образом, в каждом документе или шаблоне может быть несколько модулей. Объединение подпрограмм в модули позволяет иметь в одном документе или в одном шаблоне несколько различных подпрограмм с одинаковыми именами, хранящихся в различных модулях. При "ручном" программировании модулям можно "поручить" дополнительную нагрузку. В редакторе Visual Basic для каждого модуля (так же как для каждого документа в Microsoft Word) открывается свое окно. Когда вы работаете с документом, вам доступны подпрограммы, которые хранятся в самом документе, в шаблоне Normal.dot, а также в шаблоне документа и загруженных общих

шаблонах *(см. гл. 7)*. Каждому такому документу или шаблону в окне **Project Explorer** соответствует проект (project).

В окне **Project Explorer** проекты и их компоненты представлены в виде дерева с раскрывающимися ветвями и листьями, похожего на дерево папок и файлов в проводнике операционной системы Windows. Так же как и в проводнике, значки со знаками плюс и минус позволяют разворачивать или сворачивать определенные ветви дерева. Модули являются "листьями" этого "дерева" (о других листьях мы в этой книге не рассказываем).

В окне проекта всегда выделен один из проектов или один из компонентов какого-либо проекта. Этот проект является текущим и его имя отображается в заголовке окна после слова **Project**. Это имя играет важную роль, поскольку оно используется при назначении кнопок и сочетаний клавиш для макросов, содержащихся в проектах.

Обратите внимание, по умолчанию имя проекта, соответствующего текущему документу, всегда **Project**, а имя присоединенного шаблона или любого общего шаблона — **TemplateProject** (слово *template* переводится как шаблон). В окне **Project Explorer** после имени проекта указано имя файла, позволяющее различать различные общие шаблоны, но при назначении макросам сочетаний клавиш все шаблоны, имеющие одинаковые имена проектов неразличимы. Это неприятно и превращается в серьезную проблему, если в таких шаблонах есть модули с одинаковыми именами, содержащие одноименные подпрограммы.

Чтобы такие неприятности не происходили с вашими шаблонами, мы настоятельно рекомендуем изменять используемые по умолчанию имена проектов.

Чтобы изменить имя проекта для шаблона, выполните следующие действия:

1. Откройте шаблон в приложении Microsoft Word (так же как вы открываете обычный документ) или присоедините шаблон к текущему документу в качестве его шаблона в диалоговом окне **Шаблоны и надстройки** (меню **Сервис**, команда **Шаблоны и надстройки**).
2. Если данный шаблон загружен в качестве общего, его следует выгрузить, т. е. снять флажок в окне **Общие шаблоны и надстройки** этого же диалогового окна. Загруженные общие шаблоны недоступны для изменения в редакторе Visual Basic.
3. Перейдите в редактор Visual Basic.
4. Выделите имя нужного шаблона в окне **Project Explorer**.

   В заголовке окна свойств появится надпись **Properties - TemplateProject**
5. В окне свойств в поле **Name** (Имя) измените имя проекта. Мы рекомендуем в качестве имени проекта использовать имя соответствующего ему файла.

Таким образом с помощью окон **Project Explorer** (Окно проектов) и **Properties Window** (Окно свойств) вы можете назначить для своего шаблона имя проекта, которое позволит его отличать от других шаблонов при назначении макросам кнопок и сочетаний клавиш.

Аналогичным способом можно изменить имя модуля в любом доступном для изменения проекте. Для этого следует в окне проектов раскрыть дерево проекта, выбрать нужный модуль и в окне свойств изменить его имя.

Вы знаете, как можно создать новый модуль с помощью вкладки **Макросы** окна **Организатор** *(см. гл. 7)*. Для этого достаточно переименовать автоматически создаваемый при записи макросов модуль **NewMacros** либо скопировать модуль из другого шаблона или документа.

Модуль можно создать непосредственно в редакторе Visual Basic.

Чтобы создать модуль, выполните следующие действия:

1. В окне проектов выберите доступный для изменения проект, в котором требуется создать новый модуль.
2. В меню **Insert** (Вставка) выберите команду **Module** (Модуль).

   В дереве проектов раскроется выбранный проект и в нем появится новый элемент с именем **Module1** (если модуль с таким именем уже существует, то цифра на конце имени нового элемента будет другой). Одновременно в заголовке окна свойств появится надпись **Properties - Module1**, в поле **Name** в окне свойств появится значение **Module1**, а справа в окне Visual Basic появится новое окно модуля.
3. (Необязательное действие.) Переименуйте модуль в поле **Name** в окне свойств.

Для тренировки откройте в Microsoft Word новый документ и создайте в нем модуль **Example** (Пример).

Логично предположить, что если есть возможность создавать модули, должна быть предусмотрена возможность удалять их. Это действительно так.

Чтобы удалить модуль **Example**, выполните следующие действия:

1. В окне проектов разверните ветвь доступного для изменения проекта, из которого требуется удалить модуль.
2. Выберите предназначенный для удаления модуль и щелкните его правой кнопкой.
3. В раскрывшемся контекстном меню выберите пункт, начинающийся словом **Remove** (удалить), после которого следует имя выбранного модуля.

   На экран будет выведено диалоговое окно, показанное на рис. 10.3.

Рис. 10.3. Запрос, появляющийся при попытке удаления модуля. Примерный перевод текста сообщения: "Экспортировать «Example» перед удалением?"

4. Нажмите кнопку **Да**, если требуется выполнить экспорт модуля перед удалением, **Нет**, если экспорт выполнять не нужно, или **Отмена**, если вы передумали удалять модуль.

   Если вы решите экспортировать модуль, то откроется диалоговое окно **Export File** (Экспорт файла), в котором потребуется выбрать папку для экспортируемого файла, которому по умолчанию назначается имя, состоящее из имени модуля и расширения bas.

Возможность экспорта и импорта модулей является простым и удобным средством распространения макросов. Эти задачи легко выполняются с помощью команд **Import File** (Импортировать файл) и **Export File** (Экспортировать файл) меню **File** (Файл).

Полученный в результате экспорта модуля файл — это обычный текстовый файл, занимающий мало место на диске. Благодаря этому его удобно хранить и пересылать по электронной почте. Обратите внимание, что в этом файле не содержится информация ни о кнопках, ни о сочетаниях клавиш, назначенных макросам, содержащимся в данном модуле.

Чтобы импортировать модуль из файла с расширением bas, выполните следующие действия:

1. В окне проектов выделите доступный для изменения проект, в который требуется импортировать модуль.
2. В меню **File** (Файл) выберите пункт **Import File** (Импортировать файл).
3. В окне **Import File** (Импорт файла) перейдите к папке, в которой содержится нужный файл с расширением bas, и нажмите кнопку **Открыть**.
4. В выбранном проекте добавится модуль, содержащийся в импортируемом файле. Если модуль с таким именем уже есть в проекте, к имени добавляемого модуля добавится число (1, 2 и т. д.), чтобы имя добавляемого модуля отличалось от имен модулей, содержащихся в данном проекте.

Теперь выполним свое обещание и расскажем о сохранении изменений, произведенных в проектах. Именно в проектах, поскольку единицей хранения является файл, а каждому файлу соответствует проект. Учтите, что, ра-

ботая с одним документом Microsoft Word, вы можете внести изменения не только в этот документ, но и в шаблон этого документа, и в шаблон Normal.dot. Поэтому, когда вы хотите сохранить сделанные в редакторе Visual Basic изменения, вы должны сохранить именно тот проект, к которому эти изменения относятся.

На самом деле надо смотреть на проблему шире: если вы вносите в программу изменения, то должны понимать, где эти изменения сохранятся (в документе или в одном из шаблонов) и для каких документов будет доступна программа, в которую вносятся изменения. Вспомните, когда вы записываете макрос, вы выбираете, куда его записывать: в текущий документ, в шаблон Normal.dot или в шаблон текущего документа. От этого зависит, для каких документов будет доступен этот макрос *(см. гл. 7)*. Это та же самая ситуация.

Чтобы сохранить проект, следует выбрать этот проект в окне проектов и в меню **File** (Файл) выбрать команду, начинающуюся словом **Save** (сохранить), после которого следует имя проекта (как и в Microsoft Word, можно воспользоваться сочетанием клавиш <Ctrl>+<S> или значком ). Если вы забудете сохранить проект, то при закрытии приложения Microsoft Word последует запрос о необходимости сохранения соответствующего этому проекту документа или шаблона, на который необходимо дать утвердительный ответ, иначе вся работа может пропасть.

10.3. Окна модулей

Последняя по счету, но не по значимости функция окна проектов, о которой мы хотим рассказать в этой главе, это открытие нужного модуля и переход от одного модуля к другому. Так же как вы это делаете в Проводнике, раскройте дерево нужного проекта и дважды щелкните необходимый вам модуль. Откроется соответствующее этому модулю окно программы. Если же это окно уже было открыто, то произойдет переход к нему.

С окном модуля **NewMacros** проекта **Normal** вы уже работали при изучении предыдущих глав. Если это окно развернуто, имя файла (именно имя файла без расширения, а не имя проекта, хотя в данном примере они совпадают) и имя модуля отображаются в заголовке окна Visual Basic (рис. 10.4).

Рис. 10.4. Фрагмент заголовка окна редактора Visual Basic, в котором активно окно модуля **NewMacros** шаблона Normal.dot

Если окно не развернуто, имя файла и имя модуля отображаются в заголовке окна.

Справа под заголовком окна модуля расположено поле с алфавитным списком процедур, содержащихся в данном модуле. *Процедурой* (procedure) называется наименьшая часть программы, у которой есть имя. Примером процедуры является подпрограмма, а других процедур вы пока не знаете. С помощью этого списка удобно переходить к нужной процедуре.

Вам уже несколько раз приходилось копировать подпрограмму. Удобнее это делать в режиме, когда в основном окне модуля отображается только одна подпрограмма. Для перехода в этот режим поместите курсор внутри текста нужной подпрограммы и нажмите кнопку **Procedure View** (представление процедуры)  , расположенную в левом нижнем углу окна **Normal - NewMacros (Code)**.

В этом режиме, чтобы выделить весь текст подпрограммы, в меню **Edit** (правка) выберите команду **Select All** (Выделить все) или нажмите сочетание клавиш <Ctrl>+<A>.

Далее скопируйте выделенный текст в буфер обмена. Для этого в меню **Edit** (Правка) выберите команду **Copy** (Копировать) (сочетание клавиш <Ctrl>+ +<C> или значок ).

Вставлять скопированную подпрограмму в текст модуля удобнее в режиме, в котором отображаются все содержащиеся в нем процедуры. Чтобы перейти в этот режим, нажмите кнопку **Full Module View** (Представление полного модуля)  , расположенную в левом нижнем углу окна модуля справа от кнопки **Procedure View**.

После этого в окне модуля будут отображаться все содержащиеся в нем процедуры.

Установите курсор в то место, куда вы хотите вставить новую подпрограмму. Будьте внимательны: курсор должен находиться между процедурами, иначе можно испортить находящуюся в этом месте процедуру.

В меню **Edit** (Правка) выберите команду **Paste** (Вставить из буфера обмена) (сочетание клавиш <Ctrl>+<V> или значок ).

В модуле появятся две одинаковые подпрограммы с одинаковыми именами.

Измените имя одной их этих подпрограмм в инструкции `Sub`. Если этого не сделать, то при попытке выполнения любого макроса из данного модуля откроется редактор Visual Basic, в окне этого модуля отобразится текст одной из одинаковых подпрограмм с выделенным заголовком подпрограммы и на экране появится сообщение, показанное на рис. 10.5.

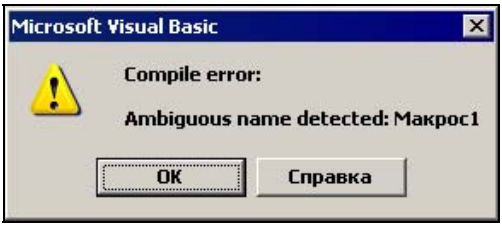


Рис. 10.5. Сообщение об ошибке во время компиляции: обнаружено повторение имен

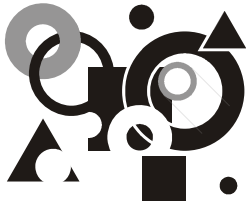
После текста сообщения об ошибке указано повторяющееся имя.

На этом мы прерываем рассказ о редакторе Visual Basic. Чтобы закрепить изложенный в этом разделе материал, мы настоятельно рекомендуем выполнить несколько раз все описанные в этой главе манипуляции с проектами, модулями и подпрограммами, пока вы не убедитесь, что можете делать их без шпаргалки.

Дальнейшие сведения о редакторе Visual Basic мы будем сообщать вам по мере изложения основного материала, но это не заменит вам систематического описания его функций и возможностей, которое содержится во многих книгах по VBA. О некоторых таких книгах мы расскажем в *Заключении*.

10.4. Что нового в этой главе

В этой главе подробно рассказано о различных способах перехода в редактор Visual Basic и о переключении между окном редактора Visual Basic и окном активного документа Microsoft Word. Читатели знакомятся с использованием окна проектов и окна свойств редактора Visual Basic. Приводятся сведения о возможности конфликта имен, связанного с одинаковым именованием проектов для всех шаблонов. Читатели узнают, как дать проектам уникальные имена, чтобы можно было различать шаблоны при назначении макросам кнопок и сочетаний клавиш. Даются инструкции по созданию, переименованию и удалению модулей, экспорту и импорту модулей. Рассказано о сохранении изменений, выполненных в различных проектах в одном сеансе работы с редактором Visual Basic. Читатели знакомятся с основными элементами окна модуля, о переходе в окно нужного модуля. Даны инструкции по копированию модулей с использованием различных режимов отображения окна модуля.



Глава 11

Популярные объекты

Объект и ссылка на объект. Использование свойств для ссылок на объекты. Глобальные свойства объекта `Global`. Понятие семейства. Ссылки на элементы семейств. Объект `Range`. Использование свойств для возвращения значений.

В *гл. 9* мы познакомились со структурой автоматически записываемых подпрограмм. Осталось выучить содержащиеся в этих программах слова и можно будет их читать и понимать. Что это за слова, вы уже знаете, это ключевые слова, имена подпрограмм, свойства и методы, аргументы и константы. Вы обратили внимание, что мы не упомянули про объекты? Это не случайно. Надо признаться, что в тексте подпрограмм имена объектов, как ни странно, не встречаются. Это так, несмотря на то, что мы с вами разбирали инструкции, начинающиеся со слов `Selection` и `Options`, и говорили при этом об объектах `Selection` и `Options`. Просим прощения. Мы не хотели с самого начала вдаваться в эти тонкости, и хотя при описании инструкций мы все время называли первое слово в инструкции не объектом, а ссылкой на объект, мы так и не сказали, что такое ссылка на объект и чем она отличается от объекта. На самом деле это важно, но мы хотели, чтобы вы сначала получили некоторый опыт работы с объектами, чтобы можно было рассказывать на конкретных примерах — так и понятнее, и интереснее. Эта тема составляет основное содержание данной главы. Кроме того, мы познакомимся с новым типом объектов, так называемым семейством, и научимся ссылаться на объекты, являющиеся элементами семейства. Рассказывать все это мы будем на примерах подпрограмм, предназначенных для работы с закладками, которые были получены при автоматической записи макросов. Об этих макросах мы рассказывали в первой части книги. Попутно мы расскажем об объектах, свойствах и семействах, встречающихся в подпрограммах этих макросов.

11.1. Объект *Bookmarks* и свойство *Bookmarks*

Начнем с того, что посмотрим на текст подпрограммы, автоматически созданной *(см. разд. 1.2)* при записи макроса **Marker**, устанавливающего закладку с таким же именем (листинг 11.1).

Листинг 11.1

```
Sub Marker()
'
' Marker Макрос
' Вставка закладки Marker
'
    With ActiveDocument.Bookmarks
        .Add Range:=Selection.Range, Name:="Marker"
        .DefaultSorting = wdSortByName
        .ShowHidden = False
    End With
End Sub
```

Здесь мы встречаем знакомые нам инструкции **Sub**, **End Sub**, **With** и **End With**. Мы уже знаем *(см. разд. 9.3.1)*, что после ключевого слова **With** записывается ссылка на объект. В данном случае ссылка состоит из слов `ActiveDocument` (активный документ) и `Bookmarks` (закладки), разделенных точкой. Объект, на который делается ссылка, — это совокупность всех закладок в активном документе.

На этом примере нам будет удобно объяснить, в чем состоит разница между объектом и ссылкой на него. Как известно, файлы, папки, закладки и макросы имеют имена. Мы их различаем по именам и используем их имена, чтобы сослаться на них. В данном случае применяется другой принцип идентификации объектов. У объектов есть "родовое имя", например, если у вас открыто несколько документов, каждый из них является объектом `Document` (документ). Другими словами, каждый из открытых документов является *экземпляром класса* `Document` (мы уже упоминали, что слово *класс* используется для обозначения "абстрактного" объекта, когда надо отличить его от конкретного объекта). Однако чтобы сослаться на определенный документ, недостаточно сказать, что это объект `Document`. Надо его как-то выделить из множества этих объектов, например, указать, что это активный документ. В *разд. 9.3.3* мы уже говорили, что слово `ActiveDocument` исполь-

зуется именно для ссылки на активный документ. А сейчас добавим, что это ссылка на один из возможных экземпляров класса `Document`.

Другой пример. Если открыто несколько окон с документами, то при переходе в любое окно мы увидим или курсор, или выделенный участок текста. Это означает, что в каждом окне есть свой объект `Selection` (выделение), но только один из этих объектов находится в активном окне, и, хотя мы и не говорили об этом, во всех разобранных ранее подпрограммах, где была ссылка на объект `Selection`, это была ссылка именно на объект, находящийся в активном окне.

В подпрограмме `Marker` мы встречались с объектом `Bookmarks` (закладки). Закладки могут быть в любом документе, и, чтобы сослаться на закладки, находящиеся в определенном документе, сначала указывается ссылка на этот документ, ставится точка, а вслед за точкой без пробела пишется слово `Bookmarks`. Например, ссылка на закладки в активном документе выглядит следующим образом: `ActiveDocument.Bookmarks`.

Вы знаете, что также, как эта ссылка, записываются и свойства объектов. Это не случайное совпадение. В данном случае `Bookmarks` — это не название объекта, хотя записывается оно точно также. Среди свойств объектов `Document` есть свойство `Bookmarks`, значением которого является ссылка на конкретный объект `Bookmarks`, представляющий совокупность закладок в конкретном документе, ссылка на который указана перед точкой. Другими словами, чтобы сослаться на закладки в активном документе, надо указать свойство `Bookmarks` объекта, представляющего активный документ. Еще раз повторяем: названия объектов в тексте подпрограмм не встречаются, `Bookmarks` — это не объект, а свойство, у которого такое же имя.

Это общее правило: чтобы сослаться на один объект, надо найти другой объект, свойством которого является первый объект, и использовать для ссылки это свойство.

Проницательный читатель уже увидел внутреннее противоречие такого способа ссылки на объекты — непонятно, как сослаться на самый первый объект в этой цепочке ссылок. Кроме того, в тех примерах, которые мы разбирали раньше, ссылка на объект состояла только из одного слова, такого как `Selection`, `Options` или `ActiveDocument`.

Действительно, чтобы разрешить это противоречие, авторы языка Visual Basic сделали исключение из этого общего правила для большей части свойств объекта `Application` (приложение), в нашем случае представляющего приложение Microsoft Word. Эти свойства можно указывать сокращенно, без ссылки на объект `Application`. Точка-разделитель также опускается. Например, чтобы указать свойство `ActiveDocument` объекта `Application`, достаточно просто указать глобальное свойство `ActiveDocument` без ссылки на объект `Application`.

Такие свойства называют *глобальными свойствами*. Придумали даже специальный объект `Global` (глобальный), смысл которого только в том, что он обладает всеми глобальными свойствами и методами объекта `Application`, и только ими. Глобальными являются также свойство `Selection` и свойство `Options`, с помощью которых мы ссылались на знакомые нам одноименные объекты.

В отличие от свойства `ActiveDocument`, название этих свойств совпадает с названиями соответствующих объектов, но это не приводит к путанице, а наоборот, позволяет не запоминать лишних названий.

> ***Примечание***
>
> То, что свойством объекта может быть другой объект, видимо, не очень понятно. Это все равно, что сказать, что ножки стула являются его свойством. Но так принято в VBA, и к этому надо привыкнуть. Ничего другого не остается.

Обратите внимание, для каждого объекта есть класс, экземпляром которого он является. Однако класса `ActiveDocument` не существует. Повторяем, `ActiveDocument` — это имя глобального свойства, значением которого является ссылка на конкретный объект `Document`, представляющий активный документ. Иными словами, активный документ является экземпляром класса `Document`.

Вы, может быть, уже задумались, зачем вам знать, чем отличается объект от класса, а ссылка на объект от объекта.

Докладываем. Наша цель — дать основные сведения, которые позволили бы вам в дальнейшем пользоваться справочной литературой и электронной справкой самостоятельно. По своему опыту мы знаем, что основную сложность при этом составляет не изучение новых объектов, свойств и методов, а освоение новых понятий и соответствующей терминологии. Поэтому мы по ходу дела рассказываем о тех достопримечательностях страны Visual Basic, которые встречаются нам на пути, и постепенно изучаем язык, на котором говорят ее аборигены.

11.2. Семейства

Объект `Bookmarks` (закладки) состоит из объектов `Bookmark` (закладка).

Такие объекты, как `Bookmarks`, состоящие из набора родственных объектов, называются `семействами` (collection). По-русски их чаще называют *коллекциями*, но мы будем придерживаться терминологии, принятой в справке Microsoft Office. Объекты, из которых состоит семейство, называют его *элементами* (element).

Обычно название объекта и семейства, в которое он входит, отличаются только буквой `s` на конце слова (для тех, кто совсем не знает английского языка, сообщаем, что так в нем образуется множественное число существительных).

Сейчас мы немного поближе познакомимся с семейством `Bookmarks`. Посмотрим на первую исполняемую инструкцию подпрограммы `Marker`:

```
.Add Range:=Selection.Range, Name:="Marker"
```

Она является частью блока **`With`** и, следовательно, может быть записана в эквивалентном виде:

```
ActiveDocument.Bookmarks.Add Range:=Selection.Range, Name:="Marker"
```

Если вы не можете сразу распознать, что в этой инструкции применяется метод `Add`, следует повторить *разд. 8.6*. Эта инструкция служит для добавления нового элемента в семейство `Bookmarks`, т. е. для создания новой закладки в активном документе. В этой инструкции сначала делается ссылка на объект `Bookmarks`, представляющий все закладки в активном документе, а затем для этого объекта применяется метод `Add` (добавить) семейства `Bookmarks`. Метод с таким названием имеется у многих семейств и всегда используется для добавления в них нового элемента, но набор аргументов этого метода зависит от семейства, к которому он применяется. У нашего метода два аргумента: `Range` (диапазон), задающий участок области документа, на который устанавливается закладка, и `Name` (имя), задающий имя закладки.

Значением аргумента `Range` является ссылка на объект `Range` (т. е. на экземпляр класса `Range`). Для первого знакомства с объектом `Range` (диапазон) нам будет достаточно знать совсем немного: он представляет так называемый *диапазон* непрерывную часть документа или определенную позицию в документе. В окне документа ему ничего не соответствует, он "невидим".

Чтобы задать диапазон, соответствующий выделенной части документа или положению курсора, можно воспользоваться свойством `Range` объекта `Selection`. Значением этого свойства является ссылка на объект `Range`, представляющий или выделенную часть в активной области документа, или, если в ней ничего не выделено, позицию, в которой находится курсор.

Обратите внимание, что здесь опять для задания объекта `Range` используется свойство с тем же именем. Кроме того, также называется и упомянутый ранее аргумент метода `Add`. В *гл. 16* рассмотрен метод `Range` объекта `Document`. Когда первый раз сталкиваешься с тем, что такие разные вещи называются одинаково, голова идет кругом. Однако постепенно к этому привыкаешь, и уже считаешь это достоинством — не надо запоминать много слов, а смысл всегда ясен из контекста.

Следующие две инструкции в подпрограмме `Marker`:

```
.DefaultSorting = wdSortByName
.ShowHidden = False
```

появились в ней потому, что при записи макроса мы открывали диалоговое окно **Закладки**, в котором в числе прочего можно задавать порядок сортировки в списке закладок и устанавливать или снимать флажок, задающий необходимость отображения в этом списке скрытых закладок (мы рассказывали об этих параметрах диалогового окна в *разд. 1.2.2* и еще вернемся к ним в *разд. 15.1*). В записанном нами макросе свойство `DefaultSorting` (порядок сортировки) объекта `Bookmarks` получает значение, заданное константой `wdSortByName` (сортировать по имени), а свойство `ShowHidden` (показывать скрытые) — значение **`False`** (ложь).

Мы уже знаем *(см. разд. 9.2)*, что в таких случаях макрос обладает побочным эффектом. Чтобы от него избавиться, эти лишние инструкции следует удалить. Заодно, также как в *разд. 9.3.2*, целесообразно исключить блок **`With`**, поскольку в нем осталась только одна инструкция.

В результате в модифицированной подпрограмме остается только одна исполняемая инструкция (листинг 11.2).

Листинг 11.2

```
Sub Marker1()
'
' Вставка закладки Marker
' Лишние инструкции удалены

    ActiveDocument.Bookmarks.Add Range:=Selection.Range, Name:="Marker"
End Sub
```

Аналогично можно модифицировать созданный в *разд. 1.1.2* макрос `GoToMarker` перехода к закладке **Marker** (листинг 11.3).

Листинг 11.3

```
Sub GoToMarker()
'
' GoToMarker Макрос
' Переход к закладке Marker
'
    Selection.GoTo What:=wdGoToBookmark, Name:="Marker"
```

```
    With ActiveDocument.Bookmarks
        .DefaultSorting = wdSortByName
        .ShowHidden = False
    End With
End Sub
```

Для перехода к метке **Marker** используется метод `GoTo` (перейти) объекта `Selection`. Аргументы этого метода: `What` (какой) и `Name` (имя). Аргумент `What` задает тип элемента в документе, на который происходит переход. В данном случае значение этого аргумента — `wdGoToBookmark` (перейти к закладке) указывает, что переход происходит к закладке. Аргумент `Name` задает имя закладки.

Обратите внимание, что действие происходит с выделением, а не с закладкой как таковой, поэтому для выполнения этого действия используется метод объекта `Selection` (выделение), а не объекта `Bookmark` (закладка).

Так же как и в предыдущем примере, "лишние" инструкции, содержащиеся в блоке **With**, следует удалить. В результате в модифицированной подпрограмме остается только одна исполняемая инструкция (листинг 11.4).

Листинг 11.4

```
Sub GoToMarker1()
'
' Переход к закладке Marker
' Лишние инструкции удалены
'
    Selection.GoTo What:=wdGoToBookmark, Name:="Marker"
End Sub
```

Теперь методом автоматической записи макроса создайте подпрограмму удаления закладки **Marker**. Назовите ее `DeleteMarker` (удалить Marker), а в качестве описания используйте фразу "Удаление закладки Marker". Наш вариант приведен в листинге 11.5.

Листинг 11.5

```
Sub DeleteMarker()
'
' DeleteMarker Macro
' Удаление закладки Marker
```

```
'
    ActiveDocument.Bookmarks("Marker").Delete
    With ActiveDocument.Bookmarks
        .DefaultSorting = wdSortByName
        .ShowHidden = False
    End With
End Sub
```

Для удаления закладки служит инструкция

```
ActiveDocument.Bookmarks("Marker").Delete
```

в которой применяется метод `Delete` (удалить) объекта `Bookmark` (закладка), представляющего одну закладку. Тут нет опечатки: в инструкции указано семейство `Bookmarks`, а мы говорим о методе объекта `Bookmark`, представляющего один элемент этого семейства. Дело в том, что в этой инструкции используется новый для вас способ ссылки на объект, согласно которому, чтобы сослаться на элемент семейства, можно после ссылки на семейство указать в скобках строку, содержащую имя этого элемента.

В данном случае необходимо сослаться на объект, представляющий содержащуюся в активном документе закладку с именем **Marker**. Сначала мы записываем ссылку на семейство, содержащее этот объект (поэтому и появилась буква `s` на конце слова), а затем в кавычках имя закладки. Кавычки, окружающие имя закладки, позволяют редактору Visual Basic понять, что `"Marker"` это *строка* (string).

Таким образом, ссылка на объект `Bookmark`, представляющий закладку **Marker** в активном документе, имеет следующий вид:

```
ActiveDocument.Bookmarks("Marker")
```

Далее к этому объекту применяется метод `Delete`. Обратите внимание, что это метод объекта `Bookmark`, а не объекта `Bookmarks`, потому что удаляется конкретная закладка, а не все закладки документа.

Как и в предыдущем случае, блок **`With`** можно исключить, и в результате программа принимает вид, представленный в листинге 11.6.

Листинг 11.6

```
Sub DeleteMarker1()
'
' Удаление закладки Marker
```

```
' Лишние инструкции удалены
'
    ActiveDocument.Bookmarks("Marker").Delete
End Sub
```

11.3. Свойство возвращает значение

До сих пор все наши действия сводились к удалению и небольшому преобразованию инструкций, полученных при автоматической записи макросов. Следующий шаг — макросы с инструкциями, которые не могут быть созданы при автоматической записи макроса. Вообще говоря, речь о них пойдет в следующей главе, однако кое-чему можно научить вас уже сейчас на базе имеющихся у вас знаний. Начнем с того, что разберем подпрограмму, полученную при автоматической записи макроса `ColorMarker` *(см. разд. 1.1.2)*, представленную в листинге 11.7.

Листинг 11.7

```
Sub ColorMarker()
' Вставка светло-зеленой закладки Marker
'
    With ActiveDocument.Bookmarks
        .Add Range:=Selection.Range, Name:="Marker"
        .DefaultSorting = wdSortByName
        .ShowHidden = False
    End With
    Selection.MoveRight Unit:=wdWord, Count:=1, Extend:=wdExtend
    Options.DefaultHighlightColorIndex = wdBrightGreen
    Selection.Range.HighlightColorIndex = wdBrightGreen
    Selection.MoveLeft Unit:=wdCharacter, Count:=1
End Sub
```

Можете порадоваться. Этот текст вы уже способны читать с пониманием, несмотря на то, что некоторые слова в нем вам не знакомы.

Инструкция

```
Options.DefaultHighlightColorIndex = wdBrightGreen
```

была создана во время записи макроса в тот момент, когда вы устанавливали ярко-зеленый цвет в инструменте выделения цветом. Она знакомит вас с

новым свойством объекта `Options` (параметры). В этой инструкции устанавливается свойство `DefaultHighlightColorIndex` (стандартный цвет подкрашивания) равным значению константы `wdBrightGreen` (ярко-зеленый). Обратите внимание, что это свойство объекта `Options` (параметры) не связано ни с каким параметром диалогового окна **Параметры**. Таким образом, у диалогового окна и объекта, имеющих в англоязычной версии Microsoft Word одинаковые имена, набор параметров не полностью совпадает с набором свойств. Вспомните также о параметре **стиль абзаца по умолчанию**, расположенном в группе **Свободный ввод** на вкладке **Правка** (о нем мы говорили в конце *гл. 9*). Соответствующее этому параметру свойство отнюдь не является свойством объекта `Options`.

Следующая инструкция

```
Selection.Range.HighlightColorIndex = wdBrightGreen
```

подкрашивает ярко-зеленым цветом выделенный диапазон. В ней устанавливается значение `wdBrightGreen` свойства `HighlightColorIndex` (цвет подкрашивания) объекта `Range` (диапазон), представляющего диапазон, соответствующий выделенному участку документа. Вы уже знаете, что выражение `Selection.Range` представляет собой (программисты говорят "возвращает") ссылку на диапазон, соответствующий выделенному участку текста в активном документе.

Обратите внимание, что цвет подкрашивания является свойством диапазона в документе (объекта `Range`), а не выделения (объект `Selection`). Выделение может переместиться, а цвет останется.

Без первой из этих инструкции можно обойтись, и если бы перед записью макроса мы заранее установили нужный нам цвет, а во время записи макроса просто щелкнули значок **Выделение цветом**  , не выбирая цвета, то первой инструкции в подпрограмме бы не было. Вы можете легко в этом убедиться. Из-за наличия первой инструкции макрос обладает побочным эффектом. После выполнения макроса не только подкрашивается светло-зеленым выделенный участок, но и устанавливается светло-зеленый цвет средства подкрашивания на значке **Выделение цветом**. Если исключить лишнюю инструкцию, то при выполнении макроса цвет на значке не будет изменяться, что, на наш взгляд, удобнее. Однако следует признаться, что мы записали макрос с этой лишней инструкцией намеренно, и вы сейчас поймете, почему мы это сделали.

Мы уже знаем, что многим свойствам объектов можно присваивать значения, выраженные константами. Оказывается, вместо константы можно использовать любое выражение, значение которого совпадает с одной из допустимых констант. Воспользуемся тем, что свойству `Options.DefaultHighlightColorIndex` можно не только присваивать значение, но и записывать его вместо кон-

станты, соответствующей цвету, установленному на значке **Выделение цветом**  . В инструкции

```
Selection.Range.HighlightColorIndex = wdBrightGreen
```

заменим константу выражением `Options.DefaultHighlightColorIndex`, которое скопируем из первой инструкции:

```
Selection.Range.HighlightColorIndex = _
    Options.DefaultHighlightColorIndex
```

Обратите внимание, что здесь мы воспользовались знаком подчеркивания (_) с предшествующим пробелом и последующим переводом строки, чтобы записать длинную инструкцию в двух строчках. В соответствии с принципом ступенчатой записи инструкций мы в начале второй строки добавили пробелы.

Далее удалим первую инструкцию. В результате получим подпрограмму, приведенную в листинге 11.8.

Листинг 11.8

```
Sub ColorMarker1()
' Вставка закладки Marker с подсветкой текущим цветом
'
    ActiveDocument.Bookmarks.Add Range:= _
        Selection.Range, Name:="Marker"
    Selection.MoveRight Unit:=wdWord, Count:=1, Extend:=wdExtend
    Selection.Range.HighlightColorIndex = _
        Options.DefaultHighlightColorIndex
    Selection.MoveLeft Unit:=wdCharacter, Count:=1
End Sub
```

Макрос `ColorMarker1` устанавливает закладку и выделяет ее тем цветом, который установлен на значке **Выделение цветом**  . Такой макрос методом автоматической записи создать нельзя, поскольку в подпрограммах, созданных с помощью этого метода, свойствам присваиваются только константы.

На примере подпрограммы `ColorMarker1` вы увидели, что свойствам можно не только присваивать значения, но и извлекать содержащиеся в них значения (на языке программистов свойство "возвращает" значение). Однако так можно обращаться не со всеми свойствами. В разделе электронной справки по Visual Basic, в котором описывается конкретное свойство (а такой раздел есть для каждого свойства), указано, можно ли присваивать значение свой-

ству (write — запись) и можно ли извлечь значение, которым обладает свойство (read — чтение). Чаще всего вы увидите признак write/read, означающий, что можно делать и то, и другое.

11.4. Дополнительные сведения: сходство и различие объектов *Range*, *Bookmark* и *Selection*

Все три объекта: `Range` (диапазон), `Bookmark` (закладка) и `Selection` (выделение) представляют неразрывные участки документа. Одновременно может существовать столько закладок и столько диапазонов, сколько нужно программисту, но в каждой области окна может быть только одно выделение. Выделение всегда отображается на экране. Для этого оно и предназначено. Закладки отображаются на экране, только если установлен соответствующий параметр приложения Microsoft Word. В отличие от выделения и закладок диапазоны никогда не отображаются на экране, они существуют, только пока выполняется макрос, в котором они определяются и используются. Диапазоны для того и придуманы, чтобы использовать их для работы с отдельными участками текста в программах.

Для любой закладки можно определить диапазон, на который она установлена, и выделить этот диапазон. На любое выделение можно установить закладку, и для любого выделения можно определить диапазон, соответствующий этой закладке. На любой диапазон можно установить закладку, чтобы запомнить его в документе после окончания работы макроса, и любой диапазон можно выделить, чтобы показать его пользователю.

11.5. Что нового в этой главе

Вы узнали разницу между объектом и ссылкой на него. Вы научились ссылаться на объект, используя свойство другого объекта. Вы познакомились с глобальными свойствами, которыми обладает объект `Global`. На примере объекта `Bookmarks` вы усвоили понятие семейства. Получили первое представление об объекте `Range`. Узнали, как можно сослаться на объекты, являющиеся элементами какого-либо семейства. Научились использовать свойства в качестве значений, присваиваемых другим свойствам.

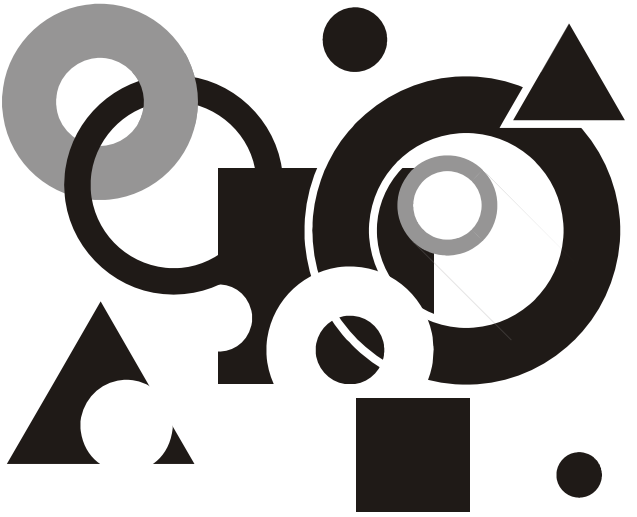
Теперь посмотрите на приведенные далее табл. 11.1—11.2. Постарайтесь вспомнить, что означают написанные в каждой ячейке слова. Может быть, некоторые из вас захотят прочитать про них еще раз.

Таблица 11.1. *Новые объекты и методы*

Объект	Метод	Аргумент	Константа
Document			
Bookmarks	Add	Range Name	
Selection	GoTo	What	wdGoToBookmark
Bookmark	Delete		

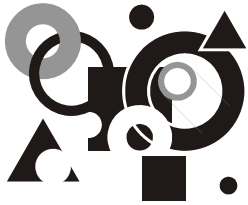
Таблица 11.2. *Новые свойства объектов*

Объект	Свойство	Константа
Document	Bookmarks	
Selection	Range	
Application	ActiveDocument Selection Options	
Global	ActiveDocument Selection Options	
Bookmarks	DefaultSorting ShowHidden	wdSortByName
Options	DefaultHighlightColorIndex	wdBrightGreen
Range	HighlightColorIndex	



ЧАСТЬ III

ЭЛЕМЕНТЫ ПРОГРАММИРОВАНИЯ



Глава 12

Поиск текста и открытие документа

Решение задач, связанных с поиском выделенного текста методом небольших изменений в тексте автоматически записанных макросов. Поиск следующего и предыдущего вхождений выделенного текста. Поиск выделенного текста в справочном документе. Инструкции для выполнения поиска и замены. Особенности поиска при использовании макросов. Инструкции открытия документа. Влияние блока `With` на выполнение программы. Подкрашивание текущим цветом всех вхождений выделенного текста.

Этой главой начинается новая часть, в которой мы будем учиться программировать на качественно новом уровне. Если до сих пор мы только немного изменяли автоматически записанные подпрограммы, то теперь мы будем добавлять в них "свои слова". В этой главе мы пока не будем использовать новых структур. Основой создаваемых нами макросов по-прежнему будут автоматически записанные подпрограммы, однако без "ручного" программирования создание таких макросов уже невозможно.

12.1. Поиск других вхождений выделенного текста

Мы уже посвятили целую главу изучению возможностей поиска в Microsoft Word и методом автоматической записи создали несколько полезных макросов, использующих эти возможности *(см. гл. 4)*. Теперь наша цель — подробно разобрать текст получившихся при этом подпрограмм и, понимая, как они работают, создать на их основе новые макросы, решающие более широкий круг задач.

12.1.1. Постановка задачи

Очень часто возникает задача поиска следующего вхождения того или иного фрагмента текста. Казалось бы, совсем несложно выделить его, скопировать в буфер обмена, открыть диалоговое окно **Найти и заменить**, вставить скопированный текст в поле **Найти** и нажать кнопку **Найти далее**. Однако если эти действия повторяются многократно, то совсем неплохо автоматизировать их с помощью макроса, который находил бы следующее (или предыдущее) вхождение выделенного текста. В *гл. 4* мы решали похожие задачи, и вы знаете, как записать макрос, с помощью которого можно найти следующее вхождение конкретного словосочетания. Однако, не прибегая к программированию, создать макрос для поиска следующего вхождения выделенного текста не удается.

Исследование

Начнем с того, что запишем пробный макрос **Test1**, в котором зафиксируем действия, которые мы выполняем при решении поставленной задачи вручную. В каком-нибудь документе выделим сочетание букв "по" (которое наверняка неоднократно в нем встречается) и попытаемся найти его следующее вхождение, включив запись макроса.

1. Выделите сочетание букв "по".
2. Приступите к записи макроса с именем **Test1** и описанием "Поиск выделенного сочетания букв «по»".
3. Скопируйте выделенный текст в буфер обмена.
4. Откройте окно **Найти и заменить**, вкладку **Заменить**.

   Зачем требуется переходить на эту вкладку, когда мы ничего заменять не собираемся, мы объясним немного позже.
5. В поле **Найти** вставьте из буфера обмена скопированный ранее текст.

   Обратите внимание, что в поле **Направление** указано значение **Вперед**. Это связано с тем, что при открытии диалогового окна **Найти и заменить** в окне документа был выделен текст. В противном случае в окне **Направление** по умолчанию было бы указано значение **Везде**.
6. Проверьте, что сняты все флажки диалогового окна **Найти и заменить**, кнопка **Снять форматирование** недоступна (в противном случае нажмите ее) и на вкладке **Заменить** в поле **Заменить на** пусто (чтобы убедиться в том, что в этом поле нет пробелов, выделите его).
7. Нажмите кнопку **Найти далее**.

   Выделится следующее вхождение сочетания букв "по".

8. Нажмите кнопку **Отмена**.
9. Остановите запись макроса.

Попробуйте выполнить записанный макрос. Нетрудно убедиться, что он умеет искать следующее вхождение выделенного текста только в том случае, если этот текст представляет собой сочетание букв "по". Чтобы разобраться в причинах нашей неудачи, посмотрим на текст макроса. Вы уже знаете, как это сделать. Автоматически записанная подпрограмма приведена в листинге 12.1 (мы только выделили курсивом инструкцию, содержащую строку "*по*").

Листинг 12.1

```
Sub Test1()
'
' Test1 Макрос
' Поиск выделенного сочетания букв "по"
'
    Selection.Copy
    Selection.Find.ClearFormatting
    With Selection.Find
        .Text = "по"
        .Replacement.Text = ""
        .Forward = True
        .Wrap = wdFindAsk
        .Format = False
        .MatchCase = False
        .MatchWholeWord = False
        .MatchWildcards = False
        .MatchSoundsLike = False
        .MatchAllWordForms = False
    End With
    Selection.Find.Execute
End Sub
```

На данном этапе мы не будем учиться читать эти инструкции. Этим мы займемся немного позже *(см. далее разд. 12.1.2)*, а сейчас воспользуемся опытом, полученным при изучении *гл. 6.* В *разд. 6.4.1* вы уже анализировали похожую подпрограмму и знаете, что текст, который ищет макрос, записан в инструкции

```
.Text = "по"
```

Естественно предположить и несложно проверить, что вместо строки "по" в эту инструкцию можно записать любую другую строку, и макрос будет ее послушно искать. Однако когда мы записывали макрос **Test1**, мы надеялись, что он будет искать текст, который выделен в окне документа. Мы скопировали в буфер обмена выделенный текст и вставили его в поле **Найти**, но в подпрограмму эти действия не записались.

К счастью, решить поставленную задачу нам может помочь уже знакомый нам объект `Selection` (выделение). У него есть свойство `Text` (текст), возвращающее текст в выделенной части активного окна документа, а если в окне ничего не выделено, оно возвращает символ, стоящий справа от курсора. Следует учитывать, что это свойство возвращает выделенный текст без информации о форматировании. В данном случае это нам подходит, поскольку при поиске не требуется принимать во внимание особенности форматирования текста. Короче говоря, вместо строки "по" в инструкции `.Text = "по"` следует записать `Selection.Text`.

Это еще один шаг на нашем пути к вершинам программирования. Мы впервые пишем "от себя", а не копируем фрагменты автоматически записанного макроса.

Для надежности скопируйте подпрограмму `Test1` и переименуйте полученную копию, дав новой подпрограмме имя `Test2` (и соответственно подправьте комментарий). Кроме того, замените строку `"по"` на `Selection.Text`. У вас получится подпрограмма, приведенная в листинге 12.2.

Листинг 12.2

```
Sub Test2()
'Поиск выделенного текста
    Selection.Copy
    Selection.Find.ClearFormatting
    With Selection.Find
        .Text = Selection.Text
        .Replacement.Text = ""
        .Forward = True
        .Wrap = wdFindAsk
        .Format = False
        .MatchCase = False
        .MatchWholeWord = False
        .MatchWildcards = False
        .MatchSoundsLike = False
```

```
        .MatchAllWordForms = False
    End With
    Selection.Find.Execute
End Sub
```

Попробуйте ее в деле. Она работает! Давайте теперь разберемся, как это у нее получается.

Первая исполняемая инструкция нам знакома. Это копирование выделенного текста в буфер обмена. Мы пользовались этой операцией при записи макроса, чтобы скопировать выделенное значение в буфер обмена, а затем вставить его в поле **Найти** диалогового окна. В текст подпрограммы операция вставки не записалась, записался только результат этой операции. Это не случайно. Сформулируем общее правило.

При записи макроса операции, которые мы выполняем, когда активно какое-либо диалоговое окно, в программу макроса не записываются. Записывается только результат этих операций. В момент закрытия диалогового окна создаются инструкции, в которых значения параметров диалогового окна присваиваются свойствам каких-либо объектов или используются как значения аргументов методов, выполняемых над некоторыми объектами. Те операции, которые мы выполняли, чтобы изменить значения параметров диалогового окна (ввод данных с клавиатуры, копирование из буфера обмена, операции с мышью) в программе макроса не записываются.

Когда мы копировали текст из буфера обмена в поле **Найти** диалогового окна **Найти и заменить**, это диалоговое окно, естественно, было открыто. Поэтому в подпрограмме никак не отражена операция вставки текста из буфера обмена в это поле. Инструкция копирования выделенного текста в буфер обмена оказалась лишней и даже вредной. Вредной потому, что эта инструкция заменяет выделенным текстом то, что было в буфере обмена до запуска макроса и в результате работы макроса теряется содержимое буфера обмена, которое, вообще говоря, нам может в дальнейшем еще пригодиться. Другими словами, из-за наличия этой инструкции макрос обладает неприятным побочным эффектом. Мы с подобной ситуацией уже сталкивались и знаем, что в таких случаях лишние инструкции следует удалять.

Далее в тексте макроса записаны инструкции, изучая которые, мы познакомимся с новым важным объектом, который заслуживает отдельного подраздела.

12.1.2. Объект *Find*

Посмотрим на вторую исполняемую инструкцию подпрограммы `Test2` (см. листинг 12.2).

```
Selection.Find.ClearFormatting
```

Начнем с перевода новых английских слов: *Find* — найти; *Clear* — очистить; *Formatting* — форматирование. Действие этой инструкции эквивалентно нажатию кнопки **Снять форматирование** в диалоговом окне **Найти и заменить** (если же эта кнопка не видна, то следует нажать кнопку **Больше**).

Конструкция `Selection.Find`, как уже понял внимательный читатель, является обращением к свойству `Find` объекта `Selection`, которое возвращает ссылку на объект `Find`.

Некоторая трудность состоит в том, что этот объект уже не такой наглядный, как те, с которыми мы знакомились до сих пор. Его назначение — предоставить пользователю свойства и методы для выполнения поиска, реализующие примерно те же возможности, что и вкладка **Найти** диалогового окна **Найти и заменить**. Например, в рассматриваемой нами инструкции вызывается метод `ClearFormatting`, который, как мы уже сказали, выполняет ту же функцию, что и кнопка **Снять форматирование**. Эту инструкцию рекомендуется записывать перед другими инструкциями, в которых задаются условия поиска, чтобы убрать нежелательные условия форматирования, которые могли остаться от предыдущего поиска. Хотя мы и не нажимали на соответствующую кнопку, предусмотрительный редактор Visual Basic эту инструкцию записал в наш макрос.

Назначение следующей инструкции

```
With Selection.Find
```

вам знакомо. Она открывает блок **`With`**, внутри которого, т. е. до инструкции **`End With`**, можно опускать слова `Selection.Find` в начале инструкций. Это сокращает запись программы, но как мы увидим немного дальше, этим его действие не ограничивается.

Инструкцию

```
.Text = Selection.Text
```

мы частично записали сами. Слева от знака равенства указано свойство `Text` (текст) объекта `Find`, соответствующее полю **Найти** вкладки **Найти** диалогового окна **Найти и заменить**. Справа мы записали свойство `Text` объекта `Selection`, которое возвращает выделенный нами текст. Знак равенства, как вы помните, обозначает операцию присваивания. Таким образом, эта инструкция вставляет выделенный текст в поле **Найти** диалогового окна **Найти и заменить**, что, разумеется, уже не является для вас неожиданностью.

В следующей инструкции

```
.Replacement.Text = ""
```

используется свойство `Replacement` (замена) объекта `Find`, которое возвращает ссылку на объект `Replacement`. Этот новый для вас объект пре-

доставляет пользователю свойства и методы, реализующие примерно те же возможности замены, что и вкладка **Заменить** диалогового окна **Найти и заменить**.

С первого взгляда не очень понятно, почему ссылка на объект `Replacement` возвращается свойством объекта `Find`. Получается, что объект `Replacement` как бы является частью объекта `Find`. После некоторого размышления можно объяснить это тем, что замену можно считать просто одной из возможностей поиска.

В рассматриваемой нами инструкции устанавливается свойство `Text` (текст) объекта `Replacement`, соответствующее полю **Заменить на** вкладки **Заменить** диалогового окна **Найти и заменить**.

Строка `""`, которая присваивается этому свойству, называется *пустой строкой*, т. е. строкой, не содержащей ни одного символа и имеющей нулевую длину. Когда свойству присваивается пустая строка, соответствующее ему поле в диалоговом окне очищается.

Сразу непонятно, почему в процедуре поиска появилась эта инструкция, когда никакой замены мы не делали. Дело здесь не в особенностях Visual Basic, а в том, что при использовании вкладки **Найти** выполнение поиска, как ни странно, зависит от того, какое значение указано на вкладке **Заменить** в поле **Заменить на**. Действительно, попробуйте ввести в это поле символ ^ (на клавиатуре он находится над цифрой 6), перейти на вкладку **Найти**, ввести в поле **Найти** какой-нибудь текст и выполнить поиск. Вы получите сообщение об ошибке. Оказывается, чтобы исправить ошибку при поиске, надо изменить значение в поле **Заменить на**. При этом ошибочное значение могло быть занесено при предыдущей попытке поиска. Если в нашей подпрограмме удалить инструкцию, которая очищает поле **Заменить на**, то при наличии ошибочного значения в строке замены макрос работать не будет. Поэтому при записи макроса мы переходили на вкладку **Заменить** и проверяли, что в поле **Заменить на** пусто.

К объекту `Replacement` мы вернемся позже, а сейчас рассмотрим следующую инструкцию:

```
.Forward = True
```

В ней задается свойство `Forward` (вперед) объекта `Find`, устанавливающее направление поиска. Обратите внимание, что это свойство не вполне соответствует полю **Направление** диалогового окна. Свойство `Forward` может принимать только два значения: **`True`** и **`False`**, соответствующие направлению поиска вперед и назад, в то время как в поле **Направление** можно задать также значение **Везде**.

Если в поле **Направление** задано значение **Вперед** или **Назад**, то на первом этапе поиск производится в указанном направлении только в выделенной час-

ти документа, а если ничего не выделено, то до конца или до начала документа соответственно. После этого на экране появляется диалоговое окно, в котором пользователю задается вопрос, следует ли продолжать поиск в оставшейся части документа. При положительном ответе поиск продолжается.

Если в поле **Направление** задано значение **Везде**, то, независимо от наличия выделенного участка текста и положения курсора поиск производится во всем документе без вывода соответствующих запросов. Чтобы запретить макросу такое поведение, используется следующая инструкция:

```
.Wrap = wdFindAsk
```

Свойство `Wrap` (обернуть) объекта `Find` может принимать три значения: `wdFindAsk`, `wdFindContinue` и `wdFindStop`. Прежде чем объяснять смысл этих значений, назовем *ближней областью поиска* выделенную часть документа, а если в документе ничего не выделено, то ту часть документа, которая расположена ниже курсора, если поиск происходит вперед (`.Forward = True`), или выше курсора, если поиск происходит назад (`.Forward = False`).

Кроме того, необходимо уточнить, что ищется именно *следующее вхождение* нужного текста, т. е. если этот текст в документе выделен, то поиск в этом выделении не производится и ближайшей областью поиска считается часть документа, расположенная ниже или выше выделенного участка в зависимости от направления поиска.

Значение свойства `Wrap` определяет, что происходит, если в ближайшей области поиска искомое значение найти не удается:

- `wdFindAsk` (*Ask* — спрашивать) — пользователь получает запрос о необходимости продолжать поиск в остальной части документа;
- `wdFindContinue` (*Continue* — продолжать) — поиск продолжается в остальной части документа без вывода запроса;
- `wdFindStop` (*Stop* — стоп) — поиск завершается.

В табл. 12.1 показано примерное соотношение между значениями в поле **Направление** и значениями свойств `Forward` и `Wrap`.

