Опыт не требуется

Книги этой серии наглядно свидетельствуют, что новичка можно научить языку и хорошему стилю программирования, не повергая его в тоску и уныние.
– Лу Гринзоу, обозреватель Dr. Dobb's Journal.

Исходные коды находятся на дискете

Сэм А. Аболрус

Программирование на PASCAL

СИМВОЛ

По договору между издательством «Символ-Плюс» и Интернет-магазином «Books.Ru – Книги России» единственный легальный способ получения данного файла с книгой ISBN 5-93286-057-X, название «Программирование на Pascal, 3-е издание» – покупка в Интернет-магазине «Books.Ru – Книги России». Если Вы получили данный файл каким-либо другим образом, Вы нарушили международное законодательство и законодательство Российской Федерации об охране авторского права. Вам необходимо удалить данный файл, а также сообщить издательству «Символ-Плюс» (piracy@symbol.ru), где именно Вы получили данный файл.

Learn Pascal in Three Days

Third Edition

Sam A. Abolrous

Wordware Publishing, Inc.

Опыт не требуется

Программирование на Pascal

Третье издание

Сэм А. Аболрус

Символ®

Санкт-Петербург — Москва
2003

Серия «Опыт не требуется»

Сэм А. Аболрус

Программирование на Pascal, 3-е издание

Перевод А. Петухова

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Научный редактор	*В. Озеров*
Редактор	*В. Овчинников*
Корректура	*С. Беляева*
Верстка	*Н. Гриценко*

Аболрус С.

Программирование на Pascal, 3-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2003. – 328 с., ил.

ISBN 5-93286-057-X

Книга Сэма Аболруса «Программирование на Pascal» – третье, обновленное издание одного из лучших руководств по Паскалю, ставшего бестселлером. Она идеально подходит для приобретения базовых знаний о структурном программировании. Легкий стиль изложения и большое количество примеров и упражнений помогут начинающим программистам изучить язык за очень короткий срок. Начав с простейших программ обработки числовых данных и вывода на печать, вы научитесь писать реальные приложения, которые будут выполняться на любой платформе.

Прилагаемая дискета содержит исходный код примеров, ответы к упражнениям и файлы, необходимые для их выполнения.

ISBN 5-93286-057-X
ISBN 1-55622-805-8 (англ)

© Издательство Символ-Плюс, 2003

Authorized translation of the English edition © 2002 Wordware Publishing Inc. This translation is published and sold by permission of Wordware Publishing Inc., the owner of all rights to publish and sell the same.

Все права на данное издание защищены Законодательством РФ, включая право на полное или частичное воспроизведение в любой форме. Все товарные знаки или зарегистрированные товарные знаки, упоминаемые в настоящем издании, являются собственностью соответствующих фирм.

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, edit@symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Налоговая льгота – общероссийский классификатор продукции
ОК 005-93, том 2; 953000 – книги и брошюры.
Подписано в печать 15.08.2003. Формат 70x100$^{1}/_{16}$. Печать офсетная.
Объем 20,5 печ. л. Тираж 3000 экз. Заказ N
Отпечатано с диапозитивов в Академической типографии «Наука» РАН
199034, Санкт-Петербург, 9 линия, 12.

Оглавление

Об авторе 9

Предисловие 10

1. Привет, Паскаль 11

1-1. Ваша первая программа на Паскале 11
1-2. Вывод текста: WRITELN, WRITE 13
1-3. Обработка чисел 14
1-4. Переменные 18
1-5. Именованные константы 21
1-6. Преобразование типов: ROUND, TRUNC 23
1-7. Чтение с клавиатуры: READLN, READ 24
1-8. Форматирование вывода 25
Заключение 27
Упражнения 28
Ответы 28

2. Элементы языка 29

2-1. Стандартные типы данных и функции 29
2-2. Числовые типы данных 30
2-3. Стандартные арифметические функции 31
2-4. Символьный тип: CHAR 36
2-5. Строковый тип 40
2-6. Тип BOOLEAN 42
Заключение 46
Упражнения 48
Ответы 48

3. Решения 49

3-1. Принятие решений 49
3-2. Простое решение: IF-THEN 50
3-3. Конструкция IF-THEN-ELSE 53

3-4. Цепочки ELSE-IF 55
3-5. Вложенные условия 57
3-6. Множественный выбор: CASE 61
3-7. Безусловный переход: GOTO 64
3-8. Возможности Турбо Паскаля: EXIT, CASE-ELSE 66
Заключение 67
Упражнения 69
Ответы 70

4. Циклы 71

4-1. Построение циклов 71
4-2. Цикл FOR 72
4-3. Пошаговое выполнение вверх и вниз 76
4-4. Вложенные циклы 78
4-5. Цикл WHILE 79
4-6. Цикл REPEAT 82
Заключение 84
Упражнения 85
Ответы 86

5. Архитектура данных 87

5-1. Порядковые типы данных 87
5-2. Секция TYPE 91
5-3. Массивы как структуры данных 94
5-4. Одномерные массивы 96
5-5. Двумерные массивы 103
Заключение 107
Упражнения 108
Ответы 109

6. Обработка текста 110

6-1. Работа с текстовыми данными 110
6-2. Советы по применению оператора OUTPUT 110
6-3. Советы по применению оператора INPUT 111
6-4. Чтение текстовой строки: EOLN 120
6-5. Чтение текстового файла: EOF 121
6-6. Обработка строк 122
6-7. Строковые функции и процедуры 125
Заключение 128
Упражнения 128
Ответы 129

7. Архитектура программы 130
7-1. Программы и подпрограммы 130
7-2. Процедуры 130
7-3. Глобальные и локальные переменные 136
7-4. Функции 138
7-5. Советы относительно области видимости переменных 140
7-6. Рекурсия 142
Заключение 143
Упражнения 144
Ответы 145

8. Множества и записи 146
8-1. Множества 146
8-2. Объявление и присваивание множеств 147
8-3. Множественные операторы и операции 149
8-4. Записи 153
8-5. Вложенные записи 158
Заключение 160
Упражнения 161
Ответы 163

9. Файлы и приложения 165
9-1. Файлы данных 165
9-2. Файлы типа TEXT 166
9-3. Чтение TEXT-файла 166
9-4. Вывод TEXT-файла на экран 172
9-5. Создание TEXT-файла: REWRITE 175
9-6. Нетекстовые файлы 182
9-7. Использование переменной файлового буфера 189
Заключение 190
Упражнения 191
Ответы 192

10. Использование вариантных записей 194
10-1. Вариантные записи 194
10-2. Пример: Улучшенная система расчета зарплаты 196
10-3. Удаление записей из файла 203
10-4. Обновление записей 212
10-5. Повышение модульности программы 215
Заключение 225
Упражнения 225
Ответы 226

11. Указатели и связанные списки 227
11-1. Динамическое выделение памяти 227
11-2. Указатели 227
11-3. Основы связанных списков 234
11-4. Поиск в списке 244
11-5. Удаление узлов из списка 251
Заключение 261
Упражнения 262
Ответы 263

A. Таблица ASCII-кодов 265

B. Зарезервированные слова и стандартные идентификаторы . . 269

C. Ответы к заданиям 271

О дискете 319

Алфавитный указатель 320

Об авторе

Сэм Аболрус (Sam Abolrous), программист с большим опытом дизайна и разработки программного обеспечения, получил степень бакалавра инженера-электрика в университете Александрии, Египет.

Аболрус публиковался в ведущих журналах по программированию, написал более 50 книг в компьютерной области от Кобола до C++, включая «Learn C in Three Days» и «Learn Pascal», опубликованные издательством Wordware Publishing. Он разработал множество программ в области исследования слуха в медицинском центре университета штата Луизиана (Louisiana State University Medical Center, LSUMC). В настоящее время работает программистом в корпорации Microsoft.

Благодарности

Я бы хотел поблагодарить мою дочь Салли Аболрус (Sally Abolrous) за помощь при редактировании этой книги.

Предисловие

На примерах, приведенных в этой книге, можно изучить Паскаль за очень короткий срок. Вы начнете с простейших программ обработки числовых данных и вывода на печать, а закончите созданием реальных приложений с использованием структурного программирования.

Паскаль был разработан в начале 70-х годов Николасом Виртом (Niklaus Wirth), швейцарским программистом, и назван в честь французского математика Блеза Паскаля (Blaise Pascal, 1623–1662). Стандарт языка появился в 1983 году и был утвержден Институтом инженеров по электротехнике и электронике (IEEE) и Американским национальным институтом стандартов (ANSI). С ростом использования микрокомпьютеров язык подвергся изменениям и дополнениям, из которых наиболее популярны UCSD Pascal (разработан Калифорнийским университетом в Сан Диего) и Turbo Pascal фирмы Borland International.

Цель книги – научить писать переносимые, платформо-независимые программы с использованием, главным образом, стандартов IEEE/ANSI. Мы обсудим также новые возможности языка, ссылаясь на источники. Здесь не рассматриваются подробно непереносимые части языка (такие как графика), но зато уделяется внимание мощным возможностям современных реализаций, полезным при обработке данных. Программы, приведенные в книге, компилировались с помощью Turbo Pascal, но вы вправе использовать компилятор по своему выбору. Лишь в редких случаях потребуются небольшие изменения, к которым есть соответствующие комментарии.

Книга содержит ряд специальных обозначений:

Примечания. Содержат дополнительную информацию по сложным темам.

Предупреждения. Предостерегают о возможной опасности совершения ошибки.

Советы. Содержат конкретные рекомендации, облегчающие понимание материала и правильное выполнение примеров.

1

Привет, Паскаль

1–1. Ваша первая программа на Паскале

Программа на Паскале может быть простой, как в примере 1-1. Она выводит на экран фразу «Привет всем».

```
{ ---------------------------- Пример 1-1 ----------------------------- }
PROGRAM FirstProgram(OUTPUT);
BEGIN
   WRITELN('Привет всем')
END.
```

Большая или маленькая, программа на Паскале должна иметь определенную структуру. В примере программа состоит, главным образом, из одного оператора (WRITELN), который и делает всю работу – выводит на экран содержимое в скобках. Оператор помещен в блок, начинающийся с ключевого слова BEGIN и заканчивающийся ключевым словом END. Он называется *основным программным блоком* (или просто *программным блоком*) и обычно представляет основной алгоритм обработки данных.

Комментарии

Рассмотрим первую строку программы:

```
{ ---------------------------- Пример 1-1 ----------------------------- }
```

Это *комментарий*, и он всегда игнорируется компилятором. Можно располагать комментарии в любом месте программы на Паскале между фигурными скобками {} или между символами (* и *), например так:

```
(* Это комментарий *)
```

Заголовок программы

Вторая строка называется *заголовком программы*. Она начинается с ключевого слова PROGRAM, затем следуют пробел и имя программы (FirstProgram). Имя программы определяется пользователем. Определенные пользователем слова называются в Паскале *идентификаторами*. Идентификатор обязательно начинается с буквы и может состоять из любого количества букв и цифр (в Турбо Паскале он может также содержать символ подчеркивания). Для программы можно выбрать любое осмысленное имя, но не рассчитывайте, что пройдут такие имена, как «BEGIN» или «PROGRAM». Это *зарезервированные слова*, которые могут быть использованы только в определенных местах программы. Зарезервированные слова Паскаля собраны в приложении В.

За именем программы в скобках следует слово OUTPUT, которое завершается точкой с запятой:

```
PROGRAM FirstProgram(OUTPUT);
```

Ключевое слово OUTPUT сообщает компилятору, что программа будет выводить данные (например, на экран). Его противоположностью является слово INPUT (обозначающее, например, ввод с клавиатуры). Слова OUTPUT и INPUT называются *файловыми параметрами*. Программа может выполнять и ввод и вывод, и в этом случае параметры принимают вид:

```
PROGRAM FirstProgram(INPUT,OUTPUT);
```

В Турбо Паскале заголовок необязателен. Можно пропустить всю строку и начать программу со слова BEGIN или указать имя программы без параметров, например так:

```
PROGRAM FirstProgram;
```

Синтаксис и соглашения

Наиболее важный элемент синтаксиса – это точка с запятой после заголовка программы (которая используется в качестве разделителя) и точка после слова END, завершающая программу. Общепринято писать ключевые слова Паскаля в верхнем регистре, а слова, определенные пользователем (идентификаторы), – в нижнем регистре, начиная с заглавной буквы. Если имя программы состоит из двух и более слов (как в нашем случае), то каждое слово начинается с заглавной буквы. Таким образом, в программах, написанных на Паскале, можно встретить такие идентификаторы:

Wages

PayRoll

HoursWorkedPerWeek

Это всего лишь соглашение, призванное облегчить чтение программы. Компилятор Паскаля не чувствителен к регистру, поэтому всю программу можно написать и в нижнем (как в примере 1-2) и в верхнем (как в примере 1-3) регистре. Все три программы скомпилируются и будут работать.

```
{ ------------------------------ Пример 1-2 ------------------------------ }
program firstprogram(output);
begin
   writeln('Привет всем')
end.
```

```
{ ------------------------------ Пример 1-3 ------------------------------ }
PROGRAM FIRSTPROGRAM(OUTPUT);
BEGIN
   WRITELN('Привет всем')
END.
```

Все пустые строки, отступы и пробелы (за исключением тех, которые стоят после ключевых слов) необязательны, но такой стиль является хорошей программистской привычкой, позволяющей писать хорошо организованные и легко читаемые программы.

1–2. Вывод текста: WRITELN, WRITE

Как показано в примере 1-4, чтобы вывести на экран несколько строк текста, необходимо написать оператор WRITELN для каждой строки. Не забудьте заключить текстовые строки в кавычки.

Для того чтобы сберечь ваше время и силы, на прилагаемую к книге дискету помещены исходные коды примеров и решения заданий. Ознакомьтесь с файлами Readme.txt или Readme.htm, имеющимися на дискете. В них приведены инструкции по установке файлов на жесткий диск.

```
{ ------------------------------ Пример 1-4 ------------------------------ }
PROGRAM LinesOfText(OUTPUT);
BEGIN
   WRITELN('Привет всем');
   WRITELN('Как вы сегодня?');
   WRITELN('Вы готовы к изучению Паскаля?')
END.
```

Теперь в программе больше одного оператора. Все операторы разделяются точкой с запятой. Это единственный признак, по которому компилятор может определить, где заканчивается оператор, хотя для последнего оператора в программном блоке точку с запятой можно опустить.

После компиляции программа выведет на экран:

```
Привет всем.
Как вы сегодня?
Вы готовы к изучению Паскаля?
```

Оператор WRITELN выводит строку текста и переходит на новую строку (перевод строки и возврат каретки). Для того чтобы вывести две строки в одной, надо применить оператор WRITE, как показано в следующей программе:

```
{ ------------------------------ Пример 1-5 ------------------------------ }
PROGRAM TwoLines(OUTPUT);
BEGIN
   WRITE('Привет всем. ');
   WRITELN('Как вы сегодня?);
   WRITELN('Вы готовы к изучению Паскаля?')
END.
```

Вывод программы:

```
Привет всем. Как вы сегодня?
Вы готовы к изучению Паскаля?
```

Как видите, второе предложение выводится в той же строке, что и первое, поскольку для вывода первой строки использовался оператор WRITE. Это единственное различие между операторами вывода WRITE и WRITELN. Для вывода пустой строки напишите оператор:

```
WRITELN;
```

Задание 1–1

Напишите программу на Паскале, которая выводит на экран следующий текст:

```
Wordware Publishing, Inc.
-------------------------
2320 Los Rios Boulevard
Plano, Texas 75074
```

1–3. Обработка чисел

Любой программе проще всего работать с числами. Оператор WRITELN (или WRITE) можно использовать и для вывода чисел, и для вычисления выражений. Арифметические выражения конструируются с помощью *арифметических операторов*:

- \+ сложение
- – вычитание
- * умножение
- / деление

Взгляните на примеры:

```
WRITELN(123);
WRITELN(1.23 * 4);
```

В первом примере выводится число, заключенное в скобки. Во втором примере выводится результат умножения двух чисел. Заметьте, что для численных значений, в отличие от текстовых строк, кавычки не используются.

Применив запятую в качестве разделителя, можно в одном операторе WRITELN вывести и числа и текст:

```
WRITELN('Результат=', 125 * 1.75);
```

Следующая программа вычисляет два выражения (умножение и деление) и отображает результат, предваряя его соответствующим текстом.

```
{ ------------------------------ Пример 1-6 ------------------------------}
PROGRAM CrunchNumbers(OUTPUT);
BEGIN
   WRITELN('Я легко могу обрабатывать числа.');
   WRITELN('Это умножение 50x4:',50*4);
   WRITELN('..а это деление 2400/8:',2400/8)
END.
```

Вывод этой программы:

```
Я легко могу обрабатывать числа.
Это умножение 50x4:200
..а это деление 2400/8: 3.0000000000E+02
```

Умножение выполнилось с ожидаемым результатом. Два операнда (50 и 4) были целыми числами, и результат (200) тоже целое число. Однако формат результата деления требует пояснений.