***Таблица 12.1**. Поле **Направление** диалогового окна **Найти и заменить** и свойства `Forward` и `Wrap` объекта `Find`*

Значение поля *Направление*	**Значение свойства `Forward`**	**Значение свойства `Wrap`**
Вперед	**`True`**	`wdFindAsk`
Назад	**`False`**	`wdFindAsk`
Везде	**`True`**	`wdFindContinue`

К сожалению, соотношение это не является строгим, поскольку поиск с помощью диалогового окна **Найти и заменить** происходит не совсем так, как поиск с помощью макроса. Дело в том, что действия, которые вы выполняли при записи макроса, не воспроизводятся с помощью записанного макроса даже при одних и тех же исходных условиях. Хоть и не часто, но такое с макросами случается.

Чтобы в этом убедиться, выделите в тексте одну букву "п" и попытайтесь найти сочетание букв "по". Выполните эту процедуру в режиме записи макроса. При записи макроса произойдет переход к следующему сочетанию букв "по" (будем считать, что в вашем тексте такое сочетание есть). Однако если вы выделите ту же букву "п" и запустите записанный вами макрос, то неожиданно получите сообщение, в котором вас попросят ответить, нужно ли продолжать поиск в оставшейся части документа.

Есть и более существенное отличие. В справке по Microsoft Word сказано, что, задав для поля **Направление** значение **Везде**, можно выполнять поиск также и в колонтитулах, сносках, примечаниях и т. п. Можете убедиться, что это действительно так. Однако макрос, записанный с использованием значения **Везде**, такой возможностью не обладает. К этой проблеме мы вернемся в *гл. 16*, а сейчас перейдем к следующей инструкции:

```
.Format = False
```

Свойство `Format` (формат) объекта `Find` определяет, учитывается ли при поиске форматирование. Принимает два значения: **`True`** и **`False`**. Присвоение свойству значения **`False`** эквивалентно применению описанного ранее метода `ClearFormatting`, а присвоение значения **`True`**, как показал наш опыт, на выполнение программы не влияет: в этом случае все зависит от установки параметров форматирования в диалоговом окне **Найти и заменить**.

На очереди инструкция:

```
.MatchCase = False
```

Свойство `MatchCase` (*Match* — соответствовать; *Case* — регистр букв) объекта `Find` определяет, учитывается ли регистр букв при поиске, т. е. различаются ли прописные и строчные буквы. Принимает два значения: **`True`** и **`False`**. Соответствует флажку **Учитывать регистр**.

Далее записана инструкция:

```
.MatchWholeWord = False
```

Свойство `MatchWholeWord` (*Whole* — целое; *Word* — слово) объекта `Find` определяет, включен ли при поиске режим поиска только целого слова. Принимает два значения: **`True`** и **`False`**. Соответствует флажку **Только слово целиком**.

В инструкции

```
.MatchWildcards = False
```

устанавливается свойство `MatchWildcards` (*Wildcards* — подстановочные знаки) объекта `Find` определяет, используются ли при поиске подстановочные знаки, специальные знаки и специальные операторы. Принимает два значения: **`True`** и **`False`**. Соответствует флажку **Подстановочные знаки**. Об использовании при поиске подстановочных знаков подробно рассказано в *разд. 4.3.2*. Отметим только, что когда этот флажок установлен (`.MatchWildcards = True`), режим поиска целого слова отключается, а регистр букв при поиске всегда учитывается.

Следующие две инструкции для русскоговорящих пользователей не очень интересны, но поскольку они нам часто встречаются в автоматически записанных подпрограммах, расскажем, что происходит при их выполнении. В инструкции

```
.MatchSoundsLike = False
```

устанавливается свойство `MatchSoundsLike` (*Sounds* — звучит; *Like* — подобно) объекта `Find`, который определяет, используется ли при поиске режим поиска похожих по звучанию слов (только для слов на английском языке). Принимает два значения: **`True`** и **`False`**. Соответствует флажку **Произносится как**.

В инструкции

```
.MatchAllWordForms = False
```

устанавливается свойство `MatchAllWordForms` (*All* — все; *Word* — слово; *Forms* — формы) объекта `Find`, который определяет, используется ли режим поиска и замены всех грамматических форм слова (только для слов на английском языке). Принимает два значения: **`True`** и **`False`**. Соответствует флажку **Все словоформы**.

Далее следует знакомая нам инструкция завершения блока **`With`**:

```
End With
```

Этой инструкцией завершается присвоение значений всем свойствам объекта `Find`, определяющим условия поиска.

Далее следует важная инструкция

```
Selection.Find.Execute
```

В ней применяется метод `Execute` (выполнить) объекта `Find`, который, собственно, и выполняет поиск. У этого метода есть аргументы, но знакомство с ними мы отложим на будущее. Действие этой инструкции эквивалентно нажатию кнопки **Найти далее**.

12.1.3. Запоминание условий поиска

Важно, что поиск в Microsoft Word обладает памятью. Для повторения поиска с теми же условиями не требуется указывать их повторно. Даже если закрыть диалоговое окно **Найти и заменить**, перейти в другой документ и вновь открыть это окно, в нем будут установлены те же параметры, которые использовались при предыдущей попытке поиска. Исключением является параметр **Направление**. Если его значение было **Назад**, это значение запоминается. Если же были установлены значения **Вперед** или **Везде**, то при следующем поиске будет установлено значение **Вперед**, если в окне будет выделенный фрагмент, или значение **Везде**, если выделенного фрагмента не будет. "Забываются" эти параметры только после закрытия приложения Microsoft Word. Такое поведения удобно для пользователя, особенно если учесть, что поиск можно продолжать в обоих направлениях, не открывая диалогового окна **Найти и заменить**, с помощью кнопок **Найти/перейти далее**  и **Вернуться назад**  , расположенных в правом нижнем углу окна документа под вертикальной полосой прокрутки, или сочетаний клавиш <Ctrl>+<Pgdn> и <Ctrl>+<Pgup> соответственно.

Многие свойства объекта `Find` однозначно связаны с соответствующими параметрами диалогового окна **Найти и заменить**. Таким образом, одновременно запоминаются свойства объекта `Find`. Благодаря этому можно продолжать поиск с условиями, установленными в результате работы макроса, но чаще из-за этого выполнение макроса, в котором проводился поиск, сопровождается неприятным побочным эффектом. Когда пользователь открывает окно **Найти и заменить**, после выполнения макроса он должен не забыть переустановить все флажки и поля, установленные ранее макросом. Поэтому целесообразно в конце подпрограммы "убрать за собой", т. е. добавить инструкции, которые установят все параметры окна **Найти и заменить** в состояние, в котором они находились при первом открытии окна **Найти и заменить** после запуска приложения Microsoft Word. Сказанное не относится к свойству `Wrap` (обернуть), которое не связано ни с каким параметром диалогового окна **Найти и заменить** и не запоминается.

12.1.4. Макросы поиска выделенного текста

В результате изучения всех инструкций подпрограммы `Test2` можно сделать вывод, что для решения поставленной в начале раздела задачи достаточно исключить инструкцию копирования выделенного текста в буфер обмена, переименовать подпрограмму и написать соответствующий комментарий. Далее следует удалить из поля поиска текст, чтобы при следующем открытии окна **Найти и заменить** его там не было. Кроме того, давайте в качестве эксперимента попробуем исправить небольшую "шероховатость" в автомати-

чески созданной подпрограмме и включим последние исполнимые инструкции в блок `With` (изучив *разд. 12.2.2*, вы узнаете, что в ряде случаев нельзя включать в блок `With` инструкцию, в которой применяется метод `Execute`).

В результате получаем подпрограмму поиска следующего вхождения выделенного текста, представленную в листинге 12.3.

Листинг 12.3

```
Sub SearchSelectionNext_1()
' Поиск следующего вхождения выделенного текста
'
    With Selection.Find
         .ClearFormatting
        .Text = Selection.Text
        .Replacement.Text = ""
        .Forward = True
        .Wrap = wdFindAsk
        .Format = False
        .MatchCase = False
        .MatchWholeWord = False
        .MatchWildcards = False
        .MatchSoundsLike = False
        .MatchAllWordForms = False
        .Execute
        .Text = ""
    End With
End Sub
```

Поскольку внутри блока **With** уже необязательно писать в начале инструкций `Selection.Find`, инструкция, в которой применяется метод `.Execute`, стала короче. Так же коротко мы записали и следующую инструкцию, благодаря которой после завершения работы макроса поле **Найти** очистится (обратите внимание, что тогда при следующем поиске в этом поле автоматически появится выделенный в документе текст).

Назначить полученному макросу кнопку инструментальной панели вы сможете самостоятельно. Пожалуй, единственный существенный недостаток этой программы заключается в том, что поиск, который начинается в основном тексте документа, не распространяется на надписи, колонтитулы, примечания и т. п. Решением этой задачи мы займемся в *разд. 16.5*.

Теперь вы сможете сами в качестве упражнения создать макрос для поиска предыдущего вхождения выделенного текста.

Назовите макрос **SearchSelectionBack_1**. Кроме того, в этом макросе целесообразно после выполнения поиска добавить еще одну инструкцию "уборки", в которой направление поиска заменялось бы на стандартное значение "вперед". Наш вариант макроса приведен в листинге 12.4.

Листинг 12.4

```
Sub SearchSelectionBack_1()
' Поиск предыдущего вхождения выделенного текста
'
    With Selection.Find
         .ClearFormatting
        .Text = Selection.Text
        .Replacement.Text = ""
        .Forward = False
        .Wrap = wdFindAsk
        .Format = False
        .MatchCase = False
        .MatchWholeWord = False
        .MatchWildcards = False
        .MatchSoundsLike = False
        .MatchAllWordForms = False
        .Execute
        .Text = ""
        .Forward = True
    End With
End Sub
```

12.2. Поиск в другом документе

Предположим, у вас есть справочный документ. Удобно, когда можно нажать кнопку, и тут откроется справочный документ и в нем найдется текст, предварительно выделенный в рабочем документе. Искать выделенный текст мы уже научились, а как открыть документ, нам еще предстоит узнать.

12.2.1. Открытие документа

Прежде всего нам потребуется открыть окно со справочным документом. Это можно сделать различными способами. На первом этапе желательно это сделать попроще, поэтому пожертвуем универсальностью макроса и предположим, что справочный документ имеет имя Reference.doc и всегда находится в определенной папке. Мы положили его в папку "D:\Documents and Settings\Администратор\Мои документы". Вы можете выбрать любую папку на своем компьютере (или, если необходимо, на другом компьютере сети).

Чтобы узнать, как записать инструкцию, с помощью которой можно открыть документ Reference.doc, откроем его в приложении Microsoft Word, включив запись макроса.

> ***Совет***
>
> Вместо того чтобы искать в справке по Visual Basic, как запрограммировать то или иное действие, можно выполнить это действие в приложении Microsoft Word с включенной записью макроса. После этого уже не сложно найти в автоматически записанной подпрограмме нужную инструкцию и получить о ней дополнительные сведения во встроенной справке или в другом источнике.

Чтобы у нас с вами в результате записи макроса получились похожие подпрограммы, перед записью макроса создайте справочный документ с именем Reference.doc и положите ее в свою папку "Мои документы". Затем откройте диалоговое окно **Открытие документа** (меню **Файл | Открыть**) и перейдите в этом окне в любую папку, отличную от папки "Мои документы".

Необходимость этих подготовительных действий объясняется тем, что во время записи макроса мы планируем перейти в папку, содержащую файл Reference.doc. Если при открытии диалогового окна **Открытие документа** мы сразу попадем в папку "Мои документы", то переходить в эту папку не понадобится и инструкция перехода к нужной папке в макрос не запишется. Кроме того, необходимо учесть, что, пока открыто приложение Microsoft Word, диалоговое окно **Открытие документа** помнит, какая папка открывалась в прошлый раз (даже если не открывать в этом окне никакого документа, а просто перейти в нужную папку и закрыть окно).

Итак, запишите макрос с именем **Test4** и описанием "Открытие документа". Во время записи макроса перейдите в папку, где находится файл Reference.doc, и откройте его. Полученная нами подпрограмма приведена в листинге 12.5.

Листинг 12.5

```
Sub Test4()
'
' Test4 Макрос
' Открытие документа
'
    ChangeFileOpenDirectory _
        "D:\Documents and Settings\Администратор\Мои документы\"
    Documents.Open FileName:="Reference.doc", ConfirmConversions:=True, _
        ReadOnly:=False, AddToRecentFiles:=False, PasswordDocument:="", _
        PasswordTemplate:="", Revert:=False, WritePasswordDocument:="", _
        WritePasswordTemplate:="", Format:=wdOpenFormatAuto
End Sub
```

В подпрограмме две исполнимые инструкции.

Первую инструкцию редактор Visual Basic записал в две строки (как мы уже рассказывали, подчеркивание после пробела в конце строки — признак переноса продолжения инструкции на следующую строку). В этой инструкции применяется глобальный метод `ChangeFileOpenDirectory` (изменить каталог открытия файла), с помощью которого указывается папка, которая открывается в диалоговом окне **Открытие документа**. Единственным аргументом этого метода является строка, задающая путь к папке, которая будет отображаться в этом диалоговом окне. На примере этой инструкции вы впервые знакомитесь с *глобальным методом* (раньше мы знакомили вас с глобальными свойствами). Так называются методы объекта `Application` (приложение), при обращении к которым необязательно указывать ссылку на объект. Все такие методы считаются методами объекта `Global` (глобальный).

Второй новой особенностью этой инструкции является способ, с помощью которого указывается значение аргумента этого метода. Это значение (в данном случае — путь к папке) просто указывается через пробел после имени метода. В нашей инструкции кроме пробела используется еще перенос на следующую строку, но это не имеет принципиального значения. Такой способ задания значений аргументов называется *позиционным*, подробнее о нем рассказано в *гл. 14*.

Обратите внимание, что наличие этой инструкции обуславливает побочный эффект в записанном нами макросе: после его выполнения папка, с которой начинается выбор нужного файла в окне **Открытие документа**, изменится на папку, указанную в данной инструкции.

Во второй инструкции применяется метод `Open` (открыть) нового объекта `Documents` (документы). Этот объект является семейством, элементами которого являются уже знакомые нам объекты `Document` (документ). Оно представляет все документы, открытые в приложении Microsoft Word. В инструкции используется глобальное свойство `Documents`, которое возвращает ссылку на объект `Documents`.

Метод `Open` (открыть), как несложно догадаться, открывает сохраненный ранее документ, добавляя новый элемент в семейство `Documents`. Аргументы этого метода соответствуют параметрам диалогового окна **Открытие документа**.

Далее в инструкции указываются аргументы метода `Open`. На них следует остановиться подробнее.

- `FileName` (имя файла) задает имя файла. Это единственный обязательный аргумент. Все остальные аргументы можно не указывать, или можно указать только те аргументы, значения которых отличаются от используемых по умолчанию. Здесь можно задать полное имя файла, т. е., указать путь к нему. Мы рекомендуем это сделать. Тогда можно удалить первую инструкцию, и макрос избавится от побочного эффекта, о котором говорилось ранее.
- `ConfirmConversions` (подтверждение преобразования). Значение **`True`** задает необходимость подтверждения преобразования файла при его открытии. Значение по умолчанию задается флажком **Подтверждать преобразование при открытии**, расположенном на вкладке **Общие** диалогового окна **Параметры** (меню **Сервис** пункт **Параметры**).
- `ReadOnly` (только чтение). Если значение аргумента — **`True`**, то файл открывается только для чтения.
- `AddToRecentFiles` (добавить к списку последних файлов). Если значение аргумента — **`True`**, то имя файла добавляется к списку последних открывавшихся файлов в нижней части меню **Файл**. Значение по умолчанию определяется состоянием флажка **Помнить список из**, расположенного на вкладке **Общие** диалогового окна **Параметры** (в меню **Сервис** пункт **Параметры**).
- `PasswordDocument` (пароль документа) — пароль для открытия документа.
- `PasswordTemplate` (пароль шаблона) — пароль для открытия шаблона.
- `Revert` (вернуться). Этот аргумент определяет, что будет происходить, если аргумент `FileName` содержит имя уже открытого документа. Если значение аргумента — **`True`**, то все изменения, сделанные в открытом документе, будут потеряны и хранящийся на диске файл откроется заново, т. е. произойдет возвращение к документу, хранящемуся на диске. Если значение аргумента — **`False`**, активируется открытый ранее документ.

- `WritePasswordDocument` (пароль на запись документа) — пароль для сохранения изменений в документе.
- `WritePasswordTemplate` (пароль на запись шаблона) — пароль для сохранения изменений в шаблоне.
- `Format` (формат) — определяет способ преобразования файла при его открытии. Возможные варианты приведены в табл. 12.2.

Таблица 12.2. *Значение аргумента* `Format` *метода* `Open`*, применяемого для открытия документа*

Значение аргумента `Format`	**Описание**
`wdOpenFormatAllWord`	Документы, сохраненные в предыдущих версиях Microsoft Word
`WdOpenFormatAuto`	Автоматическое определение типа документа
`wdOpenFormatDocument`	Документы в формате текущей версии Microsoft Word
`wdOpenFormatEncodedText`	Кодированные текстовые файлы
`WdOpenFormatRTF`	Документы в формате RTF
`wdOpenFormatTemplate`	Шаблоны
`WdOpenFormatText`	Документы в текстовом формате
`wdOpenFormatUnicodeText`	Документы в текстовом формате Юникод
`wdOpenFormatWebPages`	Документы в веб-формате

Чаще всего применяется используемое по умолчанию значение `wdOpenFormatAuto`. Если известно, что документ сохранен в текстовом виде в кодировке ANSI или Юникод, целесообразно выбрать значения `wdOpenFormatText` или `wdOpenFormatUnicodeText` соответственно. Документ будет открываться гораздо быстрее и не будет запросов к пользователю, связанных с типом кодировки. Вариант `wdOpenFormatText` следует также использовать, если необходимо открыть RTF- или HTM-файл в виде текста, содержащего коды форматирования.

При записи макроса полученный список аргументов и их значений позволяет вам узнать, какие значения используются в момент записи макроса. Если вы сохраните в тексте подпрограммы этот список аргументов, то при выполнении макроса будут использоваться те же значения аргументов, что и при записи, независимо от значений, используемых по умолчанию в момент выполнения макроса.

12.2.2. Макрос поиска выделенного текста в справочном документе

Теперь мы можем создать новую подпрограмму `SearchReference_1` на базе уже готовой подпрограммы `SearchSelectionNext_1`. Достаточно вставить инструкцию открытия документа Reference.doc перед инструкцией выполнения поиска. Однако следует заранее продумать особенности использования разрабатываемого макроса.

Во-первых, справочный документ целесообразно открывать в режиме "только для чтения". Во-вторых, для аргумента `Revert` (вернуться) рекомендуется задать значение **`False`**. Тогда при вызове макроса в то время, когда справочный документ уже открыт (а это типичная ситуация), заново открывать его уже не придется. Это удобно, но следует учитывать, что в этом случае поиск будет происходить в ранее открытом документе, в который, возможно, вы специально или случайно внесли изменения (режим "только для чтения" этому помешать не может). При этом поиск будет продолжаться с того места, на котором находился курсор в справочном документе. Поэтому для поиска целесообразно установить значение свойства `Wrap` равным `wdFindContinue`, чтобы поиск выполнялся во всем документе, а не только в области, лежащей ниже курсора.

Попробуйте написать подпрограмму `SearchReference_1` самостоятельно, проверьте, как она работает, а затем сравните с подпрограммой, приведенной в листинге 12.6.

Листинг 12.6

```
Sub SearchReference_1()
' Поиск в Reference.doc следующего вхождения выделенного текста
'
    With Selection.Find
         .ClearFormatting
        .Text = Selection.Text
        .Replacement.Text = ""
        .Forward = True
        .Wrap = wdFindContinue
        .Format = False
        .MatchCase = False
        .MatchWholeWord = False
        .MatchWildcards = False
        .MatchSoundsLike = False
```

```
        .MatchAllWordForms = False
    End With
    Documents.Open FileName:= _
"D:\Documents and Settings\Администратор\Мои документы\Reference.doc" _
        , ConfirmConversions:=False, _
        ReadOnly:=True, AddToRecentFiles:=False, PasswordDocument:="", _
        PasswordTemplate:="", Revert:=False, WritePasswordDocument:="", _
        WritePasswordTemplate:="", Format:=wdOpenFormatAuto
    Selection.Find.Execute
    Selection.Find.Text = ""
End Sub
```

Обратите внимание, что длинную строку (заключенную в кавычки) с полным именем файла мы записали без отступа. Это сделано из-за то, что внутри строки (текста, заключенного в кавычки) знак переноса инструкции на следующую строчку программы использовать нельзя. В *разд. 14.1.7* мы расскажем, что в таких случаях длинную строку представляют в виде объединения нескольких строк и знак переноса инструкции помещают рядом со знаком операции объединения строк.

Ранее мы упоминали, что глобальное свойство `Selection` (выделение) возвращает объект, представляющий собой выделенный участок *в активном окне*. Из этого следует, что до открытия справочного документа и после этого открытия свойство `Selection` возвращает разные объекты, поскольку активным становится другое окно. В первом окне запоминается выделенный текст, а во втором окне производится поиск. Поэтому блок **`With`** должен быть закрыт раньше, чем будет открыт справочный документ. Если же этого не сделать, работа будет производиться только с объектом `Selection`, относящимся к исходному документу, и поиск будет выполняться в исходном документе. Этот пример наглядно показывает, что блок **`With`** не только сокращает текст программы, но и существенно влияет на ее выполнение. По-видимому, этот факт учли разработчики и при автоматической записи макроса, выполняющего поиск; инструкция, в которой с помощью метода `Execute` (выполнить) выполняется поиск, вынесена из блока **`With`**, задающего условия поиска *(см. разд. 12.1.4)*.

12.3. Глобальное подкрашивание выделенного текста

В *гл. 4* мы записали макрос **ColorTermSeeTab** (выделить цветом "см. табл."), с помощью которого выполнялось единовременное подкрашивание желтым цветом всех выражений "см. табл.", благодаря чему их можно было легко

заметить при просмотре документа. В *гл. 6* мы проанализировали текст этого макроса и, не вдаваясь в детали, создали на его основе макрос **Pencraft** (литературный стиль), предназначенный для выделения различным цветом слов из "черного списка". В этом разделе мы продолжим эту линейку макросов и создадим макрос, который выполняет сходную задачу, позволяя выбирать подкрашиваемые слова и цвет подкрашивания "на ходу".

Постановка задачи

При нажатии кнопки выделенный обычным образом фрагмент текста подкрашивается цветом во всем документе. Цвет предварительно выбирается на панели инструментов **Форматирование** с помощью кнопки с изображением маленького черного треугольника, расположенной справа от кнопки **Выделение цветом**  .

Новые свойства и методы

Еще раз воспроизведем текст подпрограммы `ColorTermSeeTab` (листинг 12.7).

Листинг 12.7

```
Sub ColorTermSeeTab()
    Options.DefaultHighlightColorIndex = wdYellow
    Selection.Find.ClearFormatting
    Selection.Find.Replacement.ClearFormatting
    Selection.Find.Replacement.Highlight = True
    With Selection.Find
        .Text = "см. табл."
        .Replacement.Text = ""
        .Forward = True
        .Wrap = wdFindContinue
        .Format = True
        .MatchCase = False
        .MatchWholeWord = False
        .MatchWildcards = False
        .MatchSoundsLike = False
        .MatchAllWordForms = False
    End With
    Selection.Find.Execute Replace:=wdReplaceAll
End Sub
```

Теперь вам почти все знакомо. Добавилась инструкция

```
Selection.Find.Replacement.ClearFormatting
```

которая уже не должна вызывать вопросов, и хотя теперь метод `ClearFormatting` (снять форматирование) формально уже другой, поскольку теперь это метод объекта `Replacement` (замена), а не объекта `Find` (найти), вы уже догадываетесь, что он выполняет ту же задачу снятия форматирования, но уже для поля **Заменить на** вкладки **Заменить** диалогового окна **Найти и заменить**.

В инструкции

```
Selection.Find.Replacement.Highlight = True
```

вам уже известны все слова и понятен синтаксис. Вам уже не надо объяснять, что в ней присваивается значение **`True`** свойству `Highlight` (выделение цветом) объекта `Replacement`. Причем вы уже встречались со словом *Highlight* — оно входило в имена свойств, связанных с выделением цветом, с которыми вы познакомились в *гл. 11*. Поэтому вы наверняка уже поняли смысл этой инструкции: она указывает, что заменяемый текст выделяется цветом. Наличие этой инструкции соответствует появлению надписи **Формат: выделение цветом** под полем **Заменить на** на вкладке **Заменить** диалогового окна **Найти и заменить**.

Важное отличие появилось в инструкции, в которой к объекту `Find` (найти) применяется метод `Execute` (выполнить):

```
Selection.Find.Execute Replace:=wdReplaceAll
```

Теперь для этого метода указан аргумент `Replace` (заменить), которому присваивается значение, выраженное константой `wdReplaceAll` (заменить все). Этот необязательный аргумент может принимать одно из трех значений: `wdReplaceAll`, `wdReplaceNone` (не заменять ничего) или `wdReplaceOne` (заменить один раз), что соответствует нажатию на одну из кнопок **Заменить все**, **Найти далее** и **Заменить**. При этом `wdReplaceNone` является значением по умолчанию, т. е. используется, когда аргумент `Replace` не указывается.

Итак, все инструкции в подпрограмме `ColorTermSeeTab` нам знакомы. Осталось понять, какие надо сделать в ней изменения, чтобы получить из нее новую подпрограмму, решающую поставленную задачу.

Во-первых, опыт, приобретенный при изучении *разд. 11.3*, нам подсказывает, что для того чтобы выделение происходило текущим цветом (т. е. цветом, установленным с помощью инструмента **Выделение цветом** ), следует удалить первую исполнимую инструкцию, которая переопределяет текущий цвет.

Во-вторых, мы уже знаем, что для того чтобы выполнялся поиск выделенного текста, следует заменить строку "см. табл." на выражение `Selection.Text`.

В-третьих, также как мы это сделали при создании подпрограммы `SearchSelectionNext_1` в *разд. 12.1.4*, следует "убрать за собой", но в этом случае кроме очистки поля **Найти** надо также снять форматирование в поле **Заменить на**.

Вы уже знаете и умеете достаточно, чтобы после сделанных пояснений самостоятельно написать текст подпрограммы. Назовите ее `HighlightSelectionAll_1`, а в качестве описания введите строку "Выделение текущим цветом всех вхождений выделенного текста". Сравните то, что у вас получится, с подпрограммой, приведенной в листинге 12.8.

Листинг 12.8

```
Sub HighlightSelectionAll_1()
' Выделение текущим цветом всех вхождений
    With Selection.Find
        .ClearFormatting
        .Replacement.ClearFormatting
        .Replacement.Highlight = True
        .Text = Selection.Text
        .Replacement.Text = ""
        .Forward = True
        .Wrap = wdFindContinue
        .Format = True
        .MatchCase = False
        .MatchWholeWord = False
        .MatchWildcards = False
        .MatchSoundsLike = False
        .MatchAllWordForms = False
        .Execute Replace:=wdReplaceAll
        .Replacement.ClearFormatting
        .Text = ""
    End With
End Sub
```

Проверьте, как работает эта подпрограмма. Попробуйте найти ситуации, когда она не справляется со своей задачей. Одно из ограничений вы уже

знаете: так же как и в предыдущем случае, не будет происходить замена в колонтитулах, примечаниях и пр., если выделенный текст находится в основной части документа.

Итак, мы практически без программирования, пользуясь только теми инструкциями, которые были получены нами при автоматической записи макросов, смогли разработать ряд очень полезных макросов. Однако для этого нам потребовалось узнать, как эти инструкции работают, причем некоторые аспекты мы изучили достаточно глубоко. Теперь пришла пора сделать следующий шаг: научиться писать инструкции, которые при автоматической записи макросов не применяются. Это уже в чистом виде программирование.

12.4. Что нового в этой главе

Вы выяснили, что в режиме записи макроса действия, происходящие при открытом диалоговом окне, в текст макроса не записываются, поэтому использовать буфер обмена для передачи значений параметрам диалогового окна не удается. Для решения этой задачи вы научились использовать свойство `Text` объекта `Selection`. Познакомились с объектами `Find` и `Replacement`, некоторыми их свойствами и методами. Узнали, что установленные значения их свойств сохраняются в виде параметров диалогового окна **Найти и заменить** в течение всего сеанса работы с приложением Microsoft Word. Научившись с пониманием читать текст макроса, полученного при автоматической записи поиска и замены текста, создали две подпрограммы: одну для поиска следующего, а другую для поиска предыдущего вхождения выделенного текста. Познакомились с инструкциями, выполняющими открытие документа. Изучили метод `Open` объекта `Documents` и познакомились с аргументами этого метода. Составили подпрограмму для поиска выделенного текста в справочном документе. Убедились на конкретном примере, что блок `With` не только сокращает текст программы, но и существенно влияет на ее выполнение. На базе автоматически записанного макроса создали новый макрос, который позволяет подкрасить текущим цветом все вхождения выделенного текста.

Теперь посмотрите на приведенные далее табл. 12.3—12.6. Постарайтесь вспомнить, что означают написанные в каждой ячейке слова. Может быть, некоторые из вас захотят прочитать про них еще раз.

Таблица 12.3. *Объекты и их свойства*

Объект	Свойство
`Selection`	`Text`
	`Find`

Таблица 12.3 *(окончание)*

Объект	Свойство
Replacement	Text
	Highlight

Таблица 12.4*. Объекты и их методы*

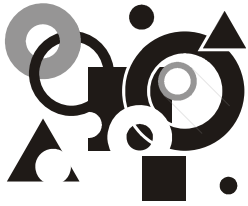
Объект	Метод	Аргумент	Константа
Find	ClearFormatting		
	Execute	Replace	wdReplaceAll wdReplaceNone wdReplaceOne
Replacement	ClearFormatting		
Global	ChangeFileOpenDirectory		

Таблица 12.5. *Свойства объекта* `Find`

Объект	Свойство	Константа
Find	Text	
	Replacement	
	Forward	
	Wrap	wdFindAsk
		wdFindContinue
		wdFindStop
	Format	
	MatchCase	
	MatchWholeWord	
	MatchWildcards	
	MatchSoundsLike	
	MatchAllWordForms	

Таблица 12.6. *Аргументы метода* `Open`

Объект	Метод	Аргумент	Константа
Documents	Open	FileName	
		ConfirmConversions	
		ReadOnly	
		AddToRecentFiles	
		PasswordDocument	
		PasswordTemplate	
		Revert	
		WritePasswordDocument	
		WritePasswordTemplate	
		Format	wdOpenFormatAllWord
			wdOpenFormatAuto
			wdOpenFormatDocument
			wdOpenFormatEncodedText
			wdOpenFormatRTF
			wdOpenFormatTemplate
			wdOpenFormatText
			wdOpenFormatUnicodeText
			wdOpenFormatWebPages



Глава 13

Память для макросов

Имя, тип и значение переменной. Инструкции присваивания. Объявление переменных. Обязательное объявление переменных. Префиксы типа в именах переменных. Новые свойства и методы объекта `Selection`.

В этой главе мы поднимаемся еще на одну ступень в изучении программирования. Мы научимся использовать переменные для хранения промежуточной информации во время выполнения макросов. При автоматической записи макросов переменные не применяются, однако это несложное усовершенствование существенно расширяет возможности подпрограмм. Мы на нескольких простых примерах постараемся научить вас пользоваться этим средством. В следующих главах переменные будут использоваться практически в каждом макросе.

13.1. Применение переменных для хранения информации

Хранение промежуточных данных в Microsoft Word осуществляется с помощью буфера обмена. Буфер обмена можно использовать и в макросах. Например, в *гл. 8* мы воспользовались буфером обмена при записи макроса **CharFlip_1**, меняющего местами две соседние буквы. Однако при разработке макросов часто возникает необходимость ненадолго, только на время работы макроса, сохранить внутреннюю информацию. Буфер обмена для таких целей использовать нельзя. Например, в *гл. 9* в конце *разд. 9.1* мы решили усовершенствовать автоматически записанный макрос **WordFlip_1**, переставляющий два соседних слова, и научить его устанавливать флажок **учитывать пробелы при копировании**, чтобы можно было с помощью макроса переставлять слова, разделенные знаком препинания. Мы решили эту задачу и в *разд. 9.3.3* составили макрос **WordFlip_3** (см. листинг 9.3), который перед перестановкой слов устанавливает этот флажок. При этом мы понимали, что такое решение подходит не для всех ситуаций.

Например, представьте себе, что вы готовите документ, в котором описывается расположение файлов на диске. В этом документе многократно встречается указание полных путей к файлам и папкам, и вы наверняка будете использовать операции копирования и вставки, чтобы не вводить длинные пути вручную. В этом случае вам целесообразно снять флажок **учитывать пробелы при копировании**, чтобы после знака "\" не добавлялся пробел после вставки. И вам совсем не надо, чтобы какой-нибудь макрос, которым вы привыкли пользоваться, незаметно для вас каждый раз устанавливал этот флажок.

Чтобы исключить такой побочный эффект, достаточно запомнить состояние флажка перед перестановкой слов, и восстановить его после перестановки. Как вы понимаете, буфер обмена для этого непригоден. Мы воспользуемся этим простым примером и составим макрос **WordFlip_4**, чтобы показать вам, как используются переменные. Однако сначала нам придется заняться теорией.

13.1.1. Имена переменных

Как мы уже сказали, для того, чтобы ненадолго, до завершения работы программы, сохранить нужное значение, применяются так называемые *переменные* (variable). У переменной есть *имя* (name), *тип* (type) и *значение* (value). Работают с переменными почти так же, как с переменными в школьной алгебре. Имена переменным дают разработчики программ.

Имена, которые мы придумываем сами (например, имена подпрограмм, закладок и переменных), должны удовлетворять следующим требованиям.

- Имя может состоять только из букв, цифр и знака подчеркивания. В имени нельзя использовать ни пробелы, ни знаки пунктуации, ни знаки арифметических операций.
- Имя должно начинаться с буквы. Оно не может начинаться с цифры или знака подчеркивания.
- Имя может содержать не более 255 знаков. Иногда предусмотрены дополнительные ограничения. Например, имя макроса не может содержать более 80 знаков.
- Имя не должно совпадать с другими именами, которые могут использоваться в данном контексте, например, с ключевыми словами, именами встроенных функций и существующих в данном модуле процедур, именами других переменных в данной процедуре.

Последнее требование звучит пугающе: неужели нужно знать наизусть все ключевые слова языка и названия всех встроенных функций? На самом деле зубрить их не нужно. С ними хорошо знаком редактор Visual Basic, и он обязательно отреагирует на ошибку, хотя не всегда можно сразу понять, что

ошибка возникла из-за совпадения имен. Немного далее, в следующем подразделе, мы расскажем о способе выбора имен переменных, позволяющем предупредить появление подобных ошибок.

Полезные сведения

Имена, отличающиеся регистром букв, редактор Visual Basic считает одинаковыми, однако позволяет использовать в записи имен переменных буквы разного регистра, причем сам следит за тем, чтобы при записи каждого имени регистр букв использовался единообразно. Например, если есть переменная с именем `str1`, нельзя другой переменной дать имя `sTr1` или `Str1`. Однако можно использовать для переменной любое из этих имен, и редактор Visual Basic сам будет изменять регистр букв таким образом, чтобы это имя всюду было записано одинаково.

Тип переменной — это тип данных, которые могут храниться в этой переменной. Например, для строк предусмотрен тип данных **`String`** (строковый). Свойства, соответствующие флажкам, как вы знаете, могут принимать два значения: **`True`** (истина) и **`False`** (ложь). Такие данные имеют тип **`Boolean`** (логический). Например, чтобы запомнить значение свойства `SmartCutPaste`, соответствующего флажку **учитывать пробелы при копировании**, нам потребуется логическая переменная.

13.1.2. Как выбирать имена для переменных

Основной принцип: имя должно быть содержательным. Можно даже использовать русские буквы, но это рискованно: при переносе макроса на другой компьютер они могут неправильно отображаться.

Совет

Используйте для переменных понятные имена. Время, затраченное на ввод длинного имени, с лихвой окупится тем, что при чтении программы вам не придется вспоминать, для чего эта переменная предназначена. Если все-таки вам жалко времени на ввод длинного имени, пользуйтесь кратким обозначением, а потом глобально во всей процедуре замените его понятным вам именем.

Совет

Используйте в именах переменных префиксы типа. Сейчас стало хорошей традицией первые буквы в именах переменных (их по-умному называют префиксом) использовать для указания типа переменной. Например, для переменных типа **`String`** использовать префикс `str`, для переменных типа **`Boolean`** — префикс `bln`. Это сделает программу нагляднее, а значит, вы быстрее ее напишете, и в ней будет меньше ошибок. Кроме того, использование префиксов убережет вас от случайного использования зарезервированного слова в качестве имени переменной.

Предупреждение

В VBA есть функции с именами `Int` и `Str`, поэтому нельзя давать переменным имена `int` и `str`, образованные от названий типов `Integer` и `String`, однако использование этих буквосочетаний в качестве префиксов приветствуется.

Например, для переменной, в которой будет храниться значение свойства `SmartCutPaste`, естественно использовать имя `blnSmartCutPaste`.

13.1.3. Инструкция присваивания

Для того чтобы в макросе **WordFlip_4** запомнить значение (**`True`** или **`False`**) свойства `SmartCutPaste` объекта `Options`, мы сохраним это значение в переменной `blnSmartCutPaste`. Другими словами, присвоим этой переменной значение свойства `Options.SmartCutPaste`. Для этой цели используется следующая *инструкция присваивания*:

```
blnSmartCutPaste = Options.SmartCutPaste
```

Вернуть свойству `Options.SmartCutPaste` значение, содержащееся в переменной `blnSmartCutPaste`, позволяет другая инструкция присваивания:

```
Options.SmartCutPaste = blnSmartCutPaste
```

Обратите внимание, что в данном случае знак равенства играет непривычную ему роль. В математике всегда можно поменять местами левую и правую части равенства, и при этом смысл равенства не меняется. В инструкции присваивания такая перестановка изменяет направление присваивания на противоположное.

С инструкциями присваивания мы многократно встречались, просто мы их так не называли, новым здесь является участие в них переменных.

13.1.4. Инструкция объявления переменных

В языке VBA есть существенный недостаток (правда, некоторые считают его достоинством). Речь идет о возможности использования переменных без предварительного объявления. Не пользуйтесь этой возможностью, всегда объявляйте все переменные.

Объявить переменную — это заранее сообщать редактору Visual Basic, что такая переменная будет использоваться в программе. После этого он каждый раз при встрече с ней будет проверять, совпадает ли регистр букв в имени переменной с тем образцом, который приведен в инструкции объявления переменных, и, если есть отличие, то исправит написание по этому образцу. Поэтому рекомендуется объявлять переменные, используя в их именах буквы в разных регистрах, а при вводе инструкций все буквы в именах переменных писать в одном регистре. Тогда вы сразу увидите, распознал ли ре-

дактор переменную, и, если переменная осталась неузнанной, поймете, что сделали опечатку.

Действуйте по принципу: если есть, что сообщить редактору Visual Basic, то не надо жалеть на это времени. Он отблагодарит дополнительными проверками, которые помогут сразу выявить ошибки, а во многих случаях благодаря полученным им сведениям программа будет занимать меньше места в памяти и быстрее выполняться.

Для объявления переменных используется инструкция **Dim** (от слова *Dimention* — размер). Например, чтобы объявить логическую переменную `blnSmartCutPaste`, можно воспользоваться инструкцией:

```
Dim blnSmartCutPaste
```

Переменную необходимо объявить до того, как ей впервые будет присвоено значение, но обычно, хотя это и не обязательно, все переменные объявляют в начале процедуры. Это облегчает чтение программы.

13.1.5. Обязательное объявление переменных и объявление типа переменных

Когда в программе десять строк и только одна переменная, которая встречается три раза, польза от приведенных далее рекомендаций не очевидна. Однако мы надеемся, что вы будете писать и более сложные программы, где будет много переменных. Поэтому мы настоятельно советуем придерживаться следующей рекомендации.

> ***Совет***
>
> Сделайте явное объявление переменных обязательным. Откажитесь от возможности использования переменных без предварительного объявления. Тогда редактор Visual Basic всякий раз, когда он встретит необъявленную переменную, будет сообщать вам об этом.

Благодаря этому вы предупредите появление многих серьезных ошибок, связанных с опечатками при вводе имен переменных. Это особенно полезно, если, следуя нашим рекомендациям, вы будете пользоваться понятными именами переменных.

Чтобы запретить использование переменных без *явного* (explicit) предварительного объявления переменных, в начале модуля поместите инструкцию включения режима явного объявления переменных:

```
Option Explicit
```

Эту инструкцию следует поместить в то же окно, где находится текст всех подпрограмм данного модуля, перед первой подпрограммой.

Тогда во время компиляции подпрограммы, происходящей перед ее первым выполнением, Visual Basic будет проверять наличие объявления всех встречающихся в ней переменных. Если встретится необъявленная переменная, появится сообщение, показанное на рис. 13.1.

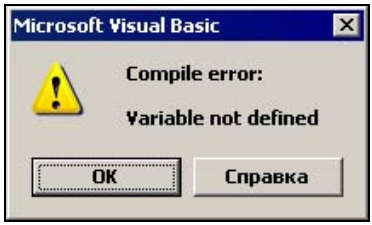


Рис. 13.1. Сообщение об ошибке при наличии необъявленной переменной: **Compile error** — ошибка при компиляции; **Variable not defined** — переменная не объявлена

Более того, мы рекомендуем сделать так, чтобы во вновь создаваемые модули редактор Visual Basic инструкцию `Option Explicit` вставлял автоматически (в существующий модуль **NewMacros** вставьте эту инструкцию сами).

Для этого в редакторе Visual Basic в меню **Tools** (Сервис) выберите пункт **Options** (Параметры) и в открывшемся одноименном окне на вкладке **Editor** (Редактор) в группе **Code Settings** (Настройки программы) установите флажок **Require Variable Declaration** (Требовать объявления переменных).

Следующая рекомендация относится к объявлению типа переменных.

> ***Совет***
>
> Объявляйте тип переменных. В инструкции **Dim** явно указывайте тип переменной, используя ключевое слово **As** (как).

Например, чтобы при объявлении переменной `blnSmartCutPaste` указать, что ее тип *логический* (boolean), можно воспользоваться инструкцией:

```
Dim blnSmartCutPaste As Boolean
```

Если тип не задавать, а написать просто:

```
Dim blnSmartCutPaste
```

то переменной по умолчанию (неявно) будет присвоен тип **Variant** (универсальный) — универсальный тип, который может использоваться для хранения данных различного типа.

Мы рекомендуем явно объявлять тип переменных, т. е. заранее сообщать редактору Visual Basic, для какого типа данных предусмотрена переменная.

Благодаря этому программа будет быстрее работать и занимать меньше места в памяти, но гораздо важнее, что вы сразу получите сообщение, если ненароком попытаетесь сохранить в переменной данные, неподходящие для этого типа. Часто это происходит из-за незамеченной ошибки. Если конкретный тип переменной не объявлен, то редактор промолчит. При этом неизвестно, когда и как себя проявит такая ошибка. Ее поиски могут потребовать значительных затрат труда и времени.

> ***Примечание***
>
> Несмотря на категоричный тон, которым сделана эта рекомендация, есть ситуации, в которых выгоднее использовать для переменных тип **Variant**. Об одном из таких случаев рассказано в *гл. 14*.

13.1.6. Практический пример

Теперь вы знаете достаточно, чтобы перейти к созданию макроса **WordFlip_4**. Возьмите за основу макрос **WordFlip_3**, приведенный в *разд. 9.3.3* (см. листинг 9.3), добавьте в нем инструкцию описания переменной `blnSmartCutPaste`, инструкцию, в которой этой переменной присваивается значение свойства `Options.SmartCutPaste`, и инструкцию, которая восстанавливает значение данного свойства из переменной. Все эти инструкции мы уже составили в предыдущих подразделах. Сравните то, что у вас получилось, с подпрограммой, приведенной в листинге 13.1.

Листинг 13.1

```
Sub WordFlip_4()
' Перестановка соседних слов, учет пробелов при копировании
    Dim blnSmartCutPaste As Boolean

    blnSmartCutPaste = Options.SmartCutPaste

    Options.SmartCutPaste = True

    Selection.MoveLeft Unit:=wdWord, Count:=1
    Selection.MoveRight Unit:=wdWord, Count:=1, Extend:=wdExtend
    Selection.Cut
    Selection.MoveRight Unit:=wdWord, Count:=1
    Selection.Paste
    Selection.MoveLeft Unit:=wdWord, Count:=1
```

```
    Options.SmartCutPaste = blnSmartCutPaste
End Sub
```

Если в тексте этой подпрограммы вам что-то непонятно, повторите пройденный материал и, разобравшись, добавьте комментарии в текст подпрограммы.

13.2. Переменные вместо буфера обмена

Во всех макросах, которые мы до сих разбирали, в качестве кратковременного хранилища информации использовался *буфер обмена* (Clipboard). Когда такие макросы выполняются, содержимое буфера обмена изменяется. Поскольку это происходит незаметно для пользователя, возможны неприятности. Представьте себе, что вы вырезали из документа большой фрагмент в буфер обмена, чтобы перенести его в другое место. Теперь, перемещаясь по документу, чтобы найти то место, куда необходимо вставить этот фрагмент, вы заметили опечатку: были переставлены две буквы. Вы с удовольствием пользуетесь макросом **CharFlip_1** *(см. гл. 8)*, чтобы исправить эту опечатку (макрос при этом подменяет содержимое буфера обмена), затем вставляете это содержимое в нужное место и... Мы не знаем, как вы обычно выражаете сильные эмоции.

13.2.1. Постановка задачи

Необходимо усовершенствовать макрос **CharFlip_1**, меняющий местами буквы, стоящие слева и справа от курсора, так, чтобы содержимое буфера обмена при этом не изменялось. Дадим макросу имя **CharFlip_2**. Его действие опишем фразой: "Перестановка соседних букв с сохранением буфера обмена".

13.2.2. Решение задачи

Еще раз воспроизведем текст продпрограммы `CharFlip_1` (листинг 13.2).

Листинг 13.2

```
Sub CharFlip_1()
'
' CharFlip_1 Макрос
' Перестановка соседних букв
'
    Selection.MoveRight Unit:=wdCharacter, Count:=1, Extend:=wdExtend
```

```
    Selection.Cut
    Selection.MoveLeft Unit:=wdCharacter, Count:=1
    Selection.Paste

End Sub
```

Как вы понимаете, вместо буфера обмена мы воспользуемся переменной. Мы уже упоминали, что для текстовой информации предусмотрен тип данных **String** (строковый). В этой переменной мы сохраним выделенный текст. Поэтому для нее естественно выбрать имя `strSelection`. Вы уже знаете, что для объявления этой переменной служит инструкция:

```
Dim strSelection As String
```

Теперь запишем инструкцию, в которой этой переменной будет присвоен выделенный текст, чтобы впоследствии перенести его в документ. Для этого мы воспользуемся уже знакомым нам свойством `Text` (текст) объекта `Selection`. Напомним, что это свойство возвращает текст, находящийся в выделенной части активного окна документа, а если в окне ничего не выделено, оно возвращает символ, стоящий справа от курсора. При этом текст возвращается без информации о форматировании. Переменные типа `String` как раз подходят для хранения таких данных. Поэтому для того, чтобы сохранить выделенный текст в переменной `strSelection`, можно использовать знакомую вам инструкцию присваивания:

```
strSelection = Selection.Text
```

Поскольку в данном макросе нам требуется запомнить только один символ, стоящий справа от курсора, предварительно выделять его не обязательно и первая исполняемая инструкция подпрограммы `CharFlip_1` нам не потребуется.

Следующая за ней инструкция `Selection.Cut` выполняет две операции: она копирует выделенный текст в буфер обмена и удаляет его из документа. Копирование в буфер обмена мы заменили приведенной ранее инструкцией присваивания, но нам еще осталось удалить стоящую справа от курсора букву из документа. Чтобы узнать, как это делается, воспользуемся испытанным приемом: запишем макрос **Test5**, который выполняет это действие. Полученная подпрограмма приведена в листинге 13.3.

Листинг 13.3

```
Sub Test5()
'
' Test5 Макрос
```

```
' Удаление буквы справа от курсора
'
    Selection.Delete Unit:=wdCharacter, Count:=1
End Sub
```

В этой подпрограмме содержится необходимая нам инструкция, в которой применяется метод `Delete` (удалить) объекта `Selection`.

Эта инструкция действует так же, как нажатие клавиши <Delete>.

Аргументы `Unit` (единица измерения) и `Count` (счет) этого метода, а также константа `wdCharacter` (символ) нам знакомы. Они такие же, как у методов `MoveLeft` и `MoveRight`. Добавим, что эти аргументы являются необязательными, а значения `Unit:=wdCharacter` и `Count:=1` являются значениями по умолчанию. Следовательно, можно позволить себе записать эту инструкцию короче:

```
Selection.Delete
```

Таким образом, вместо инструкций:

```
Selection.MoveRight Unit:=wdCharacter, Count:=1, Extend:=wdExtend
Selection.Cut
```

запишем следующие инструкции, в которых описывается переменная `strSelection`, этой переменной присваивается символ, стоящий справа от курсора, и этот символ удаляется из документа.

```
Dim strSelection As String
strSelection = Selection.Text
Selection.Delete
```

Далее перепишем из подпрограммы `CharFlip_1` третью исполнимую инструкцию, которая перемещает курсор на один символ влево. Если учесть, что используемые в ней значения аргументов являются значениями по умолчанию, ее также можно записать короче:

```
Selection.MoveLeft
```

Чтобы вставить сохраненную в переменной `strSelection` букву в то место, где теперь находится курсор, следует записать инструкцию присваивания следующим образом:

```
Selection.Text = strSelection
```

В этой инструкции ничего нового для вас нет. Однако о том, что происходит в результате выполнения этой инструкции, в справке по Visual Basic сообщается весьма скупо. Чтобы ликвидировать этот пробел, пришлось немного поэкспериментировать, и вот что у нас получилось.

При присваивании значения свойству `Text` объекта `Selection` происходит следующее:

- если во время выполнения инструкции в окне документа есть выделенный фрагмент, то текст в нем будет заменен присваиваемым значением. Вставленный текст будет отформатирован так же, как был отформатирован первый символ в замененном выделенном фрагменте;
- если в окне документа ничего не выделено, то присваиваемое значение будет вставлено в то место, где находится курсор. Вставленный текст будет отформатирован так же, как был отформатирован символ, предшествующий курсору (или как следующий за курсором символ, если курсор находился в начале абзаца);
- вставленный текст будет выделен.

Из полученной информации сейчас для нас важно то, что вставленный символ будет выделен. Чтобы снять выделение и установить курсор после выделенного символа, добавим уже знакомую нам инструкцию, в которой применяется метод `MoveRight` объекта `Selection`. Действие этой инструкции аналогично однократному нажатию клавиши <→>. Замечая, что и на этот раз используемые в ней значения аргументов являются значениями по умолчанию, ее можно записать следующим образом:

```
Selection.MoveRight
```

В качестве последнего штриха добавим блок **With**, чтобы не повторять несколько раз ссылку на объект `Selection` в начале инструкций. Полученная в результате подпрограмма `CharFlip_2` представлена в листинге 13.4.

Листинг 13.4

```
Sub CharFlip_2()
' Перестановка соседних букв с сохранением буфера обмена
'
    Dim strSelection As String
    With Selection
        strSelection = .Text
        .Delete
        .MoveLeft
        .Text = strSelection
        .MoveRight
    End With
End Sub
```

При создании этой небольшой подпрограммы мы многому научились и повторили пройденное в предыдущих главах. В конце этой главы мы создадим еще один вариант макроса, переставляющего соседние буквы, чтобы продемонстрировать, как с помощью переменной можно присвоить значение аргументу метода.

Начнем с того, что, включив запись макроса **Test6**, введем с клавиатуры буквосочетание "по". В результате получим подпрограмму с единственной исполняемой инструкцией, приведенную в листинге 13.5.

Листинг 13.5

```
Sub Test6()
'
' Test6 Макрос
' Ввод буквосочетания "по"
'
    Selection.TypeText Text:="по"
End Sub
```

В инструкции

```
Selection.TypeText Text:="по"
```

применяется метод `TypeText` (печатать текст) объекта `Selection` (выделение). У этого метода единственный аргумент `Text` (текст), принимающий строковые значения.

Поэкспериментируйте с макросом **Test6**. Убедитесь, что во всех ситуациях результат работы макроса такой же, как при вводе текста с клавиатуры. Как мы видели, при вставке текста в документ с помощью присваивания значения свойству `Text` результат был другим.

Заменим строку "по" переменной `strSelection`. Если, как в нашем случае, в документе ничего не выделено, инструкция

```
Selection.TypeText Text:=strSelection
```

вставляет текст, хранящийся в строке `strSelection`, непосредственно перед курсором.

> ***Примечание***
>
> На будущее учтите, что если в документе есть выделенный текст, то так же как при вводе текста с клавиатуры, результат применения метода `TypeText` зависит от состояния флажка **Заменять выделенный фрагмент**, расположенного на вкладке **Правка** диалогового окна **Параметры**, открываемого из меню **Сер-**

вис командой **Параметры**. Если флажок установлен, вставляемый текст заменяет выделенный фрагмент. Если флажок снят, текст вставляется перед выделенным фрагментом.

Эта инструкция в нашем случае (когда в окне ничего не выделено) эквивалентна двум инструкциям

```
Selection.Text = strSelection
Selection.MoveRight
```

Разумеется, в блоке **With** `Selection` в начале инструкций ссылку на объект `Selection` можно опускать.

Таким образом, можно предложить еще один чуть более короткий вариант макроса для перестановки соседних букв (листинг 13.6).

Листинг 13.6

```
Sub CharFlip_3()
'
' Перестановка соседних букв с сохранением буфера обмена
'
    Dim strSelection As String
    With Selection
        strSelection = .Text
        .Delete
        .MoveLeft
        .TypeText Text:=strSelection
    End With
End Sub
```

Мы рассказали о методе `TypeText` для того, чтобы подготовить вас к встрече с ним. Когда при записи макроса вы вводите текст в документ, Visual Basic пользуется именно этим методом. Обратите внимание, что одни и те же результаты можно получить, применяя как свойства, так и методы. При присваивании значения свойству `Selection.Text` и при применении метода `Selection.TypeText` производятся сходные действия. Принципиальная разница в том, что в свойствах хранится информация, которую обычно можно и изменить, и извлечь. Сам по себе метод информацию не хранит. Метод может получать необходимую ему информацию с помощью аргументов. Он может вернуть информацию, полученную в результате его применения (как, например, метод `IsEqual`, о котором рассказывается в *разд. 15.3*), но для хранения информации методы не предназначены.

13.3. Что нового в этой главе

В этой вы научились использовать переменные при программировании макросов. Вы познакомились с такими понятиями, как переменная, имя переменной, тип переменной и значение переменной. Вы узнали о требованиях, которых надо придерживаться при выборе имен переменных, узнали о строковых и логических типах и о практике применения префиксов типа в именах переменных. Вы научились присваивать переменным значения, присваивать свойствам значения переменных и использовать переменные в качестве аргументов методов. Вы уже можете написать инструкцию объявления переменных, указать в этой инструкции имя и тип переменной. Более того, вы знаете, зачем это нужно, и умеете сделать объявление переменных обязательным. На конкретных примерах вы познакомились с тем, как с помощью переменных можно избавить макросы от побочного эффекта, связанного с использованием буфера обмена. По ходу дела вы научились писать инструкции для вставки текста, применяя свойство `Text` или метод `TypeText` объекта `Selection`.

Чтобы еще раз вспомнить об инструкциях и методах, с которыми вы познакомились в этой главе, посмотрите на табл. 13.1—13.2.

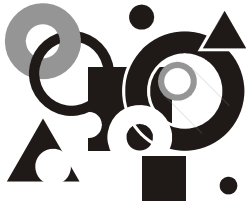
Таблица 13.1. *Новые инструкции*

`Dim` имя_переменной **`As`** тип_переменной
`Option Explicit`

Таблица 13.2. *Методы и их аргументы*

Объект	Метод	Аргумент	Константа
`Selection`	`Delete`	`Unit`	`wdCharacter`
		`Count`	
	`TypeText`	`Text`	

Глава 14



Классика программирования

Вычисления в Visual Basic. Вывод результатов. Встроенные функции. Ввод данных. Описание числовых переменных. Именованные аргументы. Объединение строк. Функции `Val` и `Replace`. Инструкции условного перехода. Логические выражения и логические операции. Сообщения об ошибках во время выполнения. Отладка программ. Цикл `For` и цикл `Do`. Прерывание цикла. Функции `Trim`, `Len` и `Mid`. Создание функций. Вызов функции и подпрограмм в функциях и подпрограммах. Область видимости переменных и процедур.

Эта глава занимает особое положение в нашей книге. Мы постарались сделать так, чтобы, с одной стороны, ее можно было изучать независимо от предыдущих глав, а с другой стороны, изложенный в ней материал можно было использовать не только в Microsoft Word, но и в других приложениях Microsoft Office. В этой главе рассказывается о базовых концепциях и приемах программирования, которые были реализованы во многих алгоритмических языках еще до наступления эры объектно-ориентированного программирования.

Особенности применения Visual Basic в том или ином приложении Microsoft Office определяются объектами, которые можно в нем использовать. А поскольку в этой главе мы не касаемся никаких объектов, то она не связана с конкретным приложением. Однако в следующей главе мы уже будем применять полученные при изучении этой главы знания для объектов Microsoft Word.

Переходя к изучению этой главы, вы делаете следующий важный шаг. Дело в том, что подпрограммы, которые создаются при автоматической записи макросов, обладают весьма ограниченным интеллектом. Они просто воспроизводят те действия, которые вы выполняли при записи макроса. Те же средства языка Visual Basic, которые еще несколько десятилетий назад позволили создавать программы с такими возможностями, что философы всерьез обсуждали вопрос "Могут ли машины мыслить?", в автоматически создаваемых редактором Visual Basic подпрограммах никогда не используются.

В этой главе мы хотим научить вас применять такие средства программирования.

- Во-первых, они позволяют макросам производить вычисления, используя арифметические, строковые и логические операции, а также встроенные математические и строковые функции. Тем самым макросы получают возможность создавать новые данные на базе существующей информации.
- Во-вторых, они обеспечивают взаимодействие макроса с пользователем, позволяя макросам запрашивать у пользователя необходимую им информацию, а также сообщать пользователю о полученных результатах.
- В-третьих, и это принципиальный момент, они позволяют макросам самим принимать решения о выполнении тех или иных инструкций на основе имеющихся у них данных.
- В-четвертых, эти средства позволяют программистам создавать независимые программные элементы — функции и подпрограммы, — из которых можно быстро собирать готовые программные продукты. Благодаря этому становится возможной разработка крупных программных проектов совместными усилиями многих программистов.

14.1. Калькулятор: функции и выражения

Поскольку вы уже познакомились с редактором Visual Basic, мы рекомендуем вам использовать его вместо калькулятора. Выполнять вычисления с помощью этого редактора очень удобно, и поэтому мы решили показать вам, как это делается, а заодно рассказать о некоторых основных средствах языка VBA на примере вычислительных задач.

Предварительно надо сказать, что редактор Visual Basic, с помощью которого производится программирование на языке VBA, используется не только в Microsoft Word, но и в других приложениях Microsoft Office, таких как Excel, Access, PowerPoint и Outlook. Разница только в наборе объектов и способах работы с ними. Если до сих пор мы основное внимание уделяли изучению объектов Microsoft Word, то в этом разделе мы в основном будем рассматривать универсальные средства программирования, которые можно применять во всех перечисленных приложениях.

14.1.1. Подготовка рабочего места

В этой главе мы приводим много примеров подпрограмм. Все они содержатся в шаблоне MacrosBook.dot *(см. гл. 7)*, но будет гораздо больше пользы, если вы будете записывать их самостоятельно. Для этой цели мы рекомендуем создать отдельный модуль **Classic** (классический) в шаблоне Normal.dot.

В *гл. 10* мы рассказали вам, как создать модуль, но на тот случай если вы забыли, как это делается, приводим соответствующие инструкции.

1. Откройте редактор Visual Basic (**Сервис | Макрос | Редактор Visual Basic**).
2. В редакторе Visual Basic откройте (если оно не открыто) окно **Project** (Проект) (**View** (Вид) **| Project Explorer** (Проводник проектов)).
3. В редакторе Visual Basic откройте (если оно не открыто) окно **Properties** (Свойства) (**View** (Вид) **| Properties Window** (Окно свойств)).

   Далее в проекте **Normal** (он соответствует шаблону Normal.dot) мы с вами создадим модуль **Classic** (классический). При этом мы предполагаем, что в своем шаблоне Normal.dot вы такого модуля не создавали. Если случайно такой модуль у вас в этом проекте есть, придумайте для нового модуля другое имя.
4. В окне **Project — <название_проекта>** выберите проект **Normal** (если он не выбран).
5. В меню **Insert** (Вставить) выберите пункт **Module** (модуль).
6. В окне **Properties — Module1** измените у выделенного свойства **Name** (Имя) значение **Module1** на **Classic**.

   В окне **Project — Normal** раскроется ветвь, ведущая к модулю **Classic**. В окне редактора откроется и станет активным окно модуля с заголовком **Normal — Classic (Code)**.

В результате вы получили в свое распоряжение отдельное чистое окно, в котором будете создавать подпрограммы. В окне должна быть только одна строка

```
Option Explicit
```

представляющая собой инструкцию, требующую обязательного объявления переменных. Она появляется у тех читателей, которые последовали нашему совету, приведенному в *разд. 13.1.5*.

Теперь можно приступить к разработке программ.

14.1.2. Вывод результатов вычислений

Прежде, чем что-то вычислять, подумаем, как мы узнаем, что у нас получилось. Вообще говоря, результаты можно вывести в окне сообщения, либо в строке состояния, либо в окне **Immediate**. Кроме того, их можно также вставить в текст документа или записать в файл (рассказ о выводе результатов в файл выходит за рамки нашей книги). Отметим, что средства вывода в строку состояния и в документ зависят от конкретного приложения. Поэтому мы начнем с универсальной функции `MsgBox` (*message* — сообщение; *box* —

окно). Напишем подпрограмму, которая выведет сообщение "Мое первое сообщение". Назовем ее `MyMsg1` (здесь *my* — мое).

Введите в окне модуля после строки **`Option Explicit`** следующий текст (ввод второй строки начините клавишей <Tab>):

```
sub MyMsg1()
    msgbox "Мое первое сообщение"
end sub
```

Если вы нигде не ошибетесь, в окне макроса этот текст преобразуется к виду, приведенному в листинге 14.1.

Листинг 14.1

```
Sub MyMsg1()
    MsgBox "Мое первое сообщение"
End Sub
```

После инструкции **`Option Explicit`** появится сплошная линия, ключевые слова станут темно-синего цвета и их первая буква станет прописной, в названии функции `MsgBox` появятся две прописные буквы. Все это означает, что Visual Basic понял введенный текст.

> ***Совет***
>
> Вводите слова, знакомые редактору Visual Basic, в нижнем регистре. Редактор сам запишет их "как надо", и тем даст вам знать, что опечаток нет.

Чтобы выяснить результат работы подпрограммы, установите курсор на любой из инструкций подпрограммы и выберите в меню **Run** (Запустить) пункт **Run Sub/UserForm** (Выполнить подпрограмму или пользовательскую форму). Вместо этого можно нажать на клавиатуре клавишу <F5> или на панели инструментов нажать кнопку **Run Sub/UserForm** (рис. 14.1).

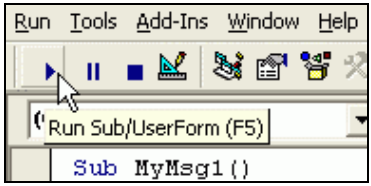


Рис. 14.1. Кнопка запуска подпрограммы в редакторе Visual Basic

На экране появится приложение Microsoft Word, а поверх окна документа откроется диалоговое окно с сообщением (рис. 14.2).

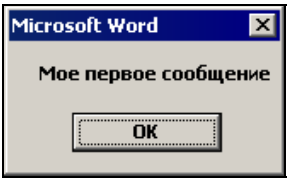


Рис. 14.2. Результат работы подпрограммы `MyMsg1`

Чтобы закрыть это окно, следует, как принято в программах Microsoft, нажать мышью кнопку **OK** (эквивалент — клавиша <Enter> или клавиша ввода пробела), что означает подтверждение, или крестик в правом верхнем углу окна (эквивалент — клавиша <Esc>), что означает отмену. Хотя в нашем примере все эти действия эквивалентны, в дальнейшем *(см. разд. 14.2)* мы научим вас, как в своих программах отличать подтверждение от отмены. Теперь скопируйте подпрограмму `MyMsg1`, измените ее имя на `MyMsg2` и вместо текста сообщения введите цифру 5 (числа можно вводить без кавычек).

Получится подпрограмма, приведенная в листинге 14.2.

Листинг 14.2

```
Sub MyMsg2()
    MsgBox 5
End Sub
```

Вы уже знаете, как ее запустить, чтобы убедиться, что функция `MsgBox` может выводить числа. Простейший калькулятор почти готов.

Вместо цифры 5 можно записать числовые выражения, состоящие из чисел, круглых скобок и знаков арифметических действий. Чуть подробнее остановимся на арифметических действиях. Знак плюс (+) вопросов не вызывает. В качестве знака вычитания и признака отрицательного числа используется дефис (-). В качестве знака умножения используется звездочка (*), знака деления — косая черта (/), знака возведения в степень — "крышка" (^). Кроме того, предусмотрен знак деления нацело (\) и ключевое слово **Mod** для вычисления остатка от деления, но эти операции требуют более подробного обсуждения и на них мы останавливаться не будем. Установлены естественные правила старшинства (приоритета) операций, но, если вы не вполне уверены, что применяете их правильно, используйте скобки. Кроме того, в выражениях можно использовать переменные и функции.

Например, для вычисления площади круга диаметром 6 см по формуле $\pi d^2/4$ подпрограмма может выглядеть следующим образом (листинг 14.3):

Листинг 14.3

```
Sub Calc_1()
    MsgBox 3.141593*(6^2)/4
End Sub
```

В выражении `3.141593*(6^2)/4` скобки ставить необязательно, поскольку операция возведения в степень имеет наивысший приоритет, но со скобками оно читается легче.

На будущее учтите, что в качестве разделителя дробной и целой частей числа в тексте программы всегда используется точка (независимо от настройки Windows), а группы разрядов пробелом не отделяются.

Сразу видны преимущества Visual Basic по сравнению с обычным калькулятором. Все данные, введенные с клавиатуры, находятся перед глазами, формулы легко читаются и очень просто вносить исправления. Для запоминания промежуточных результатов очень эффективно можно пользоваться переменными. Более того, для одноразовых вычислений можно даже не писать подпрограмму, а воспользоваться окном **Immediate** (Непосредственный), предназначенным, в первую очередь, для отладки программ.

Чтобы открыть это окно, в меню **View** (Вид) выберите команду **Immediate Window** (Окно непосредственных вычислений), воспользуйтесь сочетанием клавиш <Ctrl>+<G> или нажмите на панели инструментов **Debug** (Отладка) кнопку  . Если панель инструментов отсутствует, вызвать ее на экран можно теми же способами, что и в приложении Microsoft Word *(см. гл. 3)*.

Полезные сведения

Выполнять вычисления в окне **Immediate** очень просто. Начните строку со знака вопроса, затем введите выражение, которое необходимо вычислить, и нажмите клавишу <Enter>. В следующей строке сразу появится результат. В этом окне можно, используя переменные (без их предварительного описания), выполнять инструкции присвоения. Можно также использовать привычные средства редактирования текста, например, скопировать результат вычислений в буфер обмена. Содержимое окна **Immediate** после закрытия редактора Visual Basic не сохраняется.

14.1.3. Встроенные математические функции

Мы уже упоминали, что в выражениях можно использовать функции. В VBA предусмотрено много встроенных функций. Среди них есть математические

функции, которые записываются аналогично тому, как мы записывали функции в школе. Например, чтобы вычислить (2/3)sin(π/6), можно использовать следующее выражение: `(2/3)*Sin(3.141593/6)`. Надо только не забывать явно записывать знак умножения.

Перечень некоторых математических функций приведен в табл. 14.1.

Таблица 14.1. *Перечень некоторых математических функций*

Функция	Название
`Abs`	Абсолютная величина числа
`Atn`	Арктангенс
`Cos`	Косинус
`Exp`	Экспонента
`Log`	Натуральный логарифм (не десятичный!)
`Sin`	Синус
`Sqr`	Корень квадратный
`Tan`	Тангенс

Если вычисления по одной и той же формуле выполняются регулярно, целесообразно написать для этого специальную подпрограмму, которую можно вызывать как обычный макрос, не переходя в редактор Visual Basic. Однако в такой подпрограмме необходимо предусмотреть ввод исходных данных.

14.1.4. Ввод данных с помощью функции *InputBox*

Ввод данных можно организовать с помощью встроенной функции `InputBox` (окно ввода). Эта функция выводит на экран диалоговое окно, в котором пользователь может ввести строку. После нажатия в нем кнопки **OK** окно закрывается, а программа продолжает работать, используя эту строку в качестве значения функции. Программисты говорят, что функция "возвращает" введенную пользователем строку.

У функции `InputBox` первый аргумент является обязательным. Это строка, которая отображается как сообщение в диалоговом окне. Остальные аргументы необязательны. Второй аргумент — строка, которая будет отображаться в заголовке диалогового окна, третий аргумент — строка, которая

будет отображаться в поле ввода при открытии диалогового окна. Остальные аргументы используются редко, и описывать мы их не будем.

В листинге 14.4 приведена подпрограмма `AreaOfCircle_1` (площадь круга), в которой пользователь вводит диаметр круга в диалоговом окне, а результат читает в окне сообщения. За основу взята программу `Calc_1`, в которой цифра 6 заменена функцией `InputBox`, которая возвращает число, введенное пользователем.

Листинг 14.4

```
Sub AreaOfCircle_1()
'Вычисление площади круга
'Простейший вариант
    MsgBox 3.141593 * InputBox("Введите диаметр круга" _
        , "Вычисление площади круга", "сюда...") ^ 2 / 4
End Sub
```

Выполните эту программу из редактора Visual Basic одним из способов, о котором мы уже рассказали ранее *(см. разд. 14.1.2)*. Произойдет переход к окну Microsoft Word, на фоне которого появится диалоговое окно ввода (рис. 14.3).

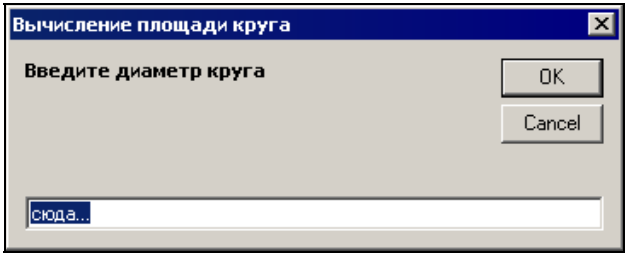


Рис. 14.3. Ввод данных с помощью встроенной функции `InputBox` в подпрограмме `AreaOfCircle_1`

Наверное, вы уже знаете, что слово *Cancel* переводится как отмена. Про эту кнопку мы поговорим немного позже. А сейчас введите число, например, 3,5 и нажмите кнопку **OK**. Сразу получите результат, приведенный на рис. 14.4.

Программа работает, однако формула, по которой производятся вычисления, в тексте программы читается плохо. Это серьезный недостаток, поскольку непонятный текст — потенциальный источник ошибок. Если

вычисления громоздкие, наглядность записи формул приобретает первостепенное значение.

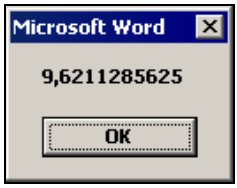


Рис. 14.4. Вычисление площади круга диаметром 3,5 с помощью подпрограммы `AreaOfCircle_1`

Для того чтобы формулы в тексте программы легче воспринимались читателем, следует использовать переменные.

Здесь необходимо сделать два замечания, которые противоречат нашим собственным (а также общепринятым) рекомендациям. Жизнь сложна, и не всегда мы действуем по тем правилам, которые мы сами же для себя устанавливаем.

14.1.5. Рекомендации по описанию переменных для числовых данных

Первое замечание относится к выбору типа переменных для хранения числовых данных. Мы уже рассказывали *(см. гл. 13)* как описывать и применять переменные, причем мы вам рекомендовали не только явно описывать переменные, но и явно указывать тип переменных. Однако чтобы выполнить эти рекомендации, следует хорошо понимать, к каким последствиям может привести выбор того или иного типа данных. Типы числовых данных (`Byte`, `Integer`, `Long`, `Currency`, `Single`, `Double`) отличаются количеством разрядов, выделенных для записи чисел, и способом представления чисел. Правильный выбор типа данных позволяет повысить скорость выполнения арифметических операций, разместить в оперативной памяти большее количество чисел и обеспечить необходимую точность вычислений. В некоторых случаях явное описание типов данных позволяет редактору Visual Basic обнаружить ошибку раньше, чем она повлияет на результат вычислений.

Вместе с тем мы предполагаем, что наши читатели не специализируются на вычислениях, и изучение типов числовых данных они могут отложить до того времени, когда у них возникнет в этом потребность. На данном этапе мы рекомендуем использовать для чисел универсальный тип `Variant`, который задается по умолчанию, т. е. тогда, когда тип переменных явно не указан. Если в своей программе вы храните в переменных меньше десяти тысяч

чисел и скорость ее работы вас удовлетворяет, вам не надо беспокоиться о том, что переменные этого типа занимают много памяти и вычисления с их применением производятся медленнее. Обычно точность вычислений с данными этого типа достаточна, и в переменных могут храниться данные различного типа, включая строки и логические константы. Если же в вашей практике возникнут проблемы, связанные с недостатком оперативной памяти для хранения переменных или с недостаточным быстродействием программы, прочитайте о типах данных в справке редактора Visual Basic или в любом систематическом описании языка VBA.

В дополнительных сведениях *(см. разд. 14.6.2)* мы немного рассказываем о погрешностях, связанных с формой представления чисел в памяти компьютера.

Второе замечание относится к выбору обозначений. Здесь основной принцип — повышение наглядности программ, поэтому, если у вас уже есть привычные обозначения для числовых переменных и вы выбираете для всех числовых переменных тип `Variant` (указывая его явным образом или просто не указывая тип переменных), откажитесь от использования префиксов типа для таких переменных вопреки нашим предыдущим рекомендациям. Однако не отказывайтесь от явного описания переменных, поскольку это уменьшит число ошибок, связанных с опечатками в именах переменных.

С учетом сделанных замечаний введем переменные в подпрограмму `AreaOfCircle_1`, назвав новый вариант `AreaOfCircle_2` (листинг 14.5).

Листинг 14.5

```
Sub AreaOfCircle_2()
'Вычисление площади круга
'Описание переменных
    Dim pi '3.141593
    Dim d 'диаметр круга
    Dim strTitle As String 'заголовок диалогового окна ввода
    Dim strPrompt As String 'сообщение диалогового окна ввода
    Dim strDefault As String 'исходное значение поля ввода
    pi = 3.141593
    strTitle = "Вычисление площади круга"
    strPrompt = "Введите диаметр круга"
    strDefault = "сюда..."
    d = InputBox(strPrompt, strTitle, strDefault)
    MsgBox pi * d ^ 2 / 4
End Sub
```

Здесь мы использовали пять переменных: `d`, `s`, `strTitle`, `strPrompt` и `strDefault`. Их смысл поясняется в комментариях, которые, как вы видите, могут располагаться и в конце инструкций. Размер программы значительно увеличился, зато читать ее стало гораздо легче.

Обратите внимание, что функция `InputBox` возвращает строку, а в формуле мы эту строку умножаем на число. Несмотря на то, что в памяти компьютера строки и числа хранятся различным образом, ошибок не возникает. Редактор Visual Basic автоматически выполнит необходимые преобразования даже в том случае, если для переменной `d` явно указать тип **`String`**. В некоторых книжках и даже в справке Visual Basic вы прочтете, что при этом появляются сообщения об ошибке, однако опыт показывает, что это не так.

14.1.6. Именованные аргументы функций

Теперь, когда вы немного познакомились с функциями `MsgBox` и `InputBox`, вам будет полезно узнать, что у этих функций имеются так называемые *именованные аргументы*. С именованными аргументами мы уже многократно встречались, хотя и не акцентировали на этом внимания. Такие аргументы используются у методов объектов. Это означает, что инструкцию, в которой вызывается функция `InputBox`,

```
d = InputBox(strPrompt, strTitle, strDefault)
```

можно записать также и в другом эквивалентном виде:

```
d = InputBox(Prompt:=strPrompt, Title:=strTitle, Default:=strDefault)
```

Здесь `Prompt` (*запрос*), `Title` (заголовок) и `Default` (стандарт) — наименования аргументов функции `InputBox`. Обратите внимание, имена переменным мы придумываем сами, а имена именованных аргументов для каждой функции задаются разработчиками функций так же, как и имя самой функции. Когда инструкция записывается в виде

```
d = InputBox(strPrompt, strTitle, strDefault)
```

соответствие между аргументами и их значениями определяется их порядком следования: первый аргумент — это сообщение в окне, второй аргумент — заголовок окна и т. д. В этом случае говорят, что применяется *позиционная передача аргументов*.

Преимуществами использования именованных аргументов является большая наглядность, а также возможность менять порядок аргументов при обращении к функции. Поскольку имена именованных аргументов редактору Visual Basic известны, он сможет распознать аргументы функции независимо от того, в каком порядке они записаны.

Приведем еще один пример. Воспользуемся именованными аргументами, чтобы передать третий (необязательный) аргумент для функции `MsgBox`. Таким образом вместо инструкции

```
MsgBox pi * d ^ 2 / 4
```

используем инструкцию

```
MsgBox Prompt:= pi * d ^ 2 / 4, Title:=strTitle
```

Здесь `Prompt` (запрос) и `Title` (заголовок) — наименования аргументов функции `MsgBox`. И по форме и по содержанию они совпадают с соответствующими аргументами функции `InputBox`. Разница в том, что у функции `InputBox` это первый и второй аргументы, а у функции `MsgBox` это первый и третий аргументы. Если бы мы воспользовались позиционной передачей аргументов, обращение к функции `MsgBox` выглядело бы следующим образом:

```
MsgBox pi * d ^ 2 / 4, , strTitle
```

В этой записи второй (необязательный) аргумент пропущен, но запятая осталась. Она показывает, что пропущен именно второй аргумент.

Для редактора Visual Basic эти две формы обращения к функции `MsgBox` полностью эквивалентны, но для человеческого восприятия первая запись нагляднее, что немаловажно.

У внимательного читателя наверняка возник вопрос: почему при обращении к функции `InputBox` список аргументов берется в скобки, а при обращении к функции `MsgBox` — нет. Постараемся внести ясность.

Предварительно заметим, что если мы называем `MsgBox` функцией, значит, она возвращает значение. Просто в наших примерах это значение не используется. Теперь можно сформулировать правило.

Использование скобок при обращении к функции

- Если значение функции используется (например, она входит в состав выражения), то список аргументов заключается в скобки.
- Если значение функции не используется, то можно записывать обращение к функции без заключения списка аргументов в скобки. Тогда значение функции ничему не присваивается.
- Если же значение функции не используется, то можно перед обращением к функции написать ключевое слово **`Call`**. Тогда список аргументов должен быть заключен в скобки.

Необязательное ключевое слово **`Call`** (вызов) показывает, что в данной инструкции происходит обращение к процедуре (функции или подпрограмме). Его полезно использовать, когда название процедуры может быть незнакомо читателю, и особенно в тех случаях, когда у процедуры нет аргументов. Это повышает наглядность программы.

Например, можно использовать такую запись:

```
Call MsgBox(Prompt:= pi * d ^ 2 / 4, Title:=strTitle)
```

> ***Совет***
>
> Чтобы не думать каждый раз, писать ли скобки при обращении к функции, можно взять за правило всегда заключать в скобки список аргументов, а если значение функции не используется, добавлять ключевое слово `Call`.

Мы продолжим эту тему, когда будем говорить о том, как создавать функции самостоятельно и как обращаться из одной подпрограммы к другой *(см. разд. 14.4)*.

14.1.7. Объединение строк

Еще один вопрос, который мы хотим затронуть в связи с подпрограммой вычисления площади круга, связан с тем, что в окне вывода приводится только численный результат без сопроводительного текста. Чтобы исправить этот недостаток, нам надо научиться производить операции над строками. Фактически существует только одна операция, которую можно выполнять над строками, операция объединения двух строк (точнее говоря, *строковых выражений*, т. е. выражений, значениями которых являются строки). В качестве знака операции *объединения* двух строковых выражений используется амперсанд (`&`). Допускается использовать знак плюс (`+`), но чтобы не перепутать эту операцию со сложением чисел, этого делать не рекомендуется. Часто эту операцию называют *конкатенацией*.

Предположим, вам потребовалось передать функции `MsgBox` в качестве значения аргумента `Prompt` строковое выражение, составленное из строки `"Площадь круга: "` и численного выражения `pi * d ^ 2 / 4` (обратите внимание на пробел после двоеточия).

Задача осложняется тем, что необходимо превратить в часть строки численное выражение. Оказывается, можно использовать операцию объединения строки с численным выражением и редактор Visual Basic воспримет это как должное: в таких случаях предусмотрено автоматическое преобразование численного значения в строковое.

Таким образом, с помощью операции объединения можно составить искомое строковое выражение:

```
"Площадь круга: " & pi * d ^ 2 / 4
```

которое можно передать в качестве аргумента `Prompt` функции `MsgBox`.

Теперь попробуем сделать так, чтобы в окне результатов отображалось бы и исходное значение диаметра круга, причем в отдельной строке.

В Microsoft Windows переход на новую строку задается сочетанием двух знаков: знака возврата коретки и знака перевода строки. Для этого сочетания в VBA предусмотрена специальная строковая константа `vbCrLf` (vb — сокращение Visual Basic; Cr — сокращение от Carriage Return — возврат каретки, Lf — сокращение от Line Feed — перевод строки). Чтобы программа легче читалась, введем дополнительные строковые переменные `strPrompt1` и `strPrompt2`, из которых затем сформируем значение для переменной `strPrompt`.

Все вышеизложенное суммируем в виде подпрограммы `AreaOfCircle_3` (листинг 14.6).

Листинг 14.6

```
Sub AreaOfCircle_3()
'Вычисление площади круга
'Вывод исходных данных - слияние строк
    Dim pi '3.141593
    Dim d 'диаметр круга
    Dim s 'площадь круга
    Dim strTitle As String 'заголовок диалогового окна ввода
    Dim strPrompt As String 'сообщение диалогового окна ввода
    Dim strDefault As String 'исходное значение поля ввода
    Dim strPrompt1 As String 'первая строка диалогового окна вывода
    Dim strPrompt2 As String 'вторая строка диалогового окна вывода
    pi = 3.141593
    strTitle = "Вычисление площади круга"
    strPrompt = "Введите диаметр круга"
    strDefault = "сюда..."
    d = InputBox(Prompt:=strPrompt, Title:=strTitle, Default:=strDefault)
    s = pi * d ^ 2 / 4
    strPrompt1 = "Диаметр круга: " & d
    strPrompt2 = "Площадь круга: " & s
    strPrompt = strPrompt1 & vbCrLf & strPrompt2
    Call MsgBox(Prompt:=strPrompt, Title:=strTitle)
End Sub
```

Результат работы этой подпрограммы показан на рис. 14.5.

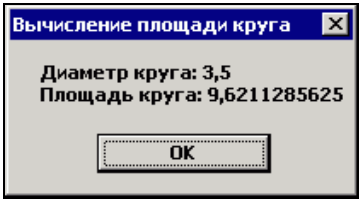


Рис. 14.5. Пример вывода результатов работы подпрограммы `AreaOfCircle_3`

14.1.8. Встроенные функции *Val* и *Replace*

Числа и строки хранятся в компьютере в различном представлении, а арифметические действия производятся не над строками, а над числами. Даже если в поле ввода мы записываем число, функция `InputBox` возвращает строку, т. е. значение типа `String`. В наших примерах эту функцию или переменную, содержащую строку, возвращенную этой функцией, мы без тени смущения использовали в арифметическом выражении, когда вычисляли площадь круга. Во время выполнения подпрограммы автоматически происходит преобразование строки в число, с которым далее производятся соответствующие арифметические действия. Все это происходит незаметно для пользователя, и мы могли бы вам об этом не рассказывать, если бы пользователь никогда не делал ошибок при вводе числа. Например, попробуйте при вводе значения диаметра вместо запятой ввести точку или букву. Преобразовать такую строку в число не удается, и на экране появится сообщение об ошибке (рис. 14.6).

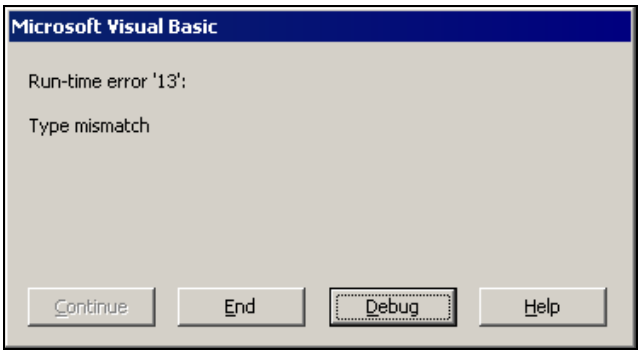


Рис. 14.6. Сообщение об ошибке при неудачной попытке преобразовать сроку в число

Это одно из стандартных диалоговых окон, которые появляются, когда во время выполнения программы, написанной на языке VBA, возникает ситуация, препятствующая ее нормальной работе.

Тексты в этом диалоговом окне переводятся следующим образом:

- **Run-time error** — ошибка во время выполнения
- **Type mismatch** — несоответствие типов
- **Continue** — продолжить
- **End** — закончить
- **Debug** — отладка
- **Help** — справка

Кнопка **Continue** (Продолжить) в данном окне недоступна, поскольку возникла ситуация, когда продолжение невозможно. Кнопка **End** (Закончить) закрывает это окно и завершает работу программы. Кнопка **Debug** (Отладка) доступна, если документ или шаблон с выполняемой подпрограммой доступен для изменения, т. е. открыт или присоединен к документу *(см. гл. 10)*. Если нажать эту кнопку, то произойдет переход к тексту программы и будет выделена инструкция, в которой выявилась ошибка. В редакторе Visual Basic предусмотрены разнообразные средства отладки программы. С их помощью опытный программист, умеющий работать с редактором Visual Basic, может проанализировать ход выполнения программы и понять, что явилось причиной ошибки. Кнопка **Help** (Справка) вызывает раздел справочной системы, в котором на английском языке рассказывается, в каких случаях может произойти эта ошибка, но поскольку эти сведения относятся к особенностям языка VBA, а не к конкретным инструкциям данной программы, чаще всего их недостаточно, чтобы понять, в чем именно кроется причина ошибки. На этот раз мы знаем, в чем ошибка, поэтому можно нажать на кнопку **End**.

Опыт показывает, что из стандартных сообщений об ошибках непросто понять, что же явилось причиной ошибки и как ее исправить. Поэтому желательно составлять программы таким образом, чтобы пользователь получал понятное сообщение об ошибках во входной информации до того, как будет сделана попытка выполнить недопустимую операцию или, что еще хуже, будут получены ошибочные результаты.

По-видимому, самым простым способом сделать так, чтобы пользователь не получал приведенное ранее сообщение об ошибке в случае опечатки при вводе чисел, является отказ от автоматического преобразования введенной строки в число. Вместо этого можно использовать для этой цели встроенную функцию `Val` (от *value* — значение). Аргументом этой функции является строка, а значением — число. Функция `Val` пытается проинтерпретировать начальную часть строки как число и возвращает это число в числовом

формате. Пробелы, символы табуляции и символы перевода строки игнорируются. Используются все те символы в начале строки, которые в совокупности можно проинтерпретировать как число. В качестве разделителя целой и дробной частей числа функция `Val` воспринимает только точку, независимо от настройки системы Windows. Если же в начале строки отсутствует число (как мы уже сказали, пробелы, символы табуляции и символы перехода на другую строку игнорируются), функция `Val` возвращает ноль без сообщения об ошибке. Текст или знаки препинания, следующие за числом, тоже игнорируются без каких-либо сообщений. Ошибки во входной информации не проверяются, но поскольку вместе с вычисленной площадью на экран выводится исходное значение диаметра, у пользователя есть возможность его проверить самому.

Чтобы использовать функцию `Val`, опишем переменную `strD` типа `String` и присвоим этой переменной введенное строковое значение с помощью инструкции:

```
strD = InputBox(Prompt:=strPrompt, Title:=strTitle, Default:=strDefault)
```

Далее воспользуемся функцией `Val` для преобразования строки в число:

```
d= Val(strD)
```

После этого будем выполнять арифметические действия с полученным в результате числом.

Сообщений об ошибке преобразования типа мы больше получать не будем, однако нам надо будет все время помнить, что при вводе дробных чисел в качестве разделителя целой и дробной частей следует использовать не запятую, а точку, иначе при вычислениях будет использоваться только целая часть введенного числа. Чтобы ликвидировать этот недостаток, можно во введенной строке всегда заменять все запятые точками и про точки больше не вспоминать.

В языке VBA для выполнения замены в строке предусмотрена функция `Replace` (заменить). Например, получить строку `strD1`, отличающуюся от строки `strD` тем, что в ней запятые заменены точками, можно с помощью следующей инструкции:

```
strD1 = Replace(Expression:=srtD, Find:=",", Replace:=".")
```

Аргументы этой функции: `Expression` (выражение) — строка, в которой выполняется замена, `Find` (найти) — строка, которая ищется в строке `Expression` и которая подлежит замене, `Replace` (заменить) — строка, которая заменяет в строке `Expression` все вхождения строки `Find`. Функция возвращает строку, полученную в результате замены.

Функция `Replace` имеет и другие аргументы, но поскольку они необязательны, то их можно и не указывать.

Воспользуемся тем, что в инструкциях присвоения можно использовать переменную, которой присваивается значение, так же как и в присваиваемом выражении, т. е. и слева, и справа от знака равенства. В нашем случае это означает, что дополнительную строку `strD1` использовать не обязательно. Результат замены можно сохранить в той же переменной, в которой до этого находилась изменяемая строка:

```
strD = Replace(Expression:=srtD, Find:=",", Replace:=".")
```

В дальнейшем мы часто будем пользоваться такой возможностью, но хотим предупредить, что применение одной и той же переменной для различных целей, хотя и экономит два слова в инструкции описания, снижает наглядность и осложняет поиск ошибки.

В итоге усовершенствованная подпрограмма вычисления площади круга приобретает вид, показанный в листинге 14.7.

Листинг 14.7

```
Sub AreaOfCircle_4()
'Вычисление площади круга
'Функции Val и Replace
    Dim pi '3.141593
    Dim d 'диаметр круга
    Dim s 'площадь круга
    Dim strTitle As String 'заголовок диалогового окна ввода
    Dim strPrompt As String 'сообщение диалогового окна ввода
    Dim strDefault As String 'исходное значение поля ввода
    Dim strD As String 'введенное значение диаметра
    Dim strPrompt1 As String 'первая строка диалогового окна вывода
    Dim strPrompt2 As String 'вторая строка диалогового окна вывода
    pi = 3.141593
    strTitle = "Вычисление площади круга"
    strPrompt = "Введите диаметр круга"
    strDefault = "сюда..."
    strD = InputBox(Prompt:=strPrompt, Title:=strTitle _
                                          , Default:=strDefault)
    strD = Replace(Expression:=strD, Find:=",", Replace:=".")
    d = Val(strD)
    s = pi * d ^ 2 / 4
    strPrompt1 = "Диаметр круга: " & d
    strPrompt2 = "Площадь круга: " & s
```

```
    strPrompt = strPrompt1 & vbCrLf & strPrompt2
    Call MsgBox(Prompt:=strPrompt, Title:=strTitle)
End Sub
```

Этот вариант программы более устойчив к погрешностям ввода исходной информации, поскольку можно при вводе данных использовать и точку, и запятую. Однако ничего не дается даром, и поэтому необходимо более тщательно следить, при каком значении диаметра был выполнен расчет площади круга, поскольку эта подпрограмма никак не реагирует на ошибки во входной информации.

14.2. Условный переход

Наверное, можно сказать, что уже в шарманке или в часах с боем была заложена программа воспроизведения мелодии. Но все-таки автоматическое воспроизведение даже сложной последовательности действий — это только первый шаг на пути к программированию. Следующий шаг — предоставление пользователю возможности изменять программу. Однако вряд ли вы чувствовали себя программистом, когда изменяли программу стиральной машины или настраивали автоматическую запись телефильма на видеомагнитофоне. Принципиальным моментом является наличие у программы возможности самостоятельно "принимать решение" о выборе той или иной последовательности действий в зависимости от исходных данных и полученных результатов. Теперь мы будем учиться писать такие программы. Программы, создаваемые при автоматической записи макросов, также как и те программы, которыми мы с вами занимались, этой возможностью не обладали. Мы начнем с описания конструкций, осуществляющих *условный переход*, т. е. позволяющих выбрать, какие выполнять инструкции в зависимости от некоторого условия.

14.2.1. Блок *If*

Вернемся к подпрограммам вычисления площади круга, которой мы занимались в предыдущем разделе. В них мы использовали функцию `InputBox`, которая отображает на экране диалоговое окно для ввода текста с кнопками **OK** и **Cancel** (см. рис. 14.3).

Мы уже упомянули, что **Cancel** переводится как "отмена". Опыт работы в операционной системе Windows подсказывает, что эта кнопка должна останавливать выполнение программы, однако в наших подпрограммах этого не происходит. Например, в подпрограмме `AreaOfCircle_4` при нажатии кнопки **Cancel** (или нажатии на клавиатуре клавиши <Esc>) функция `InputBox`

возвращает пустую строку, функция `Val` возвращает 0 и программа вычисляет нулевую площадь круга. Если программа пишется для себя, то на такие мелочи можно не обращать внимания, но другие пользователи будут воспринимать это как ошибку. Кроме того, в более сложных программах, где вычисления могут занимать заметное время, отмена выполнения случайно запущенной программы является весьма полезной возможностью.

Для того чтобы "обработать" (как говорят программисты) нажатие кнопки **Cancel**, будем останавливать работу программы в тех случаях, когда функция `InputBox` возвращает пустую строку.

> ***Полезные сведения***
>
> При использовании функции `InputBox` нажатие кнопки **OK** при пустом поле ввода и нажатие кнопки **Cancel** приводят к одинаковому результату. Если вам это не подходит или требуется ввести сразу несколько параметров, можно создать свое диалоговое окно с любыми кнопками, флажками, переключателями и т. п. Для этого в редакторе Visual Basic предусмотрено мощное средство создания форм, называемое **UserForm** *(форма пользователя)*. Рассказ о нем выходит за рамки нашей книги.

Для остановки подпрограммы воспользуемся инструкцией

```
End
```

Эта инструкция состоит из одного ключевого слова **End** (закончить). Она заканчивает выполнение подпрограммы и освобождает память от всех использованных в макросе переменных.

Теперь все готово, чтобы продемонстрировать новую конструкцию. Мы вставим в подпрограмму небольшой фрагмент, в котором будет проверяться, совпадает ли значение строки `d` с пустой строкой (`""`), и если это так, то будем останавливать работу программы:

```
If d = "" Then
    End
End If
```

Обсудим новые инструкции, использованные в этом фрагменте. Инструкция

```
If d = "" Then
```

является так называемой инструкцией **If**, начальной инструкцией *блока* **If**. Темно-синий цвет, используемый редактором Visual Basic (или выделение полужирным шрифтом в нашей книге), показывает, что **If** (если) и **Then** (то) — ключевые слова языка Visual Basic. Смысл этой инструкции в том, что если выполняется условие, которое указано между ключевыми словами **If** и **Then**, то исполняются инструкции, расположенные между данной инструкцией и инструкцией **End If**, которая отмечает конец блока **If**.

Если указанное условие не выполняется, то все инструкции, находящиеся в блоке **If**, пропускаются. В нашем случае пропускается только одна инструкция **End**.

14.2.2. Операции сравнения и логические выражения

Знак равенства здесь выступает в новой роли *операции сравнения*. С его помощью записывается проверяемое условие. В общем случае в качестве условия используется так называемое *логическое выражение*. Тут необходимо ненадолго отвлечься, чтобы пояснить, что это такое.

Характерным признаком логических выражений является то, что они могут быть истинными или ложными, т. е. принимают значение **True** (истина) или **False** (ложь). Примерами логических выражений могут быть собственно *логические константы* **True** и **False**, свойства объектов, которые соответствуют флажкам и принимают значения **True** и **False**. Вспомните, что в *гл. 13* мы уже использовали *логические переменные*, в которых хранили значения таких свойств, и которые также являются логическим выражениями. Еще один пример логического выражения — результат сравнения двух выражений (как в нашем случае), т. е. два выражения, между которыми расположен один из знаков операций сравнения (табл. 14.2).

Таблица 14.2. *Знаки операций сравнения*

Знак	Обозначаемая операция
<	Меньше
<=	Меньше или равно
>	Больше
>=	Больше или равно
=	Равно
<>	Не равно

Заметьте, что сравниваться могут не только числовые, но и строковые, и даже логические выражения. Например, в нашем случае строка `d` сравнивается с пустой строкой `""` (и переменная, и конкретная строка являются частными случаями выражений).

Над логическими выражениями можно выполнять *логические операции*, результатом которых является также логическое выражение. Наиболее известные из них **Not** (нет), **And** (и) и **Or** (или). Более подробные сведения о логи-

ческих операциях мы приводить не будем, отметим только, что при использовании логических выражений с несколькими операциями рекомендуется чаще применять скобки, чтобы избежать ошибок, связанных с порядком выполнения операций. Примеры логических выражений нам еще будут встречаться, и не один раз.

Замечание

В книжках по программированию есть некоторая терминологическая несогласованность. Знаки арифметических и логических операций в справке Microsoft Word и некоторых книгах называют *операторами* (в справке Visual Basic — *Operator*). При этом в других (или даже в тех же самых книгах) оператором называют предложение языка, задающее действие или описание, т. е. то, что мы, следуя терминологии Microsoft Word, называем инструкцией (в справке Visual Basic — *Statement*). Мы термин "оператор" для этих целей использовать не будем.

Вернемся к нашей программе. После добавления инструкций, в которых обрабатывается нажатие кнопки **Cancel**, подпрограмма для вычисления площади круга приобретет вид, показанный в листинге 14.8.

Листинг 14.8

```
Sub AreaOfCircle_5()
'Вычисление площади круга
'Обработка нажатия кнопки Cancel в окне ввода и инструкция End
    Dim pi, d, s
    pi = 3.141593
    d = InputBox(Prompt:="d =", Title:="Площадь круга", Default:="<d>")
    If d = "" Then
        End
    End If
    s = pi * d ^ 2 / 4
    Call MsgBox(Prompt:="d= " & d & vbCrLf & "s= " & s, _
        Title:="Площадь круга")
End Sub
```

Здесь, чтобы выделить главное, мы исключили почти все сделанные ранее усовершенствования. Хотя в программе добавились всего три коротких строчки, она поднялась на качественно новый уровень: теперь последовательность выполнения инструкций в нашей программе зависит от данных.

Обратите внимание на отступ в начале инструкции **End**, расположенной внутри блока **If**. Мы всегда, так же как внутри блока **With**, будем добавлять

отступы в таких инструкциях. Этот прием существенно повышает наглядность программы. Никогда не пренебрегайте им, это сэкономит вам много времени и нервов, когда, например, вы получите сообщение, в котором говорится об ошибке компиляции, связанной с тем, что в блоке **If** отсутствует инструкция **End If** (рис. 14.7), или сообщение, в котором говорится об ошибке компиляции, связанной с тем, что встретилась инструкция **End If** без блока **If** (рис. 14.8).

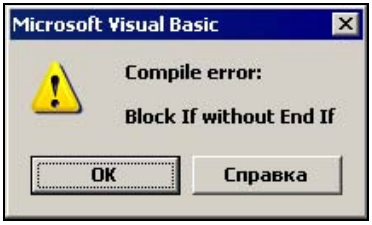


Рис. 14.7. Сообщение об ошибке компиляции при отсутствии в блоке `If` инструкции `End If`

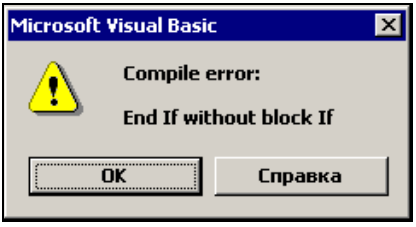


Рис. 14.8. Сообщение об ошибке компиляции при наличии лишней инструкции `End If`

Полезные сведения

Для увеличения отступа выделите группу команд и нажмите клавишу <Tab>. Для уменьшения отступа — сочетание клавиш <Shift>+<Tab>

Вместо трех вставленных нами инструкций можно было бы написать так называемую однострочную инструкцию условного перехода:

```
If d = "" Then End
```

Однако мы не рекомендуем использовать эту форму записи инструкции **If** из-за того, что она плохо заметна в тексте программы.

Потренируемся в использовании блока **If**. Вместо того чтобы останавливать программу инструкцией **End**, напишем инструкцию **If** таким образом, чтобы дальнейшие инструкции выполнялись только тогда, когда значение пере-

менной `d` не является пустой строкой. Таким образом в качестве условия запишем логическое выражение:

```
d <> ""
```

Заметим, что можно было бы использовать эквивалентное логическое выражение, содержащее логическую операцию отрицания:

```
Not (d = "")
```

В результате получится еще один вариант подпрограммы, приведенный в листинге 14.9.

Листинг 14.9

```
Sub AreaOfCircle_6()
'Вычисление площади круга
'Обработка нажатия кнопки Cancel в окне ввода без инструкции End
    Dim pi, d, s
    pi = 3.141593
    d = InputBox(Prompt:="d =", Title:="Площадь круга", Default:="<d>")
    If d <> "" Then
        s = pi * d ^ 2 / 4
        Call MsgBox(Prompt:="d= " & d & vbCrLf & "s= " & s, _
            Title:="Площадь круга")
    End If
End Sub
```

Преимущество этого варианта программы не в том, что он на одну строчку короче, а в его большей наглядности: одного взгляда на программу достаточно, чтобы увидеть, какие инструкции выполняются в зависимости от указанного условия.

Продолжим изучение условного перехода. Напишем программу вычисления корней уравнения $x^2 - 2ax + 1 = 0$ (мы понимаем, что вы не часто решаете подобные задачи, но сейчас нам нужен самый простой пример, а программами, связанными с обработкой текстов, мы займемся в следующих главах). Те, у кого сохранились в памяти школьные уроки, знают, что корни этого уравнения находятся по формуле:

$$x_{1,2} = a \pm \sqrt{a^2 - 1}.$$

Вычисление корней по этой формуле может быть записано в виде подпрограммы `Quadratic_1` (*quadratic* — квадратный), приведенной в листинге 14.10.

Листинг 14.10

```
Sub Quadratic_1()
' Решение квадратного уравнения x^2 - 2ax + 1 = 0
    Dim a, x1, x2
    a = InputBox(Prompt:="a:", Title:="x^2 - 2ax + 1 = 0")
    x1 = a - Sqr(a ^ 2 - 1)
    x2 = a + Sqr(a ^ 2 - 1)
    Call MsgBox (Prompt:="a =" & a & " x1=" & x1 & " x2=" & x2, _
        Title:="x^2 - 2ax + 1 = 0")
End Sub
```

Прочтите внимательно текст программы. Вам должно быть все понятно до мелочей. На всякий случай напоминаем, что для вычисления квадратного корня используется встроенная функция `Sqr`. Если что-то еще осталось неясно, вернитесь к предыдущим разделам. Чтобы программа была проще, мы в ней отказались от сделанных нами усовершенствований, относящихся к вводу числовых данных, но мы к ним еще вернемся.

Проверьте, что программа правильно вычисляет корни уравнения при $a = 1$, при $a = -1$, при $a = 1{,}25$ и при $a = -1{,}25$.

Однако те, кто в школе действительно учил математику, помнят, что решения нашего уравнения существуют не при всех значениях параметра a. Например, если задать $a = 0$, на экране появится окно редактора Visual Basic (если подпрограмма запускалась из окна Microsoft Word) и будет выведено сообщение, показанное на рис. 14.9.

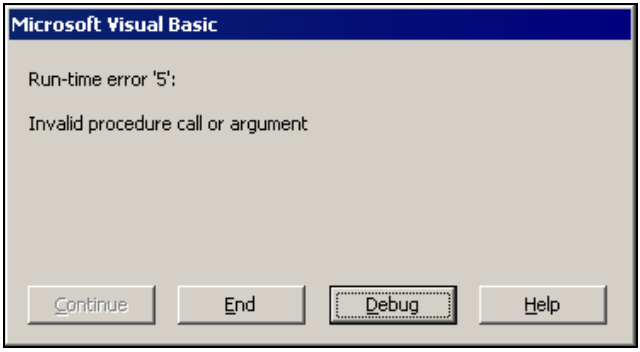


Рис. 14.9. Сообщение об ошибке при неудачном обращении к функции или подпрограмме

Это уже знакомое нам диалоговое окно (см. рис. 14.6) с сообщением об ошибке во время выполнения программы. Для удобства приведем переводы всех элементов этого окна, хотя по сравнению с окном, показанным на рис. 14.6, здесь изменились только номер ошибки и текст описания ошибки (дополнительно выделенный далее курсивом). Слово *процедура* обозначает функцию или подпрограмму. О вызове подпрограмм сказано в *разд. 14.4*.

- **Run-time error** — ошибка во время выполнения
- ***Invalid procedure call or argument*** — недопустимый вызов процедуры или недопустимый аргумент
- **Continue** — продолжить
- **End** — закончить
- **Debug** — отладка
- **Help** — справка

На этот раз воспользуемся возможностями отладки, которые предоставляет редактор Visual Basic.

14.2.3. Средства отладки: первое знакомство

Нажмите кнопку **Debug** (Отладка). Активируется редактор Visual Basic и в нем открывается окно с соответствующей подпрограммой (рис. 14.10). Стрелкой слева и желтым цветом выделяется инструкция, при выполнении которой обнаружена ошибка. Поскольку нам сообщили, что причиной ошибки может быть недопустимое значение аргумента, целесообразно проверить, какое значение имеет подкоренное выражение.

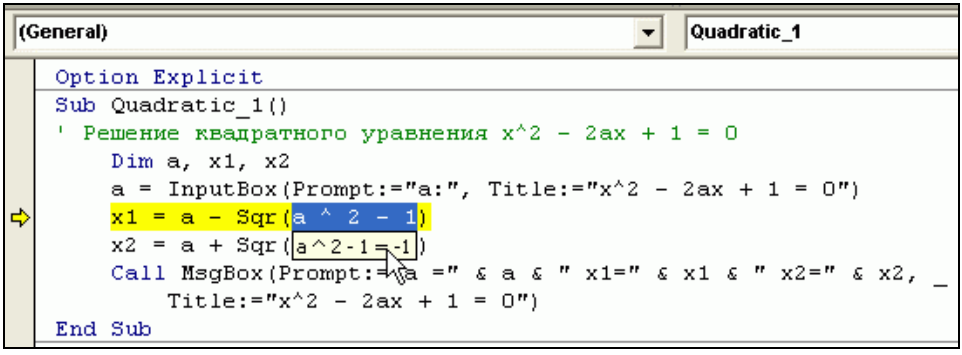


Рис. 14.10. Отладка в редакторе Visual Basic

Если подвести курсор, к переменной a, всплывет желтый прямоугольник с надписью "a = 0", а если выделить аргумент функции Sqr и подвести к выделенному фрагменту курсор, услужливый Visual Basic подсчитает для нас значение выделенного выражения (рис. 14.10).