Целые и вещественные числа

Деление, выполненное оператором /, называется *вещественным делением* и всегда дает в результате вещественное число. Вещественные числа могут быть записаны в форме с фиксированной точкой, например 300.0, или в экспоненциальной форме (scientific), например 3.0E+02, но в Паскале по умолчанию вещественные числа выводятся всегда в экспоненциальной форме. Число в экспоненциальной форме состоит из двух частей, разделенных буквой «E» (или «e»). Левая часть называется *мантиссой* и представляет собой число со знаком, а правая называется *экспонентой*. Экспонента – это степень десяти, которая определяет положение десятичной точки. Так, в этом примере число:

3.0000000000E+02 – это то же самое, что и 3×10^2.

Это же число в форме с фиксированной точкой будет выглядеть так:

300.0

Если перед экспонентой стоит минус:

3.124E–2

то десятичная точка сдвигается в данном случае на две позиции влево. Тогда число будет таким:

0.03124

В случае отрицательного числа минус размещается перед мантиссой:

–0.0124E–02

Если число положительное, то знак перед мантиссой не ставится:

1.23E02

Оператор деления (/) называется *оператором вещественного деления*, так как независимо от типа операндов результат всегда вещественный.

Для *целочисленного деления* надо применять оператор DIV, например:

```
WRITELN(2400 DIV 8);
```

В результате получим 300.

При целочисленном делении дробные части отбрасываются, например:

```
WRITELN(9 DIV 4);    результат выполнения равен 2.
```

Другой важный оператор предназначен для получения остатка целочисленного деления (деление по модулю), например:

```
WRITELN(9 MOD 4);    результат выполнения равен 1,
WRITELN(3 MOD 4);    результат выполнения равен 3.
```

Операторы DIV и MOD принимают только целые операнды и выдают целый результат.

Остальные операторы (+, – и *) выдадут вещественный результат, если хотя бы один из операндов будет вещественным.

Задание 1–2

Вычислите следующие выражения и напечатайте результат в виде целого числа (если результат целочисленный) или в виде вещественного числа (если результат вещественный):

A. 144 / 12 B. 144 DIV 12

C. 17 MOD 5 D. 3 MOD 5

E. 3e+02 + 3 F. 345E–01 – 1

Вычисление арифметических выражений

При создании более сложных арифметических выражений необходимо следить за приоритетом входящих в выражение операторов. Взгляните на эти два выражения:

```
2 + 10 / 2
(2 + 10) / 2
```

Оба выражения состоят из одних и тех же чисел и операторов, но первое равно 7, а второе – 6. Дело в том, что в первом выражении деление предшествует сложению, а во втором для изменения порядка вычислений используются скобки, при этом первым вычисляется выражение в скобках. В основном арифметические операторы Паскаля имеют два уровня приоритета: *высокий* и *низкий*.

Операторы + и – имеют низкий приоритет, все остальные – высокий. Если два оператора в выражении имеют одинаковый приоритет, они выполняются слева направо. Рассмотрим пример:

```
5 + 3 * 2 - 6 DIV 2
```

Первой должна быть выполнена операция умножения:

```
5 + 6 - 6 DIV 2
```

Вторая операция высшего приоритета – деление:

```
5 + 6 - 3
```

Оставшиеся две операции имеют равный приоритет. Они выполняются слева направо, давая в результате:

```
8
```

Когда для изменения порядка вычислений используются скобки, они образуют подвыражение, которое вычисляется первым. Рассмотрим тот же пример с вложенными скобками:

```
((5 + 3) * 2 - 6) DIV 2
```

Это выражение вычисляется по шагам:

```
(8 * 2 - 6) DIV 2
(16 - 6) DIV 2
10 DIV 2
5
```

Приоритеты и свойства арифметических операторов приведены в табл. 1-1.

Знаки + и – используются в качестве *унарных операторов* (для обозначения положительных и отрицательных чисел). Унарные операторы имеют тот же низкий приоритет, что и бинарные операторы + и –. Если бинарный оператор предшествует унарному, например 5*–4, то унарный оператор вместе с числом необходимо заключить в скобки: 5*(–4). Компилятор может принять первую форму записи, но лучше ее не использовать.[1]

[1] Современные компиляторы языка Паскаль не требуют при совершении арифметических действий заключать отрицательные числа в скобки. – *Примеч. науч. ред.*

Таблица 1-1. Арифметические операторы

Оператор	Арифметическая операция	Операнды	Результат	Приоритет
+	Сложение	REAL/INTEGER	REAL/INTEGER	Низкий
–	Вычитание	REAL/INTEGER	REAL/INTEGER	Низкий
*	Умножение	REAL/INTEGER	REAL/INTEGER	Высокий
/	Вещественное деление	REAL/INTEGER	REAL	Высокий
DIV	Целочисленное деление	INTEGER	INTEGER	Высокий

Задание 1–3

Вычислите следующие выражения и напечатайте результат в виде целого числа (если результат целочисленный) или в виде вещественного числа (если результат вещественный):

A. 15 – 15 DIV 15
B. 22 + 10 / 2
C. (22 + 10) / 2
D. 50 * 10 – 4 MOD 3 * 5 + 80

1–4. Переменные

Данные хранятся в памяти по определенным адресам. Однако программисты ссылаются на них с помощью переменных. При использовании переменных в программе им отводится определенное место в памяти. Значением переменной в действительности является содержимое этого участка памяти. При обработке данных программой содержимое любого участка памяти может измениться, меняя, таким образом, значение соответствующей переменной. Имена (идентификаторы) назначаются переменным в соответствии с упомянутыми ранее правилами.

Объявление переменных

Перед тем как использовать переменные в программе на Паскале, их имена и типы необходимо объявить в специальном разделе программы, называемом *объявлением*.

Он начинается с ключевого слова VAR, как показано в примере:

```
VAR
   a :INTEGER;
   x :REAL;
```

Переменная `a` имеет тип INTEGER, то есть она может принимать только целые значения, такие как 4, 556, 32145. Переменная `x` объявляет-

ся с типом REAL и может содержать только вещественные числа, такие как 3.14, 44.567, 3.5E+02.

Для того чтобы объявить более одной переменной того же типа, каждую из них можно поместить в отдельную строку:

```
VAR
   a :INTEGER;
   b :INTEGER;
   c :INTEGER;
   x :REAL;
   y :REAL;
```

Можно также объявить все однотипные переменные списком, например:

```
VAR
   a, b, c    :INTEGER;
   x, y       :REAL;
```

Ключевые слова INTEGER и REAL определяются как *стандартные идентификаторы*, которые предопределены в Паскале. Программист может переопределить стандартные идентификаторы, но это крайне не рекомендуется.[1] Стандартные идентификаторы перечислены в приложении B.

В следующей программе объявляются три переменные: целочисленные a и b и вещественная x. Содержимое каждой их них выводится на экран с помощью оператора WRITELN:

```
{ ------------------------------ Пример 1-7 ------------------------------ }
PROGRAM Variables(OUTPUT);
{ Объявление переменных }
VAR
   a, b :INTEGER;
   x :REAL;
{ Программный блок }
BEGIN
   WRITELN('Значение a=',a);
   WRITELN('Значение b=',b);
   WRITELN('Значение x=',x)
END.
```

Вывод программы может быть таким:

```
Значение a=0
Значение b=631
Значение x=2.7216107254E-26
```

Заметьте, что значения переменных a и b отображаются как целые, а значение x выводится в вещественном формате. Однако этот вывод чисел бесполезен, поскольку никаких значений переменным присвоено

[1] Не все компиляторы позволяют это сделать. – *Примеч. науч. ред.*

не было. Пока вы не присвоите переменным значения, они будут содержать то, что находилось в соответствующей области памяти ранее.[1]

Оператор присваивания

Для занесения значения в переменную можно использовать *оператор присваивания* (:=), например:

```
a := 55;
x := 1.5;
y := 2.3E+02;
```

Не используйте следующую форму записи вещественного числа:

.1234

Правильно записанное вещественное число в Паскале должно иметь цифру слева от десятичной точки, например:

0.1234

Точно так же число:

123.

может быть отвергнуто некоторыми компиляторами. Лучше использовать разрешенную форму записи:

123.0

В следующей программе объявляются два целых числа в разделе объявлений, а затем в программном блоке им присваиваются целые значения. После этого для вычисления и вывода результатов различных арифметических операций над этими числами применяется оператор WRITELN.

```
{ ------------------------------ Пример 1-8 ------------------------------ }
PROGRAM Arithmetic(OUTPUT);
{ Объявление переменных }
VAR
   a, b :INTEGER;
{ Программный блок }
BEGIN
   a := 25;
   b := 2;
   WRITELN('a=',a);
   WRITELN('b=',b);
   WRITELN('a+b=',a+b);
   WRITELN('a-b=',a-b);
   WRITELN('a*b=',a*b);
   WRITELN('a/b=',a/b);
   WRITELN('a div b=',a DIV b);   { используется только с целыми }
```

[1] Современные компиляторы могут инициализировать переменные начальными данными, но полагаться на действия компилятора не стоит. – *Примеч. науч. ред.*

```
   WRITELN('a mod b=',a MOD b)    { используется только с целыми }
END.
```

Вывод этой программы:

```
a=25
b=2
a+b=27
a-b=23
a*b=50
a/b= 1.2500000000E+01       ----> Вещественное деление
a div b=12                  ----> Целочисленное деление
a mod b=1                   ----> Остаток целочисленного деления
```

Можно присвоить одной переменной другую:

```
x := y;
```

В этом случае значение переменной `y` копируется в переменную `x`. Можно также переменным присваивать арифметические выражения, например:

```
z := a + b - 2;
GrossPay := PayRate * HoursWorked;
```

В этих операторах вычисляется выражение справа от оператора присваивания и запоминается в переменной, стоящей слева от оператора присваивания (`z` или `GrossPay`).

Задание 1–4

Напишите программу на Паскале, которая выполняет следующее:

A. Присвоить значение 2 переменной `a` и значение 9 переменной `b`.

B. Вывести на экран значения выражений:

a+b DIV 2

(a+b) DIV 2

1–5. Именованные константы

Значения данных (во многих языках, включая Паскаль) называются *константами*, поскольку они не изменяются в процессе выполнения программы. В Паскале имеется два типа констант:

- Литеральные константы
- Именованные константы

Литеральные константы[1] – это значения данных, такие как числа или текстовые строки, а *именованные константы* – это «переменные-

[1] Иногда их называют типизированными константами. – *Примеч. науч. ред.*

константы». Именованная константа отличается от переменной тем, что ее значение не изменяется в процессе выполнения программы. Подобно переменным, именованной константе дается имя, и она должна быть объявлена в разделе объявлений. В действительности раздел объявлений состоит из двух частей, CONST и VAR; секция CONST предшествует секции VAR.[1] Предположим, вы хотите использовать в вычислениях значение 3.14159 (численная константа, известная как π) много раз.

Было бы удобно дать ей имя и использовать его в коде. Именованные константы можно объявлять следующим образом:

```
CONST
   Pi = 3.14159;
   ThisYear = 1992;
   Department= 'OtoRhinoLaryngology';
```

Некоторые константы в Паскале предопределены как стандартные идентификаторы. Одна из полезных именованных констант – это MAXINT, которая представляет максимально возможное целое. Ее значение зависит от используемого компьютера.[2] Значение MAXINT для вашей машины можно выяснить с помощью оператора WRITELN:

```
WRITELN(MAXINT);
```

Как правило, это 32,767 (два байта).

В следующей программе именованная константа Pi используется для вычисления длины окружности.

```
{ ------------------------------ Пример 1-9 ---------------------------- }
PROGRAM Constants(OUTPUT);
{ Объявление констант}
CONST
   Pi = 3.14159;
{ Объявление переменных }
VAR
   Radius, Perimeter :REAL;
{ Программный блок }
BEGIN
   Radius := 4.9;
   Perimeter := 2 * Pi * Radius;
   WRITELN('Длина окружности=', Perimeter)
END.
```

Вывод этой программы:

```
Длина окружности= 3.0787582000E+01
```

---

[1] Современные компиляторы допускают любой порядок расположения секций. – *Примеч. науч. ред.*

[2] Или компилятора. – *Примеч. науч. ред.*

В Delphi или Турбо Паскале не надо переопределять константу Pi, поскольку она предопределена как стандартный идентификатор.

1–6. Преобразование типов: ROUND, TRUNC

Можно присвоить целое значение переменной вещественного типа, но не наоборот. Причина в том, что для целого числа отводится меньше памяти, чем для вещественного. Если бы это было разрешено, то данные, перемещаемые в область памяти недостаточного размера, могли бы потеряться или испортиться. Однако есть возможность выполнить преобразование при помощи следующих двух функций:

ROUND(*n*) округляет *n* до ближайшего целого

TRUNC(*n*) отсекает дробную часть *n*

где *n* – вещественная переменная или выражение.

Рассмотрим примеры:

`ROUND(8.4)` возвращает 8
`ROUND(8.5)` возвращает 9
`TRUNC(8.4)` возвращает 8
`TRUNC(8.5)` возвращает 8

Как видно из примеров, эти две функции могут возвращать разные значения для одного и того же аргумента.

В следующей программе эти функции используются для получения округленного и усеченного значения вещественной переменной `Perimeter`.

```
{ ------------------------------ Пример 1-10 ---------------------------- }
PROGRAM Functions1(OUTPUT);
{ Объявление констант }
CONST
  Pi = 3.14159;
{ Объявление переменных }
VAR
  Perimeter, Radius                    :REAL;
  RoundedPerimeter, TruncatedPerimeter :INTEGER;
{ Программный блок }
BEGIN
  Radius := 4.9;
  Perimeter := 2*Pi*Radius;
  RoundedPerimeter := ROUND(Perimeter);
  TruncatedPerimeter := TRUNC(Perimeter);
  WRITELN('Длина окружности=', Perimeter);
  WRITELN('Длина окружности (округленная)=', RoundedPerimeter);
  WRITELN('Длина окружности (усеченная)=', TruncatedPerimeter)
END.
```

Вывод:

```
Длина окружности= 3.0772000000E+01 ----> Исходное значение
Длина окружности (округленная)=31  ----> Округленный результат
Длина окружности (усеченная)=30    ----> Усеченный результат
```

1–7. Чтение с клавиатуры: READLN, READ

Предыдущая программа вычисляла длину окружности данного радиуса, значение которого было «зашито» в коде программы. Более полезной была бы программа, которая получала бы значение от пользователя, производила вычисления и выдавала результат. Для того чтобы программа делала паузу и ожидала пользовательский ввод, можно применить оператор READLN или READ. Оператор READLN предназначен для чтения значений в одну или несколько переменных. Его основная форма:

READLN(список переменных);

Для чтения значения с клавиатуры в переменную x можно применить оператор:

```
READLN(x);
```

Чтобы прочитать значения в три переменные x, y и z, напишите оператор:

```
READLN(x, y, z);
```

Значения, вводимые более чем в одну переменную (например, x, y и z), должны разделяться хотя бы одним пробелом или нажатием клавиши <Enter>.

Замените оператор присваивания в предыдущей программе на оператор READLN:

```
READLN(Radius);
```

После запуска программа будет ждать, пока вы не введете число и не нажмете <Enter>. Затем она произведет вычисления и выведет результат на экран. К сожалению, оператор READLN не отображает информацию, вводимую пользователем. Это можно сделать с помощью оператора WRITE (или WRITELN):

```
WRITE('Пожалуйста, введите радиус:');
```

Измененная программа:

```
{ ------------------------------ Пример 1-11 ----------------------------- }
PROGRAM KeyboardInput(OUTPUT);
{ Объявление констант }
CONST
  Pi = 3.14159;
{ Объявление переменных }
VAR
```

```
  Perimeter, Radius                    :REAL;
  RoundedPerimeter, TruncatedPerimeter :INTEGER;
{ Программный блок }
BEGIN
  WRITE(' Пожалуйста, введите радиус:');
  READLN(Radius);
  Perimeter := 2*Pi*Radius;
  RoundedPerimeter := ROUND(Perimeter);
  TruncatedPerimeter := TRUNC(Perimeter);
  WRITELN('Длина окружности=', Perimeter);
  WRITELN('Длина окружности (округленная)=', RoundedPerimeter);
  WRITELN('Длина окружности (усеченная)=', TruncatedPerimeter)
END.
```

Выполнение теста дает следующий результат:

```
Пожалуйста, введите радиус:4.9 ----> Напечатайте число и нажмите ENTER
Длина окружности= 3.0787582000E+01
Длина окружности (округленная)=31
Длина окружности (усеченная)=30
```

На этом этапе вы можете использовать или READ, или READLN, поскольку разница между ними для наших приложений пока не имеет значения.