Мы видим, что значение подкоренного выражения (в школе оно называлось *дискриминантом*) равно −1, а корень квадратный можно вычислять только из неотрицательных чисел. Поэтому и произошло прерывание вычислений с сообщением о "недопустимом вызове процедуры или недопустимом аргументе".

В школе нас учили, что если дискриминант отрицательный, то у квадратного уравнения корней нет, а если дискриминант равен нулю, то у уравнения только один корень. Значит, прежде чем вычислять корни, надо проверять значение дискриминанта. Хорошо, что мы уже умеем программировать условный переход. Один из возможных вариантов подпрограммы приведен в листинге 14.11.

Листинг 14.11

```
Sub Quadratic_2()
' Решение квадратного уравнения x^2 - 2ax + 1 = 0
' Инструкции If и End If
    Dim a, x1, x2
    Dim d 'дискриминант
    Dim s 'корень из дискриминанта
    Dim strPrompt As String 'строка вывода
    Dim strTitle As String 'заголовок окон ввода и вывода
    strPrompt = "a:"
    strTitle = "x^2 - 2ax + 1 = 0"
    a = InputBox(Prompt:=strPrompt, Title:=strTitle)
    strPrompt = "a =" & a & ", решение не существует"
    d = a ^ 2 - 1
    If d >= 0 Then
        s = Sqr(d)
        x1 = a - s
        x2 = a + s
        strPrompt = "a =" & a & " x1=" & x1 & " x2=" & x2
    End If
    Call MsgBox (Prompt:=strPrompt, Title:=strTitle)
End Sub
```

Здесь вводятся еще четыре переменные: `d`, `s`, `strTitle` и `strPrompt`, смысл которых поясняется в комментариях. Эти переменные позволяют исключить повторные вычисления. Сначала переменной `strPrompt` присваивается значение для случая, когда решения не существует. После этого проверяется условие существования решения, и, если решение есть, вычисляются корни уравнения и переменной `strPrompt` присваивается другое значение.

Посмотрите еще раз на подпрограмму `Quadratic_2`. Убедитесь, что вы полностью понимаете, что в ней написано и как эта подпрограмма работает.

Теперь взглянем на эту подпрограмму с другой стороны. Если рассматривать ее как прообраз ваших будущих сложных программ, то к ней следует предъявлять гораздо более жесткие требования.

Обратите внимание, что дискриминант в ней вычисляется один раз, а используется три раза. В нашем примере время вычисления и размер программы несущественны, однако в тех случаях, когда важно быстродействие, можно значительно сократить время выполнения благодаря использованию переменной, в которой будет храниться первоначально вычисленное значение, и которая будет использоваться всякий раз, когда это значение потребуется. Это положительный пример, достойный подражания.

Однако и в этой программе есть особенность, которая при более сложных вычислениях превратилась бы в недостаток. Представьте себе, что нам в зависимости от определенного условия необходимо выполнить один из двух вариантов достаточно продолжительных расчетов. Если использовать подход, реализованный в подпрограмме `Quadratic_2`, то в том случае, когда необходимо выполнять второй вариант расчета, все равно придется потратить время на вычисления первого варианта. Действительно, структура подпрограммы `Quadratic_2` такова, что сначала выполняется расчет для первого варианта, а затем проверяется условие и, если необходимо, рассчитывается второй вариант.

В нашем простейшем случае это несущественно, поскольку никаких расчетов при `d < 0` не выполняется, а только присваивается значение переменной `strPrompt`, но если расчеты сложные, такой подход неэффективен, поскольку каждый из вариантов может потребовать много времени. Тогда целесообразно использовать другую структуру.

14.2.4. Конструкция *If...Else*

Для тех случаев, когда в зависимости от определенного условия требуется выбрать один из двух вариантов расчета, инструкция **`If`** дополняется инструкцией **`Else`** (иначе).

Пользуясь этой инструкцией, программу `Quadratic_2` можно переписать в виде, представленном в листинге 14.12.

Листинг 14.12

```
Sub Quadratic_3()
' Решение квадратного уравнения x^2 - 2ax + 1 = 0
' Инструкции If, Else и End If
    Dim a, x1, x2
    Dim d 'дискриминант
    Dim s 'корень из дискриминанта
    Dim strPrompt As String 'строка вывода
    Dim strTitle As String 'заголовок окон ввода и вывода
    strPrompt = "a:"
    strTitle = "x^2 - 2ax + 1 = 0"
    a = InputBox(Prompt:=strPrompt, Title:=strTitle)
    d = a ^ 2 - 1
    If d < 0 Then
        strPrompt = "a =" & a & ", решение не существует"
    Else
        s = Sqr(d)
        x1 = a - s
        x2 = a + s
        strPrompt = "a =" & a & " x1=" & x1 & " x2=" & x2
    End If
    Call MsgBox (Prompt:=strPrompt, Title:=strTitle)
End Sub
```

В этой конструкции два блока исполняемых инструкций — они выделены отступом. Инструкции первого блока расположены между инструкцией **If** и инструкцией **Else**. Они выполняется, если условие, записанное в инструкции **If**, истинно. В противном случае выполняются инструкции другого блока, расположенные между инструкцией **Else** и инструкцией **End If**.

Такая конструкция не только обеспечивает большую эффективность программы, но и существенно нагляднее.

Мы все время говорим о наглядности записи программ. Это важно, потому что на чтение и понимание алгоритма уходит основное время программиста, т. е. теперь это ваше время. Если программа наглядная, то снижается вероятность ошибки. Это тоже экономия вашего времени. А еще подумайте о тех, кто будет читать ваши замечательные программы, чтобы использовать ваш опыт.

Теперь поставим задачу несколько иначе. Требуется вычислить число корней уравнения $x^2 - 2ax + 1 = 0$, не решая его. Мы знаем, что если дискриминант отрицательный, корней нет, если дискриминант нулевой, то корень один, а если дискриминант положительный, то корня два.

Один из вариантов подпрограммы расчета числа корней может иметь вид, представленный в листинге 14.13.

Листинг 14.13

```
Sub Quadratic_4()
' Число корней квадратного уравнения x^2 - 2ax + 1 = 0
' Вложенные инструкции If
    Dim a
    Dim d 'дискриминант
    Dim n ' число корней
    Dim strPrompt As String 'строка вывода
    Dim strTitle As String 'заголовок окон ввода и вывода
    strPrompt = "a:"
    strTitle = "x^2 - 2ax + 1 = 0"
    a = InputBox(Prompt:=strPrompt, Title:=strTitle)
    d = a ^ 2 - 1
    If d < 0 Then
        n = 0
    Else
        If d = 0 Then
            n = 1
        Else
            n = 2
        End If
    End If
    strPrompt = "a =" & a & ", число корней =" & n
    Call MsgBox (Prompt:=strPrompt, Title:=strTitle)
End Sub
```

В этом простом примере вам должно быть все понятно без комментариев. Он демонстрирует, что инструкции **If** могут быть вложенными, т. е. внутри любого из блоков, которые относятся к инструкции **If**, могут быть распо-

ложены другие инструкции **If** вместе с соответствующими блоками. Благодаря этому появляется возможность программировать сколь угодно сложные логические конструкции.

14.2.5. Конструкция *If...ElseIf...Else*

Эта задача позволяет нам показать еще одну инструкцию **ElseIf** (иначе, если). С помощью этой инструкции можно несколько упростить подпрограмму `Quadratic_4`, записав ее в виде, представленном в листинге 14.14.

Листинг 14.14

```
Sub Quadratic_5()
' Число корней квадратного уравнения x^2 - 2ax + 1 = 0
' Инструкции If, ElseIf, Else и End If
    Dim a
    Dim d 'дискриминант
    Dim n ' число корней
    Dim strPrompt As String 'строка вывода
    Dim strTitle As String 'заголовок окон ввода и вывода
    strPrompt = "a:"
    strTitle = "x^2 - 2ax + 1 = 0"
    a = InputBox(Prompt:=strPrompt, Title:=strTitle)
    d = a ^ 2 - 1
    If d < 0 Then
        n = 0
    ElseIf d = 0 Then
        n = 1
    Else
        n = 2
    End If
    strPrompt = "a =" & a & ", число корней =" & n
    Call MsgBox (Prompt:=strPrompt, Title:=strTitle)
End Sub
```

Как видно из этого примера, инструкция **ElseIf** позволяет упростить текст подпрограммы `Quadratic_4`. На наш взгляд, ее смысл полностью понятен из перевода составляющих ее слов и приведенных примеров. Необходимо сделать лишь два замечания. Во-первых, эта инструкция должна располагаться

после инструкции **If** и до инструкции **Else** (если инструкции **Else** нет, то до инструкции **End If**). Во-вторых, вместе с одной инструкцией **If** можно использовать несколько инструкций **ElseIf** (за каждой такой инструкцией следует блок инструкций, выполняемых в зависимости от условия, указанного в соответствующей инструкции **ElseIf**).

Таким образом, мы получаем в свое распоряжение конструкцию, удобную для программирования алгоритма, в котором нужно выполнить один из нескольких вариантов расчетов, а после выполнения любого из этих вариантов продолжить выполнение программы с одного и того же места.

Заканчивая описания различных форм инструкции условного перехода, скажем, что вместо термина "условный переход" часто используют термин *ветвление*, имея в виду различные ветви, по которым может проходить выполнение программы. Кроме того, упомянем инструкцию `Select Case` и функции `IIf`, `Switch` и `Choose`. Эти средства языка VBA также можно использовать для программирования действий, зависящих от значений логических выражений, но, несмотря на то, что в определенных случаях их применение удобно, мы решили о них здесь не рассказывать. Во-первых, без них всегда можно обойтись, а во-вторых, ничего принципиально нового для вас в их использовании нет — вы уже подготовлены для их самостоятельного изучения по другим источникам.

14.3. Циклы

В данном разделе мы познакомимся с инструкциями, которые позволяют многократно повторять уже выполненные инструкции, при необходимости изменяя используемые ими данные. Такие конструкции называются *циклами*. Разумеется, должны быть предусмотрены средства, позволяющие в нужный момент *выйти из цикла* и перейти к выполнению инструкций, следующих за циклом. Условные переходы и циклы в принципе обеспечивают возможность программной реализации любой логической задачи.

14.3.1. Цикл *For*

Возможно, вы обращали внимание, что когда с помощью мыши устанавливаются отступы или позиции табуляции на горизонтальной линейке, маркеры перемещаются дискретно с шагом 1/8 дюйма. Это происходит даже в том случае, когда в качестве единицы измерения выбраны сантиметры, т. е. когда в диалоговых окнах мы задаем отступы или позиции табуляции в сантиметрах. Если же мы хотим использовать диалоговые окна и мышь параллельно, то удобно задавать значения кратными 1/8 дюйма. Для реализации этой идеи нам понадобилось выразить в сантиметрах набор значений, крат-

ных 1/8 дюйма. Мы решили написать макрос, который напечатал бы нам список таких значений. Для этого макроса мы выбрали название `Inch2cm` (Inch — дюйм; *cm* — см, а цифрой 2 иногда заменяют предлог to — соответствующий в данном случае русскому предлогу "в", — потому что по-английски этот предлог и цифра 2 произносятся одинаково).

В этой главе мы решили описывать только средства программирования, не зависящие от конкретного приложения, однако в виде исключения для вывода результатов воспользуемся приложением Microsoft Word. Нам потребуется вывести число и выполнить перевод строки.

В *разд. 13.2.2* мы уже использовали метод `TypeText` (*Type text* — печатать текст) объекта `Selection` с аргументом `Text` и знаем, что его можно использовать для вывода строк. Мы уже научились использовать константу `vbCrLf` для перевода строки. Поэтому для решения поставленной задачи нам осталось только научиться автоматически повторять выполнение инструкции

```
Selection.TypeText Text:=s & vbCr
```

в которой значение переменной `s` изменялось бы в диапазоне от 0 до 21, увеличиваясь с каждым разом на 2,54/8. Здесь мы учли, что ширина стандартной страницы 21 см, а в одном дюйме 2,54 см.

Для подобных задач в VBA предусмотрены две инструкции, с помощью которых организуется так называемый *цикл* ***For***. Подпрограмма, решающая нашу задачу с помощью этих инструкций, представлена в листинге 14.15.

Листинг 14.15

```
Sub Inch2cm
' Шкала в сантиметрах с шагом 1/8 дюйма
Dim s
    Documents.Add
    For s = 0 To 21 Step 2.54 / 8
        Selection.TypeText Text:=s & vbCr
    Next s
End Sub
```

В этой подпрограмме после описания переменной `s` следует инструкция

```
Documents.Add
```

С этой инструкцией мы познакомились в *разд. 11.2*. Она открывает новый документ Microsoft Word.

После нее записана новая для нас инструкция **For**. Начнем с перевода ключевых слов.

- **For** — для
- **To** — до
- **Step** — шаг
- **Next** — следующий

Если попытаться перевести с языка VBA на русский язык фразу,

```
For s = 0 To 21 Step 2.54 / 8
```

получится примерно следующее:

"Выполнить *для* переменной `s`, которая изменяется от значения `0` *до* значения `21` с *шагом* `2.54 / 8`, следующую группу инструкций" (напоминаем, что в тексте программ в качестве разделителя целой и дробной части чисел используется точка).

Инструкция

```
Next s
```

ограничивает группу выполняемых в цикле инструкций. После ключевого слова **Next** указывается *переменная цикла*, к которому относится данная инструкция.

> ***Совет***
>
> Переменную цикла после ключевого слова `Next` указывать не обязательно, но мы настоятельно рекомендуем не экономить одно слово. Редактор Visual Basic будет старательно проверять соответствие инструкций, составляющих цикл и, если обнаружит ошибку, сразу о ней сообщит. Разумеется, существенная польза от такой проверки будет только в тех случаях, когда в программе используется несколько циклов (часто внутри одного цикла используются другие циклы), но лучше сразу привыкать к хорошему стилю программирования.

Переменная цикла — это обычная переменная, ее можно использовать при вычислениях и даже присваивать ей значения.

> ***Совет***
>
> Изменять переменную цикла внутри цикла правилами языка не запрещается, но мы настоятельно рекомендуем этого не делать. Бывает, что одна и та же переменная случайно используется и как переменная цикла, и как переменная для хранения другой информации. Ошибки, которые при этом возникают, очень сложно отлавливать. Старайтесь выбирать для переменных цикла понятные имена, и никогда не используйте эти имена для других целей.

В результате выполнения подпрограммы `Inch2cm` откроется новый документ и будет напечатан столбик нужных нам чисел. Конечно, можно было

бы оформить результат вычислений более наглядно, но сейчас у нас другие цели.

Продолжим изучение циклов. Рассмотрим следующую задачу. Требуется определить, можно ли введенную строку использовать для имени файла. Таким образом, надо проверить, что введенная строка:

- содержит не только пробелы;
- имеет длину, не превышающую 255 знаков;
- не содержит следующих символов: `\ / : * ? " < > |`

Составим подпрограмму `CheckFileName` (проверить имя файла). В ней мы введем строку, подсчитаем число знаков в этой строке и в цикле проверим, что каждый из символов этой строки не является запрещенным (листинг 14.16).

Листинг 14.16

```
Sub CheckFileName()
'Проверка допустимости имени файла
Dim strFileName As String 'проверяемая строка
Dim str1 As String 'очередной символ строки
Dim i 'переменная цикла
Dim boo As Boolean 'признак допустимости имени файла
strFileName = _
    InputBox("Введите имя файла", "Проверка допустимости имени файла")
If Trim(strFileName) = "" Then
    Call MsgBox("Введена пустая или содержащая одни пробелы строка")
    End
End If
boo = True
For i = 1 To Len(strFileName)
    str1 = Mid(String:=strFileName, Start:=i, Length:=1)
    If str1 = "\" Or str1 = "/" Or str1 = ":" _
            Or str1 = "*" Or str1 = "?" Or str1 = """" _
            Or str1 = "<" Or str1 = ">" Or str1 = "|" Then
        boo = False
        Exit For
    End If
Next i
If boo Then
```

```
    Call MsgBox ("""" & strFileName & """ - допустимое имя файла")
Else
    Call MsgBox ("В строке """ & strFileName & """ на позиции " & i _
        & " недопустимый символ " & str1)
End If
End Sub
```

Эта подпрограмма заслуживает подробного разбора.

После ввода строки проверяется, что она не пустая и в ней содержатся не только пробелы. Для этой цели используется встроенная функция `Trim` (подрезать), возвращающая копию заданной строки, из которой удалены начальные и конечные пробелы. Эта функция часто используется для автоматического исправления ошибок пользователя при вводе исходных данных.

Для проверки наличия недопустимых символов используется логическая переменная `boo`. В конце программы проверяется ее значение. Если значением логической переменной является константа **`True`** (истина), то имя файла допустимо, если **`False`** (ложь), то нет. Это значение формируется следующим образом: сначала выполняется инструкция

```
boo = True
```

Затем в цикле проверяется каждый символ введенной строки. Если обнаруживается, что он недопустим, выполняется инструкция

```
boo = False
```

и цикл проверки прерывается. Если недопустимых символов нет, то цикл дойдет до конца и значение переменной `boo` останется равным **`True`**.

Прерывание цикла выполняется с помощью новой для нас инструкции

```
Exit For
```

передающей управление инструкции, которая следует за инструкцией

```
Next i
```

После этого производится вывод результатов.

Теперь рассмотрим подробнее новые элементы.

Инструкция **`Exit For`** состоит из нового ключевого слова **`Exit`** (выход) и уже известного вам ключевого слова **`For`**, которое здесь обозначает название цикла. Эта инструкция досрочного прерывания цикла является необязательной составляющей цикла **`For`**.

В этой программе использована также новая для нас особенность функции `InputBox`: длина строки, которую можно ввести с помощью этой функции, не превышает 254 знака (сведения об этом в документации мы не обнару-

жили, но 255-й знак напечатать в поле ввода не получается). Поэтому ограничение в 255 знаков для длины имени файла можно не проверять. Это оказалось удобно, но с другой стороны, формально для имени файла можно использовать строку длиной 255 знаков, а такую строку с помощью нашей подпрограммы проверить нельзя. Конечно, можно ввести такую строку в два приема, но мы решили смириться с таким недостатком нашей программы и не усложнять ее.

В инструкции

```
For i = 1 To Len(strFileName)
```

два новшества: отсутствие ключевого слова **Step** и встроенная функция `Len`.

Действительно, в инструкции **For** ключевое слово **Step**, указывающее шаг цикла, является необязательным. Если оно не указано, то переменная цикла на каждом шаге увеличивается на 1. Другими словами, по умолчанию шаг цикла равен 1.

Встроенная функция `Len` (от *length* — длина) возвращает количество знаков в строке, которая задается в качестве аргумента этой функции.

Таким образом вычисляется количество знаков в строке, содержащейся в переменной `strFileName`. Переменная цикла `i` изменяется с шагом 1 от начального значения 1 до конечного значения, равного количеству знаков в строке. Для каждого значения переменной цикла выполняются инструкции, расположенные внутри цикла, т. е. расположенные между данной инструкцией **For** и инструкцией **Next** `i`.

Внимательный читатель, наверное, заметил, что последнее утверждение верно только в том случае, если в ходе выполнения цикла не выполнится инструкция **Exit For**, прерывающая выполнение цикла.

Первой внутри цикла выполняется следующая инструкция:

```
str1 = Mid(String:=strFileName, Start:=i, Length:=1)
```

В этой инструкции строковой переменной `str1` присваивается символ, находящийся на *i*-м месте во введенной строке `strFileName`.

Функция `Mid` (средний) позволяет получить новую строку из фрагмента имеющейся строки. У функции три именованных аргумента.

- Первый аргумент — `String` (строка) — исходная строка. Этот аргумент обязателен.
- Второй аргумент `Start` (начало) — число, задающее номер символа в строке, с которого начинается нужный фрагмент строки. Этот аргумент тоже обязателен. Если значение этого аргумента превышает длину строки, возвращается пустая строка.

- Третий аргумент `Length` (длина) — количество знаков в извлекаемом фрагменте. Этот аргумент необязательный. Если его опустить, извлекается фрагмент, содержащий все символы до конца строки. Такой же результат получится, если задать этот аргумент большим, чем число символов в таком фрагменте.

В нашем случае результатом является строка, содержащая один символ, находящийся на *i*-м месте: таким образом по очереди извлекаются все символы, содержащиеся во введенной сроке.

Далее внутри цикла **`For`** расположен блок **`If`**. Важное условие: блок **`If`**, начатый внутри цикла, должен целиком помещаться в этом цикле. Также и цикл, начатый в блоке **`If`**, должен целиком помещаться в этом блоке, т. е. внутри цикла, начатого в блоке **`If`**, не должны содержаться инструкции **`ElseIf`**, **`Else`** или **`EndIf`**, относящиеся к этому **`If`**.

Теперь посмотрим на записанное в несколько строчек условие блока **`If`**:

```
If str1 = "\" Or str1 = "/" Or str1 = ":" _
        Or str1 = "*" Or str1 = "?" Or str1 = """" _
        Or str1 = "<" Or str1 = ">" Or str1 = "|" Then
```

Оно представляет собой девять равенств, объединенных в одно логическое выражение знаками логической операции **`Or`** (или). Смысл его понятен: условие истинно, если истинно хотя бы одно из равенств, т. е. если строка `str1` совпадает с одним из запрещенных символов. Единственная новая для нас особенность связана с обозначением *кавычек внутри строки*. Поскольку сами строки заключаются в кавычки, кавычки внутри строки удваиваются. Поэтому строка, состоящая из одной кавычки, записывается четырьмя кавычками (включая кавычки, ограничивающие строку).

После инструкций цикла следуют инструкции, в которых проверяется, дошел ли цикл до конца или был прерван из-за того, что строке встретился недопустимый символ. В зависимости от результата проверки печатается соответствующее сообщение.

Цикл **`For`** удобен, когда известно, сколько раз надо повторять содержащиеся в цикле инструкции. Он также удобен для задач, в которых используется автоматически формируемая переменная цикла, изменяющаяся с заданным шагом. Можно задать шаг отрицательный, и тогда переменная цикла будет убывать. Если начальное значение переменной цикла превышает значение, заданное в качестве конечного, а шаг задан положительным, цикл не будет выполняться ни одного раза, что тоже бывает удобно, поскольку не требуется программировать дополнительных проверок. Однако есть задачи, для которых предпочтительнее использовать так называемый цикл **`Do`**.

14.3.2. Цикл *Do*

Вернемся к вычислению площади круга. Предположим, требуется составить программу для случая, когда вычисления необходимо повторять, каждый раз задавая новое значение диаметра до тех пор, пока пользователь не нажмет кнопку **Cancel**.

Для решения этой задачи можно, разумеется, использовать цикл **For**, если задать такие условия, чтобы цикл выполнялся очень большое число раз, а прерывание цикла осуществлять с помощью инструкции **Exit For**. Однако для данного случая удобнее использовать *цикл* ***Do***. Соответствующую подпрограмму можно записать в виде, представленном в листинге 14.17.

Листинг 14.17

```
Sub AreaOfCircle_7()
'Многократное вычисление площади круга
'Цикл Do и инструкция Exit Do
    Dim pi, d, s
    Dim strPrompt As String 'сообщение диалогового окна
    pi = 3.141593
    strPrompt = "Введите диаметр круга" & vbCrLf _
        & "и нажмите кнопку OK." & vbCrLf & vbCrLf _
        & "Чтобы выйти," & vbCrLf & "нажмите кнопку Cancel."
    Do
        d = InputBox(Prompt:=strPrompt, Title:="Площадь круга", _
            Default:="<d>")
        If d = "" Then
            Exit Do
        End If
        s = pi * d ^ 2 / 4
        strPrompt = "d= " & d & vbCrLf & "s= " & s _
            & vbCrLf & vbCrLf & "Введите новый диаметр круга" _
            & vbCrLf & "и нажмите кнопку OK." _
            & vbCrLf & vbCrLf & "Чтобы выйти," _
            & vbCrLf & "нажмите кнопку Cancel."
    Loop
End Sub
```

В этой подпрограмме три новые инструкции: **Do** (выполнять), **Loop** (цикл) и **Exit Do** (выход из цикла Do).

Инструкции **Do** и **Loop** — это инструкции начала и конца цикла. Инструкции, заключенные между ними, будут повторяться. Инструкция **Exit Do** служит для прерывания цикла **Do**. Она действует так же, как инструкция **Exit For** для цикла **For**.

Подпрограмма работает следующим образом. Сначала подготавливается текст сообщения для диалогового окна ввода данных. Этот текст запоминается в переменной `strPrompt`. Все остальные действия повторяются в цикле до тех пор, пока пользователь в диалоговом окне ввода не нажмет кнопку **Cancel**. Эти действия запрограммированы с помощью инструкций, находящихся внутри цикла (т. е. между новыми для нас инструкциями **Do** и **Loop**). В первой из них вызывается диалоговое окно ввода данных, для которого мы подготовили текст сообщения. Эта инструкция нам знакома. Диалоговое окно, открывающееся при первом выполнении этой инструкции, показано на рис. 14.11.

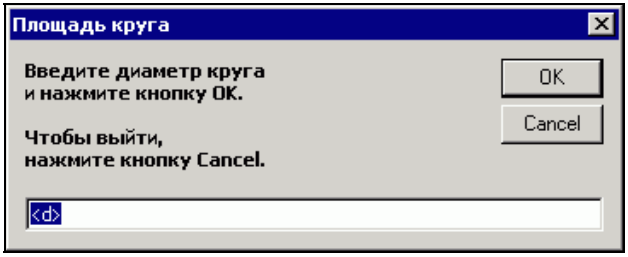


Рис. 14.11. Первое окно ввода данных в подпрограмме `AreaOfCircle_7`

Далее следуют также знакомые нам инструкции условного перехода, в которых проверяется, не нажал ли пользователь кнопку **Cancel**, и, если эта кнопка была нажата, то выполняется новая для нас инструкция выхода из цикла **Do**. Далее осуществляется вычисление площади круга и готовится строка со следующим сообщением для диалогового окна ввода данных. Это сообщение выводится, начиная со второго прохода цикла. В этом сообщении не только даются указания по вводу новых данных, но также приводятся результаты последнего вычисления площади круга (рис. 14.12).

Собственно цикл организуется с помощью только двух инструкций **Do** и **Loop**, и редактор Visual Basic не проверяет наличие инструкций, которые прерывают выполнение цикла. Если же вы забудете запрограммировать прерывание цикла, то инструкции, расположенные между ними, будут выполняться, пока не закончится терпение пользователя или по какой-либо другой заранее непредусмотренной причине. Обычно в таких случаях на экране

"висят часы", т. е. курсор имеет вид песочных часов и ничего не происходит. Как говорят программисты, программа "зациклилась".

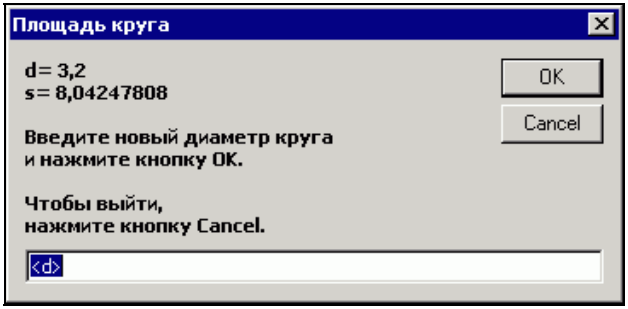


Рис. 14.12. Окно вывода результатов расчета и ввода новых данных в подпрограмме `AreaOfCircle_7`

Полезные сведения

Чтобы прервать выполнение макроса, следует нажать сочетание клавиш <Ctrl>+ +<Break>.

После нажатия сочетания клавиш <Ctrl>+<Break> на экране появится диалоговое окно, изображенное на рис. 14.13.

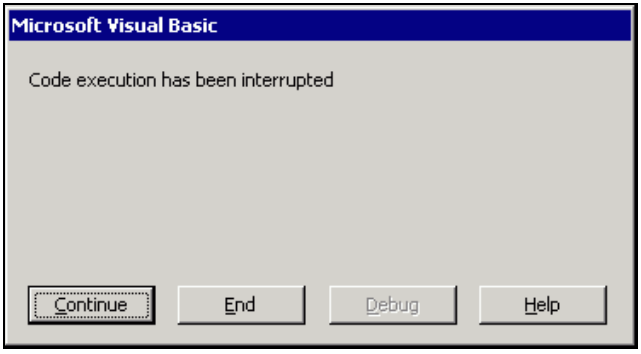


Рис. 14.13. Сообщение при прерывании выполнения программы пользователем

Надписи в этом диалоговом окне переводятся следующим образом:

- **Code execution has been interrupted** — выполнение программы прервано
- **Continue** — продолжить

- **End** — закончить
- **Debug** — отладка
- **Help** — справка

Если нажать кнопку **End**, то выполнение подпрограммы прекратится. Если доступна кнопка **Debug**, а она доступна, когда подпрограмма записана в шаблоне Normal.dot или в присоединенном шаблоне (подробнее о шаблонах *см. в гл. 7*), то можно, воспользовавшись этой кнопкой, перейти в редактор Visual Basic и продолжать работать в режиме отладки. Кнопку **Continue** нажимать не советуем. Наши компьютеры в этом случае зависали, и приходилось снимать Microsoft Word с помощью диспетчера задач (для этого применяется сочетание клавиш <Ctrl>+<Alt>+<Del>).

Таким образом, при использовании цикла `Do` лучше не забывать про условие выхода из цикла. По возможности следует использовать другие формы записи цикла `Do`, в которых условие выхода из цикла записывается в инструкции `Do` или в инструкции `Loop`. Чтобы продемонстрировать одну из таких форм, подпрограмму `AreaOfCircle_7` запишем в виде, представленном в листинге 14.18.

Листинг 14.18

```
Sub AreaOfCircle_8()
'Многократное вычисление площади круга
'Инструкция Do While
    Dim pi, d, s
    Dim strPrompt As String 'сообщение диалогового окна
    pi = 3.141593
    strPrompt = "Введите диаметр круга" & vbCrLf _
        & "и нажмите кнопку OK." & vbCrLf & vbCrLf _
        & "Чтобы выйти," & vbCrLf & "нажмите кнопку Cancel."
        d = InputBox(Prompt:=strPrompt, Title:="Площадь круга", _
            Default:="<d>")
    Do While d <> ""
        s = pi * d ^ 2 / 4
        strPrompt = "d= " & d & vbCrLf & "s= " & s _
        & vbCrLf & vbCrLf & "Введите новый диаметр круга" _
        & vbCrLf & "и нажмите кнопку OK." _
        & vbCrLf & vbCrLf & "Чтобы выйти," & vbCrLf _
        & "нажмите кнопку Cancel."
```

```
        d = InputBox(Prompt:=strPrompt, Title:="Площадь круга", _
            Default:="<d>")
    Loop
End Sub
```

В этой подпрограмме инструкция-заголовок цикла **Do** записана следующим образом:

```
Do While d <> ""
```

В этой инструкции новое ключевое слово **While** (пока истинно). Эту инструкцию можно перевести на русский язык примерно так: *Выполнять, пока истинно условие* `d <> ""`. После ключевого слова **While** записано условное выражение, истинность которого проверяется перед каждым проходом цикла, в том числе и перед первым проходом. Если оно истинно, то инструкции внутри цикла выполняются. Если же оно ложно, то эти инструкции пропускаются и управление передается инструкции, записанной после инструкции **Loop**.

Фактически инструкция **Do While** `d <> ""` заменяет следующие четыре инструкции:

```
Do
    If Not d <> "" Then
        Exit Do
    End If
```

Подпрограмма `AreaOfCircle_8` работает следующим образом.

Перед началом цикла выполняется функция `InputBox`, которая, как вы знаете, возвращает строку, содержащую введенное значение диаметра, или пустую строку, если пользователь нажал кнопку **Cancel**. Эта строка присваивается переменной `d`. В инструкции-заголовке цикла проверяется значение этой переменной. Если это пустая строка, то цикл пропускается и подпрограмма заканчивается. Если же строка непустая, то выполняются вычисления и в последней инструкции цикла с помощью функции `InputBox` выводятся результаты вычисления и вводится новое значение диаметра. Если пользователь нажмет кнопку **Cancel**, то переменной `d` присвоится пустая строка. После этого опять будет производиться проверка в заголовке цикла и т. д.

Основное преимущество инструкции **Do While** состоит в том, что в ней предусмотрено условие продолжения цикла и тем самым исключается возможность забыть его запрограммировать.

В VBA предусмотрены и другие способы организации цикла **Do**. Они обеспечивают возможность записать в инструкции **Do** условие окончания (а не

продолжения) цикла, а также возможность проверки условий продолжения или окончания цикла в инструкции **Loop**. Они применяются реже, но мы решили рассказать о них, чтобы вы лучше разобрались в логике программирования циклов.

В подпрограмме `AreaOfCircle_8` (листинг 14.18) вместо инструкции

```
Do While d <> ""
```

можно написать полностью эквивалентную ей инструкцию, которая представляет собой еще одну форму цикла **Do**:

```
Do Until d = ""
```

Она заменяет следующие четыре инструкции:

```
Do
    If d = "" Then
        Exit Do
    End If
```

Отличие от предыдущего варианта состоит в том, что знак "не равно" сменился знаком "равно" и исчезла операция **Not** в начале условного выражения.

Здесь новое ключевое слово **Until** (пока ложно). Эту инструкцию можно перевести на русский язык примерно так: *Выполнять, пока ложно условие* `d = ""`. Это то же самое, что и *Выйти из цикла, если* `d = ""`.

Безразлично, в какой из этих двух форм записывать инструкцию **Do**. Эквивалентный переход от одной формы к другой можно выполнить, добавив или удалив ключевое слово **Not** перед логическим выражением в условии или, что то же самое, заменив логическое выражение на противоположное (два логических выражения противоположны, когда одно из них ложно, а другое истинно).

Одним словом, после ключевого слова **While** записывается условие продолжения цикла, а после ключевого слова **Until** — условие окончания цикла. Выбор между **While** и **Until** зависит от того, какое из этих двух условий удобнее применить.

Ключевые слова **While** и **Until** можно использовать и в инструкции **Loop**. Так образуются еще две формы цикла **Do**. Продемонстрируем их на примере задачи вычисления площади круга с предварительной проверкой исходных данных. При неправильном вводе данных будет повторяться вывод диалогового окна ввода данных. Чтобы отказаться от ввода данных, необходимо будет нажать кнопку **Cancel**. Посмотрите на подпрограмму (листинг 14.19), решающую эту задачу.

Листинг 14.19

```
Sub AreaOfCircle_9()
'Вычисление площади круга с проверкой исходных данных
'Цикл Do и инструкция Loop Until
    Dim pi, d, s
    Dim strPrompt As String 'сообщение диалогового окна
    Dim strD As String 'введенное значение диаметра
    pi = 3.141593
    strPrompt = "Введите ненулевой диаметр круга" & vbCrLf _
        & "и нажмите кнопку OK." & vbCrLf & vbCrLf _
        & "Чтобы выйти," & vbCrLf & "нажмите кнопку Cancel."
    Do
        strD = InputBox(Prompt:=strPrompt, Title:="Площадь круга", _
            Default:="<d>")
        strD = Replace(Expression:=strD, Find:=",", Replace:=".")
        d = Val(strD)
    Loop Until strD = "" Or d <> 0
    If d <> 0 Then
        s = pi * d ^ 2 / 4
        Call MsgBox(Prompt:="d= " & d & vbCrLf & "s= " & s, _
            Title:="Площадь круга")
    End If
End Sub
```

В подпрограмме `AreaOfCircle_9` новая для вас разновидность инструкции **Loop** (цикл):

```
Loop Until strD = "" Or d <> 0
```

Смысл ее — *прекратить* выполнение цикла, если *выполняется* условие (другими словами, истинно логическое выражение), указанное после ключевого слова **Until** (пока ложно). Можно сказать иначе: *продолжать* выполнение цикла, если *не выполняется* условие, указанное после ключевого слова **Until**. Эта инструкция заменяет следующие четыре инструкции:

```
    If strD = "" Or d <> 0 Then
        Exit Do
    End If
Loop
```

Принципиальный момент: отличие инструкции **Loop Until** от инструкции **Do Until** состоит в том, что в инструкции **Loop Until** проверка осуществляется не до, а после каждого прохода цикла. Это гарантирует, что будет выполнен по меньшей мере один проход цикла.

Вы наверняка уже догадываетесь, что эта инструкция может быть заменена инструкцией

```
Loop While Not (strD = "" Or d <> 0)
```

Все, что мы говорили ранее о ключевых словах **While** и **Until** применительно к инструкции **Do**, можно повторить, но делать этого мы не будем.

Обратите внимание на скобки. Без них порядок выполнения операций был бы другим. Мы не будем останавливаться на приоритете различных операций (об этом можно посмотреть в разделе "Operator Precedence" справки VBA или в любом руководстве по VBA). Во всех сомнительных случаях используйте скобки.

Для любителей логических рассуждений заметим, что логическое выражение `Not (strD = "" Or d <> 0)` эквивалентно следующему: `strD <> "" And d = 0`. Таким образом, при желании можно обойтись без операции отрицания.

Теперь мы знаем достаточно для того, чтобы разобраться, как работает подпрограмма `AreaOfCircle_9`. Перед началом цикла в переменную `strPrompt` заносится текст, который будет выводиться в качестве сообщения в диалоговом окне ввода диаметра. Инструкция, выводящая на экран это окно, будет повторяться до тех пор, пока пользователь не введет допустимую строку или не нажмет кнопку **Cancel**. Для определения допустимости строки в ней сначала все запятые заменяются точками с помощью функции `Replace`, а потом к ней применяется функция `Val` (о функциях `Replace` и `Val` рассказано в *разд. 14.1.8*). Если функция `Val` может проинтерпретировать эту строку как ненулевое число (`d <> 0`), считается, что введена допустимая строка и можно закончить цикл и переходить к вычислениям. Выход из цикла произойдет и в том случае, когда пользователь нажмет кнопку **Cancel** (или введет пустую строку). При этом функция `InputBox` возвратит пустую строку (`strD = ""`), что также является условием окончания цикла.

Далее в подпрограмме проверяется, по какой причине произошел выход из цикла, и, если была введена допустимая строка, то вычисляется площадь круга и выводится на экран диалоговое окно с исходным диаметром и результатом вычислений.

Алгоритм, использованный в подпрограмме `AreaOfCircle_9`, обладает определенными преимуществами.

Пользователь может отказаться от выполнения расчетов, нажав на кнопку **Cancel**.

В программе предусмотрено автоматическое исправление некоторых ошибок ввода: в качестве разделителя целой и дробной части можно вводить и точки, и запятые; пробелы, знаки табуляции и любые другие символы, введенные после числа, игнорируются.

Если введенное число равно нулю или его начальная часть не является правильной записью числа, то выполнение программы не останавливается, а пользователю предлагается ввести число заново. Эта возможность будет особенно ценной для тех ваших программ, в которых ввод данных будет происходить после того, как уже потрачено существенное время на предварительные расчеты.

Прежде чем закончить раздел, посвященный циклам, и перейти к следующему разделу, упомянем еще об одной разновидности цикла, реализуемой с помощью инструкции `For Each`. Она позволяет выполнить последовательность инструкций для каждого элемента массива или семейства. О массивах мы рассказывать не будем, а поскольку семейства относятся к объектам, об этой разновидности цикла рассказано в *гл. 15*, где описана работа с семействами.

14.4. Процедуры в процедурах

Самостоятельные программные единицы, такие как подпрограммы и функции, называются *процедурами*. Существуют также процедуры-свойства (*Property*), но рассказ о них выходит за рамки нашей книги. В предыдущих разделах мы разбирали и составляли подпрограммы, которые можно использовать как макросы Microsoft Word. Также мы познакомились с несколькими встроенными функциями, которые можно вызывать из подпрограмм. В этом разделе вы научитесь составлять процедуры (функции и подпрограммы) и вызывать их из других процедур. При этом вы не только узнаете, как передавать управление из одной процедуры в другую, но и как обеспечить обмен информацией между ними. Основная ценность процедур в том, что ими можно пользоваться, не зная, как они устроены. Для этого достаточно знать, что они делают и как к ним обращаться. После того как процедуры запрограммированы и проверены, их можно передавать другим разработчикам и многократно использовать как готовые блоки при построении различных программ. Этот принцип модульности реализован во всех современных языках программирования; пользуясь им, коллективы программистов могут работать параллельно, выполняя крупные проекты, он сокращает время и повышает качество разработок.

14.4.1. Создание собственных функций

Предположим, что способ проверки ввода ненулевых чисел, использованный в подпрограмме `AreaOfCircle_9` (см. листинг 14.19), вам так понравился, что вы захотели использовать его в своих расчетных программах вместо встроенной функции `InputBox`. Это означает, что во все такие подпрограммы необходимо будет скопировать соответствующие инструкции, поменять в них имена переменных и изменить заголовок диалогового окна. Однако это еще не все. Наиболее ответственная часть работы — проверить, что после изменений все будет работать должным образом.

Это типичная задача. Она возникает всякий раз, когда фрагмент одной программы мы хотим использовать в другой программе. Для решения таких задач в Visual Basic предусмотрены специальные средства, изучением которых мы займемся в этом разделе. Один из способов решения нашей задачи — оформить фрагмент подпрограммы в виде собственной, иначе говоря, *пользовательской функции*, к которой можно обращаться вместо встроенной функции `InputBox`.

Важно, что для обращения к функции не надо ничего знать о том, как она реализована. Обращение к ней производится, как к "черному ящику". Достаточно знать, какие данные и как следует передавать функции, какие данные и как следует получать от функции и что происходит в результате ее выполнения.

Представьте себе, что такая функция уже существует. Давайте подумаем, что необходимо знать, чтобы ее использовать?

Нужно знать имя и тип функции, имена и типы ее аргументов, какая информация передается в функцию, какая информация возвращается функцией и какие действия выполняет функция.

Наша функция предназначена для ввода ненулевых чисел. Поэтому естественно назвать ее `InputNonZero` (*Input* — ввод, *Non* — не; *Zero* — ноль). Тип функции — это тип значения, которое она возвращает. Поскольку она должна возвращать число, тип функции будет **`Variant`**. Поскольку функция `InputNonZero` должна заменить функцию `InputBox`, то у нее должен быть такой же набор аргументов. Мы только немного изменим имена аргументов, добавив в начало префиксы типа. Таким образом, набор ее аргументов будет состоять из трех строковых переменных:

- `strPrompt` — строковая переменная, содержащая сообщение диалогового окна;
- `strTitle` — строковый аргумент, содержащий заголовок диалогового окна;
- `strDefault` — строковый аргумент, содержащий начальное значение поля ввода диалогового окна.

Функция `InputNonZero` должна возвращать численное значение, которое будет или ненулевым числом, введенным в поле ввода, или будет равно нулю, если пользователь введет пустую строку или же нажмет кнопку **Cancel**.

Мы предполагаем, что эта функция заменит следующий фрагмент подпрограммы `AreaOfCircle_9`:

```
Do
    strD = InputBox(Prompt:=strPrompt, Title:="Площадь круга", _
        Default:="<d>")
    strD = Replace(Expression:=strD, Find:=",", Replace:=".")
    d = Val(strD)
Loop Until strD = "" Or d <> 0
```

Точнее, этот фрагмент заменится следующей инструкцией:

```
d = InputNonZero(strPrompt:=strPrompt, strTitle:="Площадь круга", _
    strDefault:="<d>")
```

Тогда подпрограмму `AreaOfCircle_9` можно будет заменить подпрограммой `AreaOfCircle_10`, приведенной в листинге 14.20.

Листинг 14.20

```
Sub AreaOfCircle_10()
'Вычисление площади круга с проверкой исходных данных
'Цикл Do, инструкция Loop Until, функция InputNonZero
    Dim pi, d, s
    Dim strPrompt As String 'сообщение диалогового окна
    Dim strD As String 'введенное значение диаметра
    pi = 3.141593
    strPrompt = "Введите ненулевой диаметр круга" & vbCrLf _
        & "и нажмите кнопку OK." _
        & vbCrLf & vbCrLf & "Чтобы выйти," & vbCrLf & "нажмите кнопку
Cancel."
    d = InputNonZero(strPrompt:=strPrompt, strTitle:="Площадь круга", _
        strDefault:="<d>")
    If d <> 0 Then
        s = pi * d ^ 2 / 4
        Call MsgBox(Prompt:="d= " & d & vbCrLf & "s= " & s, _
            Title:="Площадь круга")
    End If
End Sub
```

Программирование функции выполняется аналогично программированию подпрограммы. В отличие от подпрограммы начальной инструкцией функции является инструкция **Function** (функция), а завершающей инструкцией — инструкция **End Function** (конец функции). В сущности, функции отличаются от подпрограмм только тем, что функции можно использовать в выражениях наряду с числами, строками и переменными.

Посмотрите на листинг 14.21, в котором приведена функция `InputNonZero`.

Листинг 14.21

```
Function InputNonZero(strPrompt As String, strTitle As String, _
    strDefault As String) As Variant
' Ввод ненулевого числа
' Диалоговое окно отображается на экране до тех пор, пока не будет
'     введено ненулевое число,
'     введена пустая строка или
'     нажата кнопка Cancel
' Возвращается
'     введенное число, если введено ненулевое число, или
'     ноль, если введена пустая строка или нажата кнопка Cancel
' strPrompt - сообщение диалогового окна
' strTitle - заголовок диалогового окна
' strDefault - исходное значение поля ввода
    Dim d As Variant, strD As String
    Do
        strD = InputBox(Prompt:=strPrompt, Title:=strTitle, _
            Default:=strDefault)
        strD = Replace(Expression:=strD, Find:=",", Replace:=".")
        d = Val(strD)
    Loop Until strD = "" Or d <> 0
    InputNonZero = d
End Function
```

Новой здесь является инструкция **Function**. Аналогично тому, как после ключевого слова **Sub** записывается название подпрограммы, после ключевого слова **Function** записывается название функции. Функция может возвращать значение, поэтому для функции можно указать тип, который является типом возвращаемого значения. Он указывается после закрывающей скобки с использованием ключевого слова **As** так же, как в инструкции **Dim**. В дан-

ном случае функция возвращает численное значение, для которого, согласно нашей договоренности *(см. разд. 14.1.5)*, мы используем тип **Variant**. Если тип функции не указан, то по умолчанию используется тип **Variant**. Таким образом, в нашем случае тип функции можно и не задавать.

В скобках указывается перечень аргументов. Перечень аргументов в нашем примере выглядит точно так же, как описание переменных в инструкции **Dim**, в нем указан тип каждого аргумента с использованием ключевого слова **As**. Если же тип аргумента не указан, то, как вы уже догадываетесь, используется тип **Variant**. Мы не будем рассказывать обо всех возможностях, связанных с передачей аргументов, но учтите, что список аргументов может содержать гораздо больше информации (об этом подробно написано в других книгах). Обратите внимание, наличие перечня аргументов не является отличительной чертой функций. У подпрограмм тоже могут быть аргументы, но об этом мы поговорим в следующем подразделе.

Еще одной особенностью функции является наличие в ней инструкции, в которой присваивается значение, которое будет возвращаться функцией. В нашем примере это инструкция

```
InputNonZero = d
```

В этой инструкции имя функции используется так же, как имя переменной, которой присваивается значение, однако для других целей "внутри" функции ее имя использовать нельзя.

Все остальные инструкции в функции `InputNonZero` нам уже знакомы. Аргументы функции в них используются как переменные. Можно считать, что при выполнении инструкции, в которой происходит обращение к функции, эти аргументы заменяются соответствующими переменными или выражениями, указанными в этой инструкции.

Для практики запрограммируем функцию `RootDistance` (расстояние между корнями), которая вычисляет расстояние между корнями квадратного уравнения

$$x^2 - 2ax + 1 = 0.$$

Мы уже приводили формулу для вычисления его корней:

$$x_{1,2} = a \pm \sqrt{a^2 - 1}.$$

Из этой формулы следует, что расстояние между корнями квадратного уравнения вычисляется по формуле:

$$2\sqrt{a^2 - 1}.$$

Если подкоренное выражение отрицательное, то корней у квадратного уравнения не существует. Пусть в этом случае функция возвращает число -1.

Один из возможных вариантов функции `RootDistance` представлен в листинге 14.22.

Листинг 14.22

```
Function RootDistance(a As Variant) As Variant
' Расстояние между корнями квадратного уравнения x^2 - 2ax + 1 = 0
    Dim d As Variant 'дискриминант
    d = a ^ 2 - 1
    If d < 0 Then
        RootDistance = -1
        Exit Function
    End If
    RootDistance = 2 * Sqr(d)
End Function
```

В этой функции использована новая для нас инструкция **Exit Function** (выход из функции), завершающая выполнение функции и передающая управление подпрограмме (или функции), из которой вызывалась функция `RootDistance`. Обратите внимание, что прежде чем завершить выполнение функции, необходимо выполнить инструкцию присвоения возвращаемого значения. В этой функции нам понадобились две такие инструкции.

Важный момент: инструкции **Exit Function** и **End** действуют по-разному. Если инструкция **End** останавливает выполнение макроса, то инструкция **Exit Function** завершает работу функции и передает управление в ту точку, откуда подпрограмма вызывалась. Таким образом, инструкция **Exit Function** действует так же, как инструкция **End Function**, но не является последней инструкцией функции.

Чтобы воспользоваться созданной нами функцией, необходимо написать подпрограмму-макрос, из которой вызывается функция. Эту подпрограмму можно написать, например, в виде, представленном в листинге 14.23.

Листинг 14.23

```
Sub Quadratic_6()
' Расстояние между корнями квадратного уравнения x^2 - 2ax + 1 = 0
    Dim a
    Dim r ' расстояние между корнями
    Dim strPrompt As String 'строка вывода
    Dim strTitle As String 'заголовок окон ввода и вывода
```

```
    strPrompt = "a:"
    strTitle = "Расстояние между корнями уравнения x^2 - 2ax + 1 = 0"
    a = InputBox(Prompt:=strPrompt, Title:=strTitle)
    r = RootDistance(a)
    If r = -1 Then
        strPrompt = "a =" & a & ", корни не существуют"
        Call MsgBox(Prompt:=strPrompt, Title:=strTitle)
        Exit Sub
    End If
    strPrompt = "a =" & a & ", расстояние между корнями =" & r
    Call MsgBox(Prompt:=strPrompt, Title:=strTitle)
End Sub
```

Обратите внимание, что здесь используется стандартный подход, согласно которому ввод-вывод и вычисления разделены: в подпрограмме производится ввод исходных данных и вывод результатов, а вычисления сосредоточены в функции.

В подпрограмме `Quadratic_6` только одна новая инструкция **`Exit Sub`** (выход из подпрограммы). Она завершает выполнение подпрограммы аналогично тому, как инструкция **`Exit Function`** завершает работу функции. Инструкция **`Exit Sub`** действует так же, как **`End`**, если запуск подпрограммы `Quadratic_6` производится из Visual Basic или Microsoft Word. Однако, если вызывать подпрограмму `Quadratic_6` из макроса, действие этих инструкций будет различным: **`Exit Sub`** будет возвращать управление вызывающей подпрограмме.

14.4.2. Сравнение функций и подпрограмм

Как мы уже отмечали, функции отличаются от подпрограмм тем, что функции могут использоваться в выражениях. Между теми функциями и подпрограммами, которые мы разбирали или составляли до сих пор, было еще одно отличие — у подпрограмм не было аргументов, а у функций аргументы были всегда. Однако есть функции без аргументов, например, нет аргументов у встроенной функция `Time` (время), которая возвращает текущее время, а сейчас мы познакомимся с подпрограммами, использующими аргументы. То, что до сих пор у наших подпрограмм не было аргументов, объясняется тем, что только такие подпрограммы могут быть макросами. Если у подпрограммы есть аргументы, то обратиться к ней напрямую из Microsoft Word нельзя. Ей нельзя назначить ни кнопку, ни сочетание клавиш, она не появляется в списке макросов в диалоговом окне **Макрос** (которое вызывается из меню **Сервис | Макрос | Макросы**). Такие подпрограммы, также как и функ-

ции, предназначены исключительно для вызова из других подпрограмм или функций.

Чтобы понять, как составляются подпрограммы с аргументами, посмотрите, как выглядит подпрограмма `InputNonZero1`, решающая ту же задачу, что и функция `InputNonZero` (листинг 14.24).

Листинг 14.24

```
Sub InputNonZero1(strPrompt As String, strTitle As String, _
    strDefault As String, varResult As Variant)
' Ввод ненулевого числа
' Диалоговое окно отображается на экране до тех пор, пока не будет
'     введено ненулевое число,
'     введена пустая строка или
'     нажата кнопка Cancel
' Возвращается
'     введенное число, если введено ненулевое число, или
'     ноль, если введена пустая строка или нажата кнопка Cancel
' strPrompt - сообщение диалогового окна
' strTitle - заголовок диалогового окна
' strDefault - исходное значение поля ввода
' varResult - возвращаемое значение
Dim strD As String
    Do
        strD = InputBox(Prompt:=strPrompt, Title:=strTitle, _
            Default:=strDefault)
        strD = Replace(Expression:=strD, Find:=",", Replace:=".")
        varResult = Val(strD)
    Loop Until strD = "" Or varResult <> 0
End Sub
```

Новое здесь то, что в инструкции **Sub** в скобках указывается список аргументов, который составляется по тем же правилам, что и список аргументов инструкции **Function**. Обратите внимание, что список аргументов содержит и входные значения, и результат работы подпрограммы. Часто бывает, что один и тот же аргумент служит и для задания входной информации, и для возврата полученных результатов.

Заметим, что точно также может использоваться список аргументов и у функций.