1–8. Форматирование вывода

Возможно, вы подумали, что экспоненциальная форма – не лучший формат для вывода, особенно для бизнеса или финансовых расчетов. Вы правы. Экспоненциальная нотация хороша только для очень больших или очень маленьких чисел, когда степень десяти представляет порядок величины числа.

Для того чтобы отобразить число в форме с фиксированной точкой, используйте такое *описание формата*:

```
WRITELN(Wages :6:2);
```

Формат «:6:2» определяет поле длиной 6 символов, включая два десятичных знака. Таким образом, если переменная `Wages` равна 45.5, то она отобразится так:

```
B45.50
```

где буква «B» обозначает пробел. Если количество цифр в выводимом числе меньше ширины поля вывода, результат будет сдвинут вправо. Если число не помещается в поле вывода, поле будет увеличено и на экран выведется число целиком. Можно добавить символ (например, символ доллара) слева от числа:

```
WRITELN('$',Wages :6:2);
```

Тогда будет выведено:

```
$ 45.50
```

Используя меньшую ширину поля, можно сдвинуть число влево, к символу доллара:

```
WRITELN('$',Wages :0:2);
```

В результате будет выведено:

```
$45.50
```

Этим методом можно форматировать любой тип данных, за исключением целых чисел и текстовых строк, когда ширина поля определяется без десятичных знаков.

В следующей программе различные типы данных форматируются применительно к определенным полям вывода:

```
{ ------------------------------ Пример 1-12 ----------------------------- }
PROGRAM Format(OUTPUT);
{ Объявление переменных }
VAR
  a :INTEGER;
  b :REAL;
{ Программный блок }
BEGIN
  b := 1.2e+02;
  a := 320;
  WRITELN('Я текстовая строка, начинающаяся с позиции 1.');
  WRITELN('Теперь я сдвинута к правому концу поля.':50);
  WRITELN(Я неформатированное целое:', a);
  WRITELN('Я целое, записанное в поле шириной 6 символов:', a:6);
  WRITELN('Я количество денег, записанное в поле шириной 8 символов:$',b:8:2);
  WRITELN('Я количество денег, сдвинутое влево:$',b:0:2)
END.
```

Вывод:

```
Я текстовая строка, начинающаяся с позиции 1.
    Теперь я сдвинута к правому концу поля.
Я неформатированное целое:320
Я целое, записанное в поле шириной 6 символов:   320
Я количество денег, записанное в поле шириной 8 символов:$  120.00
Я количество денег, сдвинутое влево:$120.00
```

Если вывести одни числа без текста, то будет:

```
320
   320
$  120.00
$120.00
```

Задание 1–5

Напишите программу, вычисляющую зарплату сотрудников по формуле:

```
Wages := HoursWorked * PayRate;
```

Введите значения переменных `HoursWorked` и `PayRate` с клавиатуры, а значение `Wages` выведите на экран в нотации с фиксированной точкой, поместив в начало символ доллара.

Заключение

В этой главе вы познакомились с наиболее важными инструментами программирования на Паскале.

1. Теперь вы хорошо знаете, что такое:
   - Заголовок программы
   - Раздел объявлений
   - Секция CONST
   - Секция VAR
   - Тело программы между BEGIN и END
2. Вы изучили два важных типа данных, INTEGER и REAL, и научились строить и вычислять арифметические выражения с обоими типами данных.
3. Вы знаете арифметические операторы Паскаля, их свойства и порядок старшинства.

   \+ – * / DIV MOD
4. Вы знаете, как объявлять переменные обоих типов, как именовать их, как присваивать им значения (с помощью оператора присваивания (:=) или путем ввода значения с клавиатуры) и как отображать их на экране.
5. Вы научились применять следующие функции преобразования для усечения и округления вещественных выражений:

   TRUNC(*n*) отсекает дробную часть *n*

   ROUND(*n*) округляет *n* до ближайшего целого
6. Вы знаете, как объявлять именованные константы и использовать их в программе.
7. Во время первого путешествия по Паскалю вы изучили следующие операторы вывода для отображения и переменных и численных или строковых констант:

   WRITELN

   WRITE

а также операторы ввода для чтения значений с клавиатуры:

READLN
READ

8. И наконец, вы научились форматировать численный или строковый вывод, получая результат в желаемой форме.

Упражнения

1. В чем различие между литеральной и именованной константами?
2. В чем различие между именованной константой и переменной?
3. Напишите объявления переменных, используя подходящий тип данных, для следующих величин:
   a. Цена автомобиля в долларах и центах.
   b. Площадь поверхности цилиндра.
   c. Количество учащихся в классе.
4. Напишите объявления констант для следующих объектов:
   a. Имя компании (придумайте сами).
   b. Коэффициент преобразования миль в километры.
5. Напишите выражения для подсчета следующих величин:
   a. Площадь круга при известном радиусе.
   b. Общий балл ученика за три класса.
   c. Цена покупки, включая налог, равный 8%.
6. Вычислите следующие выражения:
   a. 10 + 5 * 2 – 6 / 2
   b. (10 + 5) * 2 – 6 / 2
   c. (10 + 5 * 2 – 6) / 2
   d. ((10 + 5) * 2 – 6) / 2
7. Напишите программу на Паскале, выводящую на экран ваше имя, домашний адрес и адрес электронной почты в отдельных строчках.
8. Вычислите следующие выражения:
   a. 1.0/2.0
   b. 1.0/2
   c. 1/2
   d. 1 DIV 2
   e. 1 MOD 2
   f. 10 / 3
   g. ROUND(10/3)

Ответы

6. a. 17.0 b. 27.0 c. 7.0 d. 12.0
8. a. 0.50 b. 0.50 c. 0.50 d. 0 e. 1 f. 3.33 g.3

2

Элементы языка

2–1. Стандартные типы данных и функции

Данные, с которыми работает любая программа, могут состоять из целых или вещественных чисел или из текстовых строк, но каждый тип данных хранится и обрабатывается по-разному. Паскаль поддерживает следующие типы данных (называемые также простыми или *скалярными* типами):

INTEGER
REAL
CHAR
BOOLEAN

Вы уже применяли типы INTEGER и REAL как для констант, так и для переменных. Вы также использовали арифметические операторы с переменными и константами для построения арифметических выражений и познакомились с некоторыми функциями, такими как ROUND и TRUNC. Эта глава представляет полную картину числовых типов данных и соответствующих функций и выражений. Она также вводит тип данных CHAR для представления единичного символа и тип BOOLEN для представления логических значений. Рассмотрение символьного типа включает в себя обзор представления строк в стандартном Паскале, а также их представление в современных реализациях, таких как Turbo Pascal и UCSD Pascal (использующих тип STRING).[1]

[1] Современные реализации, такие как Delphi, поддерживают гораздо больше типов данных. К моменту написания книги Turbo Pascal был самой последней реализацией. – *Примеч. науч. ред.*

2–2. Числовые типы данных

Интервал чисел, которые могут быть представлены как целые (или как вещественные), зависит от реализации. Для типа INTEGER он определяется следующими пределами:

MAXINT	максимальное положительное целое
–(MAXINT+1)	максимальное отрицательное целое

Повторим, значение MAXINT зависит от реализации.

Как правило, для вещественных чисел выделяется большее количество байт, чем для целых, но они имеют ограниченную точность. Дроби 0.333333 и 0.666666, независимо от количества цифр, представляющих число, никогда не будут обладать такой точностью, как точные значения $^1/_3$ и $^2/_3$. По этой причине не рекомендуется проверять два вещественных числа на равенство. Вместо этого лучше сравнивать их разность с некоторым конечным малым значением. В Турбо Паскале существуют дополнительные числовые типы, которые вводятся в следующем разделе.

Числовые типы в Турбо Паскале

В Турбо Паскале есть два дополнительных целых типа (наряду с INTEGER). Они показаны в табл. 2-1 вместе с необходимым для них объемом памяти и пределами значений, которые могут храниться в каждом типе.

В одном байте можно хранить либо SHORTINT, либо BYTE. В действительности BYTE – это беззнаковый SHORTINT, то есть он может хранить только положительные числа. Как видно из таблицы, если знак не используется, то максимальные значения типа удваиваются. Это верно и для INTEGER, и для WORD, поскольку WORD – это положительное целое с удвоенным максимальным значением.

Таблица 2-1. Целые типы Турбо Паскаля

Тип данных	Размер (в байтах)	Интервал значений
SHORTINT	1	от –128 до +127
BYTE	1	от 0 до 255
INTEGER	2	от –32 768 до +32 767
WORD	2	от 0 до 65 535

LONGINT – это наибольшее целое, которое может существовать в Турбо Паскале. Можно проверить его величину, выведя на экран значение константы MAXLONGINT:

```
WRITELN(MAXLONGINT);
```

Заметим, что отрицательный предел любого знакового типа на единицу больше положительного предела (то есть +127 и −128), поскольку ноль считается положительным числом.

Запятые в больших числах используются здесь только для улучшения читаемости. Их никогда не будет в выводе программы, и они не будут восприниматься как часть литеральной константы. Так, число 2,147,483,647 должно быть представлено в виде 2147483647.

Как показано в табл. 2-2, в Турбо Паскале существует также дополнительный вещественный тип (наряду с типом REAL). Для описания точности числа как максимального количества точных цифр в таблицу для вещественных чисел добавлена новая колонка.

Таблица 2-2. Вещественные типы Турбо Паскаля

Тип данных	Размер (в байтах)	Точность (вплоть до)	Интервал значений
SINGLE	4	7 цифр	от 0.71E−45 до 3.4E+38
REAL	6	11 цифр	от 2.94E−39 до 1.7E+38
DOUBLE	8	15 цифр	от 4.94E−324 до 1.79E+308
EXTENDED	10	19 цифр	от 3.3E−4932 до 1.18E+4932
COMP	8	только целые	±9.2E+18

Посмотрев на интервал значений типа SINGLE, вы обнаружите, что он довольно близок к интервалу типа REAL, особенно в области очень больших чисел. Главное отличие состоит в экономичности хранения чисел типа SINGLE (4 байт по сравнению с 6 байт), но достигается это за счет потери точности (7 цифр по сравнению с 11). Без математического сопроцессора вещественные типы, за исключением REAL, недоступны. Тип COMP в действительности принадлежит к множеству целых типов, так как не поддерживает дробных чисел, но обычно упоминается среди вещественных типов, поскольку требует наличия сопроцессора.

2–3. Стандартные арифметические функции

Паскаль включает в себя большое количество предопределенных функций, которые могут быть использованы в выражениях вместе с константами и переменными. В табл. 2-3 приведены стандартные арифметические функции, разделенные на три группы:

- Функции преобразования
- Тригонометрические функции
- Смешанные функции

Любая функция работает с параметром, заключенным в скобки. Параметр – это выражение определенного типа (заметим, что выражением может быть одиночная переменная или константа). Перед тем как использовать функцию, необходимо определить тип параметра и тип возвращаемого функцией значения (который называется также типом функции). Функции преобразования, например, принимают вещественный параметр, а возвращают результат целого типа. Другие функции работают с параметрами целого типа или вещественного, а возвращают значения различных типов. Тип возвращаемого значения важен, когда значение функции присваивается переменной.

Таблица 2-3. Стандартные арифметические функции

Функция	Формат возвращаемого значения	Тип параметра	Тип результата
Функции преобразования			
ROUND(x)	x округляется до ближайшего целого	REAL	INTEGER
TRUNC(x)	Дробная часть x отсекается	REAL	INTEGER
[a]Тригонометрические функции			
ARCTAN(x)	Арктангенс x	REAL/INTEGER	REAL
COS(x)	Косинус x	REAL/INTEGER	REAL
SIN(x)	Синус x	REAL/INTEGER	REAL
Смешанные функции			
ABS(x)	Абсолютное значение x	REAL/INTEGER	REAL/INTEGER
EXP(x)	Экспоненциальная функция x (e^x)	REAL/INTEGER	REAL
LN(x)	Натуральный логарифм x	REAL/INTEGER	REAL
SQR(x)	x в квадрате (x^2)	REAL/INTEGER	REAL/INTEGER
SQRT(x)	Корень квадратный из x (^x)	REAL/INTEGER	REAL

[a.] Все углы должны быть в радианах.

Взгляните на примеры:

```
SQR(3)=9
SQR(2.5)=6.25
SQRT(9)=3.00
ABS (-28.55)=28.55
LN(EXP(1))=1.00
ARCTAN (1)=45 градусов
```

Заметьте, что тип результата, возвращаемого функцией SQR, тот же, что и тип параметра, а функция SQRT возвращает значение вещественного типа независимо от типа параметра. Заметим также, что в ка-

честве параметра любой функции может выступать другая функция, например LN(EXP(1)).

Результат, полученный последней функцией (ARCTAN), преобразован в градусы, но без преобразования он будет выведен в радианах. Программа, с помощью которой получены эти результаты, приведена в примере 2-1. Обратите внимание на формат дескрипторов (описателей), с помощью которых отформатирован вывод:

```
{ ----------------------------- Пример 2-1 ------------------------------ }
{ Стандартные арифметические функции }
PROGRAM FunctionDemo(OUTPUT);
CONST
  Pi = 3.14159; { Эта часть в Турбо Паскале не обязательна}
BEGIN
  WRITELN('SQR(3)=',SQR(3));
  WRITELN('SQR(2.5)=',SQR(2.5):0:2); { Обратите внимание на формат }
  WRITELN('SQRT(9)=',SQRT(9):0:2);
  WRITELN('ABS(-28.55)=',ABS(-28.55):0:2);
  WRITELN('LN(EXP(1))=',LN(EXP(1)):0:2);
  WRITELN('ARCTAN(1)=',ARCTAN(1)* 180/Pi:0:0,' градусов')
{ Обратите внимание на преобразования и формат }
END.
```

Пример: Функция возведения в степень

В Паскале, в отличие от других языков (таких как FORTRAN и BASIC), не существует оператора возведения в степень, но это можно сделать с помощью арифметической функции. Можно, конечно, возвести в небольшую степень с помощью функции SQR:

SQR(x)	степень 2
SQR(x) * x	степень 3
SQR(SQR(x))	степень 4

Также можно возводить в степень при помощи следующего математического уравнения:

$x^y = EXP(LN(x) * y)$

В следующей программе это выражение используется для возведения числа в любую степень. Программа просит ввести основание a и степень b, а затем выводит на экран отформатированный результат.

```
{ ----------------------------- Пример 2-2 ------------------------------}
{ Стандартные арифметические функции }
PROGRAM PowerOperator(INPUT,OUTPUT);
VAR
  a, b :REAL;
BEGIN
  WRITE('Введите основание и степень через пробел:');
```

```
  READLN(a,b);
  WRITELN('Значение ',a:0:2,', возведенное в степень, ',b:0:2,' равно ',
          EXP(LN(a)*b):0:2)
END.
```

Результат выполнения примера:

```
Введите основание и степень через пробел: 2 10
Значение 2.00, возведенное в степень 10.00, равно 1024.00
```

Пример: Бакалейная лавка

В бакалейной лавке требуется быстро определять количество и номинал монет для подсчета сдачи с доллара, поэтому запрограммированный алгоритм был бы большой помощью кассиру. Следующая программа считывает с клавиатуры цену покупки (предположим для простоты, что она меньше доллара) и выдает количество монет номиналом в 25, 10, 5 и 1 цент на сдачу с одного доллара. В программе применены операторы DIV и MOD.

```
{ ------------------------------ Пример 2-3 ------------------------------ }
{ Бакалейная лавка }
PROGRAM Grocery(INPUT,OUTPUT);
VAR
  Change, TotalPrice,
  Dollars, Quarters, Dimes, Nickels, Cents :INTEGER;
BEGIN
  WRITE('Введите полную стоимость в центах: ');
  READLN(TotalPrice);
  Change := 100 - TotalPrice;
{ 25-центовые монеты }
  Quarters := Change DIV 25;
  Change := Change MOD 25;
{ 10-центовые монеты }
  Dimes := Change DIV 10;
  Change := Change MOD 10;
{ 5-центовые монеты }
  Nickels := Change DIV 5;
  Change := Change MOD 5;
{ 1-центовые монеты }
  Cents := Change;
  WRITELN('Сдача:');
  WRITELN(Quarters,' 25-центовые монеты');
  WRITELN(Dimes,' 10-центовые монеты');
  WRITELN(Nickels, ' 5-центовые монеты');
  WRITELN(Cents, ' 1-центовые монеты')
END.
```

Результат выполнения примера:

```
Введите полную стоимость в центах: 22 ----> Наберите 22 и нажмите ENTER
```

```
Сдача:
3 25-центовые монеты
0 10-центовые монеты
0 5-центовые монеты
3 1-центовые монеты
```

Задание 2–1

Измените приведенную выше программу так, чтобы она принимала любую сумму в качестве `TotalPrice` (включая часть доллара) и любую сумму в качестве `AmountPaid` (уплаченная сумма).