Обращение к подпрограммам с аргументами выполняется с помощью уже знакомой нам инструкции **Call** (вызов). Например, чтобы в подпрограмме `AreaOfCircle_10` заменить обращение к функции `InputNonZero` обращением к подпрограмме `InputNonZero`, достаточно инструкцию

```
d = InputNonZero(strPrompt:=strPrompt, strTitle:="Площадь круга", _
    strDefault:="<d>")
```

заменить инструкцией

```
Call InputNonZero1(strPrompt:=strPrompt, strTitle:="Площадь круга", _
    strDefault:="<d>", varResult:=d)
```

Таким образом, можно сделать вывод, что функции и подпрограммы при наличии аргументов взаимозаменяемы. Действительно, любую функцию можно заменить подпрограммой, добавив еще один аргумент для передачи возвращаемого значения. Если эта функция входила в выражение, понадобится ввести еще одну переменную, которая будет предварительно вычисляться с помощью подпрограммы, а затем использоваться в выражении вместо функции. И наоборот, любую подпрограмму можно заменить функцией, которая будет возвращать одно из вычисляемых в подпрограмме значений. Можно даже просто заменить ключевое слово **Sub** ключевым словом **Function** в инструкциях **Sub**, **Exit Sub** и **End Sub** и больше ничего нигде не менять. Получится функция без возвращаемого значения, которую можно вызывать с помощью ключевого слова **Call** так же, как вызывается подпрограмма.

Естественно возникает вопрос: когда следует пользоваться функцией, а когда подпрограммой? Прежде чем на него ответить, заметим, что аргументы всех встроенных функций VBA содержат только входную информацию. Поэтому, встретив в подпрограмме обращение к функции, большинство пользователей будет предполагать, что ее аргументы не содержат возвращаемых значений. Следовательно, чтобы не вводить читателей вашей программы в заблуждение, мы рекомендуем вам использовать только такие функции, у которых в аргументах содержится только исходная информация. Во всех остальных случаях лучше использовать подпрограммы.

- Во-первых, для создания макросов, как мы уже говорили, могут использоваться только подпрограммы без аргументов.
- Во-вторых, если процедура предназначена для выполнения определенных действий и не возвращает никаких данных, ее лучше оформлять как подпрограмму, поскольку функция без возвращаемого значения будет только путать читателя.
- В-третьих, если процедура возвращает несколько значений, то, как мы уже сказали, лучше оформить ее как подпрограмму.

14.4.3. Область видимости переменных и процедур

Когда вы начнете широко применять вызов процедур (т. е. функций и подпрограмм) из других процедур, у вас обязательно возникнет вопрос о том, к каким переменным и процедурам можно обратиться из вашей процедуры, т. е. какова *область видимости* процедур и переменных, и как можно ее сужать и расширять. Ясно, что широкая область видимости расширяет возможности как вызова процедур, так и совместного использования данных разными процедурами. Однако за все хорошее приходится платить. Проблемы, возникающие при появлении в области видимости двух одинаковых имен, зачастую сводят на нет преимущества расширения области видимости.

Второй вопрос — до какого момента сохраняется значение, присвоенное переменной, т. е. каково *время жизни* переменной? Ответы на эти два вопроса мы постараемся дать в этом подразделе.

В этом разделе мы будем использовать понятия *проект* и *модуль*, о которых рассказывалось в *гл. 10*. Если необходимо, вспомните, что они означают.

Переменные, которые мы описываем в процедурах, по умолчанию действуют только внутри той процедуры, в которой они описаны. Точно также аргументы процедур действуют только внутри соответствующих процедур. Таким образом, передача информации от одной процедуры к другой, осуществляется только через обращения к процедурам. При написании процедур не надо заботиться о том, какие имена даны переменным и аргументам в других процедурах.

В VBA предусмотрена возможность описания переменных, которые можно использовать во всех процедурах модуля или во всех модулях проекта или даже во всех проектах. Эта возможность используется в тех случаях, когда необходимо передавать из процедуры в процедуру очень большое число переменных. Тогда приходится следить за тем, чтобы имена этих переменных не использовались для других целей. Кроме того, из инструкции обращения к процедуре в этих случаях нельзя узнать, какая информация передается между процедурами. Все это затрудняет написание и отладку программ. Мы рекомендуем не пользоваться этой возможностью без особой необходимости. В этой книге мы описывать ее не будем.

Процедуры по умолчанию доступны во всех модулях и проектах. Проблему конфликта имен макросов и способы ее разрешения мы уже обсуждали в *гл. 7*, когда рассказывали о применении дополнительных шаблонов с макросами. В одном модуле не может быть двух процедур с одинаковыми именами. Допустимо наличие процедур с одинаковыми именами, если эти процедуры находятся в разных модулях. Для того чтобы можно было отличить две процедуры с одинаковыми именами, находящиеся в разных модулях, следует при обращении к ним указывать перед именем процедуры имя мо-

дуля, отделенное точкой от имени процедуры. Точно так же можно перед именем модуля добавить имя проекта с последующей точкой. С этим способом именования мы встречались, когда обсуждали задачу назначения сочетаний клавиш макросам из различных модулей и шаблонов *(см. гл. 10)*. В то же время можно обратиться к процедуре, находящейся в другом модуле точно так же, как к процедуре, находящейся в "своем" модуле. Если в "своем" модуле процедуры с указанным именем нет, то она будет найдена в другом модуле. Конфликт имен возникает, если в своем модуле нужной процедуры нет, а в других модулях есть больше одной процедуры с указанным именем.

Чтобы не беспокоиться о возможности случайного совпадения имен процедур, можно сужать область видимости процедур. Чтобы сделать процедуру доступной только внутри того модуля, где она находится, достаточно перед ключевым словом **Sub** (подпрограмма) или **Function** (функция) добавить ключевое слово **Private** (закрытый). Учтите, что подпрограммы, в заголовке которых есть такое ключевое слово, не могут быть макросами, т. е. их нельзя запустить непосредственно из окна Microsoft Word или редактора Visual Basic.

"Жизнь" переменной начинается с момента ее описания (если вы решили не описывать переменные явным образом, то их описание происходит неявно в момент первого использования). До сих пор мы говорили и будем продолжать говорить только о переменных, область видимости которых ограничивается процедурой, в которой они описаны. Жизнь таких переменных заканчивается в момент выхода из соответствующей подпрограммы. Значение переменной при этом теряется. В VBA предусмотрена возможность продлить жизнь переменных, но рассказывать мы об этом не будем.

14.5. Самостоятельное задание

Для тренировки попробуйте самостоятельно на основе подпрограммы `CheckFileName` создать подпрограмму с аргументами `CheckFileName1`, которая будет проверять допустимость имени файла, и подпрограмму без аргументов `CheckFileName1_test` (*test* — проверка), которая будет делать то же самое, что и подпрограмма `CheckFileName`, но с использованием подпрограммы `CheckFileName1`.

Одно из возможных решений этой задачи приведено в листингах 14.25 и 14.26.

Листинг 14.25

```
Sub CheckFileName1(strFileName As String, No As Variant, _
    strSymbol As String)
```

```
'Проверка допустимости имени файла
' Вход:
' strFileName - проверяемая строка
' Выход:
' No=0 - признак допустимости строки
' No=-1 - введена пустая или содержащая одни пробелы строка
' No>0 - номер недопустимого символа
' strSymbol - недопустимый символ
Dim str1 As String 'очередной символ строки
Dim i 'переменная цикла
If Trim(strFileName) = "" Then
    No = -1
    Exit Sub
End If
No = 0
For i = 1 To Len(strFileName)
    str1 = Mid(String:=strFileName, Start:=i, Length:=1)
    If str1 = "\" Or str1 = "/" Or str1 = ":" _
            Or str1 = "*" Or str1 = "?" Or str1 = """" _
            Or str1 = "<" Or str1 = ">" Or str1 = "|" Then
        No = i
        strSymbol = str1
        Exit For
    End If
Next i
End Sub
```

Листинг 14.26

```
Sub CheckFileName1_test()
'Проверка допустимости имени файла
'Вызов функции CheckFileName1()
Dim strFileName As String 'проверяемая строка
Dim No 'No=0 - признак допустимости строки
'       No=-1 - введена пустая или содержащая одни пробелы строка
'       No>0 - номер недопустимого символа
Dim str1 As String 'недопустимый символ
```

```
strFileName = InputBox("Введите имя файла", _
    "Проверка допустимости имени файла")
Call CheckFileName1(strFileName:=strFileName, No:=No, strSymbol:=str1)
If No = 0 Then
    Call MsgBox("""" & strFileName & """ - допустимое имя файла")
ElseIf No = -1 Then
    Call MsgBox("Введена пустая или содержащая одни пробелы строка")
Else
    Call MsgBox("В строке """ & strFileName & """ на позиции " & No _
        & " недопустимый символ " & str1)
End If
End Sub
```

14.6. Дополнительные сведения

14.6.1. Инструкция *Stop*

Мы уже познакомились с инструкцией **End** (закончить). Она прекращает выполнение подпрограммы и освобождает память от всех использованных в макросе переменных. Однако есть еще одна инструкция, прерывающая работу программы. Это инструкция **Stop** (остановить). Она также состоит из одного ключевого слова. Отличие этих двух инструкций проявляется только в том случае, когда во время работы макроса открыт редактор Visual Basic. Тогда инструкция **Stop** после прерывания работы макроса активизирует редактор, выводит в нем окно с текстом соответствующей подпрограммы и выделяет в нем стрелкой слева и желтым цветом инструкцию **Stop**. При этом в отличие от инструкции **End** сохраняются значения всех переменных, а редактор Visual Basic переходит в режим отладки, о возможностях которого вам еще много предстоит узнать. В частности, можно продолжить выполнение программы, выбрав в меню **Run** (Запустить) пункт **Run Sub/UserForm** (Выполнить подпрограмму или пользовательскую форму). Для продолжения программы можно нажать на клавиатуре клавишу <F5> или на панели инструментов нажать кнопку **Run Sub/UserForm** (см. рис. 14.1).

14.6.2. Безусловный переход

Наряду с условным переходом в языке VBA возможен и безусловный переход, выполняемый с помощью инструкции **GoTo** (перейти). Он позволяет без каких бы то ни было условий передать управление другой инструкции, находящейся в той же процедуре. Несмотря на простоту, использование

инструкции `GoTo` снижает наглядность программы и затрудняет ее проверку и отладку. Чтобы было понятно, на каком основании сложилось такое мнение, рассмотрим типичную ситуацию. Предположим, что во время выполнения нашей программы Visual Basic обнаруживает ошибку и мы переходим в режим отладки. При этом редактор Visual Basic показывает, при выполнении какой инструкции обнаружена ошибка. Анализируя значения переменных, используемых в этой инструкции, можно понять, какие из них посчитаны неправильно. Теперь необходимо выяснить, как получились неправильные значения. Таким образом, надо проверить инструкции, которые выполнялись до того, как была обнаружена ошибка. Если же среди этих инструкций могла быть инструкция `GoTo`, такая проверка становится гораздо более сложной.

У вас должны быть веские основания для применения инструкции безусловного перехода в своей программе. В наших примерах эту инструкцию вы не встретите, и о ее использовании мы в этой книге не рассказываем, хотя есть ситуации (например, при программной обработке ошибок), когда без нее не обойтись.

14.6.3. О погрешностях вычислений

Как вы думаете, что у нас получилось в результате выполнения следующей подпрограммы, вычисляющей значение выражения 1/2 − 1/6 − 1/3?

```
Sub Test1()
    MsgBox 1 / 2 - 1 / 6 - 1 / 3
End Sub
```

В окне Microsoft Word открылось диалоговое окно, показанное на рис. 14.14.

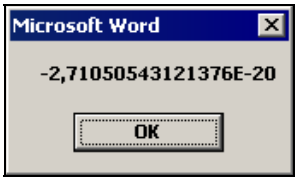


Рис. 14.14. Результат вычисления выражения 1/2 − 1/6 − 1/3

Мы получили в результате очень маленькое число (для тех, кто этого не знает, сообщаем, что так в компьютерах записывается число $-2{,}71050543121376 \quad 10^{-20}$). Не удивляйтесь, если у вас получится другой результат: он зависит от версии установленного на вашем компьютере программного обеспечения. Этот пример показывает, что, во-первых, вы-

числения выполняются с определенной погрешностью, а во-вторых, эта погрешность весьма мала и в большинстве случаев ею можно пренебречь.

Однако при сравнении числовых выражений наличие погрешности вычислений следует учитывать. Как вы думаете, истинно или ложно логическое выражение $1/2 - 1/6 - 1/3 = 0$, и что получится в результате выполнения следующей подпрограммы, вычисляющей значение этого логического выражения?

```
Sub Test2()
    MsgBox 1 / 2 - 1 / 6 - 1 / 3 = 0
End Sub
```

Наверное, теперь для вас не будет неожиданностью, что в результате выполнения этого макроса появится диалоговое окно, показанное на рис. 14.15.

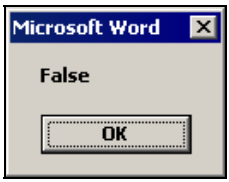


Рис. 14.15. Результат вычисления логического выражения 1/2 – 1/6 – 1/3 = 0

Теперь вы понимаете, что если результатом вычисления логического выражения $1/2 - 1/6 - 1/3 = 0$ получается логическая константа **`False`** (ложь), то не надо писать возмущенное письмо Биллу Гейтсу. Тем не менее, проблема остается. Как же учесть влияние погрешности вычисления на результат сравнения? Если это возможно, то лучше всего преобразовать формулу так, чтобы сравнивались целочисленные выражения, которые точно представляются значениями в ячейках памяти компьютера. Например, умножив левую и правую части равенства $1/2 - 1/6 - 1/3 = 0$ на шесть, мы получим эквивалентное условие $3 - 1 - 2 = 0$, при вычислении которого погрешности не возникают. Другой подход к решению проблемы основан на оценке погрешностей вычисления. Например, если мы уверены, что абсолютная погрешность вычисления чисел не превышает 10^{-6}, то вместо условия `1 / 2 - 1 / 6 - 1 / 3 = 0` можно использовать следующее логическое выражение: `Abs(1 / 2 - 1 / 6 - 1 / 3) < 0.000003`

В этом выражении использована встроенная функция `Abs`, которая возвращает абсолютную величину числа.

14.7. Что нового в этой главе

Вы научились с помощью встроенной функции `MsgBox` выводить на экран диалоговое окно с результатами вычислений и определенными вами сообщениями. Научились использовать Visual Basic для вычислений по формулам с использованием переменных и встроенных математических функций. Узнали, как можно использовать для этой цели окно **Immediate**. Научились вводить данные в свои программы с помощью встроенной функции `InputBox`. Мы с вами обсудили выбор типа переменных для хранения числовых данных. Вы узнали об автоматическом преобразовании типов данных при выполнении вычислений. Познакомились с применением именованных аргументов при обращении к функциям и с использованием позиционной передачи аргументов. Освоили применение скобок и ключевого слова **`Call`** при обращении к функциям. Научились использовать операцию объединения строк. Узнали о строковой константе `vbCrLf` и об автоматическом преобразовании типов при формировании строковых выражений. Научились пользоваться встроенными функциями `Val` и `Replace` для преобразования строк в числа и выполнения замены в строках. Познакомились с диалоговым окном, в котором выводятся сообщения об ошибках во время выполнения программ. Научились применять различные формы условного перехода, используя инструкции **`If`**, **`Else`**, **`ElseIf`**, **`End If`**. Научились завершать выполнение программы с помощью инструкции **`End`**. Познакомились с логическими выражениями. Узнали об операциях сравнения и логических операциях. Научились изменять величину отступов в тексте программ. Получили первое представление о возможностях отладки программ, предусмотренных в редакторе Visual Basic. Научились применять циклы в своих программах с использованием инструкций **`For`**, **`Next`** и **`Exit For`**, а также инструкций **`Do`**, **`Loop`** и **`Exit Do`**. Познакомились со встроенными функциями `Trim`, `Len` и `Mid`. Научились "вручную" прерывать работу макроса. Освоили использование логических выражений для выхода из цикла. Научились создавать собственные функции и подпрограммы с аргументами. Научились организовывать обмен информацией с функциями и подпрограммами с помощью аргументов. Освоили использование инструкций **`Function`**, **`End Function`**, **`Exit Function`**, **`Exit Sub`**. Узнали о различии в использовании функций и подпрограмм. Познакомились с понятиями видимости переменных и процедур, а также временем жизни переменных. Узнали о применении ключевого слова **`Private`** для сужения области видимости процедур и о способах разрешения конфликта имен.

Новые элементы языка VBA приведены в табл. 14.3—14.4.

Таблица 14.3. *Новые инструкции*

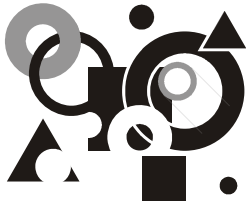
Call имя процедуры (список аргументов)
End
Stop
IF логическое выражение **Then**
ElseIf логическое выражение **Then**
Else
End If
For переменная цикла **=** нач. значение **To** конеч. значение **Step** шаг
Next переменная цикла
Do
Do While логическое выражение
Do Until логическое выражение
Exit Do
Loop
Loop While логическое выражение
Loop Until логическое выражение
Private Function имя функции **(**список аргументов**) As** тип функции
Private Sub имя функции **(**список аргументов**)**
Exit Function
End Function

Таблица 14.4. *Новые функции*

Функция	Значение	Аргумент	Константа
Msgbox		Prompt	
		Title	
InputBox		Buttons	
		Title	
		Default	

Таблица 14.4 *(окончание)*

Функция	Значение	Аргумент	Константа
Abs			
Atn			
Cos			
Exp			
Log			
Sin			
Sqr			
Tan			
Val			
Replace		Expression Find Replace	
Len			
Mid		String Start Lenght	
Time			



Глава 15

Закладки на все случаи жизни

Удаление закладки с проверкой ее наличия. Удаление всех закладок. Удаление нескрытых закладок. Установка закладки с уникальным понятным именем. Решение проблемы перехода к закладкам в документах с колонтитулами, надписями, сносками и примечаниями. Программирование сверху вниз. Знакомство с цепочками. Дополнительные сведения о функции `MsgBox`.

В этой главе мы будем пожинать плоды просвещения. Материал предыдущих глав позволил вам познакомиться с некоторыми возможностями, предоставляемыми макросами, вы научились читать и изменять подпрограммы, познакомились с основами программирования. В этой главе мы будем применять полученные навыки для решения задач, связанных с закладками.

Вы уже умеете устанавливать закладку с фиксированным именем, переходить к этой закладке и удалять закладку. В этом разделе мы усовершенствуем макросы, решающие эти задачи, и напишем несколько новых полезных макросов. В результате вы узнаете много нового о свойствах закладок, попрактикуетесь в разработке программ и продолжите знакомство с возможностями программирования на языке Visual Basic.

15.1. Корректное удаление закладок

Начнем с того, что усовершенствуем макрос, удаляющий закладку **Marker**. Этот макрос был создан еще в первой части книги, когда учились записывать макросы в автоматическом режиме. Мы назвали его **DeleteMarker** (удалить Marker). Затем в *разд. 11.2* мы удалили из него лишние инструкции и измененную подпрограмму назвали `DeleteMarker1`. Для удобства еще раз приведем текст программы в листинге 15.1.

Листинг 15.1

```
Sub DeleteMarker1()
'
' Удаление закладки Marker
' Лишние инструкции удалены
'
    ActiveDocument.Bookmarks("Marker").Delete
End Sub
```

В этом макросе неявно предполагается, что в документе установлена закладка **Marker**. Если его запустить, когда в документе этой закладки нет, то выполнение макроса прерывается и на экране появляется сообщение об ошибке (рис. 15.1).

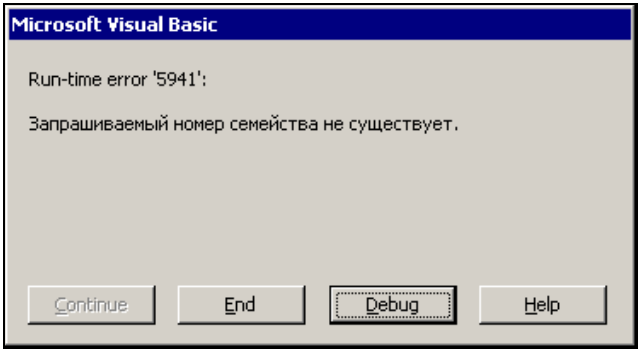


Рис. 15.1. Сообщение при попытке удалить несуществующую закладку

Нажав кнопку **Debug** (Отладка), вы перейдете на инструкцию подпрограммы `DeleteMarker`, в которой удаляется закладка **Marker**. Если вы еще не забыли, что совокупность закладок является объектом-семейством, то, скорее всего, догадаетесь, что в сообщении содержится намек на то, что закладки с таким именем не существует. Проверить это несложно (меню **Вставка | Закладка**). Однако для тех пользователей, которые не знают, в чем заключается связь между закладками и семейством, это будет непонятное и, следовательно, не очень полезное сообщение. Поэтому лучше его заменить своим более понятным. Для этого необходимо перед удалением закладки проверить, что она существует, и, при необходимости, сообщить пользователю, что такой закладки в документе нет.

Удостовериться в наличии закладки можно с помощью метода `Exists` (существует) семейства `Bookmarks` (закладки). У этого метода один обязательный аргумент `Name` (имя) — строка, содержащая имя закладки. Возвращает этот метод логическое значение **`True`** или **`False`** в зависимости от того, существует ли в документе закладка с указанным именем.

Если вы внимательно читали предыдущие главы, то этих сведений вам будет достаточно, чтобы самостоятельно написать усовершенствованную подпрограмму `DeleteMarker2`. Эта программа сначала должна проверять, существует ли закладка **Marker**, и если такая закладка существует, то удалять ее, а при ее отсутствии сообщать пользователю: "Закладку «Marker» удалить не удалось, поскольку такой закладки в документе нет" (напоминаем, что кавычка внутри строки обозначается двумя кавычками).

У вас должна получиться примерно такая же подпрограмма, как приведенная в листинге 15.2.

Листинг 15.2

```
Sub DeleteMarker2()
' Удаление закладки Marker
' если она существует
    If ActiveDocument.Bookmarks.Exists("Marker") = True Then
        ActiveDocument.Bookmarks("Marker").Delete
    Else
        MsgBox Prompt:="Закладку ""Marker"" удалить не удалось, " _
            & "поскольку такой закладки в документе нет" _
        , Title:="Удаление закладки ""Marker"""
    End If
End Sub
```

С этой подпрограммой общаться гораздо приятнее, чем с подпрограммой `DeleteMarker1`. Она не "ругается" непонятными словами, если пользователь обратится к ней в ситуации, когда выполнить его желание невозможно, а четко ему докладывает о сложившейся ситуации.

Если в процессе работы с документом установлено много временных закладок, то их ручное удаление на завершающем этапе работы с документом занимает заметное время. В Microsoft Word возможность удаления сразу всех закладок не предусмотрена, но можно самому написать макрос, эффективно решающий эту задачу. Простейший вариант такой программы с именем `DelBookmarks` (удалить закладки) приведен в листинге 15.3.

Листинг 15.3

```
Sub DelBookmarks()
' Удаление всех закладок
    Dim bkm As Bookmark
    For Each bkm In ActiveDocument.Bookmarks
        bkm.Delete
    Next bkm
End Sub
```

В этой небольшой подпрограмме много нового. Начнем с того, что в инструкции

```
Dim bkm As Bookmark
```

при описании переменной `bkm` встретился новый тип `Bookmark`. Это так называемый *объектный тип*. Оказывается, переменные можно использовать не только для данных, но и для объектов. Мы знаем, что существует объект (точнее, класс) `Bookmark`, представляющий одну закладку. В этой инструкции описывается *объектная переменная* `bkm`, в которой могут храниться объекты `Bookmark`. Аналогичным образом описываются объектные переменные и для других объектов. Чуть позже вы узнаете, зачем тут понадобилась объектная переменная.

Дальше следует конструкция, напоминающая цикл **For**. Это действительно цикл, и смысл его в том, что все инструкции, находящиеся между инструкциями начала и конца цикла (инструкцией **For Each** и инструкцией `Next`), выполняются для каждого элемента семейства. Эта же конструкция используется для цикла по всем элементам массива, но рассказ о массивах выходит за рамки нашей книги.

Рассмотрим подробнее инструкцию

```
For Each bkm In ActiveDocument.Bookmarks
```

В этой инструкции вам встретились два новых ключевых слова: **Each** (каждый) и **In** (в). Перевести на русский язык всю эту инструкцию в целом можно следующим образом: "Для каждого значения переменной `bkm` в семействе `ActiveDocument.Bookmarks` выполнить следующие инструкции". После слов **For Each** указывается объектная переменная цикла, а после слова **In** указывается ссылка на семейство. Цикл выполняется для каждого элемента семейства, и на каждом шаге цикла для ссылки на очередной элемент семейства используется объектная переменная. В нашем примере выполняется цикл по семейству `Bookmarks`, представляющему все закладки в документе. Тип объектной переменной `bkm`, как и положено, соответствует типу

элемента семейства, т. е. `Bookmark`. На каждом шаге цикла для ссылки на объект, представляющий очередную закладку, используется объектная переменная `bkm`.

В нашем случае между инструкцией **`For Each`** и инструкцией **`Next`** расположена только одна инструкция

```
bkm.Delete
```

Это знакомый нам метод `Delete` объекта `Bookmark`, который удаляет соответствующую закладку из документа.

Следующая инструкция нам тоже знакома:

```
Next bkm
```

После ключевого слова **`Next`** указывается переменная цикла. Новым здесь является только то, что это объектная переменная. Указывать объектную переменную необязательно, но мы рекомендуем это делать.

Для тестирования этой процедуры воспользуйтесь каким-либо документом, в котором имеются оглавление, перекрестные ссылки и закладки. Если у вас нет под рукой такого документа, то создайте его. Как уже отмечалось в предыдущих главах книги, при создании оглавления и перекрестных ссылок в документе создаются так называемые скрытые закладки. Мы уже упоминали о них в *разд. 1.2.2* и *11.2*.

Откройте окно **Закладка** (меню **Вставка | Закладка**). Установите флажок **Скрытые закладки**. Вы увидите много закладок, имена которых начинаются со знака подчеркивания. Это и есть скрытые закладки, которые были автоматически созданы, когда вы вставляли в документ оглавление и перекрестные ссылки. Если же эти закладки удалить, то нельзя будет перейти из оглавления к соответствующему заголовку, и перестанут работать перекрестные ссылки. Можете проверить это сами. Убедитесь также, что макрос **DelBookmarks** удаляет скрытые закладки тогда и только тогда, когда в диалоговом окне **Закладки** установлен флажок **Скрытые закладки**. Удалите оглавление и создайте его заново. Повторите эту операцию несколько раз, чтобы убедиться, что при создании оглавления появляются новые скрытые закладки, которые остаются и после удалении оглавления.

Из этих экспериментов можно сделать следующие выводы.

- Удаление закладок — совсем не безобидная операция. Нельзя допустить, чтобы при случайном вызове макроса в результате удаления скрытых закладок незаметно для пользователя испортился документ. Необходимо перед удалением закладок предупредить пользователя о такой возможности и получить у него подтверждение на выполнение этой операции.
- Следует четко разграничивать две разные задачи: удаление скрытых и удаление нескрытых закладок. Необходимость удаления скрытых закла-

док возникает относительно редко, например, после многократной переделки подробного оглавления можно несколько сократить размер файла, удалив скрытые закладки. Более актуальна задача удаления нескрытых закладок, поскольку желательно, чтобы в готовом документе не содержалось никаких "лишних" элементов. Однако скрытые закладки необходимо оставить, чтобы не нарушать работу перекрестных ссылок и оглавления. Поэтому следует предусмотреть, чтобы во время удаления закладок был снят флажок **Скрытые закладки**.

Итак, перед нами стоят две задачи. Во-первых, получить подтверждение пользователя на удаление закладок и, во-вторых, снять флажок **Скрытые закладки**. Приступим к их решению.

Начнем с получения подтверждения пользователя. Вы уже многократно применяли функцию `MsgBox` (окно сообщений), но мы ни разу не сказали вам о том, что она не только позволяет выводить сообщения, но также предоставляет пользователю возможность выбрать один из предлагаемых в сообщении вариантов. Для этой цели предназначен второй аргумент функции `MsgBox`, имя которого — `Buttons` (кнопки). С его помощью можно указать, какой набор кнопок будет выводиться в диалоговом окне и какой значок будет в нем отображаться.

По умолчанию в диалоговом окне отображается только одна кнопка **OK**, а значок вообще отсутствует. Чтобы отображалась пара кнопок **Да** и **Нет**, следует в качестве значения аргумента `Buttons` выбрать константу `vbYesNo`, а если мы еще хотим, чтобы в диалоговом окне отображался значок в виде знака вопроса, к этой константе надо добавить еще одну константу `vbQuestion` (вопрос). Когда мы говорим "добавить", то мы имеем в виду обычное арифметическое сложение — эти константы являются просто обозначением конкретных чисел.

Если второй аргумент позволяет вывести в диалоговом окне набор кнопок, чтобы пользователь мог сообщить свое решение, то узнать, в чем состоит его решение, позволяет значение, возвращаемое функцией. В зависимости от того, какие кнопки или клавиши нажимал пользователь, чтобы закрыть диалоговое окно, функция `MsgBox` возвращает различные целые числа. Для обозначения этих чисел также используются константы. Например, если нажата кнопка **Да**, то функция `MsgBox` возвращает число, соответствующее константе `vbYes`. Кнопке **Нет** соответствует константа `vbNo`. Дополнительные сведения о функции `MsgBox` приведены в *разд. 15.5*.

Для возвращаемого значения опишем переменную `ReturnValue` (возвращаемое значение) без указания типа (если вы помните, мы договорились не указывать явно тип переменных, предназначенных для числовых значений).

Чтобы обратиться к функции `MsgBox`, воспользуемся следующей инструкцией:

```
ReturnValue _
    = MsgBox(Prompt:="Будут удалены все нескрытые закладки." _
    & vbCr & vbCr & "Продолжать?", Buttons:=vbYesNo + vbQuestion _
    , Title:="Удаление всех нескрытых закладок")
```

Здесь вам все должно быть понятно. Если же возникают вопросы, то перечитайте четыре предыдущих абзаца. Если это не поможет, то придется вернуться к *гл. 14* и еще раз прочитать про обращение к функциям.

В результате выполнения этой инструкции появляется диалоговое окно, изображенное на рис. 15.2.

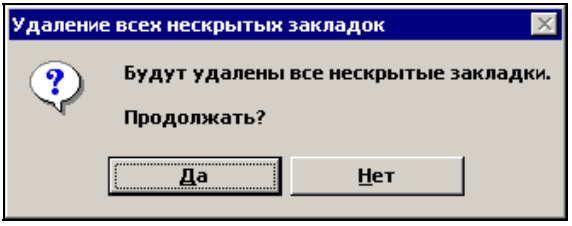


Рис. 15.2. Предупреждение усовершенствованного макроса, предназначенного для удаления закладок

Обратите внимание, что кнопка **Да** выделена. Это означает, что те, кто предпочитают пользоваться не мышью, а клавиатурой, могут нажать эту кнопку с помощью клавиши <Enter> или клавиши для ввода пробела. Чтобы нажать с помощью клавиатуры кнопку <Нет>, надо ее предварительно выделить с помощью клавиши <Tab>.

Теперь требуется проверить, какую кнопку нажал пользователь в ответ на заданный в диалоговом окне вопрос. Удалять закладки следует только в том случае, если пользователь нажмет кнопку **Да**. Как записать инструкции, в которых проверяется, какая нажата кнопка, вы уже знаете. Первая из них может быть, например, такой:

```
If ReturnValue = vbYes Then
```

Нам еще осталось решить вторую из поставленных ранее задач: предусмотреть, чтобы не были удалены скрытые закладки, т. е. позаботиться, чтобы был установлен флажок **Скрытые закладки**. В *разд. 11.2* вы соответствующую инструкцию удаляли из автоматически записанной подпрограммы `Marker`. Теперь мы можем вставить ее обратно. Запишем эту инструкцию следующим образом:

```
ActiveDocument.Bookmarks.ShowHidden = False
```

Теперь вы знаете достаточно, чтобы самостоятельно написать новый вариант подпрограммы для удаления всех нескрытых закладок из активного документа. Наш вариант приведен в листинге 15.4.

Листинг 15.4

```
Sub DelBookmarks1()
' Удаление всех нескрытых закладок
    Dim bkm As Bookmark
    Dim ReturnValue
    ReturnValue _
        = MsgBox(Prompt:="Будут удалены все нескрытые закладки." _
        & vbCr & vbCr & "Продолжать?", Buttons:=vbYesNo + vbQuestion _
        , Title:="Удаление всех нескрытых закладок")
    If ReturnValue = vbYes Then
        ActiveDocument.Bookmarks.ShowHidden = False
        For Each bkm In ActiveDocument.Bookmarks
            bkm.Delete
        Next bkm
    End If
End Sub
```

В результате мы получили полезный макрос и многому научились.

15.2. Автоматическое именование закладок

Закладки, которые мы устанавливаем в документе, можно разделить на временные и постоянные. Временные закладки мы используем, например, чтобы отметить место, в которое потом надо вернуться. Постоянные закладки широко применяются для автоматизации задач технического редактирования и при подготовке электронной документации. Например, чтобы сослаться на номер страницы, где приводится текст подпрограммы `Marker`, мы перед этим текстом устанавливаем закладку, а в том месте, где в тексте ссылки указывается номер страницы, вставляем перекрестную ссылку.

Полезные сведения

Чтобы вставить перекрестную ссылку, указывающую на закладку, следует в меню **Вставка** выбрать пункт **Перекрестная ссылка**. В диалоговом окне **Перекрестные ссылки** в поле со списком **Тип ссылки** выбрать значение **Закладка**, в поле со списком **Вставить ссылку на** выбрать значение **Номер страницы**,

снять флажок **Вставить как гиперссылку**, снять флажок **Добавить слово "выше" или "ниже"** и в списке **Для какой закладки** выделить нужную закладку. После этого в тексте ссылки появляется так называемое *поле* с номером страницы, на которой установлена закладка. От обычного текста это поле отличается тем, что в случае изменения номера страницы содержимое поля можно обновить (выделив фрагмент документа, содержащий это поле, и нажав клавишу <F9>). Можно также настроить автоматическое обновление всех полей документа перед печатью (меню **Сервис | Параметры | Печать | Обновлять поля**). В результате в отпечатанном документе в тексте ссылки всегда будет указываться правильный номер страницы.

Когда вы добавляете новую закладку для перекрестной ссылки, то ей нужно дать такое имя, чтобы закладка была узнаваемой и ее имя не совпадало с именами уже имеющихся закладок (иначе одноименная "старая" закладка будет удалена из документа без предупреждения). Если в документе требуется создать много закладок, их установка займет много времени. Мы предлагаем написать макрос, ускоряющий решение этой задачи.

Оказывается, Microsoft Word 2000 (или любая из более поздних версий) умеет расставлять уникальные закладки с узнаваемым именем, но он это скрывает. Мы так говорим потому, что Microsoft Word вставляет скрытые закладки, о которых редко кто знает, нигде в документации об этом не упоминается, а воспользоваться этой возможностью, чтобы установить нескрытую закладку с такими же свойствами, не удается. Microsoft Word применяет это свое умение, когда вы вставляете гиперссылки на заголовки, отмеченные стандартными стилями Заголовок 1, Заголовок 2 и т. д. (меню **Вставка | Гиперссылка | Связать с: местом в документе | Выберите место в документе | Заголовки**). Как отображать скрытые закладки в диалоговом окне **Закладка** (меню **Вставка | Закладка**), мы рассказали в *разд. 11.2.*

Имя скрытой закладки Microsoft Word формирует примерно следующим образом:

1. Берется текст заголовка. Если в заголовке более 35 символов, то берутся первые 35 символов (в имени закладки должно быть не более 40 символов).
2. Все символы, не являющиеся буквами или цифрами, заменяются знаками подчеркивания.
3. В начало сформированной таким образом строки добавляется знак подчеркивания (имена всех скрытых закладок в Microsoft Word начинаются со знака подчеркивания). В результате получается начальный вариант имени закладки.
4. Если же в документе уже есть закладка с таким же именем, в конец имени добавляется подчеркивание и цифра 1 (или 2, 3 и т. д.), чтобы имя закладки было уникальным. В результате получается новый вариант имени закладки.

Мы применим такой же алгоритм, но в качестве исходной строки воспользуемся не заголовком, а выделенным текстом или, если в документе ничего не выделено, текстом, расположенным справа от курсора. Вспомним, что мы предназначаем создаваемый нами макрос для установки постоянных закладок. Чтобы отличить эти закладки от остальных и случайно не удалить, добавим в начало букву b с подчеркиванием "b_". Поскольку мы таким образом зафиксировали два символа в начале закладки, а не один знак подчеркивания, то имя закладки будем формировать не из 35, а из 34 символов, расположенных справа от курсора.

Реализуем этот алгоритм в виде подпрограммы `AutoBookmark`. Чтобы текст подпрограммы читался легче, разобьем программу на части и для каждой части напишем отдельную процедуру. Функция `AutoBookmarkI` будет формировать начальный вариант имени закладки, функция `AutoBookmarkII` создаст окончательный вариант, а подпрограмма `AutoBookmarkIII` будет устанавливать закладку, если в этом месте еще не установлена закладка с таким же начальным вариантом имени.

Подпрограмма `AutoBookmark` (листинг 15.5) состоит только из вызова этих процедур и передачи информации от одной процедуры в другую.

Листинг 15.5

```
Sub AutoBookmark()
'Вставка закладки с автоматически сформированным именем
    Dim str0 As String 'начальный вариант имени закладки
    Dim strName As String 'конечный вариант имени закладки
    str0 = AutoBookmarkI() 'формирование начального варианта
    strName = AutoBookmarkII(str0) 'формирование конечного варианта
   'Установить закладку с именем strName
   'если в этом месте нет закладки, начинающейся с str0
    Call AutoBookmarkIII(str0, strName)
End Sub
```

Чтобы можно было проверить, правильно ли работает подпрограмма `AutoBookmark`, напишем так называемую "заглушку" для каждой процедуры. *Заглушка* — это простейший вариант разрабатываемой процедуры. Обращение к заглушке происходит так же, как и к основной программе, она получает и возвращает тот же комплект данных, но на этом сходство и кончается. Данные, возвращаемые заглушкой, должны обеспечить возможность отладки остальных частей программы, но могут не иметь ничего общего с реальностью. После отладки главной подпрограммы заглушки постепенно

заменяются настоящими процедурами. Сейчас вы на конкретном примере познакомитесь, как это все работает.

Такой подход называется *программированием сверху вниз*. При его применении не нужно специально создавать процедуры для тестирования вложенных функций и подпрограмм, а ошибки в процедурах верхнего уровня (практика показывает, что у этих ошибок самые неприятные последствия) обнаруживаются и исправляются в первую очередь.

Итак, напишем процедуры-заглушки (листинг 15.6).

Листинг 15.6

```
Function AutoBookmarkI() As String
'Начальный вариант имени закладки
    AutoBookmarkI = "b_test"
End Function

Function AutoBookmarkII(str0 as String) As String
'Конечный вариант имени закладки
    AutoBookmarkII = str0
End Function

Sub AutoBookmarkIII(str0 As String, strName As String)
'Вставка закладки,
'если на этом месте нет закладки, начинающейся с str0
    ActiveDocument.Bookmarks.Add Name:=strName, Range:=Selection.Range
End Sub
```

То, что делает каждая из процедур-заглушек, вы легко поймете сами.

Чтобы одновременно хранить процедуры-заглушки и реальные процедуры с одинаковыми именами, мы в шаблоне MakrosBook.dot создали для заглушек отдельный модуль: **Listings_15_Stubs** (*stubs* — заглушки). Мы рекомендуем вам в своем шаблоне Normal.dot создать модуль **Bookmarks**, скопировать туда подпрограмму `AutoBookmark` и процедуры-заглушки.

О том, как создавать модуль, мы рассказывали в *разд. 14.1.1*. Как вы знаете *(см. разд. 14.4.3)*, вызывать процедуру можно из любого модуля, но если в "своем" модуле (в том модуле, откуда производится вызов) есть процедура с тем же именем, то будет вызываться процедура из "своего" модуля.

Когда вы проверите, как подпрограмма `AutoBookmark` работает с заглушками, мы рекомендуем по одной заменять заглушки реальными процедурами из модуля **Listings_15** и тестировать их работу.

Итак, проверим, как работает подпрограмма `AutoBookmark`. Этот обязательный этап разработки любых программ программисты называют тестированием. В процессе тестирования необходимо убедиться, что подпрограмма выполняет поставленную задачу. В нашем случае, когда вместо реальных вложенных процедур используются заглушки, задача состоит в том, чтобы установить в точку расположения курсора или на выделенный фрагмент документа закладку **b_test**. Вы, надеемся, помните, как можно проверить, что закладка установлена: для этого следует открыть диалоговое окно **Закладка** (меню **Вставка | Закладка**). Место установки закладки видно на экране, если в диалоговом окне **Параметры** (меню **Сервис | Параметры**) на вкладке **Вид** в группе **Показывать** установлен флажок **Закладки**.

Вы, может быть, думаете, что такую простую программу можно и не проверять, а уж если подпрограмма правильно сработала хотя бы один раз, то дальнейшее тестирование — это пустая трата времени. Однако это далеко не так. Вот и в данном случае при тестировании мы обнаружили, что поставленную задачу эта подпрограмма выполняет не всегда. Если перед запуском подпрограммы выделить фрагмент, содержащий последний в документе знак абзаца, то закладка устанавливается не на весь выделенный фрагмент. Последний знак абзаца никогда не попадает в диапазон, охваченный закладкой. Эта особенность приложения Microsoft Word, к сожалению, не отражена в документации. Немного позже вы узнаете, какие она может иметь последствия для нашего макроса, а пока просто запомним эту особенность.

Приступим к реализации "настоящей" функции `AutoBookmarkI`. Начнем с плана. Составим план в виде последовательности комментариев (листинг 15.7). Запомните этот прием. Он удобен тем, что, во-первых, такой план можно использовать в качестве заглушки, а во-вторых, написанные комментарии, по крайней мере, частично будут использованы в окончательном варианте процедуры. Вариант функции `AutoBookmarkI`, приведенный в листинге 15.7, мы не стали помещать в шаблон MakrosBook.dot, поскольку выполнение его не предполагается.

Листинг 15.7

```
Function AutoBookmarkI() As String
'Начальный вариант имени закладки
 Dim str34 As String, strName As String
    'Если ничего не выделено,
        'Переменной str34 присвоить 34 символа справа от курсора
    'иначе
```

```
        'Переменной str34 присвоить первые 34 символа выделенного текста
    'В переменной strName сформировать начальный вариант имени закладки:
        'Переменной strName присвоить "b_"
        'К переменной strName присоединить строку str34, заменив в ней
        '     все символы, отличные от букв и цифр, на знак подчеркивания
    AutoBookmarkI = strName
End Function
```

Мы понимаем, что у вас недостаточно знаний для реализации этого плана. Не волнуйтесь, сейчас мы вместе с вами заполним этот пробел.

Во-первых, нужно проверить, есть ли в активном окне документа выделенный текст. Для этого можно воспользоваться свойством `Type` (тип) объекта `Selection`. Оно возвращает различные целочисленные значения в зависимости от типа выделения. В Visual Basic для каждого такого значения предусмотрена константа. В частности, константа `wdSelectionNormal` (обычное выделение) означает, что в активном окне документа обычным образом выделен текст, а константа `wdSelectionIP` (*Insertion Point, IP* — точка вставки) — в окне не выделенный текст, а курсор. Предусмотрены также и другие константы, например, константа `wdSelectionRow` означает, что в документе выделена строка таблицы. Обратите внимание на то, что свойство `Type` предназначено "только для чтения", т. е. его можно использовать в выражении, но ему нельзя присваивать значения.

Теперь вам уже должно быть ясно, как записать логическое выражение, принимающее значение **`True`**, когда в активном окне документа отображается не выделение, а курсор:

```
Selection.Type = wdSelectionIP
```

Если это выражение истинно, то переменной `str34` необходимо присвоить 34 символа справа от курсора. Вы уже знаете, что для этого можно выделить эти символы и воспользоваться свойством `Text` объекта `Selection`. Затем, чтобы в результате наших действий в документе все осталось по-старому, снимем выделение, перемесив курсор влево.

```
If Selection.Type = wdSelectionIP Then
    'Если ничего не выделено, выделить 34 символа справа от курсора
    Selection.MoveRight Unit:=wdCharacter, Count:=34, Extend:=wdExtend
    str34 = Selection.Text
    Selection.MoveLeft Unit:=wdCharacter, Count:=1
Else
```

Если в активном окне документа есть выделенный фрагмент, то строковой переменной `str34` потребуется присвоить первые 34 символа выделенного

текста. Чтобы извлечь из этой строки первые 34 символа, воспользуемся функцией `Left` (левый).

У функции `Left` два именованных аргумента: первый аргумент `String` задает строку, из которой извлекаются символы, а второй аргумент `Length` (длина) — количество извлекаемых символов. Возвращается строка, содержащая заданное число символов, находящихся в начале заданной строки.

Для нас важна следующая особенность: если в строке, из которой функция `Left` извлекает символы, символов меньше, чем заданное число, она возвращает всю строку.

Вообще говоря, можно было бы использовать знакомую вам функцию `Mid` *(см. разд. 14.3.1)*, однако мы решили познакомить вас с новой функцией, применение которой повышает наглядность программы (подумайте, как можно заменить функцию `Left` функцией `Mid`).

С помощью функции `Left` можно записать инструкцию, в которой переменной `str34` присваиваются первые 34 символа выделенного текста (или весь выделенный текст, если длина его меньше 34 символов):

```
'Переменной str34 присвоить первые 34 символа выделенного текста
str34 = Left(String:=Selection.Text, Length:=34)
```

Теперь приступим к формированию начального варианта имени закладки. В *разд. 14.3.1* мы уже использовали цикл для перебора всех символов, содержащихся в строке. Там было рассказано, как записать инструкции, которые в строковую переменную `str1` будут по очереди извлекать каждый символ, содержащийся в строке `str34`:

```
For iFor = 1 To Len(str34)
    'Присвоить переменной str1 очередной символ строки str34
    str1 = Mid(String:=str34, Start:=iFor, Length:=1)
```

Надо только не забыть описать числовую переменную цикла `iFor` и строковую переменную `str1`.

Осталось научиться проверять, что символ, содержащийся в переменной `str1`, не является ни буквой, ни цифрой.

Чтобы проверить, содержится ли цифра в переменной `str1`, можно воспользоваться операциями сравнения для строк. Мы уже упоминали, что сравнивать можно не только числа, но и строки. Сравнение основано на том, что для символов установлен определенный порядок сортировки и считается, что каждый следующий символ больше предыдущего. Для нас важно, что в упорядоченной последовательности всех символов цифры от 0 до 9 расположены подряд и по возрастанию.

Отсюда следует, что если, как в нашем случае, `str1` содержит один символ, то условие

```
str1 >= "0" And str1 <= "9"
```

истинно в том и только том случае, когда этот символ является цифрой.

Чтобы проверить, содержится ли буква в переменной `str1`, воспользуемся тем, что в отличие от остальных символов буквы бывают строчными и прописными. В языке Visual Basic предусмотрены встроенные функции `UCase` и `LCase`. У обеих этих функций по одному строковому аргументу `String` (строка), и возвращает каждая из них строку. Функция `UCase` преобразует все буквы строки в прописные, а `LCase` — в строчные. Названия их произошли от английских терминов *Upper Case* (верхний регистр) и *Lower Case* (нижний регистр). Если символ строки не является буквой, то эти функции его не изменяют. Значит, условие

```
UCase(str1) <> LCase(str1)
```

истинно в том и только том случае, когда в строке `str1` содержится буква.

Теперь мы можем написать условие, в котором проверяется, что символ в строке `str1` является буквой или цифрой

```
If (str1 >= "0" And str1 <= "9") Or UCase(str1) <> LCase(str1) Then
```

В зависимости от этого условия к строке `strName` будет добавляться или очередной символ строки `str34`, или знак подчеркивания. Обратите внимание на стандартный прием: перед циклом в строку `strName` заносится начальное значение:

```
strName = "b_"
```

Затем на каждом шаге цикла в строку добавляется по одному символу:

```
strName = strName & str1
```

или

```
strName = strName & "_"
```

Мы рекомендуем вам попытаться самостоятельно закончить программирование функции `AutoBookmarkI`, а затем посмотреть на наш вариант (листинг 15.8).

Листинг 15.8

```
Function AutoBookmarkI() As String
'Начальный вариант имени закладки
 Dim str34 As String, strName As String, str1 As String, iFor
    If Selection.Type = wdSelectionIP Then
```

```
        'Если ничего не выделено, выделить 34 символа справа от курсора
         Selection.MoveRight Unit:=wdCharacter, Count:=34 _
             , Extend:=wdExtend
         str34 = Selection.Text
         Selection.MoveLeft Unit:=wdCharacter, Count:=1
    Else
         'Переменной str34 присвоить первые 34 символа выделенного текста
         str34 = Left(String:=Selection.Text, Length:=34)
    End If
    strName = "b_"
    'К переменной strName присоединить строку str34, заменив в ней
    'все символы, отличные от букв и цифр, на знак подчеркивания
    For iFor = 1 To Len(str34)
         'Присвоить переменной str1 очередной символ строки str34
         str1 = Mid(String:=str34, Start:=iFor, Length:=1)
         If (str1 >= "0" And str1 <= "9") _
                 Or UCase(str1) <> LCase(str1) Then
             'Если в str1 буква или цифра,
             strName = strName & str1
         Else
             strName = strName & "_"
         End If
    Next iFor
    AutoBookmarkI = strName
End Function
```

Теперь все готово для тестирования функции `AutoBookmarkI`. Приведенный в листинге 15.8 вариант можно скопировать из модуля **Listings_15** шаблона MacrosBook.doc в модуль **Bookmarks** шаблона Normal.dot на место одноименной заглушки. Тестирование выполните самостоятельно.

Приступим к программированию функции `AutoBookmarkII`. Вспомним, что должна делать эта функция.

"Если в документе уже есть закладка с таким же именем, в конец имени добавляется подчеркивание и цифра 1 (или 2, 3, и т. д.), чтобы имя закладки было уникальным. В результате получается новый вариант имени закладки".

Реализовать этот алгоритм можно, например, с помощью функции, приведенной в листинге 15.9.

Листинг 15.9

```
Function AutoBookmarkII(str0 As String) As String
'Конечный вариант имени закладки
Dim iCount, strName As String
    strName = str0
    iCount = 0 'обнуление счетчика
    Do While ActiveDocument.Bookmarks.Exists(strName)
        'если strName совпадает с именем существующей закладки
        iCount = iCount + 1 'наращение счетчика
        strName = str0 & "_" & iCount 'изменение strName
    Loop
    AutoBookmarkII = strName
End Function
```

В инструкциях этой функции нет новых для вас элементов языка Visual Basic, однако приемы, использованные в этой процедуре, заслуживают внимания.

Посмотрите, как выполняется наращение счетчика. Это стандартный прием, который вы сможете использовать в своих программах.

Мы хотим также обратить ваше внимание, что в инструкции

```
strName = str0 & "_" & iCount
```

происходит объединение числа со строкой и это вполне допустимая операция, поскольку Visual Basic, когда применяет операцию объединения строк (&), автоматически преобразует число в строку.

Мы рекомендуем вам, прежде чем читать дальше, внимательно проанализировать эту подпрограмму, и если возникнут вопросы, то повторить пройденный ранее материал.

Выполните тестирование подпрограммы `AutoBookmark` с реальными процедурами `AutoBookmarkI` и `AutoBookmarkII` и заглушкой `AutoBookmarkIII`. Надо признаться, что первоначально мы планировали подпрограмму `AutoBookmark` именно в таком виде, но при тестировании обнаружили у нее существенный недостаток. Если запустить ее несколько раз подряд, не меняя положения курсора или выделенного текста, она устанавливает несколько закладок на одно и то же место. Иногда это действительно необходимо, но чаще всего это неоправданно увеличивает число закладок, что затрудняет работу с ними. Мы решили усовершенствовать подпрограмму, чтобы она не добавляла закладку, если в этом месте уже есть закладка с таким же начальным вариантом имени. Для проверки наличия закладки, уста-

новленной в указанном месте, можно использовать один из методов объекта `Range`. Воспользуемся этим поводом, чтобы узнать о некоторых важных возможностях, которые предоставляет нам этот объект.

15.3. Закладки и диапазоны

Начиная с этого раздела, мы будем рассказывать о другом подходе к разработке программ. До сих пор мы старались как можно больше использовать инструкции из автоматически записанных макросов. Теперь мы все больше будем применять средства программирования, которые в автоматически записанных подпрограммах не встречаются. Это позволит вам немного ближе познакомиться с некоторыми инструментами "настоящих" программистов и существенно расширит ваши возможности. Фактически вы постепенно поднимаетесь еще на одну маленькую ступеньку на пути к высотам программирования. Если вам интересно, читайте дальше. Если у вас складывается впечатление, что излагаемый материал имеет специальный характер, в любой момент можете прервать чтение и пользоваться готовыми макросами, не думая о том, как они устроены. Учтите только, что материал этого раздела будет применяться в следующей главе.

Чтобы проверить, есть ли в данном месте закладка, надо сравнить диапазон, на который мы собираемся установить закладку, с диапазонами всех установленных ранее закладок. Такая возможность предусмотрена в Visual Basic и мы сейчас научимся ее использовать, но сначала вспомним об одной особенности Microsoft Word, которую мы обнаружили при тестировании подпрограммы `AutoBookmark` с процедурами-заглушками, приведенными в листинге 15.6.

Речь идет о методе `Add` объекта `Bookmarks`, который используется в инструкции

```
ActiveDocument.Bookmarks.Add Name:=strName, Range:=Selection.Range
```

подпрограммы-заглушки `AutoBookmarkIII`.

При тестировании мы обратили внимание, что если закладка устанавливается на диапазон, охватывающий символ абзаца в конце документа, то в диапазон закладки этот символ абзаца не попадает.

Предположим, что мы выделили диапазон, охватывающий последний символ абзаца, и выполнили подпрограмму-заглушку `AutoBookmarkIII`, чтобы установить на него закладку. Последний символ абзаца в диапазон закладки не попадет. Еще раз выделим тот же диапазон (охватывающий последний символ абзаца) и попытаемся проверить, не установлена ли уже закладка на выделенном диапазоне. Сравнение его с диапазоном установленной закладки, покажет, что диапазоны отличаются (разница в последнем символе аб-

заца). Получается, что таким способом невозможно узнать, устанавливалась ли закладка на выделенный участок текста.