Программа должна читать `AmountPaid` и `TotalPrice` и выводить на экран сдачу в монетах номиналом 25, 10, 5 и 1 цент.

Дополнительные функции Турбо Паскаля

Турбо Паскаль включает в себя значительное количество дополнительных арифметических функций. Две из них особенно полезны:

FRAC(n) возвращает дробную часть вещественного числа n

INT(n) возвращает целую часть вещественного числа n

Например:

```
WRITELN(FRAC(8.22):2:2); выводит 0.22
WRITELN(INT(8.22)2:2); выводит 8.00
```

Обе функции возвращают вещественные числа.

Можете использовать эти функции в задании 2-1.

Другая пара функций генерирует случайные числа:

RANDOM(n) возвращает случайное целое число между 0 и целым n (включая 0)

RANDOM возвращает случайное вещественное число между 0 и 1 (включая 0)

Попробуйте эти два оператора:

```
WRITELN(RANDOM:2:2);
WRITELN(RANDOM(n));
```

где n – целая переменная, введенная с клавиатуры.

Несколько раз запустите программу с этими операторами и посмотрите на результаты. Они должны отличаться при каждом запуске.

Задание 2–2

Напишите следующие выражения на Паскале:

1. Квадратное уравнение: $Ax^2 + Bx + C$
2. Детерминант: $B^2 - 4AC$
3. Корень квадратный детерминанта
4. Абсолютное значение детерминанта

Затем напишите программу вычисления корней уравнения с учетом ввода значений A, B и C. Используйте тестовые значения для получения вещественных корней. Типовые значения:

A=1, B=2 и C=1 дают решение: X1= X2= −1.00
A=1, B=4 и C=2 дают решение: X1= −0.59, X2= −3.41

2–4. Символьный тип: CHAR

Тип CHAR используется в Паскале для хранения одиночных символов. Например, переменную типа CHAR можно объявить так:

```
VAR
   SingleLetter : CHAR;
```

В главном теле программы (между BEGIN и END) переменной `SingleLetter` можно присвоить одиночный символ:

```
   SingleLetter := 'A';
```

Как понятно из примера, постоянный литерал типа CHAR должен состоять ровно из одного символа, заключенного в одинарные кавычки:

'A' '3' '*' '$' ' '

Для того чтобы записать одинарную кавычку (или апостроф) как символьную константу, используйте двойные кавычки ''''.

Для вывода на экран символьной константы или символьной переменной можно использовать операторы WRITELN или WRITE:

```
WRITELN('A');
WRITELN(SingleLetter);
```

Внутреннее представление множества символов кодируется однобайтным целым.[1] Для малых компьютеров обычно используют код ASCII

[1] Современные операционные системы (Windows 2000 и выше) работают только с Unicode, то есть для хранения одного символа отводится два байта. Данная операция прозрачна для программиста, так как преобразования в обе стороны делаются самой системой автоматически и незаметно для пользователя. Подробнее см. *http://www.unicode.org*. – *Примеч. науч. ред.*

(American Standard Code for Information Interchange, американский стандартный код для обмена информацией). ASCII-код включает 256 символов от 0 до 255 (см. приложение А). Первая половина кодов ASCII (от 0 до 127) является стандартом для всех персональных компьютеров. Он включает следующие символы:

- Буквы в верхнем регистре (A–Z): ASCII от 65 до 90
- Буквы в нижнем регистре (a–z): ASCII от 97 до 122
- Цифры (0–9): ASCII от 48 до 57

Код также содержит символы пунктуации и управляющие символы.

Вторая половина ASCII-кодов не является стандартной и на разных машинах реализована по-разному.

Относительное положение символа в ASCII-таблице называется *порядковым номером*.

Стандартные символьные функции

Существует четыре стандартных функции для работы с символами:

ORD(*c*)	возвращает порядковый номер символа *c*
CHR(*n*)	возвращает символ, представленный порядковым номером *n*
PRED(*c*)	возвращает символ, предшествующий символу *c* в соответствии с порядковым номером
SUCC(*c*)	возвращает символ, следующий за символом *c* в соответствии с порядковым номером

Можно получить порядковый номер любого символа при помощи функции ORD, например:

```
WRITELN(ORD('A'));
```

Этот оператор выводит на экран порядковый номер символа «A», равный 65.

В следующей программе пользователь вводит символ, а программа выводит на экран соответствующий порядковый номер.

```
{ ------------------------------ Пример 2-4 ------------------------------ }
{ Вывод на экран порядкового номера символа }
PROGRAM OrdinalNumber(INPUT,OUTPUT);
VAR
  SingleChar :CHAR;
BEGIN
  WRITE('Дайте мне, пожалуйста, символ: ');
  READLN(SingleChar);
  WRITELN('Порядковый номер этого символа равен ', ORD(SingleChar));
  READLN { Программа будет ждать, пока вы не нажмете ENTER }
END.
```

Результат выполнения примера:

```
Дайте мне, пожалуйста, символ: A          ----> Наберите A и нажмите ENTER
Порядковый номер этого символа равен 65   ----> Вывод программы
```

Обратите внимание на последний оператор READLN. Применение READLN без скобок заставляет программу ждать нажатия клавиши <ENTER>. Нельзя использовать с этой целью READ. Этой форме оператора READLN обычно предшествует подсказка пользователю, например:

```
WRITELN(' Для продолжения нажмите ENTER...');
```

Парой функции ORD является функция CHR, которая в качестве параметра принимает порядковый номер, а возвращает символ, соответствующий этому номеру. Посмотрите на пример:

```
WRITELN(CHR(66));
```

Этот оператор выводит на экран символ «B».

В следующей программе пользователь вводит порядковый номер, а программа выводит на экран соответствующий символ.

```
{ ------------------------------ Пример 2-5 ------------------------------ }
{ Вывод на экран символа по его порядковому номеру }
PROGRAM CharDisplay(INPUT,OUTPUT);
VAR
  OrdinalNum :BYTE;
BEGIN
  WRITE('Дайте мне, пожалуйста, номер от 0 до 255: ');
  READLN(OrdinalNum);
  WRITELN('Это соответствует символу "', CHR(OrdinalNum),'"');
  WRITELN('Для продолжения нажмите ENTER...');
  READLN { Программа будет ждать, пока вы не нажмете ENTER }
END.
```

Результат выполнения примера:

```
Дайте мне, пожалуйста, номер от 0 до 255: 66     ----> Введите число 66
Это соответствует символу "B"                    ----> Вывод программы
Для продолжения нажмите ENTER...
```

Обратите внимание на использование в Турбо Паскале типа BYTE для хранения порядковых номеров, которые представляют собой положительные целые числа от 0 до 255. Если на вашем компьютере такого типа нет, используйте тип INTEGER.

Следующая программа демонстрирует использование функций PRED и SUCC. Вы вводите символ, а программа выводит на экран предшествующий и последующий символы.

```
{ ------------------------------ Пример 2-6 ------------------------------ }
{ Предшествующий и последующий символы }
PROGRAM CharPredAndSucc(INPUT,OUTPUT);
VAR
```

```
  Letter: CHAR;
BEGIN
  WRITE('Пожалуйста, введите символ: ');
  READLN(Letter);
  WRITELN(' Предшествующий символ - "',PRED(Letter),'"');
  WRITELN(' Последующий символ - "',SUCC(Letter),'"');
  WRITELN(' Для продолжения нажмите ENTER...');
  READLN
END.
```

Результат выполнения примера:

```
Пожалуйста, введите символ:K                ----> Введите символ K
Предшествующий символ - "J"                 ----> Ответ программы
Последующий символ - "L"
Для продолжения нажмите ENTER...
```

Для проверки этой программы годятся номера любых специальных символов на вашей клавиатуре. Однако помните, что на некоторых машинах (мэйнфреймах) используется другая кодировка, известная как EBCDIC (Extended Binary-Coded Decimal Interchange Code) – расширенный двоично-десятичный код (для) обмена (информацией)).

Можно также использовать функцию ORD с типом INTEGER. В этом случае она возвращает целые из множества целых чисел (от –(MAXINT+1) до MAXINT):

ORD(0)=0, ORD(1)=1, ORD(255)=255 и ORD(–22)=–22

Функции SUCC и PRED работают с целыми точно так же:

SUCC(1)=2 и PRED(1)=0

Некоторые программисты увеличивают значение счетчиков таким способом:

```
Counter := SUCC(Counter);
```

Можно протестировать эти соотношения, заменив в последней программе (пример 2-6) тип CHAR на тип INTEGER.

Строки в стандартном Паскале

Как упоминалось ранее, можно представлять строковые константы, заключая символы в одинарные кавычки, например:

```
'Это строка, заключенная в одинарные кавычки'
```

Чтобы включить апостроф в текстовую константу, его надо удвоить:

```
'Это апостроф '', включенный в строку'
```

Можно также присвоить строку именованной константе.

```
CONST
```

```
Name = 'Sally Shuttleworth';
```

После такого объявления можно использовать именованную константу Name вместо самой строки, но помните, что в программе нельзя присвоить именованной константе любое значение.

В Паскале не существует понятия строковой переменной. Строка в Паскале хранится в PACKED ARRAY OF CHAR (упакованный массив символов), который можно объявить, например, так:

```
VAR
   Name : PACKED ARRAY[1..15] OF CHAR;
```

Это объявление позволяет хранить строку длиной ровно 15 символов – не больше и не меньше.

Взгляните на следующий пример, где объявляется переменная Message и ей присваивается строка "Press any key . . . ". Лишний пробел добавлен в конец строки, чтобы она соответствовала размеру переменной Message, которая объявлена как PACKED ARRAY OF CHAR длиной 19 символов.

```
{ ------------------------------ Пример 2-7 ------------------------------ }
{ Упакованный массив символов}
PROGRAM PackedArray(OUTPUT);
VAR
  Message :PACKED ARRAY[1..19] OF CHAR;
BEGIN
  Message := 'Нажмите клавишу... ';
  WRITELN(Message)
END.
```

Вывод:

```
Нажмите клавишу...
```

2–5. Строковый тип

Вам никогда не придется иметь дело с PACKED ARRAY OF CHAR, если только у вас не одна из старых реализаций Паскаля на мэйнфрейме. В современных реализациях (таких как Турбо или UCSD) определен тип STRING.

Объявление строки

Можно объявить переменную типа STRING, например:

```
VAR
   StudentName : STRING;
```

Это объявление позволяет хранить в переменной StudentName строку некоторой длины. Хотя максимальная длина строки в Турбо Паскале равна 255 (в UCSD – 80), действительная длина строки (называемая также *динамической длиной*) определяется количеством символов в строке. Можно объявить строковую переменную и ее максимальную длину в одном операторе:

```
VAR
   StudentName : STRING[20];
```

В этом случае максимальная длина строки, хранимой в переменной StudentName, равна 20 символам. Приведем программу, которая читает имя длиной максимум в 20 символов и выводит его на экран:

```
{ ------------------------------ Пример 2-8 ------------------------------ }
{ Строковый тип в Турбо Паскале}
PROGRAM StringDemo(INPUT,OUTPUT);
VAR
  Name :STRING[20];
BEGIN
  WRITE('Пожалуйста, введите имя длиной не более 20 символов:');
  READLN(Name);
  WRITELN('Вы ввели имя ',Name, '. Правильно?')
END.
```

Результат выполнения примера:

```
Пожалуйста, введите имя длиной не более 20 символов: Peter Rigby
Вы ввели имя Peter Rigby. Правильно?
```

Помните, что если вы присвоите переменной Name строковую константу длиной более 20 символов, то лишние символы будут отсечены.

Длина строки

Функция LENGTH позволяет определять динамическую длину строки. Для того чтобы определить, например, длину строки Name в последней программе, можно воспользоваться выражением:

```
LENGTH(Name)
```

Выведя значение этого выражения на экран, получим точное количество символов, содержащихся в строковой переменной, включая пробелы. Длина пустой строковой переменной равна нулю. В следующей программе вводится имя, а программа отображает действительную длину переменной до и после присваивания.

```
{ ------------------------------ Пример 2-9 ------------------------------ }
{ Динамическая длина строки }
PROGRAM StringLen(INPUT,OUTPUT);
VAR
  Name :STRING[20];
```

```
BEGIN
  WRITELN('Динамическая длина строки равна сейчас ',LENGTH(Name), ' симво-
лов');
  WRITE('Пожалуйста, введите имя длиной не более 20 символов: ');
  READLN(Name);
  WRITELN('Динамическая длина строки равна сейчас ',LENGTH(Name), ' симво-
лов')
END.
```

Результат выполнения примера:

```
Динамическая длина строки равна сейчас 0 символов
Пожалуйста, введите имя длиной не более 20 символов: Dale Sanders
Динамическая длина строки равна сейчас 12 символов
```

Введение типа STRING заполнило пробел в Паскале и добавило мощный инструмент, особенно в области обработки текстов.

2–6. Тип BOOLEAN

Значения булева типа (иногда называемые *логическими значениями*) представлены двумя константами:

TRUE и FALSE

Они названы в честь английского математика Джорджа Буля (George Boole, 1815–1864). В Паскале можно объявить переменную типа BOOLEAN, которая может принимать значения двух логических констант, следующим образом:

```
VAR
   Result : BOOLEAN;
```

Простые логические выражения

Логической переменной можно присвоить логическую константу, например:

```
Result := TRUE;
```

Можно также переменной присвоить логическое выражение, например:

```
Result := A > B;
```

Если A, например, равно 22.5, а B равно 2.3, то выражение A>B (A больше B) вычисляется как TRUE. Если A равно 1.8, то соотношение не выполняется, и выражение вычисляется как FALSE. Логические выражения можно конструировать с помощью *операторов отношений*, показанных в табл. 2-4.

Таблица 2-4. Операторы отношений

Оператор	Значение	Пример
>	Больше чем	A > B
<	Меньше чем	C < 54
>=	Больше или равно	x >= 16.8
<=	Меньше или равно	A+B <= 255
=	Равно	SQR(B)=4*A*C
<>	Не равно	CHR(a)<>'N'

Операторы отношений можно использовать с любыми типами данных: числовыми, символьными или логическими. Несколько примеров:

Числовой: y > 66.5
Y = A * x + B

Символьный: FirstCharacter = 'B'
CHR(x) > 'A'

Булев: TRUE > FALSE (всегда TRUE)
TRUE < FALSE (всегда FALSE)

Символьное выражение:

```
'A' < 'B'
```

всегда истинно, так как буква «A» предшествует букве «B» в алфавите; другими словами, она имеет меньший порядковый номер. В соответствии с этой логикой следующие выражения равны TRUE:

```
'9' > '1'
'Y' < 'Z'
```

Следующая программа читает с клавиатуры два целых числа A и B и выводит на экран значение логического выражения A=B:

```
{ ------------------------------ Пример 2-10 ---------------------------- }
{ Логические переменные }
PROGRAM Compare1(INPUT,OUTPUT);
VAR
  A, B   :INTEGER;
  Result :BOOLEAN;
BEGIN
  WRITE('Пожалуйста, введите два целых числа: ');
  READLN(A, B);
  Result := (A = B);
         { или,
           Result := A = B;
           Скобки можно опустить. }
  WRITELN('Сравнение - ', Result)
END.
```

Результат двух запусков программы:

```
Пожалуйста, введите два целых числа: 5 5
Сравнение - TRUE
Пожалуйста, введите два целых числа: 50 55
Сравнение - FALSE
```

Как упоминалось ранее, нельзя проверять два вещественных числа на равенство из-за их ограниченной точности. В следующей программе считается, что две вещественные переменные равны, если их разность меньше некоторого малого значения `Difference`.

```
{ ------------------------------ Пример 2-11 ---------------------------- }
{ Сравнение вещественных значений }
PROGRAM Compare2(INPUT,OUTPUT);
CONST
  Difference = 0.0001;
VAR
  x, y   :REAL;
  Result :BOOLEAN;
BEGIN
  WRITE(' Пожалуйста, введите два вещественных числа: ');
  READLN(x, y);
  Result := ABS(x - y) < Difference;
  WRITELN('Разность равна ', ABS(x-y):2:6);
  WRITELN('Сравнение - ', Result)
END.
```

Результат запуска примера:

```
Пожалуйста, введите два вещественных числа: 4.5 4.50001
Разность равна 0.000010
Сравнение - TRUE
```

Сложные логические выражения

Логические выражения, содержащие операторы отношений, называются *простыми логическими выражениями* (в других языках они называются относительными выражениями). *Сложные логические выражения* – это выражения, содержащие булевы операторы (называемые также логическими операторами): AND, OR и NOT.

Чтобы понять, как работают сложные логические выражения, рассмотрим пример:

```
(x = 4) AND (y < 50)
```

Это выражение вычисляется как TRUE, если оба условия, x = 4 и y < 50, равны TRUE.

Теперь рассмотрим тот же пример, но с оператором OR:

```
(x = 4) OR (y < 50)
```

Это выражение вычисляется как TRUE, если любое из условий равно TRUE. Например, если x равно 4, то выражение равно TRUE независимо от значения y.

Оператор NOT предназначен для замены значения булева выражения на противоположное. Предположим, что переменная UnderAge типа BOOLEAN означает, что возраст меньше 18 лет:

```
UnderAge := Age < 18;
```

Переменная UnderAge будет содержать значение TRUE, если Age меньше 18.

Тогда выражение:

```
NOT(UnderAge)
```

вычисляется как TRUE, если значение Age равно или больше 18.

Операторы Турбо Паскаля

В Турбо Паскаль также включен логический оператор XOR, который называется *исключающее ИЛИ*. Пример его применения:

```
(x = 4) XOR (x = 400)
```

Значение выражения равно TRUE, если хотя бы одно из двух условий (x = 4 или x = 400) равно TRUE, но выражение вычисляется как FALSE, если оба условия или TRUE или FALSE. В любой реализации Паскаля можно использовать оператор <> как исключающее ИЛИ. Можно переписать предыдущее выражение:

```
(x = 4) <> (x = 400)
```

Старшинство операторов

Как и в случае арифметических выражений, при построении булевых выражений (относительных или логических) необходимо учитывать старшинство операторов. Относительное старшинство всех операторов, которыми мы уже пользовались, иллюстрирует табл. 2-5.

Таблица 2-5. Старшинство операторов Паскаля

Оператор	Старшинство
NOT	Приоритет 1 (высший)
* / DIV MOD AND	Приоритет 2
+ – OR (XOR в Турбо Паскале)	Приоритет 3
= > < >= <= <>	Приоритет 4 (низший)

Чтобы понять влияние старшинства, рассмотрим выражение:

```
x = 4 OR x = 400
```

Оператор OR имеет более высокий уровень старшинства, чем равенство, поэтому выражение не пройдет компиляцию, поскольку будет интерпретировано как:

```
x = (4 OR x) = 400
```

а такое выражение недопустимо.

Задание 2–3

Напишите логические выражения для следующих условий:

1. A меньше 55.5
2. X равно Y или X больше или равно Z
3. Или X=40, или Y=80, или оба выражения истинны
4. Или X=40, или Y=80, но не оба выражения истинны

Заключение

В этой главе вы изучили:

1. Четыре стандартных типа данных:
   - INTEGER
   - REAL
   - CHAR
   - BOOLEAN
2. Числовые типы Турбо Паскаля:

   Целые:
   - SHORTINT
   - BYTE
   - INTEGER
   - WORD
   - LONGINT

   Вещественные:
   - SINGLE
   - REAL
   - DOUBLE
   - EXTENDED
   - COMP
3. Стандартные арифметические функции, разделенные на три группы:

   Функции преобразования:
   - ROUND

- TRUNC

Тригонометрические функции:

- ARCTAN
- COS
- SIN

Смешанные функции:

- ABS
- EXP
- LN
- SQR
- SQRT

4. Некоторые арифметические функции Турбо Паскаля, такие как:
   - FRAC
   - INT
   - RANDOM
5. Вы научились писать математические выражения, используя арифметические операторы и функции.
6. Вы познакомились с четырьмя символьными функциями:
   - CHR
   - ORD
   - PRED
   - SUCC
7. Вы изучили некоторые свойства строковых переменных стандартного Паскаля и теперь знаете, что они объявляются как PACKED ARRAYS OF CHAR. В расширениях Паскаля, таких как Турбо Паскаль и UCSD Паскаль, тип STRING добавлен в язык вместе со свойствами и функциями.

   Вы также изучили строковую функцию:

   LENGTH

   которая определяет длину строки.
8. Вы научились строить простые и сложные логические выражения с помощью арифметических операторов, операторов отношений и логических операторов и узнали, как использовать тип BOOLEAN.

   Вы также знаете логические операторы:

   - NOT
   - AND
   - OR

   и можете записать исключающее ИЛИ двумя способами:

   - используя относительный оператор <>

- используя оператор Турбо Паскаля XOR

9. И наконец, вы прошли последний курс по операторам Паскаля и изучили их относительное старшинство.

Упражнения

1. Дано:

   A := 99

   B := 98

   X := A > B

   Y := A = B

   Z := A < B

   Вычислите следующие логические выражения:

   - a. X
   - b. Y
   - c. Z
   - d. X OR Y
   - e. X OR Z
   - f. X AND Y OR Z
   - g. (X AND Y) OR Z
   - h. X AND (Y OR Z)

2. В условиях упражнения 1 вычислите следующие выражения:
   - a. NOT X
   - b. NOT (X OR Y)
   - c. X AND NOT Y
   - d. (X AND NOT Y) OR Z

3. Согласно теореме Пифагора сумма квадратов катетов прямоугольного треугольника равна квадрату гипотенузы, то есть катет^2+катет^2 = гипотенуза^2. Напишите программу на Паскале, которая читает длины двух катетов прямоугольного треугольника и печатает длину гипотенузы.

4. Напишите программу на Паскале, которая читает имя, отчество и фамилию, а затем печатает полное имя.

Ответы

1. a. TRUE b. FALSE c. FALSE d. TRUE e. TRUE f. FALSE g. FALSE h. FALSE
2. a. FALSE b. FALSE c. TRUE d. TRUE

3

Решения

3–1. Принятие решений

До сих пор программы в этой книге представляли собой серии инструкций, выполняемых последовательно. Однако в реальных приложениях обычно приходится менять последовательность выполнения в соответствии с определенными условиями. Иногда условия просты, например:

> «**Если** (if) холодно, **то** (then) наденьте пальто.»

В этом утверждении действия предпринимаются, если условие оценивается как TRUE (погода холодная). Однако если погода прекрасная, предложение пропускается.

Некоторые условия могут быть множественными, как в следующей беседе:

> «**Если** (if) я рано приду с работы, мы встретимся сегодня вечером; **либо если** (else if) будет слишком поздно, мы сделаем это завтра; **либо если** (else if) завтра придет мой брат, мы сможем встретиться все вместе во вторник; **либо если** (else if) вторник будет выходным днем, назначим встречу на среду; **либо** (else) я позвоню тебе, чтобы договориться о следующей встрече!»

В действительности программа сможет легко справиться с такими последовательными или вложенными условиями, если вы напишете соответствующий код.

В Паскале есть две *управляющие структуры* для работы с условиями и их результирующими действиями: конструкция двойного выбора IF-THEN-ELSE и конструкция множественного выбора CASE.

3–2. Простое решение: IF–THEN

Простое условие можно записать при помощи конструкции IF-THEN, например:

```
IF Age < 18 THEN
   WRITELN('Извините, вы несовершеннолетний.');
```

Конструкция начинается с ключевого слова IF, за которым следует логическое выражение (условие, которое необходимо проверить), затем ключевое слово THEN, затем результирующий оператор WRITELN, который выполняется, если условие принимает значение TRUE. Как видите, конструкция IF – это один оператор, заканчивающийся точкой с запятой. Если значение переменной `Age` меньше 18, то выполняется часть оператора после ключевого слова THEN, в противном случае весь оператор пропускается и программа продолжает выполняться со следующего оператора. Такой тип управления программой называется условным ветвлением.

Основная форма конструкции IF-THEN:

```
IF условие THEN
    оператор;
```

Конструкция записана в две строки исключительно ради улучшения читаемости, хотя это один оператор, заканчивающийся точкой с запятой, и нет никакой необходимости ставить лишние пробелы. Надо лишь отделять ключевые слова (такие как IF и THEN) от остальной части конструкции хотя бы одним пробелом.

Пример: Кредитная карта на Паскале

Взгляните на следующую программу, в которой проверяется лимит кредитной карты при некоторой покупке. Программа начинается с объявления константы `Limit`, которая представляет лимит кредитной карты ($1000), и переменной `Amount`, значение которой будет вводиться с клавиатуры. Программа выводит на экран сообщение «Ваш расход принят», если значение `Amount` (Сумма) меньше либо равно `Limit`. Если условие принимает значение FALSE, программа заканчивает работу без сообщений.

```
{ ----------------------------- Пример 3-1 ------------------------------ }
PROGRAM SimpleDecision(INPUT,OUTPUT);
CONST
  Limit = 1000;
VAR
  Amount :REAL;
BEGIN
  WRITE('Пожалуйста, введите сумму:');
  READLN(Amount);
```

```
  IF Amount <= Limit THEN
    WRITELN('Ваш расход принят.'); {Конец оператора IF}
  WRITELN('Для продолжения нажмите ENTER..');
READLN
END.
```

Оператор READLN приостанавливает вывод на экран во время отображения сообщения «Для продолжения нажмите ENTER». Поскольку оператор находится за пределами конструкции IF, он будет выполняться независимо от того, какое значение принимает условие, TRUE или FALSE. Результаты выполнения двух тестов:

Тест 1:

```
Пожалуйста, введите сумму:200
Ваш расход принят.
Для продолжения нажмите ENTER..
```

Тест 2:

```
Пожалуйста, введите сумму:2000
Для продолжения нажмите ENTER..
```

Для представления двух ситуаций, TRUE и FALSE, можно использовать два условных оператора. В следующей программе для обработки другого варианта (сумма превышает 1000) добавлен второй оператор IF. В этом случае на экране появляется сообщение «Сумма превышает лимит вашего кредита».

```
{ ---------------------------- Пример 3-2 ------------------------------ }
PROGRAM TwoConditions(INPUT,OUTPUT);
CONST
  Limit = 1000;
VAR
  Amount :REAL;
BEGIN
  WRITE('Пожалуйста, введите сумму:');
  READLN(Amount);
  IF Amount <= Limit THEN
    WRITELN('Ваш расход принят.');
  IF Amount > Limit THEN
    WRITELN('Сумма превышает лимит вашего кредита.');
  WRITELN('Спасибо за использование кредитной карты на Паскале.');
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Результат выполнения двух запусков программы:

Тест 1:

```
Пожалуйста, введите сумму:150
Ваш расход принят.
Спасибо за использование кредитной карты на Паскале.
Для продолжения нажмите ENTER..
```

Тест 2:

```
Пожалуйста, введите сумму:1500
Сумма превышает лимит вашего кредита.
Спасибо за использование кредитной карты на Паскале.
Для продолжения нажмите ENTER..
```

Как и ранее, заметим, что последние две строки выводятся на экран в каждом случае, поскольку не входят в условные операторы.

Использование блоков

Если для одного условия нужно выполнить несколько операторов, можно воспользоваться блоками BEGIN-END. В главном теле программы может присутствовать любое количество блоков, выделяя группу операторов ключевыми словами BEGIN и END. Блок будет считаться одной конструкцией независимо от количества входящих в него операторов. Рассмотрим следующий пример:

```
{ ------------------------------ Пример 3-3 ------------------------------ }
PROGRAM UsingBlocks(INPUT,OUTPUT);
CONST
  Limit = 1000;
VAR
  Amount :REAL;
  BEGIN
   WRITE('Пожалуйста, введите сумму:');
  READLN(Amount);
  IF Amount <= Limit THEN
    BEGIN
      WRITELN('Ваш расход принят.');
      WRITELN('Ваша цена с налогами: $',1.05*Amount:0:2)
      { Точка с запятой необязательна }
    END;
  IF Amount > Limit THEN
    BEGIN
      WRITELN('Сумма превышает лимит вашего кредита.') ;
      WRITELN('Лимит равен $',Limit)
      { Точка с запятой необязательна }
    END;
  WRITELN('Спасибо за использование кредитной карты на Паскале.');
  WRITELN('Для продолжения нажмите ENTER..');
  READLN { Точка с запятой необязательна }
END.
```

В этом примере выполняется более одного оператора в любом случае (TRUE или FALSE). Поэтому используется два блока.

Обратите внимание, что в трех местах этой программы в конце оператора нет точки с запятой, поскольку она (;) не обязательна. В каждом случае это последний оператор блока.

Результаты выполнения двух тестов:

Тест 1:

```
Пожалуйста, введите сумму:120
Ваш расход принят.
Ваша цена с налогами: $126.00
Спасибо за использование кредитной карты на Паскале.
Для продолжения нажмите ENTER..
```

Тест 2:

```
Пожалуйста, введите сумму:2000
Сумма превышает лимит вашего кредита.
Лимит равен $1000
Спасибо за использование кредитной карты на Паскале.
Для продолжения нажмите ENTER..
```

Если вы напишете программу без блоков, то обнаружите, что только первый оператор, следующий за ключевым словом THEN, входит в оператор IF, а все остальные принадлежат главному телу программы и будут выполнены независимо от условия.

Задание 3-1

Напишите программу, которая принимает символ с клавиатуры и проверяет, является ли он:

- числом;
- буквой в нижнем регистре;
- буквой в верхнем регистре.

Для каждого случая выведите на экран подходящее сообщение.

3-3. Конструкция IF-THEN-ELSE

Форма оператора IF, которую мы использовали до сих пор, на самом деле является простейшей версией полной конструкции. Полный оператор IF включает в себя два варианта действий в зависимости от результата проверки условия. Он выглядит следующим образом:

```
IF условие THEN
   оператор
ELSE
   оператор;
```

Обратите внимание, что здесь лишь одна точка с запятой, поскольку вся конструкция считается одним оператором. Рассмотрим пример:

```
IF AGE < 18 THEN
   WRITELN('Несовершеннолетний.')
ELSE
   WRITELN('Возраст подходящий.');
```

Этот оператор будет выводить на экран сообщение «Несовершеннолетний», если Age меньше 18. В противном случае будет выведено сообщение «Возраст подходящий».

При добавлении других операторов в любой из вариантов необходимо использовать блоки BEGIN-END. Новая конструкция примет вид:

```
IF AGE < 18 THEN
  BEGIN
    WRITELN('Несовершеннолетний.');
    WRITELN('Подождите пару лет.')
  END { Точка с запятой здесь не нужна }
ELSE
  BEGIN
    WRITELN('Возраст подходящий.');
    WRITELN('Вам не нужно ждать.')
  END; { Точка с запятой обязательна}
```

В этом месте применение точки с запятой становится критическим и при неверном использовании может привести к ошибкам. Обратите внимание, что ключевое слово END первого блока не заканчивается точкой с запятой (поскольку это не конец оператора), в то время как в конце второго блока стоит точка с запятой, обозначая конец условного оператора.

Вернемся теперь к программе «Кредитная карта на Паскале», чтобы усилить ее полным оператором IF-THEN-ELSE.

```
{ ----------------------------- Пример 3-4 ------------------------------ }
PROGRAM CreditCard(INPUT,OUTPUT);
CONST
  Limit = 1000;
VAR
  Amount :REAL;
BEGIN
  WRITE('Пожалуйста, введите сумму:');
  READLN(Amount);
{ Начало конструкции IF }
{ --------------------- }
  IF Amount <= Limit THEN
    BEGIN
      WRITELN('Ваш расход принят.');
      WRITELN('Ваша цена с налогами $',1.05*Amount:0:2)
    END
  ELSE
    BEGIN
      WRITELN('Сумма превышает лимит вашего кредита.');
      WRITELN('Лимит равен $',Limit)
```

```
    END;
{ Конец конструкции IF }
{ ------------------- }
  WRITELN('Спасибо за использование кредитной карты на Паскале.');
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Результаты выполнения тестов:

Тест 1:

```
Пожалуйста, введите сумму:1000
Ваш расход принят.
Ваша цена с налогами $1050.00
Спасибо за использование кредитной карты на Паскале.
Для продолжения нажмите ENTER..
```

Тест 2:

```
Пожалуйста, введите сумму:1001
Сумма превышает лимит вашего кредита.
Лимит равен $1000
Спасибо за использование кредитной карты на Паскале.
Для продолжения нажмите ENTER..
```

Задание 3–2

Модифицируйте программу, которую вы написали в задании 2-2, чтобы она вычисляла и вещественные и мнимые корни квадратного уравнения ($Ax^2 + Bx + C$).