Чтобы обойти возникшее затруднение, будем устанавливать закладку не на все выделение, а на его начальную позицию. Тогда, чтобы узнать, не установлена ли закладка в данном месте, потребуется сравнить начальную позицию выделения с диапазоном установленных таким же образом закладок. При этом наличие в выделенном тексте последнего символа абзаца на результат сравнения уже не повлияет.

Для реализации такого алгоритма необходимо научиться устанавливать закладку на начальную позицию выделенного участка документа.

Если бы мы создавали такой макрос средствами автоматической записи (не прибегая к программированию), то во время записи макроса мы бы нажали клавишу <←>, чтобы снять выделение и установить курсор на его начало, а затем установили бы закладку в эту позицию курсора. Чтобы восстановить выделение, понадобилось бы предварительно установить на него временную закладку, чтобы потом перейти на эту закладку и удалить ее. Мы хотим показать вам, как можно получить такой же результат с помощью "ручного" программирования.

В методе `Add` объекта `Bookmarks`, который мы использовали в инструкции добавления закладки, ее место задается аргументом `Range`. Значение этого аргумента — ссылка на диапазон, т. е. на объект класса `Range`. До сих пор мы в качестве этой ссылки использовали выражение `Selection.Range`, задающее диапазон выделенного текста. Теперь нам требуется ссылаться на диапазон, соответствующий началу выделенного текста.

Во-первых, опишем объектную переменную типа `Range`. В *разд. 15.1* в подпрограмме `DelBookmarks` мы уже описывали объектную переменную типа `Bookmark`. Аналогично можно описать объектную переменную `rng` типа `Range`:

```
Dim rng As Range
```

Далее присвоим этой объектной переменной выделенный диапазон.

Чтобы присвоить значение объектной переменной, недостаточно только написать знак равенства. В начале инструкции *присваивания значения объектной переменной* следует написать ключевое слово **`Set`** (установить) с последующим пробелом.

```
Set rng = Selection.Range
```

После этого воспользуемся методом `Collapse` (схлопывание) объекта `Range`. У этого метода один необязательный числовой аргумент `Direction` (направление), принимающий одно из двух значений, заданных константами `wdCollapseStart` (начало выделения) и `wdCollapseEnd` (конец выделения). Этот метод "схлопывает" диапазон в точку, расположенную в конце диапа-

зона, если значением аргумента `Direction` является константа `wdCollapseEnd`, или в начале диапазона, если значением аргумента `Direction` является константа `wdCollapseStart` или аргумент `Direction` не указан.

Метод `Collapse` можно также применять и к объекту `Selection` (выделение).

Раньше, чтобы снять выделение, мы использовали инструкцию перемещения курсора, но метод `Collapse` удобнее: когда нет выделения, при применении этого метода курсор не перемещается.

Необходимую нам инструкцию можно записать или развернуто

```
rng.Collapse Direction:=wsCollapseStart
```

или кратко

```
rng.Collapse
```

> ***Совет***
>
> Пока вы не выучили наизусть список аргументов, всегда указывайте имя аргумента и не пользуйтесь возможностью задавать значение аргумента по умолчанию. В противном случае читать подпрограмму вам будет сложно. Исключение можно сделать для аргументов, которыми вы никогда не пользуетесь.

Теперь мы будем учиться выяснять, совпадает ли данный диапазон с диапазоном какой-либо из установленных закладок.

Для перебора закладок мы воспользуемся таким же циклом, как в подпрограмме `DelBookmarks` (см. листинг 15.3), где в качестве переменной цикла используется объектная переменная `bkm` типа `Bookmark`, содержащая ссылку на закладку. Чтобы получить диапазон, соответствующий очередной закладке, мы пользуемся известным вам свойством `Range` объекта `Bookmark`. Таким образом, диапазон, соответствующий закладке `bkm`, задается выражением `bkm.Range`.

Теперь необходимо сравнить два диапазона: диапазон выделения `rng` и диапазон очередной закладки `bkm.Range`.

> ***Полезные сведения***
>
> Знаки операций сравнения можно использовать только для сравнения данных (числовых, строковых, логических), но не объектов. У тех объектов, которые можно сравнивать, для этого предусмотрены специальные методы.

Чтобы проверить, совпадают ли два диапазона, проще всего воспользоваться методом `IsEqual` (является равным). Этот метод применяется к объектам `Range` (диапазон) и `Selection` (выделение).

У метода IsEqual один аргумент Range, в котором задается ссылка на объект Range. Метод IsEqual возвращает логическое значение.

Обратите внимание, что перед вами пример метода, возвращающего значение. Чтобы выяснить, совпадают ли диапазоны rng и bkm.Range, можно составить логическое выражение rng.IsEqual(bmp.Range). Это выражение истинно, если диапазоны совпадают, и ложно, если они отличаются.

Далее, если мы обнаружим, что для выделенного участка текста (или курсора, если выделения нет) уже установлена закладка, то посмотрим на ее имя. Если начальный вариант имени устанавливаемой закладки содержится в имени уже установленной в этом месте закладки, то сообщим об этом пользователю, а устанавливать закладку не будем.

Чтобы реализовать этот план, нам необходимо выяснить имя закладки, соответствующей объектной переменной bmp. Запомните, если у объекта есть имя (например, имена есть у документов и окон), то у него наверняка есть свойство Name (имя), содержащее имя этого объекта. Свойство Name есть и у объекта Bookmark (закладка).

Мы специально включили в список аргументов подпрограммы AutoBookmarkIII аргумент str0, содержащий начальный вариант имени устанавливаемой закладки. Проверить, что у закладки bmp начало имени совпадает со значением, содержащимся в str0, как вы, наверное, догадываетесь, можно с помощью логического выражения

```
Left(bmp.Name, Len(str0)) = str0
```

Теперь все, что требуется для создания подпрограммы AutoBookmarkIII, вы знаете. Прежде чем читать дальше, попробуйте написать эту подпрограмму самостоятельно.

Наш вариант приведен в листинге 15.10.

Листинг 15.10

```
Sub AutoBookmarkIII(str0 As String, strName As String)
'Вставка закладки,
'если на этом месте нет закладки с аналогичным именем
Dim bmp As Bookmark, rng As Range
    Set rng = Selection.Range
    rng.Collapse Direction:=wdCollapseStart
    For Each bmp In ActiveDocument.Bookmarks
        If rng.IsEqual(bmp.Range) And _
                    Left(bmp.Name, Len(str0)) = str0 Then
            MsgBox "В этом месте уже установлена закладка """ _
            & bmp.Name & """", vbExclamation, "Закладка не установлена"
```

```
            Exit Sub
        End If
    Next bmp
    ActiveDocument.Bookmarks.Add Name:=strName, Range:=rng
End Sub
```

Если хотите, можете самостоятельно добавить в эту подпрограмму инструкции, которые будут выделять цветом место, на которое установлена закладка, причем цвет может быть фиксированным или задаваться перед выполнением макроса с помощью инструмента "Выделение цветом". Этому вы уже научились, когда изучали *разд. 11.3*. Можно также добавить инструкцию, которая будет в диалоговом окне **Параметры** на вкладке **Вид** устанавливать флажок **Закладки**. Мы уже рассказывали об этом флажке в *разд. 1.2.2*. Когда он установлен, закладки отображаются на экране в квадратных скобках, которые при печати документа не выводятся.

Теперь самостоятельно выполните тестирование подпрограммы `AutoBookmark`.

В начале *разд. 15.2* мы уже говорили о постоянных и временных закладках. Макрос **AutoBookmark** предназначен для установки постоянных закладок, которые не следует удалять после окончания работы над документом. Поскольку все устанавливаемые этим макросом закладки начинаются с сочетания символов "b_", можно усовершенствовать макрос удаления нескрытых закладок таким образом, чтобы он не удалял закладки, начинающиеся с этих символов. Сделайте это самостоятельно. Сравните то, что у вас получится, с нашим вариантом подпрограммы `DelBookmarks2`, приведенным в листинге 15.11.

Листинг 15.11

```
Sub DelBookmarks2()
' Удаление всех нескрытых закладок, кроме тех,
' которые начинаются с "b_"
    Dim bkm As Bookmark
    Dim ReturnValue
    ReturnValue = _
        MsgBox(Prompt:="Будут удалены все нескрытые закладки," _
        & vbCr & "кроме тех, которые начинаются с ""b_""" _
        & vbCr & vbCr & "Продолжать?", Buttons:=vbYesNo + vbQuestion _
        , Title:="Удаление временных закладок")
    If ReturnValue = vbYes Then
        ActiveDocument.Bookmarks.ShowHidden = False
```

```
        For Each bkm In ActiveDocument.Bookmarks
            If Left(bkm.Name, 2) <> "b_" Then
                bkm.Delete
            End If
        Next bkm
    End If
End Sub
```

15.4. Переход из любого места к любой закладке

Название этого раздела у некоторых читателей может вызвать недоумение. Разве сложно перейти к любой закладке, установленной в документе? Для этого в интерфейсе Microsoft Word предусмотрено диалоговое окно **Закладка** (меню **Вставка | Закладка**) и вкладка **Перейти** диалогового окна **Найти и заменить** (меню **Правка | Найти | Перейти | Объект перехода | Закладка**). Кроме того, еще в *гл. 1* наши читатели научились автоматически записывать макрос перехода к определенной закладке, а в *гл. 9* они не только научились с пониманием читать текст этого макроса, но и сами удалили лишние инструкции. Но некоторые из вас помнят, что в *разд. 1.2.2*, когда мы рассказывали о закладках, мы упомянули, о том, что если в документе есть колонтитулы, примечания, сноски или надписи, то не ко всякой закладке можно перейти из любого места в документе. Там же мы пообещали научить решать эту проблему, используя средства Visual Basic. Пришла пора выполнять обещание. Это позволит не только больше узнать о возможностях программирования, но и познакомиться с некоторыми особенностями Microsoft Word. Материал этого раздела используется также в следующей главе.

15.4.1. Недоступные закладки в Microsoft Word

В этой и следующей главе нам потребуется документ, который мы будем использовать в качестве "испытательного полигона". Создать такой документ можно на основе шаблона MacrosBook.dot. Для этого достаточно щелкнуть значок этого шаблона в той папке, в которую вы его загрузили. Откроется документ, основанный на этом шаблоне. Сохраните его под именем debug.doc. В этом документе можно запускать содержащиеся в шаблоне макросы, он содержит колонтитулы, сноски, примечания и надписи, и в каждой из таких областей документа установлены закладки. Если шаблон

MacrosBook.dot вам недоступен, то такой документ вы можете создать самостоятельно. Откройте в Microsoft Word новый документ (сочетание клавиш <Ctrl>+<N>) и сохраните его под любым именем в удобном для вас месте. Введите в документ любой текст и установите закладку в основном тексте документа, затем вставьте в документ две надписи (меню **Вставка | Надпись**, далее мышью нарисуйте прямоугольник и введите текст), в каждой из этих надписей установите закладки. Имена закладкам давайте такие, чтобы по имени можно было определить, где находится закладка, например, **Надпись_1** и **Надпись_2**. Затем добавьте два примечания (меню **Вставка | Примечание**) и по две обычные и концевые сноски (в Microsoft Word 97 и 2000: меню **Вставка | Сноска | Вставить сноску**; в Microsoft Word 2002 и 2003: меню **Вставка | Ссылка | Сноска | Положение**) с соответствующими закладками. Если вы работаете с документом не в обычном режиме, то перейдите в этот режим (**Вид | Обычный**). Откройте область сносок (**Вид | Сноски**). В верхней части области сносок находится поле со списком **Сноски**, в котором можно выбрать ту или иную область сносок. Большинство пользователей Microsoft Word и не подозревает о наличии этих областей, но тем не менее, в каждой из них можно установить закладку.

Несколько сложнее добавить по два верхних и нижних колонтитула. Колонтитулы бывают верхние и нижние, они могут быть разными на четных и нечетных страницах, а также на первой странице. Если требуется, чтобы в каждой главе документа были различные надписи в колонтитулах, следует разбить документ на так называемые разделы и каждую главу поместить в отдельный раздел (в каждом разделе документа может быть свой комплект колонтитулов, это позволяет, например, написать в правых верхних колонтитулах название соответствующей главы).

Итак, разбейте документ на разделы (установите переключатель **со следующей страницы** в группе **Новый раздел** меню **Вставка | Разрыв**). Количество различающихся колонтитулов в каждом разделе зависит от положений переключателей в диалоговом окне **Параметры страницы** на вкладке **Макет** в группе **Различать колонтитулы** (меню **Файл | Параметры страницы**).

После разбиения документа на разделы откройте верхний колонтитул (меню **Вид | Колонтитулы**), введите в нем текст и установите закладку.

Одновременно с верхним колонтитулом появляется панель инструментов **Колонтитулы**, на которой среди прочих имеются кнопки **Как в предыдущем** (), **Верхний/нижний колонтитул** (), **Переход к предыдущему** (), **Переход к следующему** (), Закрыть. Кнопка **Верхний/нижний колонтитул** () открывает нижний колонтитул, а кнопка **Переход к следующему** () — следующий колонтитул. Не удивляйтесь, если при переходе к колонтитулу следующего раздела вы увидите тот же самый колонтитул, который был в пре-

дыдущем разделе, а в правом верхнем углу колонтитула надпись — **Как в предыдущем**. Чтобы колонтитул отличался от соответствующего колонтитула предыдущего раздела, отожмите утопленную кнопку **Как в предыдущем** (). В каждом колонтитуле введите текст и установите закладку. После этого нажмите кнопку  .

Теперь попробуйте перейти на какую-нибудь закладку, находящуюся в колонтитуле. Оказывается, для этого сначала надо перейти в какой-нибудь колонтитул, находящийся в этом же разделе. Если курсор находится не в колонтитуле, то кнопка **Перейти** в диалоговом окне **Закладки** недоступна. Если попытаться из колонтитула первого раздела перейти на закладку, расположенную в колонтитуле второго раздела, то курсор попросту исчезает, что может привести пользователя в полное замешательство.

Сходные проблемы возникают и при работе с закладками, находящимися в надписях, сносках и примечаниях, в чем вы можете также убедиться самостоятельно. К сожалению, в документации об этом не упоминается.

15.4.2. Как перейти на любую закладку

Начнем с того, что исправим очевидный недостаток созданного ранее *(см. разд. 11.2)* макроса **GoToMarker1**: при отсутствии закладки **Marker** он выдает системное сообщение об ошибке (рис. 15.3).

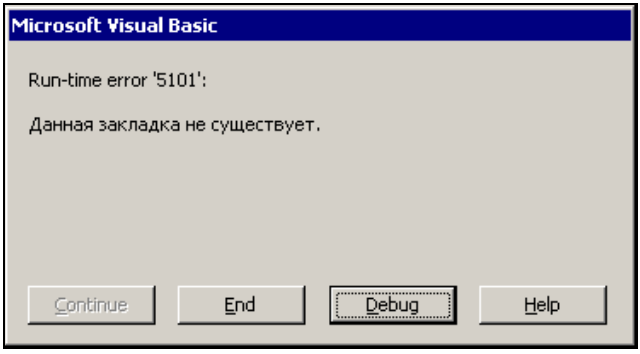


Рис. 15.3. Сообщение при попытке перейти на несуществующую закладку

На этот раз диалоговое окно содержит ясное сообщение об отсутствии закладки, однако в нем содержится также не всем понятная и одновременно лишняя информация, а также три кнопки с надписями **End** (Конец) **Debug** (Отладка) и **Help** (Справка). Неподготовленный пользователь может почув-

ствовать себя "витязем на перепутье". Поэтому, если вы предполагаете, что этой подпрограммой будут пользоваться другие люди, то лучше заранее проверить, существует ли закладка **Marker**, и при ее отсутствии сообщить об этом пользователю. Тем более, что похожая проверка уже есть в макросе **DeleteMarker2** (см. листинг 15.2) и вы уже можете сделать ее самостоятельно. Подпрограмма, решающая эту задачу, приведена в листинге 15.12.

Листинг 15.12

```
Sub GoToMarker2()
' Переход на закладку Marker, если она существует
    If ActiveDocument.Bookmarks.Exists("Marker") = True Then
        Selection.GoTo What:=wdGoToBookmark, Name:="Marker"
    Else
        MsgBox Prompt:="На закладку ""Marker"" перейти не удалось, " _
        & "поскольку такой закладки в документе нет" _
        , Buttons:=vbExclamation, Title:="Переход к закладке ""Marker"""
    End If
End Sub
```

Перейдем к тестированию подпрограммы `GoToMarker2`. Сначала проверьте, как работает эта подпрограмма, когда в документе нет закладки **Marker** и когда закладка **Marker** находится в основном тексте документа. Затем установите закладку **Marker** в каком-либо примечании (меню **Вставка | Примечание**).

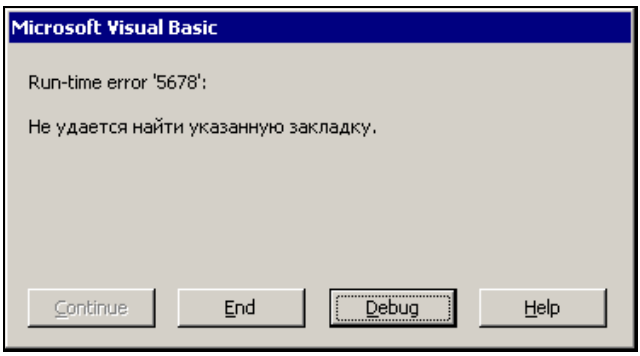


Рис. 15.4. Сообщение при попытке перейти из основного текста к закладке, установленной в колонтитуле

Убедитесь, что, когда перед запуском подпрограммы `GoToMarker2` курсор находится в области примечаний, подпрограмма работает должным образом, а когда курсор находится в области основного текста, то при запуске подпрограммы `GoToMarker2` выводится сообщение, показанное на рис. 15.4.

Теперь установите закладку **Marker** в колонтитуле (меню **Вид | Колонтитулы**). В этом случае перейти на эту закладку, пользуясь макросом **GoToMarker2**, не удается, где бы ни находился курсор. Даже если курсор находится рядом с закладкой. При этом опять выводится сообщение, приведенное на рис. 15.4. Попробуйте перейти к закладке **Marker**, находящейся в колонтитуле, "вручную", не пользуясь макросом. Это получится только в том случае, если курсор находится в каком-либо колонтитуле. Можно попытаться автоматически записать макрос перехода на эту закладку из колонтитула, но действия, которые вы будете выполнять во время записи макроса, созданный макрос не повторяет. Приходится констатировать, что в приложении Microsoft Word допущена ошибка.

К счастью, Visual Basic предоставляет нам альтернативный способ перехода к любой закладке, установленной в документе.

У объекта `Bookmark` (закладка) есть метод `Select` (выделить), который выделяет закладку, т. е. выполняет то же самое действие, которое мы выполняли методом `GoTo` объекта `Selection`. Чтобы воспользоваться этим методом, нам необходимо сослаться на объект `Bookmark`, представляющий закладку **Marker**. Как это сделать, мы рассказывали в *разд. 11.2*, когда создавали макрос **DeleteMarker1.**

Итак, можно записать подпрограмму перехода к закладке **Marker** в виде, представленном в листинге 15.13.

Листинг 15.13

```
Sub GoToMarker3()
' Переход на закладку Marker, если она существует
    If ActiveDocument.Bookmarks.Exists("Marker") = True Then
        ActiveDocument.Bookmarks("Marker").Select
    Else
        MsgBox Prompt:="На закладку ""Marker"" перейти не удалось," _
        & "поскольку такой закладки в документе нет." _
        , Buttons:=vbExclamation _
        , Title:="Переход к закладке ""Marker"""
    End If
End Sub
```

Тестирование этой подпрограммы — нелегкая задача. Наверное, проще всего методом клонирования создать набор макросов перехода к уже установленным вами закладкам в различных областях документа. Для каждой закладки следует создать отдельный макрос (другой способ — поочередно устанавливать закладку **Marker** в различных областях документа и проверять, как работает макрос **GoToMarker3** при разных исходных положениях курсора). Чтобы упростить эту работу, перепишите подпрограмму `GoToMarker3` в виде, удобном для клонирования, т. е. введите переменную, которой присвойте имя закладки, а далее всюду используйте эту переменную вместо имени закладки. Сделайте это самостоятельно. У вас должна получиться примерно такая же подпрограмма, как показанная в листинге 15.14.

Листинг 15.14

```
Sub GoToMarker4()
' Переход на закладку Marker, если она существует
    Dim strName
    strName = "Marker"
    If ActiveDocument.Bookmarks.Exists(strName) = True Then
        ActiveDocument.Bookmarks(strName).Select
    Else
        MsgBox Prompt:="На закладку """ & strName _
        & """ перейти не удалось, " _
        & "поскольку такой закладки в документе нет." _
        , Buttons:=vbExclamation _
        , Title:="Переход к закладке """ & strName & """"
    End If
End Sub
```

Совет

Используйте переменные вместо конкретных значений. Большая часть ошибок в программах делается во время внесения изменений. Разрабатывая программу, всегда рассчитывайте на то, что вы будете ее модифицировать, и предпринимайте меры для облегчения этого процесса. Использование переменной вместо конкретного значения не усложняет программу, но существенно облегчает ее правку, — как говорят, делает программу более гибкой.

Теперь вы быстро создадите нужное число макросов и сможете проверить, что при переходе к закладкам сообщения об ошибках не возникают. Однако когда закладка установлена в колонтитуле одного раздела документа, а переход к ней осуществляется из колонтитула другого раздела, курсор исчезает

точно так же, как и при попытке выполнить эту операцию без применения макросов (с помощью диалогового окна **Закладка**).

Чтобы найти способ, как обойти эту неприятную особенность Microsoft Word, продолжим тестирование. С помощью макроса удается перейти к закладке, установленной в колонтитуле, из другой части текста, т. е. не из колонтитулов. После такого перехода к закладке область колонтитулов отображается в режиме **Обычный**, что вполне приемлемо, но непривычно, поскольку при отображении колонтитулов стандартным способом (меню **Вид | Колонтитулы**) происходит переход в режим **Разметка страницы**. Однако при отображении колонтитула в режиме **Обычный** удается перейти из колонтитула одного раздела в колонтитул другого раздела, причем это можно сделать как с помощью нашего макроса, так и с помощью диалогового окна **Закладка**. Это вселяет в нас надежду на успех. Обратите внимание, что при выполнении этой операции с использованием версий Microsoft Word 97 или Microsoft Word 2000 надпись в верхней части области колонтитула, содержащая сведения о типе колонтитула и номере раздела, остается прежней, а не изменяется должным образом при переходе к другому колонтитулу, что может дезинформировать пользователя.

Эти наблюдения позволяют предложить способ решения нашей задачи. Необходимо перед переходом к закладке перейти в режим **Обычный**. Это обеспечивает переход из колонтитула одного раздела в колонтитул другого раздела. Чтобы сведения о колонтитуле отображались правильно и в том случае, когда макрос запускается в режиме **Обычный**, достаточно предварительно перейти в режим **Разметка страницы**. Однако если и закладка, и курсор (или закладка и выделение) перед выполнением макроса находятся в основном документе, или в одном колонтитуле, или в одной надписи, т. е. когда изменение режима отображения для правильной работы макроса не требуется, лучше этот режим не изменять. Иначе при переходе к расположенной в той же области документа закладке будет происходить неожиданное для пользователя изменение режима.

Для реализации этого плана вам потребуется узнать о некоторых важных возможностях Visual Basic.

Начнем с перехода в режимы **Разметка страницы** и **Обычный**.

Режим отображения документа определяется свойством `Type` (тип) объекта `View` (вид). Это свойство принимает целочисленные значения, для которых предусмотрены соответствующие константы. Режиму **Обычный** соответствует константа `wdNormalView` (обычный вид), а режиму **Разметка страницы** — `wdPrintView` (вид печати). Значения свойства `Type` можно использовать в выражениях, а присваивание этому свойству определенных констант приводит к соответствующему изменению режима отображения документа. Для ссылки на объект `View` можно использовать свойство `View` (вид) объекта `Window` (окно). Аналогично тому, как объект `Document` представляет откры-

тый документ, объект `Window` представляет открытое окно. Возможно, не все читатели знают, что один и тот же документ можно открыть в нескольких окнах (меню **Окно | Новое**) и, следовательно, открытых окон может быть больше, чем открытых документов. Аналогично семейству `Documents` (документы), представляющему все открытые документы, существует семейство `Windows` (окна), представляющее все открытые окна. Аналогично глобальному свойству `ActiveDocument` (активный документ), позволяющему сослаться на активный документ, существует глобальное свойство `ActiveWindow` (активное окно), позволяющее сослаться на активное окно, а также глобальное свойство `Windows`, позволяющее ссылаться на семейство всех открытых окон.

Мы не ставим перед собой цели подробно рассказать об этих объектах и свойствах. Нам необходимо только, чтобы вы могли с пониманием прочесть (а при необходимости и написать) инструкции перехода в режимы **Разметка страницы** и **Обычный**:

```
ActiveWindow.View.Type = wdPrintView
ActiveWindow.View.Type = wdNormalView
```

Эти инструкции предполагается выполнить перед переходом к закладке, но только в том случае, если закладка и курсор (или выделения) не находятся в основном тексте документа, одном колонтитуле, одной надписи и т. п. Чтобы однозначно сформулировать это утверждение, нам потребуется познакомить вас с понятием "цепочки".

Возможно вы замечали, что команда **Выделить все** (меню **Правка**) выделяет не весь документ, а только некоторую его область. Например, основной текст и колонтитулы одновременно выделить не удается (это обязательно надо учитывать при копировании текста из одного документа в другой, иначе можно потерять колонтитулы).

Такие части документа, которые можно выделить так, что расширить это выделение уже нельзя, называются *цепочками* (в англоязычных версиях они называются *story*).

В русских переводах это название произошло от цепочки связанных надписей, но затем цепочками стали называть и другие части документа, такие как колонтитулы, основной текст, примечания и сноски. На цепочках связанных надписей мы подробно останавливаться не будем, о них можно прочесть во встроенной справке Microsoft Word, отметим только, что называются они так потому, что отдельные надписи можно связать в одну цепочку таким образом, чтобы при заполнении их текстом он "перетекал" из одной надписи в другую.

Команда **Выделить все** выделяет одну не связанную с другими надпись, все связанные в одну цепочку надписи, все сноски или все примечания (в зависимости от того, где находится курсор), но если в документе есть несколько

колонтитулов, то эта команда выделяет только один их них. Таким образом, каждый из колонтитулов является отдельной цепочкой.

> ***Примечание***
>
> Начиная с версии Microsoft Word 2002, примечания располагаются на выносках или вместе с исправлениями в окне просмотра и командой **Выделить все** выделяется только одна выноска или же все окно просмотра. Тем не менее, совокупность всех примечаний, так же как и в предыдущих версиях, считается цепочкой.

В Visual Basic нет специального объекта, представляющего цепочки. Для их представления используется объект `Range` (диапазон). Следовательно, цепочка соответствует некоторому диапазону в документе. От других диапазонов цепочки отличаются тем, что, во-первых, их нельзя изменить, не изменяя документ, а во-вторых, любой другой диапазон целиком содержится в какой-нибудь цепочке.

Итак, нам необходимо проверить, находятся ли закладка и курсор (или закладка и выделение) в одной цепочке. Воспользуемся тем, что у объекта `Selection` (выделение) есть метод `InStory` (в цепочке), позволяющий выяснить, находится ли курсор (или выделение) в цепочке, содержащей указанный диапазон. У этого метода один аргумент `Range` (диапазон), в котором задается ссылка на диапазон, а возвращает метод значение **`True`** или **`False`** в зависимости от того, содержится ли выделенный диапазон (или курсор) в цепочке, включающей указанный диапазон.

Оформим процедуру перехода к закладке в виде функции `GoToBookmark` *(GoToBookmark* — переход к закладке), аргументом которой является имя закладки (листинг 15.15). Эта функция должна возвращать **`True`** или **`False`** в зависимости от того, существует ли в документе закладка с данным именем или нет. Этой функцией мы будем пользоваться и в следующей главе.

Листинг 15.15

```
Function GoToBookmark(strName As String) As Boolean
'Переход к закладке с именем strName
'Возвращает False, если закладки с таким именем не существует
With ActiveDocument
    If .Bookmarks.Exists(strName) = True Then
        If Not Selection.InStory(.Bookmarks(strName).Range) Then
            ActiveWindow.View.Type = wdPrintView
            ActiveWindow.View.Type = wdNormalView
        End If
```

```
        .Bookmarks(strName).Select
        GoToBookmark = True
    Else
        GoToBookmark = False
    End If
End With
End Function
```

Макрос для перехода к закладке **Marker** реализуется в виде подпрограммы `GoToMarker5` (листинг 15.16), в которой происходит обращение к функции `GoToBookmark`.

Листинг 15.16

```
Sub GoToMarker5()
' Переход на закладку Marker, если она существует
'
    Dim strName As String
    strName = "Marker"
    If GoToBookmark(strName) = False Then
        MsgBox Prompt:="На закладку """ & strName _
            & """ перейти не удалось, " _
        & "поскольку такой закладки в документе нет" _
        , Buttons:=vbExclamation, _
        Title:="Переход к закладке """ & strName & """ "
    End If
End Sub
```

Тестирование макроса **GoToMarker5** проведите самостоятельно. Из него методом клонирования вы сможете быстро и просто создать надежный макрос перехода к любой закладке.

15.5. Дополнительные сведения о функции *MsgBox*

В этой книге, как правило, мы не рассказываем подробно об аргументах встроенных функций Visual Basic. Заинтересованного читателя мы отсылаем к "толстым" книгам, а также к справке Visual Basic. Для функции `MsgBox` мы

решили сделать исключение. Мы сочли, что многим нашим читателям будет удобно иметь под рукой краткое описание этой функции.

С помощью этой функции можно не только вывести на экран результаты работы программы, но и организовать взаимодействие пользователя с программой. Она очень часто применяется в подпрограммах.

При вызове функции `MsgBox` на экран выводится диалоговое окно с сообщением, и приложение переходит в состояние ожидания, которое заканчивается, когда пользователь нажимает одну из кнопок диалогового окна. Функция возвращает число, указывающее, какая была нажата кнопка. Аргументы функции `MsgBox`, часто используемые значения аргумента `Buttons` и значения, возвращаемые функцией, приведены в табл. 15.1—15.3.

Таблица 15.1. *Именованные аргументы функции* `MsgBox`

Имя аргумента	Описание
`Prompt` (запрос)	Обязательный аргумент. Строковое выражение, отображаемое в качестве сообщения в диалоговом окне. Максимальная длина значения аргумента примерно равна 1024 символам. Она зависит от ширины используемых символов. Если же сообщение требуется разбить на несколько строк, то в него можно включить символ перевода строки (`vbCr`)
`Buttons` (кнопки)	Необязательный аргумент. Числовое выражение, представляющее собой сумму значений, задающих набор используемых кнопок, отображаемый в диалоговом окне значок, используемую по умолчанию кнопку и другие свойства диалогового окна (конкретные значения приведены ниже). Если этот аргумент опущен, то используется значение по умолчанию, равное 0
`Title` (заголовок)	Необязательный аргумент. Строковое выражение, значение которого отображается в заголовке диалогового окна. Если этот аргумент опущен, в заголовке отображается название приложения, например, Microsoft Word
`Helpfile` (файл справки) и `Context` (контекст)	Необязательные аргументы. Используются, когда необходимо связать с диалоговым окном собственную справочную систему

Таблица 15.2. *Часто используемые значения аргумента* `Buttons`

Константа	Значение	Описание
Первая группа значений		
`vbOKOnly` (только OK)	0	Только кнопка **OK**
`vbOKCancel` (OK и Отмена)	1	Кнопки **OK** и **Отмена**

Таблица 15.2 *(окончание)*

Константа	Значение	Описание
Первая группа значений		
`vbAbortRetryIgnore` (Прервать, Повтор и Пропустить)	2	Кнопки **Прервать**, **Повтор** и **Пропустить**
`vbYesNoCancel` (Да, Нет и Отмена)	3	Кнопки **Да**, **Нет** и **Отмена**
`vbYesNo` (Да и Нет)	4	Кнопки **Да** и **Нет**
`vbRetryCancel` (Повтор и Отмена)	5	Кнопки **Повтор** и **Отмена**
Вторая группа значений		
`vbCritical` (критическая ошибка)	16	Значок **Критическая ошибка**
`vbQuestion` (вопрос)	32	Значок **Вопрос**
`vbExclamation` (восклицательный знак)	48	Значок **Восклицательный знак**
`vbInformation` (информация)	64	Значок **Информация**
Третья группа значений		
`vbDefaultButton1` (кнопка по умолчанию — первая)	0	По умолчанию используется первая кнопка
`vbDefaultButton2` (кнопка по умолчанию — вторая)	256	По умолчанию используется вторая кнопка
`vbDefaultButton3` (кнопка по умолчанию — третья)	512	По умолчанию используется третья кнопка

При формировании значения аргумента `Buttons` складываются числа, соответствующие различным значениям. Из каждой группы значений может использоваться только одно число.

Таблица 15.3. *Возвращаемые значения*

Константа	Значение	Нажатая кнопка
`vbOK`	1	**OK**
`vbCancel`	2	**Отмена**
`vbAbort`	3	**Прервать**

Таблица 15.3 *(окончание)*

Константа	Значение	Нажатая кнопка
vbRetry	4	**Повтор**
vbIgnore	5	**Пропустить**
vbYes	6	**Да**
vbNo	7	**Нет**

Примечание

При нажатии клавиши <Enter> или клавиши ввода пробела возвращается значение, соответствующее кнопке, используемой по умолчанию. Если в диалоговом окне только одна кнопка **ОК**, то при любом способе закрытия диалогового окна возвращается одно и то же значение. Когда в диалоговом окне есть кнопка **Отмена**, то его закрытие без нажатия кнопок (клавишей <Esc>, нажатием кнопки с изображением крестика в правом верхнем углу окна, сочетанием клавиш <Alt>+<F4> или с помощью пункта меню, вызываемого нажатием сочетания клавиш <Alt> и клавиши ввода пробела) эквивалентно нажатию кнопки **Отмена**. Если в диалоговом окне нет ни кнопки **ОК**, ни кнопки **Отмена**, то кнопка с изображением крестика в правом верхнем углу диалогового окна неактивна, и закрыть диалоговое окно можно только нажатием одной из кнопок.

15.6. Что нового в этой главе

В этой главе мы выполняем свои обещания и совершенствуем простые макросы, рассмотренные в первой части книги, в которых удаляются закладки, устанавливаются закладки и выполняется переход к установленной закладке. При этом широко применяются средства программирования, о которых рассказывалось в предыдущих главах (переменные, условные переходы, циклы, встроенные и пользовательские процедуры). Рассказывается о проверке наличия закладки с данным именем, об описании объектных переменных и о присвоении значений таким переменным. Читатели знакомятся с инструкцией **`For Each`**, предназначенной для организации цикла по всем элементам семейства, со скрытыми закладками и инструкциями, управляющими отображением и удалением таких закладок. Приводятся сведения о функции `MsgBox`, позволяющие использовать ее для выбора вариантов действий, предлагаемых в сообщениях, а подробное описание этой функции дается в дополнительных сведениях в конце главы. Рассказывается о применении закладок для перекрестных ссылок и о задаче установки закладок с автоматически формируемым именем. Разрабатывая макрос, решающий эту задачу,

читатели знакомятся с методикой программирования "сверху вниз" и о разрешении конфликтов имен при модульном программировании. Они учатся проверять наличие выделенного фрагмента и узнают о встроенной функции `Left`, используемой для извлечения заданного числа начальных символов строки. Чтобы выяснить, относится ли символ к цифрам или буквам, используется операция сравнения строк и применяются встроенные функции, изменяющие регистр букв. Рассказывается, как сформировать строку, добавляя в цикле по одному символу. Затем демонстрируется, как организовать счетчик и как добавить число к строке. На практических примерах продолжается изучение свойств и методов объекта `Range`, рассказывается, как можно сравнивать диапазоны, соответствующие выделениям и закладкам. Отдельный раздел посвящен проблеме перехода к закладкам в документах, содержащих такие элементы, как колонтитулы, сноски, примечания и надписи. В процессе написания макроса, позволяющего решить эту проблему, читатели знакомятся с новыми объектами и методами и начинают осваивать понятие цепочки.

В заключение приведем традиционную сводку элементов языка Visual Basic (табл. 15.4—15.7), о которых идет речь в этой главе (кроме раздела с дополнительными сведениями о функции `MsgBox`).

Таблица 15.4. *Новые инструкции*

For Each объектная_переменная **In** семейство
Next объектная_переменная
Set объектная_переменная = ссылка_на_объект

Таблица 15.5. *Новые функции*

Функция	Значение	Аргумент	Константа
MsgBox	vbYes	Buttons	vbQuestion
	vbNo		vbYesNo
Left		String	
		Length	
UCase		String	
LCase		String	

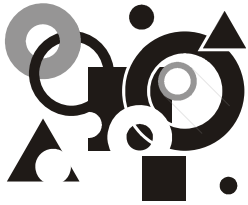
Таблица 15.6. *Новые методы*

Объект	Метод	Аргумент	Константа
Bookmarks	Exists	Name	
Range	Collapse	Direction	wdCollapseStart
Selection			wdCollapseEnd
Range	IsEqual	Range	
Selection			
Bookmark	Select		
Selection	InStory	Range	

Таблица 15.7. *Новые объекты и свойства*

Объект	Свойство	Константа
Selection	Type	wdSelectionNormal
		wdSelectionIP
		wdSelectionRow
Bookmark	Name	
View	Type	wdNormalView
		wdPrintView
Window	View	
Windows		
Global	ActiveWindow	

Глава 16



Поиск в колонтитулах, сносках, примечаниях и надписях

Программирование более сложных задач: поиск в документах с колонтитулами, примечаниями, сносками и надписями. Практическое применение приобретенных ранее навыков и новые способы работы с диапазонами, объектными переменными и цепочками.

Возможность поиска — одно из важнейших преимуществ электронных документов, и неудивительно, что мы уделяем столько внимания задачам, связанным с поиском. В *гл. 4* мы подробно рассказали о возможностях поиска и замены, предусмотренных в интерфейсе приложения Microsoft Word, в *гл. 12* детально описали инструкции, используемые редактором Visual Basic при автоматической записи макросов, содержащих операции поиска и замены. При этом мы уже обращали ваше внимание на то, что поиск реализован в Microsoft Word небезупречно. При работе с документами, содержащими надписи, колонтитулы, сноски или примечания, при поиске возникают серьезные проблемы. Если в таких документах пытаться автоматически записывать макросы с поиском и заменой, то проблем становится еще больше. С похожей ситуацией мы столкнулись, когда пытались в таких документах работать с закладками. В *гл. 15* все проблемы, связанные с закладками, удалось преодолеть с помощью программирования. На очереди — задачи поиска.

Для решения этих задач вам потребуется освоить относительно немного нового материала, основная цель, которую мы преследовали в этой главе, — предоставить вам возможность попрактиковаться в применении уже имеющихся навыков, причем для решения практически важных задач.

Эта глава предназначена в первую очередь тем, кто всерьез заинтересовался программированием и хочет узнать, как решаются задачи, более похожие на "настоящие", чем те, которые рассматривались до сих пор. Если вы заинтересованы только в применении описанных в этой главе макросов, то вам будет достаточно познакомиться лишь с постановкой задач, которые решаются в этой главе.

16.1. Ограничения возможностей поиска в Microsoft Word

Мы уже упоминали, что поиск в Microsoft Word работает не всегда и не везде. Представьте себе, что вы получили от редактора документ, в котором он выделил зеленым цветом самые удачные, на его взгляд, места, голубым — то, на что следует обратить внимание при верстке, а желтым цветом — фрагменты, которые, по его мнению, требуют литературной правки. Вам требуется найти и поправить фрагменты, выделенные желтым цветом. Документ большой и содержит сноски, колонтитулы и надписи.

Вы знаете, что в Microsoft Word предусмотрена возможность поиска выделенного цветом текста. К сожалению, искать только те фрагменты, которые выделены желтым цветом, нельзя, но можно сэкономить время, если, используя функцию поиска, пройти по всем выделенным цветом фрагментам, останавливаясь для просмотра и исправления только на фрагментах желтого цвета. В *гл. 4* мы даже записали специальный макрос **FindColor** (найти цвет), помогающий выполнить этот поиск.

Проблема в том, что и это решение не всегда можно реализовать. Убедитесь в этом сами. Когда вы изучали предыдущую главу, вы использовали специальный документ, содержащий сноски, примечания и колонтитулы *(см. разд. 15.4.1)*. Откройте его и выделите желтым и красным цветом по два фрагмента в основном тексте, в верхнем колонтитуле и в нижнем колонтитуле.

Поместите курсор в начало основного текста документа и выполните поиск выделенного текста. Для этого выполните следующие действия:

1. В окне **Найти и заменить** (меню **Правка | Найти**) перейдите на вкладку **Найти**.
2. Если кнопка **Формат** не отображается, нажмите кнопку **Больше**.
3. Нажмите кнопку **Формат** и в раскрывшемся списке выберите **Выделение цветом**.
4. Нажмите несколько раз кнопку **Найти далее**.

На первый взгляд все работает должным образом: после каждого нажатия кнопки **Найти далее** происходит переход к следующему выделенному цветом фрагменту текста. Однако попробуйте при просмотре верхнего колонтитула внести в нем исправление. Достаточно просто щелкнуть в нем кнопкой мыши, чтобы увидеть, каким цветом выделен фрагмент (когда Microsoft Word находит выделенный цветом фрагмент, он его выделяет, используя инверсный цвет, поэтому первоначальный цвет выделения фрагмента не виден). Если после этого продолжить поиск, то обнаруживается, что теперь

поиск производится только в верхнем колонтитуле, по завершении просмотра верхнего колонтитула выдается сообщение, приведенное на рис. 16.1.

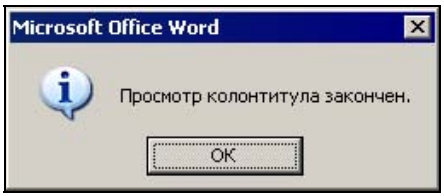


Рис. 16.1. Сообщение, выводящееся при отрицательном результате поиска в колонтитуле

Теперь необходимо самостоятельно перейти в другой колонтитул и там опять запустить поиск. Так нужно пройти по всем колонтитулам, примечаниям, сноскам и надписям (*надпись* — это графический объект для размещения текста). В большом документе, содержащем много колонтитулов и надписей, это может оказаться непростой задачей.

Можно также вернуться в основной текст и начать искать сначала, не останавливаясь на уже просмотренных участках текста, но после правки придется опять повторять поиск.

Это явная недоработка приложения Microsoft Word, и даже появление (начиная с версии Microsoft Word 2002) в окне поиска нового флажка **Выделить все элементы, найденные в** и кнопки **Найти все** не является полноценным решением проблемы, хотя и помогает пройти по всем частям документа.

16.2. Ограничения возможностей поиска в Visual Basic

В *гл. 12*, когда мы обсуждали средства программирования поиска, мы заметили, что автоматически записанный макрос, осуществляющий поиск, не выполняет в полном объеме тех действий, которые выполнялись при его записи. Сейчас мы обсудим это более подробно. Начнем с того, что еще раз запишем макрос **FindColor** (найти цвет) для поиска следующего вхождения выделенного цветом текста.

Перед записью макроса выполните следующие действия:

1. Создайте новый документ (сочетание клавиш <Ctrl>+<N>).
2. Откройте в нем верхний колонтитул (меню **Вид | Колонтитулы**).
3. Введите в колонтитул какой-нибудь текст (например, "Проверка").

4. Выделите его бирюзовым цветом (сначала выделите его мышью, а затем на панели инструментов **Форматирование** нажмите стрелку рядом с кнопкой **Выделение цветом** ). Закройте колонтитул (дважды щелкните основной текст документа, т. е. область документа вне колонтитула).

Во время записи макроса выполните следующие действия:

1. В окне **Найти и заменить** (меню **Правка | Найти**) перейдите на вкладку **Найти**.
2. Если кнопка **Формат** не отображается, то нажмите кнопку **Больше**.
3. Нажмите кнопку **Формат** и в раскрывшемся списке выберите **Выделение цветом**.
4. Нажмите кнопку **Найти далее**.

   Обратите внимание, что открылась новая область документа **Верхний колонтитул**, в которой белым шрифтом на красном фоне выделился текст.

После завершения записи макроса закройте область документа **Верхний колонтитул** (нажмите кнопку **Закрыть** в заголовке этой области) и выполните записанный макрос. В результате действия макроса ничего не происходит. Макрос не работает. Чтобы проверить, что вы все сделали правильно, откройте верхний колонтитул и запустите макрос. В этом случае выделенный цветом текст макрос успешно находит.

Не надо думать, что это ошибка. Такое поведение программ Visual Basic предусмотрено разработчиками. Чтобы объяснить, что мы имеем в виду, нам потребуется понятие цепочки, которое мы ввели в *разд. 15.4.2* (перед листингом 15.15).

Напомним, что цепочками называются части документа, которые можно выделить так, что расширить это выделение уже нельзя. Цепочкой является основной текст документа, каждый из колонтитулов, совокупность всех примечаний, каждая не связанная в цепочку надпись, каждая цепочка связанных надписей, совокупность концевых сносок и т. д.

Теперь посмотрите на автоматически записанную подпрограмму `HighlightNext`, приведенную в листинге 16.1.

Листинг 16.1

```
Sub FindColor()
'
' Поиск следующего выделенного цветом фрагмента
'
    Selection.Find.ClearFormatting
    Selection.Find.Highlight = True
```

```
    With Selection.Find
        .Text = ""
        .Replacement.Text = ""
        .Forward = True
        .Wrap = wdFindContinue
        .Format = True
        .MatchCase = False
        .MatchWholeWord = False
        .MatchWildcards = False
        .MatchSoundsLike = False
        .MatchAllWordForms = False
    End With
    Selection.Find.Execute
End Sub
```

Все инструкции, использованные в этой подпрограмме, вам уже знакомы *(см. гл. 12)*. Сначала устанавливаются параметры поиска — свойства объекта `Find` (найти), — а в заключение выполняется инструкция, с помощью которой осуществляется поиск, — к объекту `Find` применяется метод `Execute` (выполнить).

Теперь мы можем сформулировать правило, которое ограничивает возможности поиска в Visual Basic: при применении метода `Execute` к объекту `Find` поиск производится только в той цепочке, в которой содержится объект `Selection`.

Это ограничение не позволяет макросу **FindColor** повторить поиск, который выполнялся при его записи.

Такое правило в документации явно не сформулировано, однако разработчики Visual Basic об этом знают, поскольку предусмотрели средства, позволяющие обойти это ограничение и выполнять поиск во всем документе. Сейчас мы приступаем к изучению этих средств.

16.3. Перебираем цепочки

Напомним, что не существует специального объекта, представляющего цепочки. Для их представления используется объект `Range` (диапазон). От других диапазонов цепочки отличаются тем, что их нельзя расширить, т. е. любой другой диапазон целиком содержится в какой-нибудь цепочке.

Поскольку с каждым выделением и закладкой связан некоторый диапазон, то они тоже могут располагаться только внутри одной из цепочек.

Для каждого диапазона (а также выделения или закладки) можно выяснить тип цепочки, которой он принадлежит, например, верхнему колонтитулу, основному документу или концевой сноске. Для этого предусмотрено свойство `StoryType` (тип цепочки) объекта `Range`. Этим свойством обладают также объекты `Selection` и `Bookmark`.

Значениями этого свойства являются целые числа, для которых предусмотрены встроенные константы. Каждому типу цепочки соответствует числовая константа, имеющая определенное имя согласно табл. 16.1.

Таблица 16.1. *Типы цепочек*

Число	Константа	Тип цепочки
1	`wdMainTextStory`	Основной документ
2	`wdFootnotesStory`	Сноски
3	`wdEndnotesStory`	Концевые сноски
4	`wdCommentsStory`	Примечания
5	**`wdTextFrameStory`**	**Несвязанная надпись или цепочка связанных надписей**
6	**`wdEvenPagesHeaderStory`**	**Верхний колонтитул четных страниц**
7	**`wdPrimaryHeaderStory`**	**Основной верхний колонтитул**
8	**`wdEvenPagesFooterStory`**	**Нижний колонтитул четных страниц**
9	**`wdPrimaryFooterStory`**	**Основной нижний колонтитул**
10	**`wdFirstPageHeaderStory`**	**Верхний колонтитул первой страницы**
11	**`wdFirstPageFooterStory`**	**Нижний колонтитул первой страницы**
12	`wdFootnoteSeparatorStory`	Разделитель сносок
13	`wdFootnoteContinuation-SeparatorStory`	Разделитель продолжения сносок
14	`wdFootnoteContinuation-NoticeStory`	Уведомление о продолжении сносок
15	`wdEndnoteSeparatorStory`	Разделитель концевых сносок
16	`wdEndnoteContinuationSeparatorStory`	Разделитель продолжения концевых сносок
17	`wdEndnoteContinuationNoticeStory`	Уведомление о продолжении концевых сносок

В Word 97 и Word 2000 в семейство `StoryType` входило 11 типов цепочек (первые 11 строк таблицы), а в версиях Word 2002 и 2003 разработчики учли, что есть и другие цепочки, и число типов цепочек стало 17.

Обратите внимание, что строки в табл. 16.1 для некоторых типов цепочек выделены полужирным шрифтом. В документе может быть несколько цепочек каждого из этих типов. Цепочек, тип которых указан в остальных строках таблицы, в документе не может быть больше одной. Например, в документе может быть несколько несвязанных надписей и несколько цепочек связанных надписей. Каждая несвязанная надпись и каждая цепочка связанных надписей представляет собой отдельную цепочку, но все они принадлежат одному типу. В то же время все примечания составляют одну цепочку и в документе может быть только одна цепочка этого типа.

Еще один пример. В каждом разделе документа свой набор колонтитулов, и каждый колонтитул является цепочкой. Например, можно создать документ с тремя разделами и в каждом разделе создать свой верхний колонтитул. Тогда в этом документе будет три цепочки, тип которых "основной верхний колонтитул".

Это важно, потому что в Visual Basic нет семейства, состоящего из всех цепочек документа. В Visual Basic есть другое состоящее из цепочек семейство `StoryRanges` (диапазоны цепочек), но в него входят не все цепочки, а только по одной цепочке каждого типа из тех, что есть в документе (к сожалению, явно об этом в справке не сказано). Другими словами, в это семейство входит по одному представителю от каждого типа цепочек. Число элементов этого семейства равно числу типов цепочек, имеющихся в документе. Сослаться на это семейство можно с помощью свойства `StoryRanges` объекта `Document`. Например, ссылка на семейство `StoryRanges` активного документа записывается следующим образом:

```
ActiveDocument.StoryRanges
```

> ***Полезные сведения***
>
> Отличительный признак семейства — буква `s` на конце слова. В английском языке для обозначения множественного числа существительных в конце слова добавляется буква s. Поэтому буква `s` на конце слова позволяет отличить имя семейства от имени элемента этого семейства. Например, `Bookmarks` и `Bookmark`, `Documents` и `Document`. Однако не все объекты, имя которых заканчивается буквой `s`, являются семействами: объект `Options` (параметры) не является семейством.

Семейство `StoryRanges` отличается от известных нам семейств. Во-первых, элементами этого семейства являются уже известные нам объекты `Range`, специального объекта для элементов этого семейства не существует. Во-вторых, число элементов этого семейства не может быть больше, чем число возможных типов цепочек (11 или 17 в зависимости от версии Microsoft

Word). В-третьих, чтобы обратиться к отдельному элементу семейства, в качестве индекса требуется указать не порядковый номер элемента семейства, а число (или именованную константу), соответствующее типу цепочки согласно табл. 16.1. Например, если в документе есть колонтитулы, но нет сносок, то использование выражения `ActiveDocument.StoryRanges(2)`, которое представляет собой ссылку на сноски (`wdFootnotesStory` = 2, см. табл. 16.1), приведет к сообщению об ошибке, несмотря на то, что в документе больше одного типа цепочек.

Чтобы попрактиковаться в применении семейства `StoryRanges`, напишем подпрограмму, которая будет сообщать нам, сколько в документе типов цепочек (листинг 16.2).

Листинг 16.2

```
Sub StoryTypeCount()
' Подсчет типов цепочек в документе
'
    Dim rng As Range, i
    i = 0
    For Each rng In ActiveDocument.StoryRanges
        i = i + 1
    Next rng
    MsgBox Prompt:=i, Title:="Количество типов цепочек в документе"
End Sub
```

Убедитесь, что в листинге этой подпрограммы вам все понятно. Если остались вопросы, то сейчас самое время повторить пройденный материал, потому что приведенные здесь инструкции будут применяться и дальше.

Полезные сведения

Чтобы узнать, сколько элементов в семействе, не обязательно организовывать цикл по всем элементам семейства. Для этой цели предусмотрено свойство `Count` (количество), которое есть у каждого семейства. Например, выражение `ActiveDocument.StoryRanges.Count` возвращает количество типов цепочек в документе.

Однако нам требуется научиться перебрать все цепочки, а не все типы цепочек. Таким образом, для тех типов цепочек, которые выделены в таблице полужирным шрифтом (цепочек надписей и разного вида колонтитулов), надо перебрать все цепочки данного типа. Специально для этой цели в Visual Basic предусмотрено свойство `NextStoryRange` (следующая цепочка) объекта `Range` (диапазон).

Это свойство "только для чтения", ему нельзя присваивать значений.

Если у некоторой цепочки есть следующая цепочка того же типа, то свойство `NextStoryRange` возвращает объект `Range`, представляющий эту следующую цепочку. Если следующей цепочки того же типа нет (иными словами, цепочка является последней или единственной цепочкой данного типа), то свойство `NextStoryRange` возвращает *пустой объект*. Пустой объект обозначается специальным ключевым словом **`Nothing`** (ничего). У пустого объекта нет ни свойств, ни методов, ни имени.