3–4. Цепочки ELSE–IF

Оператор IF-THEN-ELSE предназначен для двойного выбора, однако его можно расширить для работы с более сложным выбором. Взгляните на новую схему конструкции, которая иногда называется цепочкой ELSE-IF:

```
IF условие-1 THEN
     оператор-1
ELSE IF условие-2 THEN
     оператор-2
ELSE IF условие-3 THEN
     оператор-3
...
ELSE
     оператор-n;
```

Условия в цепочке вычисляются сверху вниз, и всякий раз, когда условие принимает значение TRUE, выполняется соответствующий опе-

ратор, а остальная конструкция пропускается. Если не удовлетворяется ни одно из условий, в дело вступает завершающий ELSE.

Заметьте, что цепочка условий считается одним оператором и заканчивается точкой с запятой, при этом внутри цепочки точка с запятой отсутствует. Если требуется более одного результирующего оператора, используйте блоки BEGIN-END в соответствии с приведенными выше правилами.

Пример: Тестер символов

Сначала программа просит ввести букву, затем проверяет, в каком регистре введен символ – нижнем или верхнем. Программа может также распознавать числа и выдавать соответствующее сообщение, в остальных случаях выводит на экран: «Простите, это не буква».

Алгоритм программы построен на проверке ASCII-кода вводимого символа с помощью функции ORD. Символы классифицируются следующим образом:

- буквы в верхнем регистре соответствуют кодам от 65 до 90;
- буквы в нижнем регистре соответствуют кодам от 97 до 122;
- цифры соответствуют кодам от 48 до 57.

Если вы уже написали программу задания 3-1, то увидите, что цепочки ELSE-IF облегчают жизнь.

```
{ ------------------------------ Пример 3-5 ------------------------------ }
PROGRAM CharsTester(INPUT,OUTPUT);
VAR
  InputChar :CHAR;
BEGIN
  WRITE('Пожалуйста, введите алфавитный символ:');
  READLN(InputChar);
{ Начало конструкции IF }
{ --------------------- }
  IF (ORD(InputChar) > 64) AND (ORD(InputChar) < 91) THEN
    WRITELN('Это буква в верхнем регистре.')
  ELSE IF (ORD(InputChar) > 96) AND (ORD(InputChar) < 123) THEN
    WRITELN('Это буква в нижнем регистре.')
  ELSE IF (ORD(InputChar) > 47) AND (ORD(InputChar) < 58) THEN
    WRITELN('Привет, это число!')
  ELSE
    WRITELN('Простите, это не буква.');
{ Конец конструкции IF}
{ ------------------- }
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Результаты выполнения четырех тестов с различным вводом:

Тест 1:

```
Пожалуйста, введите алфавитный символ:a     ----> Ввод a
Это буква в нижнем регистре.
Для продолжения нажмите ENTER..
```

Тест 2:

```
Пожалуйста, введите алфавитный символ:B     ----> Ввод B
Это буква в верхнем регистре
Для продолжения нажмите ENTER..
```

Тест 3:

```
Пожалуйста, введите алфавитный символ:5     ----> Ввод 5
Привет, это число!
Для продолжения нажмите ENTER..
```

Тест 4:

```
Пожалуйста, введите алфавитный символ:@     ----> Ввод @
Простите, это не буква.
Для продолжения нажмите ENTER..
```

3–5. Вложенные условия

Оператор, исполняемый после проверки условия, может быть любым. На самом деле это может быть другой оператор IF, вложенный в исходный оператор IF.

Конструкции IF-THEN-ELSE могут быть вложены друг в друга, например, следующим образом:

```
IF условие-1 THEN
    IF условие -2 THEN
        ...
        IF условие -n THEN
          оператор-n1
        ELSE
          оператор -n2
        ...
    ELSE
      оператор -2
ELSE
  оператор -1;
```

Как видите, эта конструкция может справиться с любым количеством вложенных условий, но необходимо отслеживать соответствие IF и ELSE. Применим конструкцию на практике.

Пример: Результаты и отметки

Следующая программа принимает результат учащегося и выводит на экран отметку в соответствии со следующей классификацией:

1. Отметка «A» соответствует результату от 90% до 100%.
2. Отметка «B» соответствует результату от 80% до 89%.
3. Отметка «C» соответствует результату от 70% до 79%.
4. Отметка «D» соответствует результату от 60% до 69%.
5. Отметка «F» соответствует результату менее 60%.

Программа:

```
{ ------------------------------ Пример 3-6 ------------------------------ }
PROGRAM ScoresAndGrades1(INPUT,OUTPUT);
VAR
  Score :INTEGER;
BEGIN
  WRITE('Пожалуйста, введите результат:');
  READLN(Score);
  WRITELN;
{ Начало конструкции IF }
{ -------------------- }
  IF Score > 59 THEN
    IF Score > 69 THEN
      IF Score > 79 THEN
        IF Score > 89 THEN
          WRITELN('Прекрасно. Ваша оценка ''A''')
        ELSE
          WRITELN('Очень хорошо. Ваша оценка ''B''')
      ELSE
        WRITELN('Хорошо. Ваша оценка ''C''')
    ELSE
      WRITELN('Сдано. Ваша оценка ''D''')
  ELSE
    WRITELN('Удачи в следующий раз. Ваша оценка ''F''');
{ Конец конструкции IF }
{ -------------------- }
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Результаты выполнения тестов:

Тест 1:

```
Пожалуйста, введите результат:92       ----> Ввод 92
Прекрасно. Ваша оценка 'A'             ----> Ответ программы
Для продолжения нажмите ENTER..
```

Тест 2:

```
Пожалуйста, введите результат:70
Хорошо. Ваша оценка 'C'
Для продолжения нажмите ENTER..
```

Тест 3:

```
Пожалуйста, введите результат:60
Сдано. Ваша оценка 'D'
Для продолжения нажмите ENTER..
```

Тест 4:

```
Пожалуйста, введите результат:59
Удачи в следующий раз. Ваша оценка 'F'
Для продолжения нажмите ENTER..
```

Как всегда, после проверки условия можно заставить выполняться несколько результирующих операторов, объединив их в блок.

В своих программах вы можете предпочесть любой из допустимых вариантов конструкции IF-THEN-ELSE. Однако некоторые формы могут оказаться более надежными для одного приложения, чем для другого. Взгляните на программу, которая решает ту же проблему («результаты и отметки»), но использует цепочку ELSE-IF. Обратите внимание, насколько применение логических переменных A, B, C, D, и F сделало программу проще и понятнее. Заметьте также, что недопустимые значения фильтруются последним ELSE за пределами цепочки.

```
{ ------------------------------ Пример 3-7 ------------------------------ }
PROGRAM ScoresAndGrades2(INPUT,OUTPUT);
VAR
  Score         :INTEGER;
  A, B, C, D, F :BOOLEAN;
BEGIN
  WRITE(' Пожалуйста, введите результат:');
  READLN(Score);
  A := (Score >= 90) AND (Score <= 100);
  B := (Score >= 80) AND (Score < 90);
  C := (Score >= 70) AND (Score < 80);
  D := (Score >= 60) AND (Score < 70);
  F := (Score < 60) AND (Score >= 0);
  WRITELN;
{ Начало конструкции IF }
{ --------------------- }
  IF A THEN
    WRITELN('Прекрасно. Ваша оценка ''A''')
  ELSE IF B THEN
    WRITELN('Очень хорошо. Ваша оценка ''B''')
  ELSE IF C THEN
    WRITELN(' Хорошо. Ваша оценка ''C''')
  ELSE IF D THEN
    WRITELN('Сдано. Ваша оценка ''D''')
  ELSE IF F THEN
    WRITELN('Удачи в следующий раз. Ваша оценка ''F''')
  ELSE
    WRITELN('Число вне допустимых пределов.');
{ Конец конструкции IF }
{ ------------------- }
```

```
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Приемы решения головоломок IF–ELSE

Вложенность друг в друга конструкций IF может сбить с толку, поскольку трудно сказать, какой ELSE какому IF принадлежит. Взгляните на простой пример:

```
IF X >= 1 THEN
IF y >= 18 THEN
WRITELN('оператор#1.')
ELSE
WRITELN('оператор#2');
```

Согласно правилу каждый ELSE принадлежит последнему IF. Для данного примера это означает, что ELSE принадлежит второму IF. Расположение текста с отступами в соответствии с правилом делает его понятнее:

```
IF X >= 1 THEN
    IF y >= 18 THEN
        WRITELN('оператор#1.')
    ELSE
        WRITELN('оператор#2');
```

Тем не менее при необходимости связать ELSE с первым IF можно использовать блоки, например:

```
IF X >= 1 THEN
    BEGIN
        IF Y >= 18 THEN
            WRITELN('оператор#1.')
    END
ELSE
    WRITELN('оператор#2');
```

Задание 3–3

Напишите программу, которая описывает погоду в соответствии со следующей температурной классификацией:

Температура (по Фаренгейту)	**Классификация**
75 и более	Жарко
от 50 до 74	Прохладно
от 35 до 49	Холодно
менее 35	Мороз

3–6. Множественный выбор: CASE

Конструкция CASE применяется в случае множественного выбора, например для опций пользовательского меню. Ее основная форма:

```
CASE выражение OF
     метка-1 : оператор-1;
     метка-2 : оператор-2;
     ...
     метка-n : оператор-n;
END
```

Case-выражение, называемое также *селектором*, может быть типа INTEGER, CHAR или BOOLEAN (или любого *порядкового* типа, который будет рассмотрен в главе 5). В зависимости от значения этого выражения управление в программе передается на одну из *case-меток* для выполнения соответствующего оператора. Метки представляют различные возможные значения выражения. Взгляните на следующий пример.

Пример: Торговый автомат

Монеты в торговом автомате сортируются в соответствии с весом каждой монеты, который принимается равным 35 граммам для 25-центовой монеты, 7 граммам для 10-центовой и 15 граммам для 5 центов.

Алгоритм может быть запрограммирован следующим образом:

```
CASE CoinWeight OF
   35 : Amount := Quarter;
   7  : Amount := Dime;
   15 : Amount := Nickel;
END;
```

Числа 35, 7 и 15 представляют собой значения переменной `CoinWeight` (вес монеты) и используются в качестве меток. Поэтому когда, например, `CoinWeight` равна 7, выполняется оператор:

```
Amount := Dime;
```

Нет необходимости говорить, что `Dime` – это именованная константа со значением 10, а `Nickel` и `Quarter` – также именованные константы. Взгляните на законченную программу:

```
{ ------------------------------ Пример 3-8 ------------------------------ }
PROGRAM CaseOfWeights(INPUT,OUTPUT);
CONST
  Quarter = 25;
  Dime = 10;
  Nickel = 5;
VAR
```

```
  CoinWeight, Amount :INTEGER;
BEGIN
  WRITE('Пожалуйста, введите вес:');
  READLN(CoinWeight);
  CASE CoinWeight OF
    35 : Amount := Quarter;
    7  : Amount := Dime;
    15 : Amount := Nickel;
END;
  WRITELN('Сумма равна', Amount, ' центов.');
  READLN
END.
```

Результат выполнения программы:

```
Пожалуйста, введите вес:35     ----> Ввод 35
Сумма равна 25 центов.         ----> Ответ программы
```

Для одного результирующего оператора можно использовать более одной метки, что значительно сократит код по сравнению с применением IF. Рассмотрим следующий пример.

Пример: Количество дней в месяце

Предположим, вы хотите написать программу, которая читала бы номер месяца и сообщала количество дней в нем. Конструкция CASE будет выглядеть примерно так:

```
CASE Month OF
   1,3,5,7,8,10,12   : Days := 31;
   4,6,9,11          : Days := 30;
   2                 : Days := 28;
END;
```

Как видите, конструкция CASE включает три ветви, две из которых имеют более одной метки.

Все месяцы, в которых 31 день, принадлежат к первой ветви, те, в которых 30 дней, принадлежат ко второй ветви, а для февраля выделена отдельная ветвь. Для простоты предполагаем, что в феврале 28 дней, но можно расширить алгоритм определением високосного года и присваивать февралю 28 или 29 дней соответственно. В этом случае для одной ветви можно использовать блок операторов, например:

```
CASE Month OF
   1,3,5,7,8,10,12 : Days := 31;
   4,6,9,11        : Days := 30;
   2               : BEGIN
                         WRITE('Введите год:');
                         READLN(Year);
                         IF YEAR MOD 4 = 0 THEN
```

```
            Days :=29
          ELSE
            Days :=28
        END;
```

Теперь метка 2 ведет к блоку операторов. Таким образом, если в качестве номера месяца ввести 2, то программа запросит год. Год будет проверен, и вы получите 29 для високосного года и 28 в противном случае. Готовая программа:

```
{ ------------------------------Пример 3-9 ----------------------------- }
PROGRAM DaysOfMonth1(INPUT,OUTPUT);
VAR
  Days, Month, Year :INTEGER;
BEGIN
  WRITE('Пожалуйста, введите номер месяца:');
  READLN(Month);
  CASE Month OF
     1,3,5,7,8,10,12 : Days := 31;
     4,6,9,11        : Days := 30;
     2               : BEGIN
                         WRITE('Введите год:');
                         READLN(Year);
                         IF YEAR MOD 4 = 0 THEN
                           Days :=29
                         ELSE
                           Days :=28
                       END;
  END;
  WRITELN('В этом месяце ',Days,' дней.');
  READLN
END.
```

Результат выполнения тестов:

Тест 1:

```
Пожалуйста, введите номер месяца:2
Введите год:1987
В этом месяце 28 дней.
```

Тест 2:

```
Пожалуйста, введите номер месяца:2
Введите год:1984
В этом месяце 29 дней.
```

Тест 3:

```
Пожалуйста, введите номер месяца:12
В этом месяце 31 дней.
```

Обратите внимание, что в этой программе используется упрощенный алгоритм определения високосного года, который работает для годов в пределах столетия, например с 1901 по 1999. Полный алгоритм определения високосного года следующий:

- *год делится на 4 и не делится на 100*

или

- *год делится на 400.*

Для получения более детальной информации обратитесь к упражнению 3 в конце главы.

В подобных случаях использование конструкций CASE более эффективно, чем применение вложенных конструкций IF-THEN-ELSE или цепочек. Тем не менее необходимо понимать, что иногда нужны и они (например, в случае февраля).

Задание 3–4

Напишите программу, которая читает дату с клавиатуры в формате «мм дд гг» и выводит ее на экран, например так:

Январь 2-е, 1992
Октябрь 23-е, 1990
Март 5-е, 1985

3–7. Безусловный переход: GOTO

Оператор GOTO используется для передачи управления из одной точки программы в другую. Это называется *безусловным переходом*. Оператор GOTO очень легок в применении, но редко встречается в программах на Паскале, поскольку нарушает их структуру. Однако иногда он позволяет, сделав один переход, избежать большого количества уровней вложенности. Синтаксис оператора GOTO следующий:

```
GOTO метка;
```

Метка – это положительное число не более 4 цифр, предшествующее вызываемому оператору (в Турбо Паскале в качестве метки может выступать любой допустимый идентификатор, который может начинаться с цифры).

```
    GOTO 1000;
    ...
1000:
    WRITELN('Я оператор с меткой.');
    ...
```

Когда встречается оператор GOTO, управление программой передается помеченному оператору. Метка должна быть объявлена в *секции меток*

раздела объявлений программы. Секция меток начинается с ключевого слова LABEL и стоит первой в разделе объявлений стандартного Паскаля (в Турбо Паскале это не обязательно). Рассмотрим пример:

```
PROGRAM GoToDemo(INPUT,OUTPUT);
LABEL
  1000;
VAR
  InputChar :CHAR;
BEGIN
  WRITE('Пожалуйста, введите букву (или 0 для выхода):');
  READLN(InputChar);
  IF InputChar = '0' THEN
    GOTO 1000;
    { Другие операторы могут переходить сюда.. }
1000:
END.
```

В этом примере проверяется, не является ли входной символ нулем, и в этом случае управление передается на метку «1000», где программа заканчивает работу. При работе в Турбо Паскале вместо чисел можно использовать осмысленные метки, например «Wrapup» (конец) или «Start» (начало).

Повторяющиеся циклы

Можно использовать GOTO для построения замкнутых циклов. Например, если вы хотите повторять выполнение программы «Тестер символов», то можете использовать алгоритм работы, при котором управление всегда передается на метку «1000», то есть в начало программы. Условием для завершения цикла (и программы) будет проверка входного значения. Если введен ноль, управление передается на метку «2000», заканчивая программу. Если удалить это условие из программы, то она будет работать бесконечно. В этом случае единственный способ выйти из программы – нажать <Ctrl>+<Break>.[1] Такой цикл называется бесконечным.