Когда говорят "следующая цепочка", подразумевается, что цепочки каждого типа упорядочены. Надписи следуют в том порядке, в котором они создавались, а номер колонтитула определяется порядком, в котором расположены соответствующие разделы документа. При этом элементами семейства `StoryRanges` (о котором говорилось немного раньше) являются диапазоны, представляющие *первые по порядку* цепочки каждого типа.

Например, приведенная в листинге 16.3 подпрограмма `HighlightNextStory0` выделяет синим цветом цепочку, следующую за текущей цепочкой (той, в которой находится курсор или выделение) и имеющую тот же тип.

Листинг 16.3

```
Sub HighlightNextStory0()
'Выделение синим цветом следующей цепочки того же типа
    Selection.Range.NextStoryRange.HighlightColorIndex = wdBlue
End Sub
```

В приведенной в листинге 16.3 подпрограмме использовано уже знакомое вам свойство `HighlightColorIndex` (цвет подкрашивания) объекта `Range` (диапазон), указывающее цвет выделения соответствующего диапазона, и константа `wdBlue` (синий).

Если курсор или выделение находятся не в надписи и не в колонтитуле, либо в последней надписи, либо в последнем колонтитуле определенного вида, то следующей цепочки не существует, и поэтому выполнение подпрограммы `HighlightNextStory0` приведет к сообщению об ошибке (рис. 16.2).

В этом сообщении говорится, что объектной переменной или переменной блока **`With`** не присвоено значение. В нашей инструкции нет ни объектных переменных, ни блока **`With`**, поэтому непросто догадаться, что причиной ошибки было то, что выражение `Selection.Range.NextStoryRange` содержало ссылку на пустой объект, т. е. возвращало значение **`Nothing`** (сведения, приведенные в разделе справочной системы, который вызывается при нажатии кнопки **Help**, тоже не помогают выяснить причину ошибки).

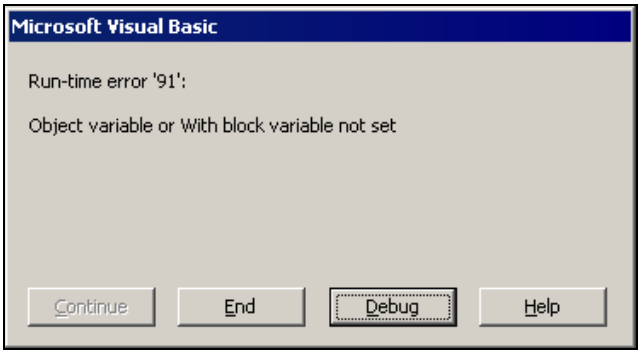


Рис. 16.2. Сообщение, появляющееся при отсутствии следующей цепочки того же типа

Чтобы предотвратить появление сообщения об ошибке, надо, прежде чем присваивать свойству `HighlightColorIndex` значение `wdBlue`, проверить, не ссылается ли выражение `Selection.Range.NextStoryRange` на пустой объект.

Для этой цели можно воспользоваться операцией **`Is`** (совпадает с), предназначенной исключительно для сравнения идентичности двух ссылок на объекты.

Например, в нашем случае, чтобы выяснить, не ссылается ли выражение `Selection.Range.NextStoryRange` на пустой объект, можно составить следующее условное выражение:

```
Selection.Range.NextStoryRange Is Nothing
```

Теперь мы можем усовершенствовать подпрограмму `HighlightNextStory0`, дополнив ее проверкой наличия следующей цепочки (листинг 16.4).

Листинг 16.4

```
Sub HighlightNextStory_1()
'Выделение синим цветом следующей цепочки того же типа
    If Selection.Range.NextStoryRange Is Nothing Then
        MsgBox Prompt:="Это последняя или единственная цепочка" _
            & "данного типа", Buttons:=vbExclamation, _
            Title:="Подкрасить следующую цепочку не удалось"
    Else
        Selection.Range.NextStoryRange.HighlightColorIndex = wdBlue
    End If
End Sub
```

Получилась простая, хотя и не очень полезная подпрограмма. Если же вы захотите убедиться, что она работает, то учтите, что переход к подкрашенной цепочке в ней не запрограммирован.

Еще раз посмотрите на листинг 16.4. Ссылка на следующую цепочку встречается два раза. Давайте эту ссылку присвоим *объектной переменной* и заменим этой переменной длинную ссылку. В нашем случае экономия получается несущественной, но этот прием вам пригодится.

Как мы уже писали в *разд. 15.3*, для присваивания значения объектной переменной написать знак равенства недостаточно. В начале инструкции *присваивания значения объектной переменной* следует написать ключевое слово **Set** (установить) с последующим пробелом.

В результате подпрограмма `HighlightNextStory_1` приобретает следующий вид, показанный в листинге 16.5.

Листинг 16.5

```
Sub HighlightNextStory_2()
'Выделение синим цветом следующей цепочки того же типа
'Применение объектной переменной
    Dim rng As Range
    Set rng = Selection.Range.NextStoryRange
    If rng Is Nothing Then
        MsgBox Prompt:="Это последняя или единственная цепочка" _
            & "данного типа", Buttons:=vbExclamation, _
            Title:="Подкрасить следующую цепочку не удалось"
    Else
        rng.HighlightColorIndex = wdBlue
    End If
End Sub
```

Теперь мы знаем и умеем достаточно, чтобы перебирать все цепочки. Начнем с того, что научимся считать общее число цепочек в документе (листинг 16.6).

Листинг 16.6

```
Sub StoryCounting()
' Подсчет цепочек в документе
    Dim rng As Range, iStory
    iStory = 0
```

```
    For Each rng In ActiveDocument.StoryRanges
        iStory = iStory + 1
        Do While Not (rng.NextStoryRange Is Nothing)
            Set rng = rng.NextStoryRange
            iStory = iStory + 1
        Loop
    Next rng
    MsgBox Prompt:=iStory, Title:="Количество цепочек в документе"
End Sub
```

В подпрограмме `StoryCounting` после начала цикла по типам цепочек в объектной переменной `rng` содержится ссылка на первую цепочку текущего типа. Сначала мы добавляем 1 в счетчик, чтобы подсчитать первую цепочку данного типа. Далее начинается цикл **Do** по всем оставшимся цепочкам данного типа. В этом цикле мы заносим в переменную `rng` ссылку на следующую цепочку и наращиваем счетчик. Когда цикл `Do` заканчивается, т. е. все цепочки данного типа просмотрены, выполняется новый элемент цикла **For Each** с новым значением объектной переменной `rng`, которая теперь содержит ссылку на первую цепочку нового типа.

16.4. Макросы с перебором цепочек

Подпрограмму `StoryCounting` можно рассматривать как схему построения различных макросов, в которых требуется перебрать все цепочки документа.

16.4.1. Выделение цветом во всех цепочках

Попробуйте самостоятельно составить макрос **NoHighlight**, который удаляет выделение цветом во всех цепочках документа. Если активно использовалось выделение цветом, такую операцию приходится выполнять каждый раз на завершающем этапе работы с документами, содержащими несколько цепочек. Для выполнения этой операции без этого макроса необходимо вручную перейти к каждой цепочке, выделить ее и нажать кнопку инструмента выделения цветом. Дополнительная сложность состоит в том, что при этом надо не пропустить ни одну цепочку.

При составлении подпрограммы `NoHighlight` за основу берется подпрограмма `StoryCounting` (листинг 16.6). Единственно, что требуется, это добавить инструкцию, которая будет удалять выделение цветом в текущей цепочке. Наверное, вы и сами догадались, что для того, чтобы узнать, как она записывается, проще всего автоматически записать соответствующий макрос.

Текст получившейся у нас подпрограммы `NoHighlight` приведен в листинге 16.7.

Листинг 16.7

```
Sub NoHighlight()
' Снятие выделения цветом со всех цепочек в документе
    Dim rng As Range
    For Each rng In ActiveDocument.StoryRanges
        rng.HighlightColorIndex = wdNoHighlight
        Do While Not (rng.NextStoryRange Is Nothing)
            Set rng = rng.NextStoryRange
            rng.HighlightColorIndex = wdNoHighlight
        Loop
    Next rng
End Sub
```

Схема перебора цепочек широко используется и в более сложных задачах. В первую очередь, это относится к задачам, связанным с поиском. Такими задачами мы уже занимались в *гл. 7* и создали несколько полезных макросов. Мы тогда честно предупредили, что все эти макросы будут работать только в основном тексте документа, и обещали исправить этот недостаток позже. Пришло время исполнять обещанное.

Начнем с макросов, которые выполняют поиск или поиск с заменой сразу во всем документе. Макросами, которые начинают поиск с текущего положения курсора и останавливаются, если находят в тексте искомое выражение, мы займемся немного позже.

В конце *гл. 12* мы усовершенствовали макрос, с помощью которого происходит поиск всех вхождений выделенного текста и найденный текст выделяется цветом. Цвет выбирается перед запуском макроса на панели инструментов **Форматирование** с помощью кнопки с изображением маленького черного треугольника, расположенной рядом с кнопкой выделения цветом. Еще раз приведем текст этой подпрограммы (листинг 16.8), немного подправив комментарии.

Листинг 16.8

```
Sub HighlightSelectionAll_1()
' Выделение текущим цветом всех вхождений выделенного текста
' в текущей цепочке
```

```
    With Selection.Find
        .ClearFormatting
        .Replacement.ClearFormatting
        .Replacement.Highlight = True
        .Text = Selection.Text
        .Replacement.Text = ""
        .Forward = True
        .Wrap = wdFindContinue
        .Format = True
        .MatchCase = False
        .MatchWholeWord = False
        .MatchWildcards = False
        .MatchSoundsLike = False
        .MatchAllWordForms = False
        .Execute Replace:=wdReplaceAll
        .Replacement.ClearFormatting
        .Text = ""
    End With
End Sub
```

Чтобы выполнить необходимую нам замену во всех цепочках, надо в каждой цепочке выполнить эту подпрограмму. Однако нельзя просто в подпрограмме перебора цепочек написать обращение к подпрограмме `HighlightSelectionAll_1`. Подпрограмму `HighlightSelectionAll_1` надо подправить, чтобы можно было сообщить ей, где выполнять замену, иначе она будет по-прежнему каждый раз выполнять замену только в той цепочке, где выделен текст.

Есть два подхода к решению этой задачи. Мы разберем оба, поскольку в обоих случаях используются заслуживающие внимания средства программирования.

Первый подход состоит в том, чтобы каждый раз перед обращением к подпрограмме выделять соответствующую цепочку. Тогда поиск будет происходить в выделенном тексте. Поскольку при этом снимется выделение с предварительно выделенного слова, которое нам понадобится для поиска, необходимо это слово запомнить. Для этой цели мы опишем строковую переменную `strText` и присвоим этой переменной значение `Selection.Text`. Переменную `strText` мы будем передавать подпрограмме поиска в качестве аргумента. Кроме того, неплохо бы в конце вернуть выделение на место. Чтобы запомнить положение исходного выделения, установим на выделение

закладку **HighlightSelectionAll**, а в конце подпрограммы перейдем на эту закладку (при этом восстановится исходное выделение) и, "чтобы замести следы", удалим ее.

Таким образом, у нас планируется подпрограмма-макрос, в которой будет выполняться перебор цепочек, и подпрограмма с аргументом `strText`, которая будет в каждой цепочке выделять цветом заданный текст. Подпрограмму-макрос назовем `HighlightSelectionAll_2`, а вызываемую из нее подпрограмму с аргументом — `HighlightInStory_1`.

При создании подпрограммы `HighlightSelectionAll_2` за основу возьмем цикл перебора цепочек, использованный в подпрограмме `StoryCounting` (листинг 16.6). В этом цикле перед каждым обращением к подпрограмме напишем инструкцию, выделяющую диапазон `rng`, соответствующий очередной цепочке:

```
rng.Select
```

В этой инструкции используется новый для вас метод `Select` (выделить) объекта `Range` (диапазон). Этот метод выделяет диапазон, представленный объектом `Range`. Вы уже знакомы с одноименным методом объекта `Bookmark`, выполняющим такую же функцию.

После успешного завершения замены сообщим об этом пользователю с помощью функции `MsgBox`.

Решение поставленной задачи, таким образом, реализуется в виде двух подпрограмм, приведенных в листингах 16.9 и 16.10.

Листинг 16.9

```
Sub HighlightSelectionAll_2()
' Выделение текущим цветом всех вхождений выделенного текста
' во всех цепочках (Selection.Find)
Dim rng As Range, strText As String
strText = Selection.Text
ActiveDocument.Bookmarks.Add Name:="HighlightSelectionAll" _
        , Range:=Selection.Range
For Each rng In ActiveDocument.StoryRanges
    rng.Select
    Call HighlightInStory_1(strText)
    Do While Not (rng.NextStoryRange Is Nothing)
        Set rng = rng.NextStoryRange
        rng.Select
```

```
        Call HighlightInStory_1(strText)
    Loop
Next rng
ActiveDocument.Bookmarks("HighlightSelectionAll").Select
ActiveDocument.Bookmarks("HighlightSelectionAll").Delete
MsgBox Prompt:="Текст """ & strText _
    & """ выделен во всем документе текущим цветом" _
    , Title:="Выделение цветом выделенного текста во всех цепочках"
End Sub
```

Листинг 16.10

```
Sub HighlightInStory_1(strText As String)
' Выделение текущим цветом всех вхождений текста strText
' в текущей цепочке
'
    With Selection.Find
        .ClearFormatting
        .Replacement.ClearFormatting
        .Replacement.Highlight = True
        .Text = strText
        .Replacement.Text = ""
        .Forward = True
        .Wrap = wdFindContinue
        .Format = True
        .MatchCase = False
        .MatchWholeWord = False
        .MatchWildcards = False
        .MatchSoundsLike = False
        .MatchAllWordForms = False
        .Execute Replace:=wdReplaceAll
        .Replacement.ClearFormatting
        .Text = ""
    End With
End Sub
```

Основная особенность данного подхода состоит в том, что объект `Range`, соответствующий очередной цепочке документа, с помощью метода `Select` представляется в виде выделенного фрагмента текста, для которого мы уже умеем решать поставленную задачу.

Другой подход основан на том, что свойство **`Find`** (найти) имеется не только у объекта `Selection` (выделение), но и у объекта `Range` (диапазон). Таким образом, поиск текста может начинаться не с выделенного текста или курсора, а с диапазона текста, представленного объектом `Range`.

Использование для ссылки на объект `Find` свойства `Find` объекта `Range`, а не объекта `Selection`, имеет определенные преимущества.

Мы знаем, что при применении свойства `Find` объекта `Selection` найденный в результате поиска текст выделяется (как при обычном поиске в Microsoft Word), т. е. объект `Selection` изменяется. Когда для поиска используется свойство `Find` объекта `Range`, найденный текст не выделяется, существующее выделение (или курсор) остаются на старом месте. Это означает, что содержимое экрана при поиске не изменяется. Такой "невидимый" поиск удобен для пользователя, поскольку на экране во время работы макроса ничего не мелькает, а поскольку перерисовывать экран не требуется, поиск работает быстрее.

Если поиск с применением свойства `Find` объекта `Range` увенчается успехом, объект `Range` изменится, он будет указывать на диапазон, содержащий найденный текст. Таким образом, на найденный текст перейдет не выделение, а диапазон. Это позволит узнать, что будет найдено в результате поиска.

Других отличий в этих способах поиска нет. Это удобно, поскольку для "невидимого" поиска с применением свойства `Find` объекта `Range` можно после минимальной переделки использовать отлаженные фрагменты подпрограмм, в которых использовалось свойство `Find` объекта `Selection`.

Для реализации второго подхода мы опять составим подпрограмму-макрос, в которой будет выполняться перебор цепочек, и подпрограмму с аргументом, которая будет в каждой цепочке выделять цветом заданный текст. Подпрограмму-макрос назовем `HighlightSelectionAll_3`, а вызываемую из нее подпрограмму с аргументом — `HighlightInStory_2`. Теперь из подпрограммы-макроса `HighlightSelectionAll_3` в подпрограмму `HighlightInStory_2` в качестве аргумента будет передаваться объектная переменная `rng` типа `Range`, содержащая ссылку на цепочку, в которой будут происходить поиск и замена.

В отличие от подпрограммы `HighlightInStory_1` в подпрограмме `HighlightInStory_2` вместо инструкции

```
With Selection.Find
```

будет использована инструкция

```
With rng.Find
```

Поскольку теперь с выделенного перед запуском макроса текста выделение не снимается, то не нужно вводить переменную для хранения выделенного текста. По той же причине не нужна закладка.

В подпрограмме `HighlightSelectionAll_3` (листинг 16.9) для того, чтобы сообщить пользователю об успешном завершении замены, мы использовали функцию `MsgBox`. Применение этой функции не всегда удобно, поскольку вывод диалогового окна сопровождается звуковым сигналом, а пользователю требуется выполнять дополнительное действие — закрывать окно. На этот раз мы сделаем по-другому: выведем нужное нам сообщение в строке состояния. Для этого достаточно присвоить текст сообщения глобальному свойству `StatusBar` (строка состояния). Такой способ не всегда подходит: он существенно меньше привлекает внимание пользователя, его даже можно не заметить. Посмотрите, как он работает в подпрограмме `HighlightSelectionAll_3`, и сами решайте, когда его применять в своих подпрограммах.

Тексты подпрограмм `HighlightSelectionAll_3` и `HighlightInStory_2` приведены в листингах 16.11 и 16.12.

Листинг 16.11

```
Sub HighlightSelectionAll_3()
' Выделение текущим цветом всех вхождений выделенного текста
' во всех цепочках (rng.Find)
Dim rng As Range
For Each rng In ActiveDocument.StoryRanges
    rng.Select
    Call HighlightInStory_2(rng)
    Do While Not (rng.NextStoryRange Is Nothing)
        Set rng = rng.NextStoryRange
        rng.Select
        Call HighlightInStory_2(rng)
    Loop
Next rng
StatusBar = "Текст """ & Selection.Text _
    & """ выделен во всем документе текущим цветом"
End Sub
```

Листинг 16.12

```
Sub HighlightInStory_2(rng As Range)
' Выделение текущим цветом всех вхождений выделенного текста
' в цепочке, содержащей диапазон rng
'
    With rng.Find
        .ClearFormatting
        .Replacement.ClearFormatting
        .Replacement.Highlight = True
        .Text = Selection.Text
        .Replacement.Text = ""
        .Forward = True
        .Wrap = wdFindContinue
        .Format = True
        .MatchCase = False
        .MatchWholeWord = False
        .MatchWildcards = False
        .MatchSoundsLike = False
        .MatchAllWordForms = False
        .Execute Replace:=wdReplaceAll
        .Replacement.ClearFormatting
        .Text = ""
    End With
End Sub
```

Итак, дорожка протоптана. Теперь можно по той же схеме усовершенствовать и другие макросы, в основе которых лежит "глобальная" замена во всем документе (так часто называют замену с использованием кнопки **Заменить все**).

Например, в *гл. 6* мы с вами написали макрос, помогающий проводить первичную литературную правку документа. Он подсвечивал указанным цветом заданные слова или части слов. Теперь вы можете сами его усовершенствовать. Вы знаете, что для решения таких задач удобно использовать процедуры с аргументами. Напишите подпрограмму `HighlightText` (выделить цветом текст) с аргументами `strText` (текст), `Color` (цвет) и `booWholeWord` (целое слово), которая будет выделять заданный текст или заданное слово заданным цветом во всех цепочках документа. Тогда макрос литературной правки `Pencraft` будет состоять только из обращений к этой подпрограмме.

Текст такого макроса вместе с подпрограммами `HighlightText` и `HighlightInStory_3` приведен в листингах 16.13—16.15.

Листинг 16.13

```
Sub Pencraft()
' Выделение цветом всех вхождений заданного текста
' или только слов целиком
Call HighlightText(strText:="польз", color:=wdBrightGreen _
                                    , booWholeWord:=False)
Call HighlightText(strText:="что", color:=wdYellow _
                                    , booWholeWord:=True)
End Sub
```

Листинг 16.14

```
Sub HighlightText(strText As String, color, booWholeWord As Boolean)
' Выделение цветом всех вхождений текста
' или только слов целиком
Dim rng As Range
Options.DefaultHighlightColorIndex = color
For Each rng In ActiveDocument.StoryRanges
    rng.Select
    Call HighlightInStory_3(rng, strText, booWholeWord)
    Do While Not (rng.NextStoryRange Is Nothing)
        Set rng = rng.NextStoryRange
        rng.Select
        Call HighlightInStory_3(rng, strText, booWholeWord)
    Loop
Next rng
End Sub
```

Листинг 16.15

```
Sub HighlightInStory_3(rng As Range _
    , strText As String, booWholeWord As Boolean)
' Выделение текущим цветом всех вхождений текста
' или только слов целиком в заданном диапазоне
'
```

```
    With rng.Find
        .ClearFormatting
        .Replacement.ClearFormatting
        .Replacement.Highlight = True
        .Text = strText
        .Replacement.Text = ""
        .Forward = True
        .Wrap = wdFindContinue
        .Format = True
        .MatchCase = False
        .MatchWholeWord = booWholeWord
        .MatchWildcards = False
        .MatchSoundsLike = False
        .MatchAllWordForms = False
        .Execute Replace:=wdReplaceAll
        .Replacement.ClearFormatting
        .Text = ""
    End With
End Sub
```

Следующий шаг — научиться переходить к следующей и предыдущей цепочкам.

16.4.2. Переход к следующей и предыдущей цепочкам

Когда вы приступаете к редактированию или переводу незнакомого документа, содержащего колонтитулы, сноски, надписи или рамки, возникает проблема: как просмотреть весь текст, ничего не пропустив. Можно написать себе памятку, чтобы не забыть просмотреть все колонтитулы сноски и примечания, но, чтобы просмотреть все надписи, их сначала надо обнаружить в документе. Даже если никакая надпись не спрятана под рамкой или другой надписью, регулярного способа просмотра всех надписей в интерфейсе Microsoft Word не предусмотрено.

Для решения этой задачи можно составить макросы перехода к следующей и к предыдущей цепочке.

Идея, лежащая в основе алгоритма перехода к следующей цепочке, проста. В *гл. 15* при программировании функции `GoToBookmark` (листинг 15.15) мы узнали, что у объекта `Selection` (выделение) есть метод `InStory` (в цепочке), позволяющий выяснить, находится ли курсор (или выделение) в ука-

занной цепочке. Мы уже умеем перебирать все цепочки документа. Во время такого перебора мы с помощью этого метода узнаем, когда дойдем до цепочки, в которой находится курсор (или выделение), и на следующей цепочке прервем перебор.

Здесь мы сталкиваемся с проблемой, аналогичной той, которую мы решали в предыдущей главе. Если курсор (или выделение) находится в колонтитуле, а следующей цепочкой является колонтитул, то при переходе в него колонтитул не отображается, а курсор (или выделение) исчезает. Это неприятное явление происходит, когда мы выполняем инструкцию `rng.Select`, пытаясь выделить диапазон `rng`, представляющий следующую цепочку — колонтитул следующего раздела. В предыдущей главе аналогичное явление происходило, когда мы пытались перейти к закладке, применяя тот же метод `Select`. Оказалось, что и в этом случае помогает последовательный переход в режимы **Разметка страницы** и **Обычный**.

Подпрограмма `SelectNextStory` (выделить следующую цепочку) и используемая в ней функция `IsInStory` (находится в цепочке) приведены в листингах 16.16 и 16.17.

Листинг 16.16

```
Sub SelectNextStory()
'Выделение следующей цепочки
    Dim rng As Range, boo As Boolean, iStory
    boo = False 'признак завершения в подпрограмме IsInStory
    iStory = 0
    For Each rng In ActiveDocument.StoryRanges
        Call IsInStory(boo, rng, iStory)
        Do While Not (rng.NextStoryRange Is Nothing)
            Set rng = rng.NextStoryRange
            Call IsInStory(boo, rng, iStory)
        Loop
    Next rng
    If boo = False Then
        Call MsgBox("Не удалось найти цепочку " _
        & "с курсором или выделенным текстом" _
        , vbCritical, " Переход к следующей цепочке ")
    ElseIf Selection.StoryType = wdMainTextStory Then
        MsgBox "В документе только одна цепочка - основной текст" _
        , vbInformation, "Переход к следующей цепочке"
```

```
    ElseIf MsgBox("Достигнут конец документа." & vbCr & vbCr _
        & "Перейти в основной текст?", vbQuestion + vbYesNo _
        , "Переход к следующей цепочке") = vbYes Then
        ActiveDocument.Range.Select
    End If
End Sub
```

Листинг 16.17

```
Sub IsInStory(boo As Boolean, rng As Range, iStory)
' Если boo, то выделяется rng и останавливается выполнение.
' Иначе возвращается boo - признак наличия курсора
'                          или выделения в цепочке rng
' Используется в подпрограмме SelectNextStory()
    iStory = iStory + 1
    If boo Then
        ' Если курсор или выделение в предыдущей цепочке
        ActiveWindow.View.Type = wdPrintView
        ActiveWindow.View.Type = wdNormalView
        rng.Select
        StatusBar = "Цепочка " & iStory
        End
    End If
    'признак наличия курсора или выделения в цепочке rng
    boo = Selection.InStory(rng)
End Sub
```

В подпрограмме `SelectNextStory` вводится логическая переменная `boo`, в которую с помощью подпрограммы `IsInStory` заносится признак наличия курсора или выделения в текущей цепочке. Обращение к этой подпрограмме происходит при переборе цепочек, который начинается с основного документа. Когда обнаруживается цепочка, содержащая курсор или выделение, значение `boo` изменяется с **`False`** на **`True`** и на следующей цепочке при обращении к функции `IsInStory` происходит выделение цепочки и инструкция `End` прекращает выполнение макроса.

Если цикл доходит до конца, то это означает, что курсор находится в последней цепочке и, следовательно, переменная `boo` содержит значение **`True`**.

Тем не менее, после окончания цикла мы предусмотрели проверку, действительно ли переменная `boo` содержит значение **`True`**. Мы надеялись, что этого никогда не произойдет, но если такое все-таки случится, то макрос не обманет пользователя, а ошибку в программе выявить и исправить будет значительно легче.

> ***Совет***
>
> Вставляйте в программу дополнительные проверки своих умозаключений. Если вы на основании своих умозаключений считаете, что некоторое событие произойти не может, мы все-таки рекомендуем вам проверить, произошло ли оно, и, если оно все-таки происходит, то сообщить пользователю об ошибке и остановить выполнение программы.

Оказалось, что сделали мы это не зря. Если в документе есть сноски, то в режиме **Обычный** можно установить курсор в отдельной области сносок (**Вид | Сноски**) и в раскрывающемся списке в заголовке области сносок выбрать, например, **Разделитель сноски**. В версиях Microsoft Word 97 и Microsoft Word 2000 эта область не считается цепочкой, и перейти из нее в следующую цепочку нельзя. Перечень таких областей приведен в табл. 16.1 в строках 12—16. Если курсор находится в любой из таких областей, то при работе в версии Microsoft Word 97 или Microsoft Word 2000 выдается сообщение о невозможности перехода в следующую цепочку.

Далее мы проверяем, не находится ли курсор или выделение в основном тексте документа, т. е. не является ли последняя цепочка первой и единственной цепочкой документа. Если это так, то мы сообщаем об этом пользователю и выполнение программы прекращается. Если же в документе больше одной цепочки, то пользователю сообщается, что достигнут конец документа. При этом ему предоставляется возможность выбора: оставить выделенной последнюю цепочку или выделить основной текст документа.

В подпрограмме `IsInStory` сначала наращивается счетчик цепочек, он нам нужен, чтобы вывести в строке состояния номер цепочки. После этого проверяется значение `boo` — признак того, что курсор или выделение находится в предыдущей цепочке. Если `boo = True`, то надо выделить текущую цепочку и закончить выполнение макроса. Перед выходом из макроса выполняется последовательный переход в режим **Разметка страницы** и режим **Обычный**, выделяется цепочка и в строке состояния выводится ее номер.

Обратите внимание на содержащуюся в подпрограмме следующую инструкцию:

```
IsInStory = Selection.InStory(rng)
```

Она выполняется, если в предыдущей цепочке нет ни курсора, ни выделения. В этой инструкции к объекту `Selection` применяется упомянутый выше метод `InStory`. У этого метода один аргумент, который должен содер-

жать ссылку на объект `Range` (в нашем случае ссылка содержится в объектной переменной `rng`). Если выделение, соответствующее объекту `Selection`, содержится в той же цепочке, что и объект `Range`, то метод `InStory` возвращает значение **`True`**. В противном случае он возвращает значение **`False`**. Тестирование макроса перехода к следующей цепочке проведите самостоятельно.

Несколько сложнее написать макрос перехода к предыдущей цепочке. Сначала мы подсчитаем, сколько всего цепочек и в какой по порядку цепочке находится курсор (или выделение). Затем потребуется еще раз перебрать цепочки, чтобы остановиться на предыдущей цепочке. Текст макроса **SelectPrevStory** (*Select Previous Story* — выделить предыдущую цепочку) и используемых в нем процедур приведен в листингах 16.18—16.21.

Листинг 16.18

```
Sub SelectPrevStory()
'Выделение предыдущей цепочки
    Dim storiesTotal, currentStory, newStoryNo
    Call storyCount(storiesTotal, currentStory)
    newStoryNo = newStory(storiesTotal, currentStory)
    Call selectStory(newStoryNo)
    StatusBar = "Цепочка " & newStoryNo & " из " & storiesTotal
End Sub
```

Листинг 16.19

```
Sub storyCount(storiesTotal, currentStory)
'Возвращается:
'storiesTotal - количество цепочек
'currentStory - номер текущей цепочки
    Dim rng As Range
    storiesTotal = 0
    For Each rng In ActiveDocument.StoryRanges
        storiesTotal = storiesTotal + 1
        If Selection.InStory(rng) Then
            currentStory = storiesTotal
        End If
        Do While Not (rng.NextStoryRange Is Nothing)
            Set rng = rng.NextStoryRange
```

```
            storiesTotal = storiesTotal + 1
            If Selection.InStory(rng) Then
                currentStory = storiesTotal
            End If
        Loop
    Next rng
End Sub
```

Листинг 16.20

```
Function newStory(storiesTotal, currentStory)
'Возвращается номер предыдущей цепочки
    If currentStory = 0 Then
        MsgBox "Не удалось найти цепочку с курсором или " _
            & "выделенным текстом" _
        , vbExclamation, "Переход к предыдущей цепочке"
        End
    End If
    If storiesTotal = 1 Then
        MsgBox "В документе только одна цепочка - основной текст" _
        , vbExclamation, "Переход к предыдущей цепочке"
        End
    End If
    If currentStory = 1 Then
        If MsgBox("Курсор (или выделение) в основном тексте " _
            & "документа." & vbCr & vbCr _
            & "Перейти в последнюю цепочку?", vbQuestion + vbYesNo _
            , "Переход к предыдущей цепочке") = vbYes Then
            newStory = storiesTotal
        Else
            End
        End If
    Else
        newStory = currentStory - 1
    End If
End Function
```

Листинг 16.21

```
Sub selectStory(newStoryNo)
'Выделение цепочки с заданным номером
    Dim rng As Range, iStory
    iStory = 0
    For Each rng In ActiveDocument.StoryRanges
        iStory = iStory + 1
        If iStory = newStoryNo Then
            ActiveWindow.View.Type = wdPrintView
            ActiveWindow.View.Type = wdNormalView
            rng.Select
            Exit Sub
        End If
        Do While Not (rng.NextStoryRange Is Nothing)
            Set rng = rng.NextStoryRange
            iStory = iStory + 1
            If iStory = newStoryNo Then
                ActiveWindow.View.Type = wdPrintView
                ActiveWindow.View.Type = wdNormalView
                rng.Select
                Exit Sub
            End If
        Loop
    Next rng
End Sub
```

В подпрограмме `SelectPrevStory` сначала вызывается подпрограмма `storyCount` (подсчет цепочек), в которой вычисляется `storiesTotal` — общее количество цепочек и `currentStory` — номер текущей цепочки. Затем с помощью функции `newStory` (новая цепочка) вычисляется `newStoryNo` — номер новой цепочки. После этого в подпрограмме `selectStory` выделяется цепочка с найденным номером. В завершение в строке состояния выводится номер новой цепочки и общее число цепочек.

В этих подпрограммах вам все должно быть понятно без комментариев.

Мы рекомендуем вам досконально разобраться в работе подпрограмм `SelectPrevStory` и `SelectNextStory`. Если у вас возникают вопросы, то по-

лезно пройти все действия по шагам в режиме отладки. Используемый здесь подход мы будем применять в более сложной ситуации, поэтому лучше все вопросы снять на этом этапе.

16.5. Макросы продолжения поиска по цепочкам

Теперь займемся макросами, которые начинают поиск с текущего положения курсора и останавливаются, если находят в тексте искомый текст.

Эта задача более сложная и более громоздкая, чем те, которыми мы занимались до сих пор. Мы ее выбрали для того, чтобы вы увидели, как совсем непростая для непрофессионального программиста задача превращается в комплекс небольших подзадач, большую часть которых вы сможете запрограммировать самостоятельно. Это очень важный и интересный аспект разработки программ, который можно освоить только на практике. Новых программных средств для решения этой задачи почти не потребуется, в основном это будет повторение и практическое применение пройденного материала в условиях, приближенных к реальным.

Сначала уточним задачу.

Мы предполагаем создать макросы для решения трех различных задач поиска:

- **storyFindSelection** — поиск следующего вхождения выделенного фрагмента текста;
- **storyFindHighlight** — поиск следующего фрагмента, выделенного заданным цветом;
- **storyFindNext** — поиск во всех цепочках документа с использованными ранее условиями поиска. Этот макрос позволяет задать условия поиска в диалоговом окне **Найти и заменить**, используя все возможности, предоставляемые этим диалоговым окном, выполнить поиск, нажав в этом диалоговом окне кнопку **Найти далее**, а затем, с помощью данного макроса продолжить поиск в оставшейся части документа (включая остальные цепочки).

Важно, что после того как будет найдено очередное вхождение искомого фрагмента, то можно будет внести в текст документа необходимые изменения, а затем продолжить поиск с переходом в другие цепочки. Обычный поиск с помощью диалогового окна **Найти и заменить** этого не позволяет — выводится сообщение, свидетельствующее о том, что поиск в данной цепочке завершен (например, такое как приведенное на рис. 16.1).

Каждый из этих трех перечисленных ранее макросов должен устанавливать закладку **storyFindFrom** в том месте, где начат поиск, и вызывать одну и ту

же для всех трех макросов подпрограмму `storyFindStep` для выполнения очередной попытки поиска. Мы будем использовать то, что условия поиска запоминаются как параметры диалогового окна **Найти и заменить** после завершения работы макроса.

Если искомый фрагмент найден, то он выделяется и макрос завершает работу. Можно продолжить поиск в еще не просмотренной части документа с использованными ранее условиями поиска с помощью четвертого макроса **storyFindContinue**. При необходимости перед продолжением поиска пользователь может внести в документ правку. Если же достигнут конец документа (точнее, конец последней цепочки документа), а искомый фрагмент не найден, то пользователю выдается запрос о необходимости продолжения поиска с начала документа. При положительном ответе устанавливается закладка **storyFindEnd** и поиск продолжается. Если документ просмотрен до того места, с которого пользователь начинал поиск, а искомый фрагмент не найден, то пользователь получает об этом сообщение, удаляются все установленные при поиске закладки, а также информация об условиях поиска, содержащаяся в диалоговом окне **Найти и заменить**. С помощью пятого макроса **storyFindStop** можно в любой момент выполнить удаление закладок и информации в окне **Найти и заменить**.

Все процедуры, относящиеся к поиску по цепочкам, хранятся в готовом виде в модуле **Listings_16** (шаблон MacroBook.dot). Для заглушек в этом шаблоне предусмотрен модуль **Listings_16_Stubs** (*stubs* — заглушки). Для отладки мы рекомендуем вам создать в своем шаблоне Normal.dot модуль **storyFindDebug** (*debug* — отладка). По мере необходимости следует переносить в модуль **storyFindDebug** заглушки из модуля **Listings_16_Stubs**, а готовые процедуры — из модуля **Listings_16**.

Начнем с макроса **storyFindHighlight**, предназначенного для поиска текста, выделенного определенным цветом.

План этого макроса можно составить в виде подпрограммы-заглушки, представленной в листинге 16.22.

Листинг 16.22

```
Sub storyFindHighlight()
'Поиск по цепочкам выделенного определенным цветом текста
'с учетом цвета выделения
    If storyFindStartCheck() Then
        '<Задание параметров поиска выделенного цветом текста>
        Call storyFindStep
    End If
End Sub
```

В этом макросе проверяется условие `storyFindStartCheck()` и, если оно принимает значение **`True`**, то задаются параметры поиска для диалогового окна **Найти и заменить** и происходит обращение к общей для всех трех макросов подпрограмме `storyFindStep`, выполняющей поиск следующего вхождения искомого фрагмента. Если условие `storyFindStartCheck()` принимает значение **`False`**, то происходит выход из подпрограммы.

Если для функции `storyFindStartCheck` и подпрограммы `storyFindStep` использовать заглушки, приведенные в листинге 16.23, то подпрограмму-заглушку `storyFindHighlight` можно выполнить и получить запрограммированное в подпрограмме-заклушке `storyFindStep` сообщение.

Листинг 16.23

```
Function storyFindStartCheck() As Boolean
'Возвращает признак возможности начала поиска
storyFindStartCheck = True
End Function

_____________________________________________

Sub storyFindStep()
'шаг поиска по цепочкам
MsgBox "Подпрограмма, выполняющая поиск очередного вхождения" _
    & " искомого фрагмента, еще не реализована.", vbExclamation _
    , "Поиск по цепочкам"
End Sub
```

В результате мы получили макрос-заглушку, с помощью которого можно отлаживать создаваемые подпрограммы.

Теперь постепенно подпрограмму-заглушку `storyFindHighlight` будем превращать в реальную подпрограмму. Для этого требуется написать инструкции, в которых будут задаваться параметры поиска выделенного цветом текста. Особенность состоит в том, что в свойствах объекта `Find`, так же как и в диалоговом окне **Найти и заменить**, нельзя задать поиск текста, выделенного определенным цветом. Поэтому мы будем искать просто выделенный цветом фрагмент, а когда он найдется, будем проверять его цвет.

Чтобы не вспоминать, как записываются нужные нам инструкции, воспользуемся испытанным приемом автоматической записи макроса. Запишите макрос **Test13**, который будет искать выделенный текущим цветом текст. Перед записью макроса выполните следующие действия:

1. Откройте диалоговое окно **Найти и заменить** (<Ctrl>+<F>).
2. Если отображается кнопка **Больше** (а не **Меньше**), нажмите ее, чтобы раскрыть диалоговое окно полностью.

3. Перейдите на вкладку **Заменить**.
4. Поля **Найти** и **Заменить на** оставьте пустыми (чтобы убедиться в том, что они не содержат пробелов, поместите в них по очереди курсор).
5. В поле **Направление** выберите значение **Вперед** и если в группе **Параметры поиска** есть установленные флажки, то снимите их.
6. Нажмите кнопку **Формат** и в раскрывшемся списке выберите значение **Выделение цветом**.

   В результате диалоговое окно примет вид, показанный на рис. 16.3.

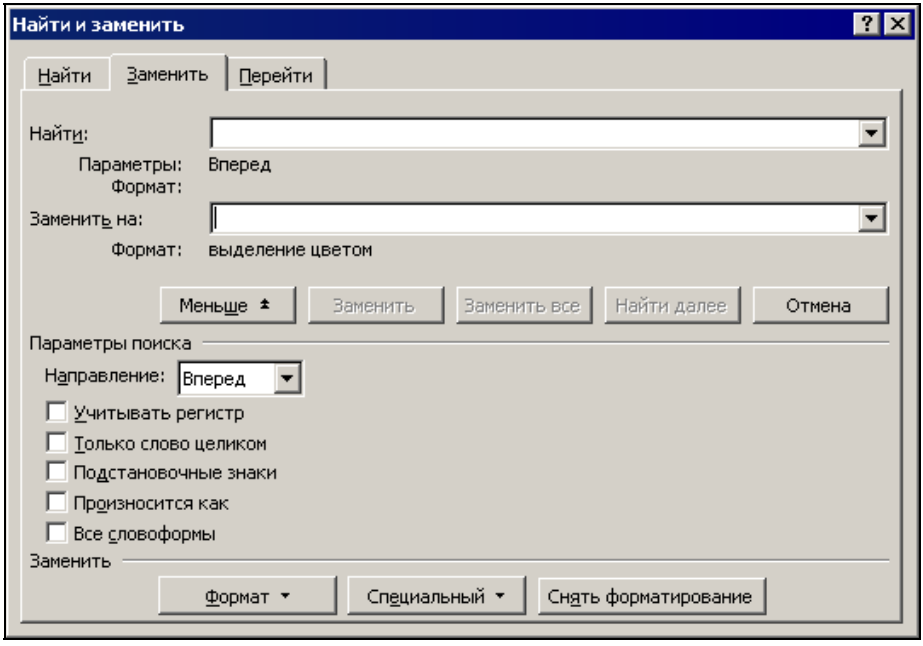


Рис. 16.3. Вид диалогового окна **Найти и заменить** перед записью макроса **Test13**

7. Нажмите кнопку **Найти далее**, чтобы попытаться найти выделенный цветом текст в текущем документе.

   Независимо от того, была ли эта попытка удачной, изменения, сделанные вами в диалоговом окне **Найти и заменить**, после такой попытки сохранятся.
8. Убедитесь, что в активном окне нет выделения, и запустите запись макроса **Test13**.

Во время записи макроса достаточно открыть диалоговое окно **Найти и заменить** и сразу закрыть его кнопкой **Отмена**. Можно было бы выполнить действия, перечисленные в шагах 1—7, не перед записью макроса, а во время его записи, но, на наш взгляд, так спокойнее.

Текст автоматически созданной подпрограммы приведен в листинге 16.24.

Листинг 16.24

```
Sub Test13()
'
' Test13 Макрос
' Поиск выделенного цветом текста
'
    Selection.Find.ClearFormatting
    Selection.Find.Highlight = True
    With Selection.Find
        .Text = ""
        .Replacement.Text = ""
        .Forward = True
        .Wrap = wdFindAsk
        .Format = True
        .MatchCase = False
        .MatchWholeWord = False
        .MatchWildcards = False
        .MatchSoundsLike = False
        .MatchAllWordForms = False
    End With
End Sub
```

В этой подпрограмме имеется одна новая инструкция

```
Selection.Find.Highlight = True
```

в которой устанавливается значение `True` для свойства `Highlight` (выделение цветом) объекта `Find` (найти). Это свойство соответствует параметру **Выделение цветом** диалогового окна **Найти и заменить** (табл. 16.2).

Таблица 16.2. *Свойство* `Highlight` *и параметр* ***Выделение цветом***

Значение свойства `Highlight`	**Значение параметра** ***Выделение цветом***	**Описание**
`True`	**выделение цветом**	Ищутся только выделенные цветом фрагменты
`False`	**невыделение цветом**	Ищутся только невыделенные цветом фрагменты
`wdUndefined`		Выделение цветом не влияет на поиск

Из текста подпрограммы `Test13` (листинг 16.24) можно целиком взять блок `With`, в котором задаются свойства объекта `Find`. Надо только заменить инструкцию

```
.Wrap = wdFindAsk
```

инструкцией

```
.Wrap = wdFindStop
```

чтобы при достижении конца цепочки поиск останавливался, а не выводилось диалоговое окно с вопросом о необходимости продолжить поиск с начала цепочки (подробно о свойстве `Wrap` рассказано в *гл. 12*). Забегая вперед, отметим, что свойства `Wrap` и `Forward` нам потребуются задавать во всех трех макросах начала поиска. Поэтому, чтобы не повторять трижды одну и ту же инструкцию, лучше задать их в общей для всех трех макросов функции `storyFindStartCheck`.

Как мы уже сказали, после того как будет найден выделенный цветом фрагмент, мы будем проверять цвет выделения. Однако эту проверку требуется выполнять только в том случае, когда поиск начат макросом **storyFindHighlight**. Следовательно, необходимо передать в процедуру, в которой будет выполняться проверка цвета выделения, информацию о том, что поиск запущен макросом **storyFindHighlight**. Обычно информация в процедуры передается с помощью ее аргументов, но в нашем случае этот вариант не подходит, поскольку проверку следует выполнять и во время продолжения поиска, который запускается макросом **storyFindContinue**, и который "не знает", каким макросом начат поиск. Чтобы передать информацию такого рода из одного макроса в другой, можно в одном макросе установить закладку с условным именем, а в другом макросе проверить, что такая закладка существует. Потому после задания параметров поиска установим закладку **storyFindHighlight**.

В итоге подпрограмма `storyFindHighlight` приобретает вид, показанный в листинге 16.25.

Листинг 16.25

```
Sub storyFindHighlight()
'Поиск по цепочкам выделенного цветом текста
'с учетом цвета выделения
    If storyFindStartCheck Then
        With Selection.Find
            .ClearFormatting
            .Highlight = True
            .Text = ""
            .Replacement.Text = ""
            .Format = True
            .MatchCase = False
            .MatchWholeWord = False
            .MatchWildcards = False
            .MatchSoundsLike = False
            .MatchAllWordForms = False
        End With
        ActiveDocument.Bookmarks.Add "storyFindHighlight"
        Call storyFindStep
    End If
End Sub
```

Можно проверить, как работает эта подпрограмма. Скопируйте ее из модуля **Listings_16** в модуль **storyFindDebug**, заменив подпрограмму-заклушку с тем же именем. После ее выполнения проверьте наличие закладки **storyFindHighlight** (<Ctrl>+<Shift>+<F5>), откройте диалоговое окно **Найти и заменить** на вкладке **Заменить** (<Ctrl>+<H>) и убедитесь, что она выглядит так же, как и на рис. 16.3. Отличаться может только значение поля **Направление**, поскольку мы планируем задать его при выполнении функции `storyFindStartCheck`, к реализации которой мы приступаем только сейчас.

Функция `storyFindStartCheck` должна установить закладку **storyFindFrom**, указывающую место в документе, с которого начинается поиск. По этой закладке мы будем определять, весь ли документ просмотрен, чтобы не выполнять поиск в уже просмотренной части документа, и будем возвращаться на эту закладку и удалять ее, когда поиск закончится.

Если при запуске нового поиска в документе обнаруживается закладка **storyFindFrom**, то это означает, что предыдущий поиск не закончен. В этом случае требуется запросить пользователя, действительно ли он намерен на-

чать новый поиск или он запустил новый поиск по ошибке. Если пользователь отказывается начинать новый поиск, то функция `storyFindStartCheck` возвращает значение **`False`** и происходит выход из подпрограммы `storyFindHighlight` без изменения документа. Если же пользователь подтверждает свое желание начать новый поиск, то выполнение функции `storyFindStartCheck` продолжается. Устанавливается закладка **storyFindFrom** и удаляется закладка **storyFindEnd** (чтобы не возникла ошибка, перед удалением закладки необходимо проверять, что закладка существует). Эта закладка устанавливается, когда при поиске достигается конец последней цепочки документа и поиск продолжается с начала основного текста документа. Если закладка **storyFindEnd** установлена и пользователь пытается продолжить поиск с места, расположенного в документе после закладки **storyFindFrom**, то это означает, что делается попытка поиска в уже просмотренной части документа и поиск выполнять не следует. Поэтому при запуске нового поиска эту закладку следует удалить. Перед завершением выполнения функции устанавливаются значения свойств `Wrap` и `Forward` объекта `Find`, о чем мы говорили раньше, и задается возвращаемое значение (**`True`**) функции.

Прежде чем смотреть текст функции `storyFindStartCheck` (листинг 16.26), мы рекомендуем вам попытаться запрограммировать ее самостоятельно.

Листинг 16.26

```
Function storyFindStartCheck() As Boolean
'Возвращает признак возможности начала поиска
'При наличии закладки storyFindFrom
'запросить пользователя о начале нового поиска,
'при отсутствии закладки storyFindFrom
'установить закладку storyFindFrom,
'удалить, если она есть, закладку storyFindEnd
'и установить свойства Wrap и Forward
'объекта Find
With ActiveDocument
    If .Bookmarks.Exists("storyFindFrom") Then
        If MsgBox("Предыдущий поиск не закончен." & vbCr & vbCr _
            & "Начать новый поиск?" & vbCr & vbCr _
            , vbQuestion + vbYesNo _
            , "Начало поиска по цепочкам") = vbNo Then
            storyFindStartCheck = False
            Exit Function
        End If
```

```
    End If
    .Bookmarks.Add "storyFindFrom", Selection.Range
    If .Bookmarks.Exists("storyFindEnd") Then
        .Bookmarks("storyFindEnd").Delete
    End If
End With
Selection.Find.Wrap = wdFindStop
Selection.Find.Forward = True
storyFindStartCheck = True
End Function
```

Скопируйте функцию `storyFindStartCheck` из модуля **Listings_16** в модуль **storyFindDebug** и проверьте, как она работает. Подумайте, как удостовериться в том, что каждый элемент функции действует должным образом.

Теперь давайте напишем подпрограмму `storyFindStartNext`, начинающую новый поиск во всех цепочках документа с условиями, оставшимися от предыдущего поиска. Эта подпрограмма задумана так, чтобы можно было задать условия поиска в диалоговом окне **Найти и заменить**, используя все возможности, предоставляемые этим диалоговым окном, выполнить поиск, нажав в этом диалоговом окне кнопку **Найти далее**, а затем, с помощью макроса выполнить тот же поиск в остальных цепочках документа. Важно, что после того как будет найдено очередное вхождение искомого фрагмента, можно внести в текст документа необходимые изменения, а затем продолжить поиск с переходом в другие цепочки. Обычный поиск с помощью диалогового окна **Найти и заменить** этого не позволяет — выводится сообщение, свидетельствующее о том, что поиск в данной цепочке завершен (например, такое как приведенное на рис. 16.1).

В отличие от подпрограммы `storyFindHighlight` параметры поиска устанавливать не требуется — используются значения, сохраненные в диалоговом окне **Найти и заменить**. Если при обращении к подпрограмме `storyFindNext` в документе выделен какой-либо фрагмент, выделение следует снять и поместить курсор в конец выделенного участка — это будет точка начала поиска. Кроме того, надо предусмотреть ситуацию, когда в документе будет установлена закладка **storyFindHighlight**. Такая ситуация может возникнуть, если не закончился поиск, выполнявшийся макросом **storyFindHighlight**. Логично ее удалить, поскольку при ее наличии поиск выделенного цветом текста происходит с учетом цвета выделения, а нам этого не требуется, т. к. параметры диалогового окна **Найти и заменить** такой возможности не предоставляют, а мы предполагаем, что наш макрос будет выполнять поиск, условия которого заданы с помощью именно этого диалого-

вого окна. При необходимости вы можете создать свой макрос, который будет выполнять поиск выделенного текста с учетом цвета выделения, просто удалив соответствующие инструкции. Попробуйте самостоятельно написать подпрограмму `storyFindStartNext` и сравните то, что у вас получится, с нашим вариантом (листинг 16.27).

Листинг 16.27

```
Sub storyFindNext()
'новый поиск во всех цепочках документа
'с использованными ранее условиями поиска
If storyFindStartCheck Then
    Selection.Collapse Direction:=wdCollapseEnd
    With ActiveDocument
        If .Bookmarks.Exists("storyFindHighlight") Then
            .Bookmarks("storyFindHighlight").Delete
        End If
    End With
    Call storyFindStep
End If
End Sub
```

Приведенный вариант подпрограммы `storyFindNext` можно переписать из модуля **Listings_16** в модуль **storyFindDebug** и проверить, как он работает.

Следующая подпрограмма — `storyFindSel`, предназначенная для поиска во всех цепочках следующего вхождения выделенного текста. Аналогичную задачу для поиска в одной цепочке мы решали с помощью подпрограммы `SearchSelectionNext_1`, описанной в *разд. 12.1.4*. В подпрограмме `storyFindSel` добавлена проверка типа выделения (*разд. 15.2*, после листинга 15.7), проверяется, что количество символов в выделенном фрагменте не должно превышать 255, а далее следует знакомая проверка условного выражения, вычисляемого с помощью функции `storyFindStartCheck`. Когда это выражение истинно, устанавливаются те же условия поиска, что и в подпрограмме `SearchSelectionNext_1` (кроме тех, которые устанавливаются в функции `storyFindStartCheck`), а затем происходит обращение к подпрограмме `storyFindStep`.

В подпрограмме `storyFindSel` нет новых для вас элементов, и вы можете попробовать составить ее самостоятельно, а потом сравнить то, что у вас получится, с нашим вариантом (листинг 16.28).

Листинг 16.28

```
Sub storyFindSel()
'Поиск выделенного фрагмента во всех цепочках документа
If Selection.Type <> wdSelectionNormal Then
    MsgBox "Должен быть выделен обычный текст для поиска." & vbCr _
        & vbCr & "Для данного типа выделения поиск не предусмотрен", _
        vbExclamation, "Неудачное начало поиска выделенного текста"
    Exit Sub
End If
If Len(Selection.Text) > 255 Then
    MsgBox "Выделено символов: " & Len(Selection.Text) & vbCr & vbCr _
        & "Для поиска можно выделить не более 255 символов." _
        , vbExclamation, "Неудачное начало поиска выделенного текста"
        Exit Sub
End If
If storyFindStartCheck Then
    With Selection.Find
         .ClearFormatting
        .Text = Selection.Text
        .Replacement.Text = ""
        .Format = False
        .MatchCase = False
        .MatchWholeWord = False
        .MatchWildcards = False
        .MatchSoundsLike = False
        .MatchAllWordForms = False
    End With
    Call storyFindStep
End If
End Sub
```

После проверки работы подпрограммы `storyFindSel` (выполните ее самостоятельно) можно приступить к подпрограмме `storyFindContinue`. Три предыдущие подпрограммы предназначены для запуска поиска по цепочкам с различными условиями поиска. Если искомое выражение найдено, то можно или сразу, или после внесения в документ необходимых изменений продолжить поиск с того места, где находится курсор. Поскольку условия

поиска запоминаются в качестве параметров диалогового окна **Найти и заменить**, для продолжения поиска с условиями, заданными любым из трех макросов, можно пользоваться одним и тем же макросом **storyFindCont**. В этом макросе проверяется, можно ли продолжать поиск, и запускается подпрограмма `storyFindStep` для выполнения очередного шага поиска по цепочкам. Если все проверки выполнять с помощью функции `storyFindContCheck`, то подпрограмма `storyFindStep` (листинг 16.29) получается такой простой, что даже нет смысла предлагать ее вам для самостоятельной работы.

Листинг 16.29

```
Sub storyFindCont()
'Продолжение ранее начатого поиска
'в оставшейся части документа
    If storyFindContCheck() Then
            Call storyFindStep
    End If
End Sub
```

Функция `storyFindContCheck` возвращает признак возможности продолжения поиска. Во-первых, она должна проверять, что закладка **storyFindStep** установлена, — если ее нет, то надо сообщить об этом пользователю и удалить оставшиеся следы поиска по цепочкам. Во-вторых, не делается ли попытка поиска в уже просмотренной части документа, т. е. не начинается ли поиск с места, расположенного после закладки **storyFindFrom** (начало просмотра), когда в документе установлена закладка **storyFindEnd** (продолжение поиска после того, как достигнут конец документа). В этом случае надо выяснить у пользователя, не требуется ли штатным образом закончить поиск. Для окончания поиска используется подпрограмма `storyFindStop`.

Проверять наличие закладки вы умеете, а как выяснить, не начинается ли поиск с места, расположенного после закладки, мы сейчас расскажем. Объекты `Selection` (выделение), `Bookmark` (закладка) и `Range` (диапазон), отмечающие участки цепочек, характеризуются начальной и конечной позицией, которые возвращают и устанавливают свойства этих объектов `Start` (начало) и `End` (конец). Значения этих свойств являются числами, сравнивая которые, можно узнать относительное расположение объектов. Каждая цепочка начинается с нулевой позиции, поэтому сравнение имеет смысл делать только для объектов, находящихся в одной цепочке. Чтобы проверить, что два объекта принадлежат одной цепочке, используется метод `InStory` объектов `Selection` и `Range`, о котором мы рассказывали в *гл. 15*. Мы рекоменду-

ем вам самостоятельно разобраться в функции `storyFindContCheck` (листинг 16.30), а проверять, как она работает, еще рано: сначала надо написать подпрограмму `storyFindStop`.

Листинг 16.30

```
Function storyFindContCheck()
'Возвращает признак возможности продолжения поиска
'Проверить условия продолжения поиска - наличие
'закладки storyFindFrom
'если этой закладки нет, сообщить пользователю и
'закончить поиск
'Если установлена закладка storyFindEnd,
'курсор (или выделение) в начальной цепочке
'и после закладки storyFindFrom, то сообщить
'пользователю и с его согласия закончить поиск
    With ActiveDocument
        If Not .Bookmarks.Exists("storyFindFrom") Then
            MsgBox "Поиск не был начат (нет закладки " _
                & "storyFindFrom)." & vbCr & vbCr _
                & "Начните поиск сначала", vbExclamation _
                , "Продолжение поиска по цепочкам"
            Call storyFindStop
            storyFindContCheck = False
            Exit Function
        End If
        If .Bookmarks.Exists("storyFindEnd") _
             And Selection.InStory(.Bookmarks("storyFindFrom").Range) _
            And Selection.Start >= .Bookmarks("storyFindFrom").End Then
            If MsgBox("Нельзя выполнить поиск в уже просмотренной " _
                & " части документа (курсор после закладки" _
                & " storyFindFrom). Можно продолжить поиск с другого" _
                & " места или продолжить поиск." & vbCr & vbCr _
                & "Закончить поиск?", vbQuestion + vbYesNo _
                , "Продолжение поиска по цепочкам") Then
                Call storyFindStop
            End If
            storyFindContCheck = False
```

```
            Exit Function
        End If
    End With
    storyFindContCheck = True
End Function
```

Подпрограмма `storyFindStop` используется в тех случаях, когда поиск по цепочкам заканчивается и необходимо "убрать за собой": при наличии закладки **storyFindFrom** перейти на нее и затем удалить ее и закладки **storyFindEnd** и **storyFindHighlight** (если они установлены в документе), после чего "почистить" окно **Найти и заменить**, удалив значение в поле **Найти** и исключив учет форматирования при поиске.

В отличие от предыдущих подпрограмм подпрограмма `storyFindStop` не предназначена для того, чтобы ее вызывать в качестве макроса. Поэтому в ее первой инструкции перед ключевым словом **Sub** рекомендуется написать ключевое слово **Private**, ограничив ее область видимости *(см. разд. 14.4.3)*, — тогда она не будет "засорять" список макросов, вызываемых в Microsoft Word.

Подпрограмма `storyFindStop` не содержит новых элементов. Инструкции перехода к закладке, возможно, находящейся в другой цепочке, мы подробно обсудили в *гл. 15*. Вам будет полезно, прежде чем посмотреть на листинг 16.31, попытаться написать эту подпрограмму самостоятельно.

Листинг 16.31

```
Private Sub storyFindStop()
'Переход к закладке в точке начала поиска
'Удаление установленных закладок
'Очистка параметров, задающих условия поиска
With ActiveDocument
    If .Bookmarks.Exists("storyFindFrom") Then
        If Not Selection.InStory(.Bookmarks("storyFindFrom").Range) Then
            ActiveWindow.View.Type = wdPrintView
            ActiveWindow.View.Type = wdNormalView
        End If
        .Bookmarks("storyFindFrom").Select
        .Bookmarks("storyFindFrom").Delete
    End If
    If .Bookmarks.Exists("storyFindEnd") Then
```

```
        .Bookmarks("storyFindEnd").Delete
    End If
    If .Bookmarks.Exists("storyFindHighlight") Then
        .Bookmarks("storyFindHighlight").Delete
    End If
End With
Selection.Find.Text = ""
Selection.Find.ClearFormatting
End Sub
```

Самостоятельно проверьте, как работают процедуры `storyFindCont`, `storyFindContCheck` и `storyFindStop`.

Следующая подпрограмма — `storyFindClose`. Она нужна для того, чтобы прервать поиск: попытаться вернуться на исходную позицию, удалить установленные при поиске закладки и очистить параметры поиска. Все эти действия выполняет уже готовая подпрограмма `storyFindStop`. Необходимо только добавить инструкции, обеспечивающие ее взаимодействие с пользователем. Если в документе установлена закладка **storyFindFrom**, следует сообщить пользователю, что поиск выполнен не во всем документе, и запросить подтверждение на выполнение дальнейших действий. Если ее нет, следует сообщить пользователю, что поиск уже закончен. В заключение следует вызвать подпрограмму `storyFindStop`.

Попробуйте написать программу `storyFindClose` самостоятельно и сравните то, что у вас получится, с нашим вариантом (листинг 16.32).

Листинг 16.32

```
Sub storyFindClose()
'Закончить поиск
'Попытаться перейти на исходную позицию
'Удалить закладки и очистить параметры поиска
    With ActiveDocument
        If .Bookmarks.Exists("storyFindFrom") Then
            If MsgBox("Поиск выполнен не во всем документе." & vbCr _
                & vbCr & "Прервать поиск?", vbQuestion + vbYesNo _
                , "Прерывание поиска по цепочкам") = vbNo Then
                Exit Sub
            End If
        Else
```

```
            Call MsgBox("Поиск не начат (нет закладки " _
                & "storyFindFrom)", vbInformation _
                , "Прерывание поиска по цепочкам")
        End If
    End With
    Call storyFindStop
End Sub
```

После того как вы проверите работу подпрограммы `storyFindClose`, можно двигаться дальше. Обратите внимание, у нас уже готово восемь процедур, но к программированию собственно поиска мы еще не приступали. В этом нет ничего удивительного: практика показывает, что разработка элементов программы, относящихся к взаимодействию программы с пользователем — ввода и проверки входной информации, сообщений об ошибках и вывода результатов — составляет "львиную долю" программного кода. И вот сейчас мы приступаем к самому интересному — реализации алгоритма поиска.

Сейчас нам предстоит заменить подпрограмму-заглушку `storyFindStep` реальной подпрограммой. В ней начинается и заканчивается поиск очередного вхождения искомого фрагмента. Программа содержит цикл **Do**, в котором перебираются все цепочки. В этом цикле сначала происходит обращение к функции `storyFindExec`, в которой выполняется поиск в текущей цепочке. Эта функция возвращает одну из трех строк, описывающих результат поиска:

1. "Искать дальше" — надо искать в следующей цепочке — происходит обращение к функции `storyFindNextStory`, которая осуществляет переход к следующей цепочке, выводит необходимые сообщения для пользователя и возвращает одну из трех строк:
   - "Stop" — курсор в области, не входящей в цепочки, или же курсор в конце документа и поиск прекращен пользователем — вызывается подпрограмма `storyFindStop` для завершения поиска по цепочкам и происходит выход из цикла **Do**.
   - "OK" — цикл поиска по цепочкам продолжается.
2. "Выйти и прекратить" — больше вхождений нет — вызывается подпрограмма `storyFindStop` для завершения поиска по цепочкам, выводится сообщение о том, что больше вхождений нет и поиск закончен, и происходит выход из цикла **Do**.
3. "Выйти и продолжать" — вхождение найдено, потом можно продолжить поиск — происходит выход из цикла **Do**.
4. В результате выполнения шага поиска выделяется найденный фрагмент или происходит возврат к закладке **storyFindFrom**. В *гл. 15* мы уже столк-

нулись с проблемой перехода к закладке, расположенной в другой цепочке. Сходные проблемы возникают и при поиске. Поэтому по окончании цикла **Do** мы предусмотрели обращение к подпрограмме storyFindGoto, с помощью которой мы будем решать эти проблемы.

Подпрограммы storyFindGoto и storyFindStep не предполагается применять в качестве макросов, поэтому в их инструкциях **Sub** следует использовать ключевое слово **Private**.

В подпрограмме storyFindStep нет новых элементов, попробуйте запрограммировать ее самостоятельно и сравните то, что у вас получится, с нашим вариантом (листинг 16.33). Обратите внимание, в нашем варианте происходят остановки (инструкции **Stop**) в тех случаях, когда возвращаемые функциями storyFindNextStory и storyFindExec значения не предусмотрены в подпрограмме storyFindStep.

Листинг 16.33

```
Private Sub storyFindStep()
'шаг поиска по цепочкам
Dim strRes As String, NextStoryRes As String
Do 'цикл по цепочкам
    strRes = storyFindExec()
    If strRes = "Искать дальше" Then 'искать в следующей цепочке
        NextStoryRes = storyFindNextStory()
        If NextStoryRes = "Stop" Then
            'пользователь остановил в конце документа или
            'курсор или выделение находится в области,
            'не входящей в цепочки
            Call storyFindStop
            Exit Do
        ElseIf NextStoryRes <> "OK" Then
            Stop
        End If
    ElseIf strRes = "Выйти и прекратить" Then
    'ничего не найдено, больше искать не надо
        Call storyFindStop
        MsgBox "Больше вхождений нет. Поиск закончен." _
                , vbInformation, "Поиск по цепочкам"
        Exit Do
    ElseIf strRes = "Выйти и продолжать" Then
```

```
            Exit Do
        Else
            Stop
        End If
    Loop
    Call storyFindGoto
    End Sub
```

Чтобы проверить, как работает подпрограмма `storyFindStep`, напишем заглушки для функций `storyFindNextStory`, `storyFindExec` и подпрограммы `storyFindGoto` (листинг 16.34).

Листинг 16.34

```
Function storyFindExec() As String
'Выполняет поиск в цепочке и возвращает результат:
    storyFindExec = "Искать дальше"
    'storyFindExec = "Выйти и прекратить"
    'storyFindExec = "Выйти и продолжать"
    'storyFindExec = "Проверка"
End Function
Function storyFindNextStory() As String
'Переставляет курсор в начало следующей цепочки
'Возвращает:
    storyFindNextStory = "Stop"
    'storyFindNextStory = "OK"
    'storyFindNextStory = "Проверка"
End Function
Private Sub storyFindGoto()
' переход к выделенному фрагменту
End Sub
```

Различные варианты строк, возвращаемых функциями, можно выбирать, удаляя или добавляя знаки комментариев перед строками. Обратите внимание на то, что для функции `storyFindNextStory` возвращаемое значение `"OK"` задавать не следует, поскольку в этом случае не произойдет выход из цикла **Do** в подпрограмме `storyFindStep` — подпрограмма зациклится (для тех, кто захочет в этом убедиться, напомним, что прервать выполнение программы можно с помощью сочетания клавиш <Ctrl>+<Break>).

После проверки работы программы `storyFindStep` можно приступить к замене функций-заглушек реальными функциями. Начнем с функции `storyFindExec`, которая выполняет поиск в одной цепочке. Несмотря на то, что в этой функции все программные элементы вам знакомы, создать ее по словесному описанию сложно. Поэтому мы рекомендуем вам разобраться в том, как работает запрограммированный нами вариант, используя листинг 16.35 и приведенное ниже описание.

Листинг 16.35

```
Function storyFindExec() As String
'Выполняет поиск в цепочке и возвращает результат
' "Искать дальше" - надо искать в следующей цепочке
' "Выйти и прекратить" - больше вхождений нет
' "Выйти и продолжать" - вхождение найдено, потом можно продолжить поиск
Dim booFinishStory As Boolean, booFound As Boolean, rng As Range
With ActiveDocument
    Selection.Find.Execute
    booFound = Selection.Find.Found
    If booFound And Selection.Find.Highlight = True And _
            .Bookmarks.Exists("storyFindHighlight") Then
        booFound = False
        Do While Selection.Find.Found
            If storyHighlightFound() Then
                booFound = True
                Exit Do
            End If
            Selection.Find.Execute
        Loop
    End If
    booFinishStory = _
        Selection.InStory(.Bookmarks("storyFindFrom").Range) _
        And .Bookmarks.Exists("storyFindEnd")
End With
If booFound = False And booFinishStory = False Then
    storyFindExec = "Искать дальше"
ElseIf booFound = False And booFinishStory = True Then
    storyFindExec = "Выйти и прекратить"
```

```
ElseIf booFound = True And booFinishStory = False Then
    storyFindExec = "Выйти и продолжать"
ElseIf booFound = True And booFinishStory = True Then
    Set rng = ActiveDocument.Bookmarks("storyFindFrom").Range
    If Selection.IsEqual(rng) Or Selection.Start >= rng.End Then
        storyFindExec = "Выйти и прекратить"
    Else
        storyFindExec = "Выйти и продолжать"
    End If
End If
End Function
```

Поскольку условия поиска уже сохранены в параметрах диалогового окна **Найти и заменить**, для выполнения поиска достаточно одной инструкции *(см. разд. 12.1.2)* `Selection.Find.Execute`.

Для нас важно знать, найден ли искомый фрагмент документа. Для этого предусмотрено свойство `Found` (найдено) объекта `Find` (найти). Оно возвращает логическое значение **`True`**, если фрагмент найден, или значение **`False`**, если искомый фрагмент найти не удалось.

Это логическое значение мы запоминаем в логической переменной `booFound`. Теперь вспомним, что при наличии закладки **storyFoundHighLight** при поиске выделенного цветом текста нам надо учитывать цвет выделения. Если в документе установлена эта закладка, в диалоговом окне **Найти и заменить** установлен параметр **выделение цветом** и `booFound = True`, то следует проверить, есть ли в найденном фрагменте участок, подкрашенный цветом, установленным на инструменте **Выделение цветом**.

Если такой участок есть, то его следует выделить и присвоить переменной `booFound` значение **`True`**, обозначающее, что искомый фрагмент найден. Если такого участка нет, следует повторить поиск в оставшейся части цепочки.

Если цепочка просмотрена до конца, а фрагмент с заданным цветом не найден, следует присвоить переменной `booFound` значение **`False`**, указав тем самым, что результат поиска отрицательный.

Эту задачу выполняет цикл **`Do`**, на каждом шагу которого происходит обращение к функции `storyHighlightFound`, в которой проверяется выделенный фрагмент на наличие в нем участка, подкрашенного заданным цветом, и выделение этого участка при его наличии (об этой функции мы подробно поговорим немного позже).

В результате переменная `booFound` получает значение, по которому можно судить, найден ли искомый фрагмент в данной цепочке.

Однако для принятия решения о том, как поступать дальше, этого нам мало. В частности, необходимо знать, не является ли текущая цепочка последней из тех, которые необходимо просмотреть. Это можно определить по наличию в цепочке закладки **storyFindFrom** и наличию в документе закладки **storyFindEnd**. Вам уже знаком метод `InStory` *(см. разд. 15.14.2)*, позволяющий выяснить, принадлежит ли курсор (или выделение) цепочке, содержащей указанный диапазон, а для проверки наличия в документе закладки вы уже неоднократно применяли свойство `Exists`. Признак того, что цепочка является последней из тех, которые необходимо просмотреть, заносится в переменную `booFinishStory`.

Теперь в зависимости от значений переменных `booFound` и `booFinishStory` можно задать возвращаемое значение функции `storyFindExec`, которое определяет дальнейшее поведение выполняемого макроса. Специального рассмотрения заслуживает случай, когда искомый фрагмент найден в последней цепочке. Если найденный фрагмент документа совпадает с диапазоном закладки **storyFindFrom** или начинается в точке, где заканчивается закладка, либо еще дальше, то поиск следует прекратить. В противном случае поиск можно продолжить в оставшейся части последней просматриваемой цепочки (иными словами, в той цепочке, откуда начат поиск, — она может не совпадать с последней цепочкой документа).

Мы надеемся, что с помощью приведенного описания вы сможете понять, как работает функция `storyFindExec`. Для использованной в ней функции `storyHighlightFound` можно запрограммировать заглушку, которая будет решать поставленную задачу для случая, когда в документе нет смежных участков, подкрашенных разным цветом (листинг 16.36).

Листинг 16.36

```
Function storyHighlightFound() As Boolean
'Проверка наличия в выделенном фрагменте участка,
'подкрашенного заданным цветом, и его выделение
    storyHighlightFound = (Selection.Range.HighlightColorIndex = _
        Options.DefaultHighlightColorIndex)
End Function
```

Эта функция-заглушка возвращает значение **True**, если у найденного (следовательно, выделенного) фрагмента документа номер цвета подкрашивания (`Selection.Range.HighlightColorIndex`) совпадает с номером цвета (`Options.DefaultHighlightColorIndex`), установленным на инструменте **Выделение цветом**. Если номера цветов не совпадают, возвращается значение **False**.

Если в документе есть участок, состоящий из частей, подкрашенных разным цветом, то при поиске выделенного цветом текста этот участок выделяется целиком. Вы уже можете самостоятельно составить простую подпрограмму, которая сообщит вам, что в том случае, когда в документе выделен такой участок, свойство `Selection.Range.HighlightColorIndex` возвращает число 9 999 999, не соответствующее никакому цвету (в документации эту информацию нам найти не удалось). Поэтому, как вы сможете скоро убедиться, при использовании функции заглушки наши макросы не позволяют найти выделенный заданным цветом текст, если к нему примыкает участок, выделенный другим цветом.

Проверку работы функции `storyFindExec` пока отложим, сначала запрограммируем функцию `storyFindNextStory`, которая переставляет курсор в следующую цепочку и возвращает строковое значение "Stop" или "OK" в зависимости от того, какие предполагается после этого выполнить действия в описанной выше подпрограмме `storyFindStep` (см. листинг 16.33).

Работа функции `storyFindNextStory` начинается с обращения к подпрограмме `storyFindStoryTrans`, которая собственно и выполняет переход в следующую цепочку. У подпрограммы `storyFindStoryTrans` два аргумента, с помощью которых она возвращает две логические переменные: `boo` и `booEOF`. Переменная `boo` содержит признак того, что курсор (или выделение) находится в области, входящей в какую-либо цепочку (мы уже упоминали, что при использовании Microsoft Word 97 или Microsoft Word 2000 в документах могут быть области, не принадлежащие никакой цепочке), а переменная `booEOF` — признак того, что поиск выполнен до конца документа, точнее, конца последней его цепочки. К этой подпрограмме мы вернемся позже, а сейчас продолжим описание функции `storyFindNextStory`.

В этой функции после обращения к подпрограмме `storyFindStoryTrans` проверяется значение переменной `boo` и, если это значение **`False`**, выводится сообщение о том, что курсор или выделение находится в области, не входящей в цепочки, и функция завершает работу, возвращая строку "Stop".

Далее проверяется значение переменной `booEOF`. Если просмотрена последняя в документе цепочка, значение этой переменной — **`True`**. В этом случае пользователь получает сообщение о том, что достигнут конец документа, и запрос о необходимости продолжить поиск с начала документа. Если пользователь отвечает отрицательно, функция завершает работу и возвращает значение "Stop". Если пользователь отвечает утвердительно, курсор устанавливается в начало основного текста документа и функция завершает работу, возвращая значение "OK", при котором продолжается цикл по цепочкам. Если значение переменной `booEOF` равно **`False`** (документ еще не просмотрен до конца), функция также возвращает значение "OK".

Для того чтобы вы могли самостоятельно реализовать этот алгоритм, вам необходимо научиться устанавливать курсор в начало основного текста документа.

Мы воспользуемся этим, чтобы рассказать о методе `Range` (диапазон) объекта `Document` (документ). У этого метода два необязательных именованных аргумента: `Start` (начало) и `End` (конец). Метод возвращает заданный этими аргументами диапазон основного текста документа — ссылку на объект класса `Range`. Если аргументы не указаны, то метод возвращает весь основной текст документа.

Обратите внимание на следующие особенности этого метода. Во-первых, то же название имеет и класс, и свойство, возвращающее объекты этого класса. Во-вторых, если аргументы опущены, то обращение к этому методу выглядит так же, как обращение к свойству, например, выражение `ActiveDocument.Range` возвращает диапазон, охватывающий весь основной текст документа, т. е. отличие этого метода от свойства только в том, что метод может иметь аргументы.

Этот метод позволяет установить курсор в начало основного текста активного документа с помощью одной инструкции:

```
ActiveDocument.Range(Start:=0, End:=0).Select
```

Теперь вы можете попробовать запрограммировать функцию `storyFindNextStory` самостоятельно и сравнить то, что у вас получилось, с нашим вариантом, приведенным в листинге 16.37.

Листинг 16.37

```
Function storyFindNextStory() As String
'Переставляет курсор в начало следующей цепочки
'Возвращает
'  "Stop" - курсор в области, не входящей в цепочки, или же курсор
'           в конце документа и поиск прекращен пользователем
'    "OK" - цикл поиска по цепочкам продолжается
Dim rng As Range, boo As Boolean, booEOD As Boolean
Call storyFindStoryTrans (boo, booEOD)
If boo = False Then
    Call MsgBox("Курсор или выделение находится в области," _
    & " не входящей в цепочки." & vbCr & vbCr _
    & "Не удалось найти цепочку с курсором или выделенным текстом" _
    , vbExclamation, "Перебор цепочек")
    storyFindNextStory = "Stop"
```

```
    Exit Function
End If
If booEOD Then 'достигнут конец документа
    If MsgBox("Достигнут конец документа." & vbCr & vbCr _
        & "Продолжить поиск с начала документа?", vbQuestion _
        + vbYesNo, "Переход к следующей цепочке") = vbYes Then
        ActiveDocument.Bookmarks.Add Name:="storyFindEnd" _
                                    , Range:=Selection.Range
        ActiveDocument.Range(Start:=0, End:=0).Select
        storyFindNextStory = "OK"
    Else
        storyFindNextStory = "Stop"
    End If
Else 'конец документа не достигнут
    storyFindNextStory = "OK"
End If
End Function
```

Теперь мы рекомендуем вам без наших комментариев изучить подпрограмму `storyFindStoryTrans` (листинг 16.38). Это будет не сложно, поскольку в ней реализован алгоритм, знакомый вам по подпрограмме `SelectNextStory` (см. листинги 16.16 и 16.17).

Листинг 16.38

```
Private Sub storyFindStoryTrans(boo As Boolean, booEOD As Boolean)
'Переход в следующую цепочку
'Возвращает
'    boo - признак того, что курсор (или выделение), находится
'          в области, входящей в какую-либо цепочку
'    booEOF - признак того, что поиск выполнен до конца последней
'          цепочки документа.
Dim rng As Range
boo = False 'признак наличия курсора или выделения в цепочке
booEOD = True 'признак конца документа
For Each rng In ActiveDocument.StoryRanges
    If boo Then 'курсор или выделение в предыдущей цепочке
        booEOD = False
```

```
            rng.Select
            Selection.Collapse
            Exit For
        End If
        If Selection.InStory(rng) Then
            boo = True 'курсор или выделение в текущей цепочке
        End If
        Do While Not (rng.NextStoryRange Is Nothing)
            Set rng = rng.NextStoryRange
            If boo Then 'курсор или выделение в предыдущей цепочке
                booEOD = False
                rng.Select
                Selection.Collapse
                Exit For
            End If
            If Selection.InStory(rng) Then
                boo = True 'курсор или выделение в текущей цепочке
            End If
        Loop
    Next rng
    End Sub
```

Остались только две подпрограммы-заглушки: `storyFindGoto` (см. листинг 16.34) и `storyHighlightFound` (см. листинг 16.36). Подпрограмма `storyFindGoto` выполняет переход к очередному найденному фрагменту документа. Если бы программное обеспечение Microsoft Word работало должным образом, то эта подпрограмма была не нужна. В результате поиска переход к найденному фрагменту должен происходить автоматически, но, к сожалению, это происходит не всегда. Сейчас мы начнем тестирование макросов, и вы сможете в этом убедиться сами.

Подготовительные действия

В *разд. 15.4.1* мы использовали специально предназначенный для проверки макросов документ debug.doc, содержащий много цепочек. Если этот документ основан на шаблоне MacrosBook.dot, то можно использовать панель **storyFindDebug** с кнопками **Sel**, **Highlight**, **Next**, **Cont**, **Close** для вызова из модуля **storyFindDebug** макросов с именами **storyFindSel**, **storyFindHighlight**, **storyFindNext**, **storyFindCont** и **storyFindClose** соответственно. Если вы созда-

вали документ debug.doc самостоятельно и не пользовались шаблоном MacrosBook.dot, то создайте такую панель сами. С ее помощью будет удобнее проверять работу наших новых макросов.

Проверка 1

1. В документе debug.doc выделите синим цветом два фрагмента, расположенных в колонтитулах, относящихся к различным разделам документа, перейдите при необходимости в режим отображения документа **Разметка страницы** и установите курсор в начале основного текста документа.
2. Нажмите кнопку **Highlight**.

   Мы надеемся, что у вас, так же как и у нас, откроется в отдельной области колонтитул и выделится найденный фрагмент. Это обычное выделение (т. е. не выделение цветом), но поскольку первоначально фрагмент уже был выделен синим цветом, то выделение будет окрашено альтернативным желтым цветом.
3. Нажмите кнопку **Cont**.

   В той же отдельной области откроется другой колонтитул и выделится второй подкрашенный фрагмент.
4. Нажмите кнопку **Cont**.

   Откроется диалоговое окно с вопросом к пользователю (рис. 16.4).

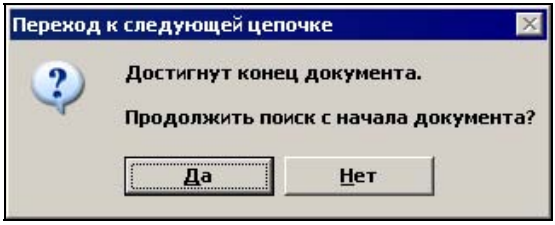


Рис. 16.4. Поиск выполнен в последней цепочке документа, искомый фрагмент не найден

5. Нажмите кнопку **Нет**.

   Работа программы завершится. Можно проверить, что закладки **storyFindFrom** в документе не осталось.

Как видите, макрос работает должным образом. Самостоятельно проверьте, что в режиме отображения документа **Обычный** макрос также работает нормально.

Проверка 2

1. В документе debug.doc по-прежнему выделены синим цветом два фрагмента, расположенные в колонтитулах, относящихся к различным разделам документа. На этот раз откройте первый из этих колонтитулов и установите курсор в его начале. Если вы действуете обычным образом (меню **Вид | Колонтитулы**), то документ после открытия колонтитулов всегда находится в режиме **Разметка страницы** и колонтитул открывается в одной области с документом. Масштаб изображения документа должен быть достаточно большим, чтобы на экране оба колонтитула одновременно не отображались.
2. Нажмите кнопку **Highlight**.

   Выделится найденный подкрашенный фрагмент. Колонтитул по-прежнему открыт в одной области с документом, режим — **Разметка страницы**.
3. Нажмите кнопку **Cont**.

   На экране появляется другой участок документа, на котором нет выделенного текста. Более того, курсор на экране отсутствует.

К сожалению, приходится констатировать, что в приложении Microsoft Word содержится ошибка. Ситуация точно такая же, как и в случае перехода к закладке, установленной в колонтитуле, из другого колонтитула *(см. разд. 15.4.2)*.

Поскольку из основного текста документа переход к выделенному фрагменту все-таки происходит, есть надежда поправить ситуацию. Вспомните, что при разработке функции `storyFindNextStory` мы воспользовались инструкцией, которая позволяет установить курсор в начало основного текста активного документа (см. листинг 16.37):

```
ActiveDocument.Range(Start:=0, End:=0).Select
```

Мы можем запомнить диапазон, в котором находится выделенный участок, перейти в основной текст документа, а затем выделить запомненный диапазон. Заменим в модуле **storyFindDebug** подпрограмму-заглушку `storyFindGoto` подпрограммой, реализующей этот план (листинг 16.39). Это только первый промежуточный вариант подпрограммы `storyFindGoto`, затем мы создадим еще один промежуточный вариант с тем же именем, и только после этого нам удастся создать окончательный вариант. Как вы знаете, а в одном модуле не может быть несколько процедур с одинаковыми именами. Поэтому для первого промежуточного варианта подпрограммы (см. листинг 16.39) мы создали отдельный модуль **Listings_16_39**, из которого эту подпрограмму вы можете скопировать в модуль **storyFindDebug**.

Листинг 16.39

```
Private Sub storyFindGoto()
' переход к выделенному фрагменту (промежуточный вариант 1)
Dim rng As Range
    Set rng = Selection.Range
    ActiveDocument.Range(Start:=0, End:=0).Select
    rng.Select
End Sub
```

Проверка показывает, что такой прием приводит к желаемому результату, но возникает та же неприятность, что и в случае перехода к закладке, установленной в другом колонтитуле. При выполнении этой операции в версиях Microsoft Word 97 или Microsoft Word 2000 надпись в верхней части области колонтитула, содержащая сведения о типе колонтитула и номере раздела, остается прежней, а не изменяется должным образом при переходе к другому колонтитулу, что может дезинформировать пользователя.

Для решения этой проблемы можно воспользоваться тем же средством, которое мы применили в программе `GoToBookmark` (см. листинг 15.15): перед выделением диапазона предварительно перейти в режим **Разметка страницы** (листинг 16.40). Этот второй промежуточный вариант подпрограммы находится в отдельном модуле **Listings_16_40**.

Листинг 16.40

```
Private Sub storyFindGoto()
' переход к выделенному фрагменту (промежуточный вариант 2)
Dim rng As Range
    Set rng = Selection.Range
    ActiveDocument.Range(Start:=0, End:=0).Select
    ActiveWindow.View.Type = wdPrintView
    rng.Select
End Sub
```

При дальнейшей проверке макросов в версиях Microsoft Word 97 или Microsoft Word 2000 мы больше ошибок не обнаружили. Однако при использовании Microsoft Word 2002 или Microsoft Word 2003 возникают новые проблемы. В этих версиях примечания отображаются в окне просмотра, а в режиме **Разметка страницы** примечания отображаются еще и на выносках. Если искомый фрагмент находится в примечаниях, то после выполнения

пробного поиска с помощью наших макросов отображается область примечаний в режиме **Обычный**, а найденный фрагмент на экране не виден и в области примечаний не выделяется. Если перейти в режим **Разметка страницы**, то можно обнаружить на выносках найденный выделенный фрагмент. Трудно объяснить, почему повторение инструкции

```
rng.Select
```

позволяет решить эту проблему. В результате подпрограмма `storyFindGoto` приобретает окончательный вид, показанный в листинге 16.41. Этот вариант содержится в модуле **Listings_16**.

Листинг 16.41

```
Private Sub storyFindGoto()
' переход к выделенному фрагменту
Dim rng As Range
    Set rng = Selection.Range
    ActiveDocument.Range(Start:=0, End:=0).Select
    ActiveWindow.View.Type = wdPrintView
    rng.Select
    rng.Select
End Sub
```

Мы рекомендуем продолжить тестирование и убедиться, что не удается найти выделенный заданным цветом текст, если к нему примыкает участок, выделенный другим цветом.

Чтобы исправить этот недостаток, необходимо заменить заглушку `storyHighlightFound` полноценной функцией, которая будет проверять, находится ли в выделенном тексте фрагмент, подкрашенный заданным цветом, и, в зависимости от результата проверки, возвращать значение `True` или `False`. Кроме того, при наличии такого фрагмента функция должна выделять его.

Мы планируем запрограммировать следующие действия:

1. Если цвет выделенного фрагмента совпадает с цветом, заданным инструментом **Выделение цветом**, вернуть значение `True` и закончить выполнение функции.
2. Если выделенный фрагмент подкрашен одним цветом, вернуть значение `False` и закончить выполнение функции, поскольку с учетом проверки, выполненной в предыдущем пункте, это означает, что заданного цвета в выделенном фрагменте нет.

3. Для каждого знака, содержащегося в выделенном фрагменте, проверить, совпадает ли цвет его подкраски с заданным цветом. Если он совпадает, то выделить все последующие знаки, подкрашенные тем же цветом, вернуть значение **True** и закончить выполнение функции.
4. Если цвет подкраски всех знаков в выделенном фрагменте отличается от заданного цвета, вернуть значение **False** и закончить выполнение функции.

Теперь постепенно будем готовить отдельные инструкции, необходимые для реализации этого плана.

Введем переменную `CurrentHighlight` для номера цвета, заданного инструментом **Выделение цветом**, и переменную `HighlightColor` для номера цвета выделенного фрагмента:

```
Dim HighlightColor As Variant, CurrentHighlight As Variant
```

Как вы уже знаете, чтобы присвоить значение переменной `CurrentHighlight`, можно воспользоваться свойством `DefaultHighlightColorIndex` объекта `Options`:

```
CurrentHighlight = Options.DefaultHighlightColorIndex
```

У объекта `Selection` нет свойства, позволяющего сразу определить цвет выделенного фрагмента. Поэтому сначала надо с помощью свойства `Range` найти диапазон, соответствующий этому выделению, а затем уже воспользоваться свойством `HighlightColorIndex` полученного объекта `Range`. Этот диапазон нам понадобится еще раз, поэтому введем объектную переменную `rngSelection`, в которой запомним ссылку на диапазон выделенного объекта:

```
Dim rngSelection As Range
Set rngSelection = Selection.Range
HighlightColor = rngSelection.HighlightColorIndex
```

Теперь можно реализовать первый пункт нашего плана:

```
If HighlightColor = CurrentHighlight Then
    storyHighlightFound = True
```

Проверить, что выделенный фрагмент подкрашен одним цветом, можно разными способами. Можно воспользоваться результатом нашего эксперимента, показывающим, что при смешанной окраске фрагмента свойство `HighlightColorIndex` возвращает значение 9 999 999. Однако нет никаких гарантий, что в другой версии Microsoft Word получится тот же результат. Мы не рекомендуем без особой нужды пользоваться недокументированными возможностями. Надежнее проверить значения всех констант, соответствующих различным цветам подкраски, и убедиться, что эти константы принимают значения от 0 до 16 (включая признак отсутствия подкраски и

подкраску белым цветом). Тогда инструкции, проверяющие, что выделенный фрагмент подкрашен одним цветом, можно записать следующим образом:

```
ElseIf 0 <= HighlightColor And HighlightColor <= 16 Then
    storyHighlightFound = False
```

Далее предполагается в цикле проверить цвет каждого подкрашенного символа в выделенном участке и в случае совпадения его цвета с указанным задать в качестве возвращаемого значения **True**. Однако сначала надо позаботиться о том, чтобы в случае, когда совпадения не найдется, вернуть значение **False**. Стандартное решение этой задачи состоит в том, чтобы определить логическую переменную, перед выполнением цикла присвоить ей значение **False**, а в случае, если на каком-то шаге обнаружится совпадение цвета, присвоить этой переменной значение **True**. После окончания цикла значение этой переменной можно использовать в качестве возвращаемого значения функции. Инструкции, реализующие этот алгоритм, схематично можно представить следующим образом:

```
Dim boo As Boolean, i As Variant
    lngSelection = <длина выделенного фрагмента>
    boo = False
    For i = 1 To lngSelection
        If <цвет i-го символа> = CurrentHighlight Then
            boo = True
            <выделить последующие знаки, подкрашенные тем же цветом>
            Exit For
        End If
    Next i
    storyHighlightFound = boo
```

Мы уже отмечали, что одни и те же действия можно программировать, используя как объект `Selection`, так и объект `Range`. Воспользуемся случаем, чтобы дать вам возможность попрактиковаться в работе с объектом `Range`. Начнем с вычисления длины выделенного фрагмента, представленного диапазоном, присвоенным переменной `rngSelection`.

Когда мы программировали функцию `storyFindContCheck` (см. листинг 16.30), мы рассказали о свойствах `Start` и `End`, которыми, в частности, обладают объекты `Range`. Напомним, что эти свойства возвращают номера позиций, соответствующие началу и концу диапазона (номер позиции отсчитывается от начала цепочки). С помощью этих свойств длину выделенного фрагмента можно найти посредством следующей инструкции:

```
lngSelection = rngSelection.End - rngSelection.Start
```

Присвоение других значений свойствам `Start` и `End` позволяет изменить границы диапазона, представленного объектом `Range`, однако таким образом нельзя переместить объект в другую цепочку. В качестве примера сформируем диапазон, представляющий *i*-й символ в выделенном фрагменте.

Начать надо с описания новой объектной переменной `rng1` и присвоения ей ссылки на диапазон, находящийся в той же цепочке, в которой находится выделение. После этого можно изменять границы этого диапазона:

```
Dim rngSelection As Range, rng1 As Range
Dim lngSelection As Variant, i As Variant
Set rngSelection = Selection.Range
Set rng1 = Selection.Range
lngSelection = rngSelection.End - rngSelection.Start
For i = 1 To lngSelection
    rng1.End = rngSelection.Start + i
    rng1.Start = rng1.End - 1
```

Обратите внимание, что вместо инструкций

```
Set rngSelection = Selection.Range
Set rng1 = Selection.Range
```

нельзя написать инструкции

```
Set rngSelection = Selection.Range
Set rng1 = rngSelection
```

поскольку тогда в обеих переменных `rng1` и `rngSelection` будут находиться ссылки на один и тот же диапазон. Это, в частности, означает, что при изменении значения `rng1.End` будет синхронно изменяться значение `rngSelection.End`.

Теперь, когда в переменной `rng1` сформирована ссылка на диапазон, представляющий *i*-й символ в выделенном фрагменте, легко записать выражение для цвета подкрашивания этого символа: `rng1.HighlightColorIndex`.

Осталось написать инструкции для выделения последующих символов, подкрашенные тем же цветом. Это можно сделать, например, следующим образом:

```
Do
    rng1.End = rng1.End + 1
    If rng1.HighlightColorIndex <> CurrentHighlight _
            Or rng1.End > rngSelection.End Then
        rng1.End = rng1.End - 1
```

```
        rng1.Select
        Exit Do
    End If
Loop
```

На каждом шаге цикла `Do` мы расширяем диапазон, соответствующий `rng1`, на один символ и, если цвет подкрашивания изменился или диапазон вышел за пределы выделенного участка, то последнее расширение отменяем, выделяем диапазон и прекращаем цикл. Проверка выхода расширенного диапазона за пределы выделенного участка нужна в том случае, когда после выделенного цветом фрагмента находится пробел. Несмотря на то, что такой пробел цветом не выделяется, свойство `HighlightColorIndex` для диапазона, соответствующего этому пробелу, возвращает значение, соответствующее цвету подкрашивания предыдущего символа. К сожалению, такая неожиданная особенность свойства `HighlightColorIndex` не документирована.

Итак, мы научились записывать все инструкции, необходимые для реализации функции `storyHighlightFound`. Попробуйте сами запрограммировать функцию и сравните то, что у вас получится, с нашим вариантом, представленном в листинге 16.42. Обратите внимание, что некоторые инструкции мы записали по-другому.

Листинг 16.42

```
Function storyHighlightFound() As Boolean
'Проверка наличия в выделенном фрагменте участка,
'подкрашенного заданным цветом, и его выделение
    Dim HighlightColor As Variant, CurrentHighlight As Variant
    Dim rngSelection As Range, rng1 As Range
    Dim lngSelection As Variant, i As Variant
    Dim boo As Boolean
    Set rngSelection = Selection.Range
    Set rng1 = Selection.Range
    HighlightColor = rngSelection.HighlightColorIndex
    CurrentHighlight = Options.DefaultHighlightColorIndex
    If HighlightColor = CurrentHighlight Then
        storyHighlightFound = True
    ElseIf 0 <= HighlightColor And HighlightColor <= 16 Then
        storyHighlightFound = False
    Else
        lngSelection = rngSelection.End - rngSelection.Start
        boo = False
```

```
        For i = 1 To lngSelection
            rng1.End = rngSelection.Start + i
            rng1.Start = rng1.End - 1
            If rng1.HighlightColorIndex = CurrentHighlight Then
                boo = True
                Do
                    rng1.End = rng1.End + 1
                    If rng1.HighlightColorIndex <> CurrentHighlight _
                            Or rng1.End > rngSelection.End Then
                        rng1.End = rng1.End - 1
                        rng1.Select
                        Exit For
                    End If
                Loop
            End If
        Next i
        storyHighlightFound = boo
    End If
End Function
```

Прежде чем расстаться с макросами поиска в цепочках, подведем некоторые итоги. Мы создали пять макросов, для вызова которых применяются пять кнопок на панели **storyFindDebug**.

Три кнопки — **Sel**, **Highlight** и **Next** — предназначены для запуска трех типов поиска.

Если искомый фрагмент найден, поиск можно продолжить в оставшейся области документа. Поскольку условия поиска запоминаются в виде параметров диалогового окна **Найти и заменить**, для продолжения всех трех типов поиска используется один и тот же макрос, вызываемый четвертой кнопкой **Cont**.

Когда просмотрена последняя цепочка документа, пользователю задается вопрос о необходимости продолжения поиска с начала документа. Когда документ просмотрен до того места, с которого начинался поиск (это место отмечается закладкой **storyFindFrom**), поиск прекращается.

Поиск можно прекратить, и не просмотрев весь документ до конца. Для того чтобы удалить установленные при поиске закладки и перейти в то место, с которого начинался поиск, предназначена пятая кнопка **Close**.

Макросы запрограммированы с помощью небольших и относительно простых процедур. Чтобы еще раз вспомнить, как они взаимодействуют, по-

смотрите табл. 16.3. Перечень всех процедур и их краткие описания расположены в первой и второй колонках таблицы. В третьей колонке для каждой процедуры приведены имена вызываемых из нее процедур.

Таблица 16.3. *Список процедур, использованных в макросах поиска по цепочкам*

Процедура	Описание	Вложенные процедуры
Подпрограммы-макросы		
`StoryFindSel`	Запуск поиска выделенного текста	`storyFindStartCheck`, `storyFindStep`
`storyFindHighlight`	Запуск поиска выделенного цветом текста	`storyFindStartCheck`, `storyFindStep`
`storyFindNext`	Запуск поиска с условиями, заданными в окне **Найти и заменить**	`storyFindStartCheck`, `storyFindStep`
`storyFindCont`	Поиск следующего вхождения	`storyFindContCheck`, `storyFindStep`
`storyFindClose`	Завершение поиска по цепочкам	`storyFindStop`
Вложенные подпрограммы и функции		
`storyFindStartCheck`	Проверка запуска поиска	
`storyFindContCheck`	Проверка поиска следующего вхождения	`storyFindStop`
`storyFindStep`	Шаг поиска следующего вхождения. Цикл по цепочкам	`storyFindExec`, `storyFindNextStory`, `storyFindStop`, `storyFindGoto`
`storyFindExec`	Поиск в одной цепочке	`storyHighlightFound`
`storyFindNextStory`	Переход к следующей цепочке и анализ результатов перехода	`storyFindStoryTrans`
`storyFindStoryTrans`	Собственно переход к следующей цепочке	
`storyFindStop`	Действия по прекращению поиска	
`storyFindGoto`	Переход к найденному фрагменту	
`storyHighlightFound`	Проверка наличия в выделенном фрагменте участка, подкрашенного заданным цветом, и его выделение	

16.6. Что нового в этой главе

В этой главе решаются задачи, связанные с поиском в документах, содержащих не только основной текст, но и колонтитулы, сноски, примечания или надписи. Сначала объясняется, что поиск в таких документах в Microsoft Word реализован недостаточно хорошо и подпрограммы, полученные при автоматической записи макросов, в которых использован поиск в таких документах, не работают должным образом. В данной главе рассказывается о том, что программирование позволяет решать проблемы, связанные с поиском в таких документах. Основной материал главы посвящен средствам работы с цепочками. Рассказывается о типах цепочек, о свойстве `StoryType` и семействе `StoryRanges`, позволяющих выяснять типы имеющихся в документах цепочек и находить в документе цепочки различных типов. Чтобы подготовить читателя к составлению программ, связанных с перебором всех имеющихся в документе цепочек, на конкретных примерах рассказывается о переходе к следующей цепочке данного типа, о пустом объекте **`Nothing`**, об операции **`Is`**. После того как составлена программа вычисления количества цепочек в документе, по той же схеме составляются макросы удаления выделения цветом во всех цепочках и выделения текущим цветом всех вхождений выделенного цвета. При этом читатель получает новые сведения о работы с объектами `Range` и `Find`. Далее полученные навыки закрепляются при создании макросов перехода к следующей и предыдущей цепочкам. Последний крупный раздел главы посвящен макросам, которые начинают поиск с текущего положения курсора и останавливаются, если находят в тексте искомый текст. Эта более сложная задача разбивается на несколько простых подзадач. Новых программных средств для решения этой задачи почти не потребуется, в основном это повторение и практическое применение пройденного материала в условиях, приближенных к реальным.

В заключение приведем сводку новых программных элементов, с которыми вы познакомились в этой главе (табл. 16.4, 16.5).

Таблица 16.4. *Новые объекты и свойства*

Объект	Свойство	Константа
`Range`	`StoryType`	`wdMainTextStory`
		`wdFootnotesStory`
		`wdEndnotesStory`
		`wdCommentsStory`
		`wdTextFrameStory`
		`wdEvenPagesHeaderStory`
		`wdPrimaryHeaderStory`

Таблица 16.4 *(окончание)*

Объект	Свойство	Константа
		wdEvenPagesFooterStory wdPrimaryFooterStory wdFirstPageHeaderStory wdFirstPageFooterStory wdFootnoteSeparatorStory wdFootnoteContinuationSeparatorStory wdFootnoteContinuationNoticeStory wdEndnoteSeparatorStory wdEndnoteContinuationSeparatorStory wdEndnoteContinuationNoticeStory
Document	StoryRanges	
StoryRanges	Count	
Range	NextStoryRange Find	
Find	HighLight	**True** **False** wdUndefined
	Found	**True** **False**
Global	StatusBar	
Selection Bookmark Range	Start End	
Nothing		

Таблица 16.5. *Новые методы*

Объект	Метод	Аргумент
Range	Select	
Document	Range	Start End

Заключение

Авторы прочитанной вами книги берут на себя смелость утверждать, что детище их совместных усилий получилось непохожим ни на одну из множества публикаций по VBA. Если вы прочли из нее хотя бы несколько глав, то сможете сами оценить ее преимущества. Мы же хотим поговорить о недостатках книги, которые, в некотором смысле, являются продолжением ее достоинств. Суть в том, что она не предназначена для подготовки программистов, в ней нет последовательного изложения материала, ни один из аспектов, затронутых в этой книге, не раскрыт полностью.

Если у нашего читателя пробудился интерес к макросам и он захочет в своей практической деятельности применить навыки, полученные при изучении нашей книги, то цель, которую мы поставили перед собой, когда писали эту книгу, можно считать достигнутой. Однако такому читателю почти наверняка потребуются дополнительные источники информации. Мы это предвидели с самого начала и старались подготовить читателя к изучению других публикаций по VBA.

Вам придется самостоятельно решить, что вы будете читать дальше. Это, разумеется, зависит от ваших потребностей и ваших возможностей. Мы можем лишь поделиться собственным опытом.

Когда вы заходите в книжный магазин или начинаете поиск в Интернете, у вас сразу возникает вопрос: каким образом по названию публикации можно узнать, что речь в ней будет идти о создании макросов в Microsoft Word.

Прежде чем ответить на этот вопрос, мы хотим обратить ваше внимание на то, что следует четко различать собственно язык VBA, поддержка которого встроена во многие приложения, и дополнительные *библиотеки объектов*, относящиеся к конкретным приложениям, таким как Microsoft Word и Microsoft Excel. Это означает, что в книге, посвященной программированию в Microsoft Access, обязательно будет дано описание языка VBA, но вряд ли вы найдете в этой книге даже упоминания об объектах Microsoft Word. В то же время, если на обложке книги, посвященной VBA, названия конкретного приложения нет, скорее всего, авторы не обходят стороной вопросы применении VBA в приложениях Microsoft Word и Microsoft Excel. Наиболее серьезные книги по этой тематике содержат в названии упоминание о средствах разработки в среде Microsoft Office.

Если в названии нет таких слов, как VBA, макросы, программирование или разработка, а просто говорится о работе в приложении Microsoft Word или в системе Microsoft Office, то, скорее всего, о программировании речь не пойдет. Возможно, будет рассказано, как создать макрос в режиме автоматической записи, как перейти в редактор Visual Basic, могут быть освещены вопросы безопасности и работа с организатором. Однако это не значит, что вам такие материалы читать не нужно. При чтении нашей книги вы не могли не заметить, что для создания макросов необходимо хорошо знать возможности Microsoft Word и уметь их применять. Это необходимое, хотя и недостаточное условие. Вообще говоря, справочная система, встроенная в Microsoft Word, содержит почти всю необходимую информацию по работе с Microsoft Word. У подавляющего большинства наших читателей есть доступ в Интернет и возможность пользоваться дополнительными ресурсами на веб-узле Microsoft Office Online, находящимся по адресу **http://office.microsoft.com/**.

В нашей книге мы привели лишь основные сведения по языку VBA и вы получили только начальные навыки программирования. Можно воспользоваться тем, что в русскоязычной версии Microsoft Office 97 редактор Visual Basic и справка по нему были на русском языке. Если вы установите на своем компьютере эту версию (наш опыт показывает, что в одной операционной системе можно установить несколько версий Microsoft Office), то увидите, что редактор Visual Basic изменился не очень сильно. Мы надеемся, что тем, кто изучил нашу книгу, встроенную справку будет читать значительно легче. Книг с описанием VBA и редактора Visual Basic очень много. Например:

- Кузменко В. Г. Программирование на VBA 2003. — БИНОМ, 2004.
- Каммингс Стив. VBA для "чайников". — Диалектика, 2001.

В этих книгах, рассчитанных на начальную подготовку специалистов по программированию в среде Microsoft Office, подробно описаны многие возможности языка VBA и, кроме того, рассказывается о его применении к разработке макросов в Microsoft Word.

Краткое, но достаточно полное описание языка VBA и редактора Visual Basic дано в следующей книге, написанной для квалифицированных пользователей и разработчиков офисных приложений:

- Матросов А. В. и др. Microsoft Office XP: разработка приложений. — СПб.: БХВ-Петербург, 2003.

Бегло ознакомившись с возможностями языка VBA, вы сможете сами решать, какие из них вам следует использовать в своей работе, и по мере необходимости будете их осваивать.

Помимо языка VBA вам потребуются сведения об объектах, содержащихся в объектной библиотеке Microsoft Word. Перед тем как приступать к созданию

макросов, надо научиться работать с соответствующими объектами (командами, диалоговыми окнами и т. п.) в самом приложении Microsoft Word, об этом мы уже говорили. Чтобы найти объект в объектной библиотеке, обычно достаточно создать макрос в режиме автоматической записи и посмотреть полученную подпрограмму. После этого можно воспользоваться справкой, встроенной в редактор Visual Basic (во всех версиях Microsoft Office она на английском языке). Обратите внимание, в Microsoft Word 2000, чтобы вызвать справку, содержащую описание объектов Microsoft Word, необходимо в окне модуля выделить элемент, относящийся к одному из этих объектов, например, `Selection`, и нажать клавишу <F1>. Иначе на экран будет выведена справка без разделов, относящихся к объектам Microsoft Word.

К сожалению, мы не смогли найти книги на русском языке, содержащей подробные сведения обо всех объектах объектной библиотеки Microsoft Word.

Из многочисленных ресурсов Интернета, посвященных разработке приложений на языке VBA, укажем только веб-узел "Microsoft Office Extensions. Разработчикам", находящийся по адресу **http://www.microsoft.ru/offext/developers/**.

На этом веб-узле имеется много статей и ссылок, в том числе на публикации в компьютерных журналах.

В добрый путь!

Указатели

Словарь-указатель английских терминов и слов языка Visual Basic

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

Указатель элементов VBA

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

А

И

К

(окончание рубрики см. на с. 452)

M

O

C

Т

Ф

Предметный указатель

Г

Д

З

И

К

Л

М

Н

О

П

Р

С

T

(окончание рубрики см. на с. 464)

У

Ф

Х

Ц

Ч

Ш

Э

Ю, Я