```
{ ------------------------------ Пример 3-10 ----------------------------- }
PROGRAM CharsTester2(INPUT,OUTPUT);
LABEL
  1000, 2000; { объявление меток }
VAR
  InputChar :CHAR;
BEGIN
1000:
  WRITE('Пожалуйста, введите букву (или 0 для выхода): ');
```

[1] Речь идет о выполнении программы под управлением операционной системы DOS. – *Примеч. науч. ред.*

```
  READLN(InputChar);
{ Начало конструкции IF }
{ -------------------- }
  IF InputChar = '0' THEN { условие выхода }
    GOTO 2000
  ELSE IF (ORD(InputChar) > 64) AND (ORD(InputChar) < 91) THEN
    WRITELN('Это буква в верхнем регистре.')
  ELSE IF (ORD(InputChar) > 96) AND (ORD(InputChar) < 123) THEN
    WRITELN('Это буква в нижнем регистре.')
  ELSE IF (ORD(InputChar) > 47) AND (ORD(InputChar) < 58) THEN
    WRITELN('Привет, это число!')
  ELSE
    WRITELN('Извините, это не буква.');
{ Конец конструкции IF }
{ -------------------- }
  GOTO 1000; { перезапуск программы }
2000: { выход из программы }
END.
```

Результат выполнения программы:

```
Пожалуйста, введите букву (или 0 для выхода:W     ----> Ввод W
Это буква в верхнем регистре.
Пожалуйста, введите букву (или 0 для выхода:e     ----> Ввод e
Это буква в нижнем регистре.
Пожалуйста, введите букву (или 0 для выхода:0     ----> Ввод 0
```

Как видите, это не лучший способ построения циклов или управления программой, поскольку он основан на переходах из одной точки программы в другую. В следующей главе вы познакомитесь со структурированными циклами Паскаля.

3–8. Возможности Турбо Паскаля: EXIT, CASE–ELSE

Если в примере 3-9 в качестве номера месяца ввести недопустимое значение, например 13, то просто получим сообщение,

```
В месяце 0 дней.
```

Для обработки неправильных данных нужно использовать подходящую конструкцию IF. В Турбо Паскале для работы с данными, которые не принадлежат ни одной из меток, к управляющей структуре CASE можно добавить ELSE. Тогда структура CASE примет вид:

```
CASE выражение OF
     метка-1 : оператор-1;
     метка-2 : оператор-2;
     ...
     метка-n : оператор-n;
ELSE
```

```
    оператор
END
```

Другая возможность Турбо Паскаля – оператор EXIT, который заканчивает выполнение программы в любом месте. Оператор EXIT относится к операторам безусловного перехода. В следующей программе демонстрируются эти две возможности. Если ввести любое число, отличное от чисел от 1 до 12, то раздел ELSE и оператор EXIT остановят выполнение программы.

```
{ ------------------------------ Пример 3-11 ---------------------------- }
PROGRAM DaysOfMonth2(INPUT,OUTPUT);
LABEL
  Start;
VAR
  Days, Month, Year :INTEGER;
BEGIN
Start:
  WRITE('Пожалуйста, введите номер месяца: ');
  READLN(Month);
  CASE Month OF
     1,3,5,7,8,10,12 : Days := 31;
     4,6,9,11        : Days := 30;
     2               : BEGIN
                         WRITE('Введите год:');
                         READLN(Year);
                         IF YEAR MOD 4 = 0 THEN
                           Days :=29
                         ELSE
                           Days :=28
                       END;
     ELSE
       EXIT           { остальные случаи}
  END;
  WRITELN('В этом месяце ',Days,' дней.');
  GOTO Start
END.
```

Результат выполнения теста:

```
Пожалуйста, введите номер месяца:1
В этом месяце 31 дней.
Пожалуйста, введите номер месяца:4
В этом месяце 30 дней.
Пожалуйста, введите номер месяца:13      ----> Выход из программы
```

Заключение

В этой главе вы познакомились со структурами управления ветвлением, которые помогают оперировать решениями в вашей программе:

1. С простым оператором IF-THEN, применяемым в простых решениях:

```
IF условие THEN
   оператор;
```

2. С полной конструкцией IF-THEN-ELSE, содержащей результат и его альтернативу:

```
IF условие THEN
   оператор
ELSE
   оператор;
```

3. Со способами реализации сложных условий с помощью цепочки ELSE-IF:

```
IF условие-1 THEN
   оператор-1
ELSE IF условие-2 THEN
   оператор-2
ELSE IF условие-3 THEN
   оператор-3
...
ELSE
   оператор-n;
```

4. С альтернативой цепочки – вложенными друг в друга конструкциями IF-THEN-ELSE:

```
IF условие-1 THEN
  IF условие-2 THEN
      ...
      IF условие-n THEN
        оператор-n1
      ELSE
        оператор-n2
      ...
  ELSE
    оператор-2
ELSE
  оператор-1;
```

5. С конструкцией множественного выбора CASE, пригодной для обработки многих случаев:

```
CASE выражение OF
     метка-1 : оператор-1;
     метка-2 : оператор-2;
     ...
     метка-n : оператор-n;
END
```

6. С особенностями конструкции CASE в Турбо Паскале – у нее больше возможностей, поскольку она может включать раздел ELSE, который обрабатывает все случаи, не соответствующие метке.

```
CASE выражение OF
      метка-1 : оператор-1;
      метка-2 : оператор-2;
      ...
      метка-n : оператор-n;
ELSE
      оператор
END
```

   Вы также знаете, что можно в любых приведенные выше выражениях заменить один оператор на блок операторов, используя блоки BEGIN-END.

7. С оператором безусловного перехода GOTO, который передает управление программой помеченному оператору:

```
GOTO метка;
```

   Метка в стандартном Паскале – это положительное целое до четырех цифр, в то время как в Турбо Паскале она может быть любым допустимым идентификатором и может начинаться с цифры. Вы также знаете, как объявлять метки в начале раздела объявлений программы. В Турбо Паскале секция LABEL не обязательно должна быть первой секцией.

8. Наконец, вы познакомились с оператором Турбо Паскаля EXIT, который останавливает работу программы в любом месте.

В следующей главе мы продолжим рассмотрение управляющих структур и научимся строить структурированные циклы.

Упражнения

1. Что выводит каждый из следующих операторов WRITELN?
   a. WRITELN(5 > 4);
   b. WRITELN((5 < 4) OR (4 > 1));
   c. WRITELN(TRUE OR FALSE);
   d. WRITELN(TRUE AND FALSE);
   e. WRITELN(TRUE OR FALSE AND TRUE);
   f. WRITELN(TRUE OR FALSE AND NOT FALSE);
2. Напишите программу на Паскале, которая читает три различных числа и печатает наибольшее из них.

3. В примерах 3-9 и 3-11 используется упрощенная версия определения високосного года. Она пригодна только для годов в пределах столетия. Полный алгоритм определения високосного года таков:

   - год делится на 4 и не делится на 100

   или

   - год делится на 400.

   Взяв за основу этот алгоритм, модифицируйте пример 3-9. Можно использовать следующие логические переменные:

```
R4   := Year MOD 4 = 0;
R100 := Year MOD 100 = 0;
R400 := Year MOD 400 = 0;
```

Ответы

1. a. True b. True c. True d. False e. True f. True

2. Следующая программа сравнивает три числа: A, B и C:

```
VAR
  A, B, C, MaximumNumber :INTEGER;
BEGIN
    READ (A, B, C);
    IF A > B THEN
      MaximumNumber := A
    ELSE
      MaximumNumber := B;
    IF MaximumNumber < C THEN
      MaximumNumber := C;
    WRITELN('Maximum Number=',MaximumNumber);
END.
```

3. Следующий оператор CASE предназначен для версии программы с определением високосного года:

```
CASE Month OF
     1,3,5,7,8,10,12 : Days := 31;
     4,6,9,11        : Days := 30;
     2               : BEGIN
                         WRITE('Введите год:');
                         READLN(Year);
                         R4 := Year MOD 4 = 0;
                         R400 := Year MOD 400 = 0;
                         R100 := Year MOD 100 = 0;
                         Leap := (NOT R100 AND R4) OR (R400);
                         IF (Leap) THEN
                           Days := 29
                         ELSE
                           Days := 28;
                       END
END;
```

4

Циклы

4–1. Построение циклов

В предыдущей главе вы научились строить повторяющиеся циклы с помощью следующих инструментов:

- Оператор перехода, такой как GOTO, многократно передающий управление в начало программы.
- Условие выхода из цикла при необходимости.

Условие может быть использовано для проверки начального значения и выхода из цикла при достижении определенного значения. Для того чтобы повторить цикл некоторое число раз, необходимо организовать счетчик. В этом случае условие используется для проверки счетчика при каждом выполнении цикла. Этот тип цикла называется *арифметическим циклом*. В следующей программе эти простые приемы применяются для пятикратного вывода на экран сообщения «Прошу прощения, повторите еще раз». Алгоритм, используемый в программе, следующий:

1. Счетчик получает нулевое начальное значение.
2. Приращение счетчика на 1.
3. Проверяется, меньше или равно 5 значение счетчика.
4. Вывод сообщения.
5. Переход на шаг 2.

```
{ ------------------------------ Пример 4-1 ------------------------------ }
PROGRAM GoToLoop(OUTPUT);
LABEL
  1000; { объявление метки}
VAR
  Kounter :INTEGER;
BEGIN
```

```
  Kounter := 0;
1000:
  Kounter := Kounter + 1;
  IF Kounter <= 5 THEN
    BEGIN
      WRITELN(' Прошу прощения, повторите еще раз..');
      GOTO 1000 { перезапуск }
    END;
  WRITELN;
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

В этой программе перед входом в цикл, начинающийся с метки «1000», счетчику присваивается нулевое начальное значение. Внутри цикла счетчик увеличивается, а затем проверяется, меньше ли он или равен 5. Если да, то выполняется оператор WRITELN, и цикл повторяется с помощью оператора GOTO. Если условие не выполняется (то есть значение счетчика больше 5), программа заканчивает работу. Вывод программы выглядит так:

```
Прошу прощения, повторите еще раз..
Прошу прощения, повторите еще раз..
Прошу прощения, повторите еще раз..
Прошу прощения, повторите еще раз..
Прошу прощения, повторите еще раз..
Для продолжения нажмите ENTER..
```

Паскаль предоставляет готовые к использованию управляющие структуры для построения циклов, поэтому можно избежать такого нескладного кода. В одной конструкции управляющей структуры содержится и оператор ветвления, и условие.

В этой главе мы рассмотрим следующие конструкции:

- Цикл FOR
- Цикл WHILE
- Цикл REPEAT

Каждый из трех циклов имеет свои возможности, подходящие для различных приложений.

4–2. Цикл FOR

Цикл FOR представляет собой арифметический цикл, повторяющий выполнение оператора или блока операторов определенное число раз. Взгляните на пример:

```
{ ----------------------------- Пример 4-2 ----------------------------- }
PROGRAM ForLoop(OUTPUT);
```

```
VAR
  Kounter :INTEGER;
BEGIN
  FOR Kounter := 1 TO 5 DO
    WRITELN('Прошу прощения, повторите еще раз..');
  WRITELN;
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Эта программа приводит к тому же результату, что и предыдущая, но она проще и лучше организована. Цикл FOR выполняет ту же работу, что и предыдущая программа. Он присваивает *управляющей переменной* (Kounter) *начальное значение* = 1, затем выполняет оператор, увеличивает значение управляющей переменной на единицу и повторяет процесс, пока значение Kounter не достигнет *конечного значения* = 5.

Основная форма конструкции FOR следующая:

```
FOR управляющая переменная := выражение-1 TO выражение-2 DO
     оператор;
```

где управляющая переменная – это счетчик цикла, выражение-1 – начальное значение, а выражение-2 – конечное значение.

Управляющая переменная, выражение-1 и выражение-2 могут иметь любой тип, за исключением REAL.[1] Все они должны быть одного типа.

Помните, что конструкция FOR – это один оператор, заканчивающийся точкой с запятой. Если по ошибке вы добавите еще одну точку с запятой, как в следующем цикле:

```
FOR Kounter := 1 TO 1000 DO;
          WRITELN('Прошу прощения, повторите еще раз..');
```

не удивляйтесь, если цикл выполнится только один раз независимо от конечного значения счетчика. Точка с запятой после ключевого слова DO прекращает выполнение цикла в этом месте.

Внутри цикла нельзя изменять значение управляющей переменной.[2] Взгляните на оператор присваивания, расположенный внутри цикла:

```
FOR K := 1 TO 10 DO
  K := 2
...
```

---

[1] Современные компиляторы понимают десятки встроенных (предопределенных) типов. Между тем, не все типы, кроме REAL, могут быть использованы в конструкции FOR. Для получения дополнительных сведений обратитесь к справочной информации по используемому вами компилятору. – *Примеч. науч. ред.*

[2] Компилятор это допускает, однако это строго не рекомендуется, разве только вы точно знаете, что делаете. – *Примеч. науч. ред.*

Даже если компилятор примет такой оператор, он отменит действие цикла, поскольку счетчик цикла все время равен 2. То же самое правило действует и для начального и конечного значений управляющей переменной.

Как обычно, использование блоков BEGIN-END позволяет включить в цикл столько операторов, сколько хотите.

Пример: Степени двойки

Число 2 и его степени играют важную роль в компьютерной сфере. Некоторые числа, такие как 1024 байт (эквивалентно 1 Кбайт) и 65 536 байт (64 Кбайт), имеют широкое применение. В следующей программе для вывода на экран степеней двойки применяется цикл FOR, а степень вычисляется по алгоритму из примера 2-2. Программа выводит степень и результат возведения числа 2 в эту степень. Начальное и конечное значения счетчика задаются пользователем в процессе выполнения программы. Таким образом, вы можете определять интервал чисел, который хотели бы протестировать:

```
{ ------------------------------ Пример 4-3 ------------------------------ }
PROGRAM ForLoop(INPUT, OUTPUT);
VAR
  Base, Power, Start, Final :INTEGER;
BEGIN
  Base := 2;
  WRITE('Введите начальную степень:');
  READLN(Start);
  WRITE('Введите конечную степень:');
  READLN(Final);
  WRITELN;
  WRITELN('Число     Степень двойки');
  FOR Power := Start TO Final DO
    BEGIN
      WRITE(Power:3);
      WRITELN(EXP(LN(Base)*Power):20:0)
    END;
  WRITELN;
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Результат выполнения примера с использованием значений степени от 1 до 20:

```
Введите начальную степень:1
Введите конечную степень:20
Число    Степень двойки
  1                 2
  2                 4
  3                 8
  4                16
```

```
 5              32
 6              64
 7             128
 8             256
 9             512
10            1024
11            2048
12            4096
13            8192
14           16384
15           32768
16           65536
17          131072
18          262144
19          524288
20         1048576
Для продолжения нажмите ENTER..
```

Задание 4-1

Напишите программу определения високосного года в пределах от 1900 до 2000 года. Выведите на экран год и результат проверки в виде:

```
Год 1990 невисокосный год.
Год 1991 невисокосный год.
Год 1992 високосный год.
Год 1993 невисокосный год.
Год 1994 невисокосный год.
Год 1995 невисокосный год.
Год 1996 високосный год.
Год 1997 невисокосный год.
Год 1998 невисокосный год.
Год 1999 невисокосный год.
Год 2000 високосный год.
```

Пример: Среднее значение

Следующая программа демонстрирует ввод данных с применением цикла. Она принимает с клавиатуры последовательность чисел и вычисляет сумму и среднее значение чисел. В начале выполнения программы вас просят ввести число элементов, которое представляет конечное значение счетчика. Внутри цикла сумма накапливается в переменной `Sum` с помощью оператора:

```
Sum := Sum + Number;
```

Во время выполнения цикла среднее значение определяется по сумме и количеству элементов оператором:

```
Average := Sum / N;
```

Вот код программы:

```
{ ------------------------------ Пример 4-4 ------------------------------ }
PROGRAM AverageProg1(INPUT,OUTPUT);
VAR
  Average, Sum, Number :REAL;
  N, Kounter           :INTEGER;
BEGIN
  Sum := 0;
  WRITE('Введите количество элементов:');
  READLN(N);
  FOR kounter := 1 TO N DO
    BEGIN
      WRITE('Введите элемент #',kounter,': ');
      READLN(Number);
      Sum := Sum + Number { Точка с запятой необязательна}
    END;
  Average := Sum / N;
  WRITELN;
  WRITELN('Сумма чисел = ', Sum:0:2);
  WRITELN('Среднее значение чисел = ', Average:0:2);
  WRITELN;
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Результат выполнения программы:

```
Введите количество элементов: 5
Введите элемент #1: 1
Введите элемент #2: 2
Введите элемент #3: 3
Введите элемент #4: 4
Введите элемент #5: 5
Сумма чисел = 15.00
Среднее значение чисел = 3.00
Для продолжения нажмите ENTER..
```

Обратите внимание, как количество элементов было выведено на экран внутри цикла с использованием значений управляющей переменной `Kounter`.

4–3. Пошаговое выполнение вверх и вниз

В предыдущих примерах счетчик цикла FOR всегда увеличивался. Это означает, что конечное значение должно быть больше начального, в противном случае цикл никогда не будет выполнен.

Альтернативная форма цикла FOR позволяет уменьшать значение счетчика путем замены ключевого слова TO на ключевое слово DOWNTO:

```
FOR управляющая переменная := выражение-1
            DOWNTO выражение-2 DO
    оператор;
```

Теперь можно начинать считать с большего значения и идти вниз, пока не будет достигнуто конечное значение.

Пример: Факториал

Факториал положительного целого числа определяется так:

```
N! = N * (N-1) * (N-2) ... * 3 * 2 * 1
```

Таким образом, факториал 4 равен 4*3*2*1, а факториал 3 равен 3*2*1. Можно написать следующие соотношения для факториала:

```
4! = 4 * 3!
3! = 3 * 2!
2! = 2 * 1!
1! = 1
```

Вообще говоря, можно написать следующий оператор Паскаля для вычисления факториала с использованием счетчика:

```
Factorial := Factorial * Kounter;
```

Переменной Kounter можно давать или положительное приращение от 1 до N, или отрицательное – от N до 1. Следующая программа использует этот алгоритм в цикле с отрицательным приращением:

```
{ ------------------------------ Пример 4-5 ------------------------------- }
PROGRAM FactorialProg1(INPUT,OUTPUT);
VAR
  Factorial       :REAL;
  Kounter, Number :INTEGER;
BEGIN
  WRITE('Дайте мне число, и я скажу вам его факториал: ');
  READLN(Number);
  Factorial := 1;
  FOR kounter := Number DOWNTO 1 DO
    Factorial := Factorial * Kounter;
  WRITELN('Факториал', Number,' равен ', Factorial:0:0);
  WRITELN;
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Обратите внимание, что начальное значение 1 должно быть присвоено переменной Factorial до начала итерации. Результат выполнения программы:

```
Дайте мне число, и я скажу вам его факториал: 8
Факториал 8 равен 40320
Для продолжения нажмите ENTER..
```

Хотя факториал всегда является целым числом, использование типа REAL (или LONGREAL в Турбо Паскале) для переменной Factorial предоставляет больше места для хранения быстро растущих результатов вычисления факториала. При использовании типа INTEGER уже после 7! программа начнет выдавать странные результаты.

Задание 4–2

Измените предыдущую программу так, чтобы она проверяла входные значения. При нулевом значении программа должна заканчивать работу, не начиная выполнение цикла. Можно использовать оператор GOTO или функцию Турбо Паскаля EXIT.

4–4. Вложенные циклы

Подобно любому другому оператору, оператор цикла FOR может быть использован внутри другого оператора. В соответствии с нуждами приложения можно вкладывать сколь угодно много операторов цикла друг в друга. Следующая программа выводит на экран массив чисел:

```
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
```

Массив состоит из трех строк и пяти столбцов. Количеством строк и столбцов можно управлять с помощью счетчиков двух вложенных циклов. Как видите, для каждого прохода счетчика внешнего цикла (`Row`) счетчик внутреннего цикла (`Column`) делает пять проходов. Значения, которые появляются на выходе, – это значения счетчика `Column`. Заметьте, что с помощью счетчика внешнего цикла после вывода всей строки на экран выводится пустая строка.

```
{------------------------------ Пример 4-6 ------------------------------}
PROGRAM NestedLoops(OUTPUT);
VAR
  Row, Column :INTEGER;
BEGIN
  FOR Row := 1 TO 3 DO { Начало внешнего цикла }
    BEGIN
      FOR Column := 1 to 5 DO { Начало внутреннего цикла }
        WRITE(Column, ' '); { Конец внутреннего цикла}
      WRITELN { Этот оператор принадлежит внешнему циклу }
    END { Конец внешнего цикла}
END.
```

Обратите внимание на два ключевых слова END в предыдущей программе, первое из которых без точки с запятой, поскольку это последний оператор главного блока (главное тело программы). Ключевое слово WRITELN, стоящее перед этим END, также не заканчивается точкой с запятой, так

как это последний оператор в блоке цикла. И хотя они не обязательны, вы можете поставить точки с запятой. Если вы добавите в конец другой оператор (например, задержки вывода на экран), то ситуация изменится.

Задание 4–3

Измените последнюю программу так, чтобы она рисовала 50 звездочек в пяти строчках, как показано ниже:

```
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
```

4–5. Цикл WHILE

Конструкция цикла WHILE содержит необходимое условие завершения цикла, но, в отличие от цикла FOR, в ней отсутствует счетчик. Ее основной вид:

```
WHILE условие DO
    оператор;
```

Эта форма говорит: «Выполняйте следующий оператор, пока условие принимает значение TRUE».

После входа в цикл проверяется условие (логическое выражение). Если его значение равно TRUE, выполняется оператор, стоящий за ключевым словом DO. Цикл будет повторяться, а оператор будет повторно выполняться, пока условие не примет значение FALSE. Программа должна иметь соответствующую логику, чтобы в нужное время условие приняло значение FALSE. В этом цикле можно использовать счетчик, но вам нужно самим заботиться о его положительном или отрицательном приращении.

Следующая программа демонстрирует тот же самый алгоритм вычисления среднего значения множества чисел, введенных с клавиатуры, но с помощью цикла WHILE. Условие состоит в сравнении значения счетчика Kounter с максимальным количеством элементов N. При достижении максимума цикл завершается.

```
{ ------------------------------ Пример 4-7 ------------------------------ }
PROGRAM AverageProg2(INPUT,OUTPUT);
VAR
  Average, Sum, Number :REAL;
  Kounter, N           :INTEGER;
BEGIN
  Sum := 0;
  Kounter := 1;
```

```
  WRITE('Введите число элементов:');
  READLN(N);
  WHILE Kounter <= N DO
    BEGIN
      WRITE('Введите элемент #',Kounter,': ');
      READLN(Number);
      Sum := Sum + Number;
      Kounter := Kounter + 1
    END;
  Average := Sum / N;
  WRITELN;
  WRITELN('Сумма чисел = ', Sum:0:2);
  WRITELN('Среднее значение чисел = ', Average:0:2);
  WRITELN;
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Обратите внимание, что счетчик получает начальное значение в начале программы, а приращение – внутри цикла. Для первого прохода используется начальное значение (`Kounter := 1`), потому что приращение выполняется в конце. Это один из способов, но в нескольких следующих программах используются другие комбинации. Не забудьте также, что если в цикле надо выполнить более одного оператора, их нужно поместить в блок BEGIN-END.

Результат выполнения программы:

```
Введите количество элементов: 3
Введите элемент #1: 1
Введите элемент #2: 2
Введите элемент #3: 3
Сумма чисел = 6.00
Среднее значение чисел = 2.00
Для продолжения нажмите ENTER..
```

Можно не вводить количество элементов заранее, а подсчитывать их в цикле. В этом случае необходим ключ для завершения цикла, например ввод отрицательного числа. Взгляните на модифицированную версию программы, где каждое вводимое число проверяется на равенство –1:

```
{ ----------------------------- Пример 4-8 ----------------------------- }
PROGRAM AverageProg3(INPUT,OUTPUT);
VAR
  Average, Sum, Number :REAL;
  Kounter              :INTEGER;
BEGIN
  Sum := 0;
  Average := 0;
  Number := 0;
  Kounter := 0;
```

```
  WHILE Number<>-1 DO
    BEGIN
      Kounter := Kounter + 1;
      Sum := Sum + Number;
      WRITE('Введите элемент #',Kounter,' (или -1 для выхода): ');
      READLN(Number)
    END;
  IF Kounter > 1 THEN
  Average := Sum / (Kounter - 1);
  WRITELN;
  WRITELN('Сумма чисел = ', Sum:0:2);
  WRITELN('Среднее значение чисел = ', Average:0:2);
  WRITELN;
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Результат выполнения программы:

```
Введите элемент #1 (или -1 для выхода): 1
Введите элемент #2 (или -1 для выхода): 2
Введите элемент #3 (или -1 для выхода): 3
Введите элемент #4 (или -1 для выхода): -1
Сумма чисел = 6.00
Среднее значение чисел = 2.00
Для продолжения нажмите ENTER..
```

Обратите внимание на следующие моменты в этой программе:

- Оператор цикла стоит в конце блока цикла, так что входное значение может быть проверено перед любой обработкой.
- Среднее значение определяется путем деления суммы на уменьшенное на единицу значение счетчика. Это нужно для того, чтобы учесть лишний проход, когда `Number` равно –1.
- Среднее значение вычисляется, только если переменная `Kounter` не равна 1. Это сделано для того, чтобы избежать ошибки «деления на ноль» в случае выхода из программы до введения данных. В таком случае будет получен следующий ответ:

```
Введите элемент #1 (или -1 для выхода): -1
Сумма чисел = 0.00
Среднее значение чисел = 0.00
Для продолжения нажмите ENTER..
```

Задание 4–4

Составьте программу, содержащую конструкцию цикла WHILE, которая выводит на экран таблицу умножения в виде:

1 * X = Y
2 * X = Y
3 * X = Y
4 * X = Y
5 * X = Y
6 * X = Y
7 * X = Y
8 * X = Y
9 * X = Y
...

Значение X принимается с клавиатуры, а значение Y представляет собой результат умножения.

4–6. Цикл REPEAT

Этот цикл применяется для выполнения группы операторов, пока не будет выполнено определенное условие. Он имеет вид:

```
REPEAT
     оператор-1;
     оператор -2;
     ...
     оператор -n;
UNTIL условие;
```

Как видно из этой формы, цикл может выполнить более одного оператора без применения блоков BEGIN-END. Другое отличие между циклом WHILE и циклом REPEAT в том, что цикл REPEAT выполняется хотя бы один раз независимо от условия, так как каждый проход начинается выполнением оператора и заканчивается проверкой условия. В некоторых приложениях эта возможность необходима.

Взгляните на алгоритм вычисления факториала с использованием цикла REPEAT:

```
  ...
  Factorial := 1;
  Kounter := Number;
REPEAT
  Factorial := Factorial * Kounter;
  Kounter := Kounter - 1;
UNTIL Kounter = 0;
```

Когда Kounter достигает нуля (показывает, что значение 1 уже использовано), то другие проходы уже не нужны, и цикл завершается. Можно также реализовать алгоритм пошагового увеличения, например:

```
...
Factorial := 1;
Kounter := 1;
REPEAT
  Factorial := Factorial * Kounter;
  Kounter := Kounter + 1;
UNTIL Kounter = Number + 1;
```

В этом случае цикл завершается, когда значение Kounter достигает Number+1, и это говорит о том, что значение Number уже было использовано.

В следующей программе цикл REPEAT вложен в цикл WHILE. Программа будет повторно выполняться, пока не будет введен 0 для ее остановки.

```
{ ------------------------------ Пример 4-9 ---------------------------- }
PROGRAM FactorialProg2(INPUT,OUTPUT);
VAR
  Factorial       :REAL;
  Kounter, Number :INTEGER;
BEGIN
  WRITE('Дайте мне число (или 0 для выхода): ');
  READLN(Number);
  WHILE Number<>0 DO { Начало цикла WHILE }
    BEGIN
      Factorial := 1;
      Kounter := 1;
      REPEAT { Начало цикла REPEAT }
        Factorial := Factorial * Kounter;
        Kounter := Kounter + 1;
      UNTIL Kounter = Number + 1; { End of the REPEAT loop }
      WRITELN('Факториал ', Number,' равен ', Factorial:0:0);
      WRITE('Дайте мне число (или 0 для выхода): ');
      READLN(Number)
    END; { Конец цикла WHILE }
  WRITELN('Я вышла!')
END.
```

Обратите внимание, что здесь используются два одинаковых оператора ввода – один перед циклом WHILE, а другой внутри. Первый нужен для присвоения начального значения переменной Kounter перед входом в цикл, чтобы было с чем сравнивать внутри цикла. Результат выполнения программы:

```
Дайте мне число (или 0 для выхода): 3
Факториал из 3 равен 6
Дайте мне число (или 0 для выхода): 5
Факториал из 5 равен 120
Дайте мне число (или 0 для выхода): 0
Я вышла!
```

Задание 4-5

Перепишите предыдущую программу, используя цикл FOR в качестве внутреннего цикла, а цикл WHILE – в качестве внешнего.

Заключение

В этой главе мы рассмотрели три управляющие структуры, предназначенные для построения циклов:

- Цикл FOR
- Цикл WHILE
- Цикл REPEAT

1. Цикл FOR применяется для повторного выполнения оператора или блока операторов определенное число раз:

```
FOR управляющая переменная := выражение-1 TO выражение-2 DO
      оператор;
```

где управляющая переменная – это счетчик цикла, выражение-1 – начальное значение, а выражение-2 – конечное значение.

2. Отрицательное приращение счетчика реализуется при помощи альтернативной формы оператора FOR:

```
FOR управляющая переменная := выражение-1
                DOWNTO выражение-2 DO
      оператор;
```

3. Цикл WHILE предназначен для выполнения оператора или блока операторов, пока определенное условие принимает значение TRUE. Основная форма конструкции:

```
WHILE условие DO
    оператор;
```

4. И в цикле FOR, и в цикле WHILE можно выполнить несколько операторов, помещая их в блок BEGIN-END.

5. Цикл REPEAT служит для выполнения группы операторов, пока определенное условие не примет значение FALSE. Его основная форма:

```
REPEAT
     оператор-1;
     оператор-2;
     оператор-n;
UNTIL условие;
```

6. Теперь вы знаете, что главное отличие цикла REPEAT от двух других операторов в том, что операторы внутри цикла REPEAT выполняются хотя бы один раз независимо от условия.
7. Вы также понимаете, что цикл REPEAT может выполнить несколько операторов без применения блоков BEGIN-END.
8. Наконец, вы изучили в этой главе, что конструкции циклов могут быть вложены в другие конструкции (включая другие циклы).

Упражнения

1. Определите, истинны или ложны следующие утверждения:
   a. Тело цикла WHILE выполняется хотя бы один раз.
   b. Тело цикла REPEAT выполняется хотя бы один раз.
   c. Тело цикла REPEAT, содержащее более одного оператора, не должно быть заключено между BEGIN и END.
   d. Тело цикла WHILE, содержащее более одного оператора, не должно быть заключено между BEGIN и END.
2. Напишите программу, печатающую нечетные (или четные) числа от 1 до 20.
3. Напишите программу, которая читает с клавиатуры 100 целых чисел и прекращает работу, если введен ноль или сто первое число, независимо от того, что произошло сначала.
4. Напишите программу, определяющую население города в конце каждого года с 1990 по 2000 год. Используйте следующие предположения:
   a. Начальная популяция в конце 1989 года = 5 миллионам
   b. Ежегодный процент смертности = 3%
   c. Ежегодный процент рождаемости = 7%

   Программа должна выводить данные на экран следующим образом:

```
 Год     Народонаселение
1990          5349999.97
1991          5724499.94
1992          6125214.90
1993          6553979.92
1994          7012758.48
1995          7503651.54
1996          8028907.12
1997          8590930.59
1998          9192295.70
1999          9835756.37
2000         10524259.29
```

Ответы

1. a. Ложно b. Истинно c. Истинно d. Ложно
2. Следующий фрагмент кода представляет алгоритм печати нечетных чисел от 1 до 20:

```
FOR I := 1 TO 20 DO
  IF (I MOD 2)<>0 THEN
    WRITELN(I);
```

   Для печати четных чисел замените оператор <> на оператор =.

3. Следующий фрагмент кода представляет основной алгоритм программы. Добавьте объявления и инициализацию переменных.

```
WHILE (K < 100) AND (DONE = FALSE) DO
  BEGIN
    READLN(n);
    K := K+1;
    IF N = 0 THEN
      DONE := TRUE
  END;
```

4. Это основной алгоритм вычисления народонаселения. Добавьте объявления и инициализацию переменных.

```
P := 5000000.0 ;
FOR I := 1990 TO 2000 DO
     BEGIN
          B:= 0.07 * P;
          D:= 0.03;
          P:= P + B - D;
          WRITELN(I:10, P:20:2);
     END
```

5

Архитектура данных

5–1. Порядковые типы данных

До сих пор рассказывалось о предопределенных в языке типах данных, которые называются простыми типами данных, в противоположность структурным типам данных. Каждая величина простого типа – это единственный элемент, в то время как величины структурных типов (такие как массивы) могут содержать их совокупность.

Простые типы делятся на две основные категории:

- Порядковый тип
- Вещественный тип

Порядковые типы включают INTEGER, CHAR и BOOLEAN. Порядковый тип характеризуется значениями, которые образуют последовательность дискретных элементов, каждый из которых имеет предшественника (за исключением первого элемента) и последователя (за исключением последнего элемента). Таким образом, целые числа образуют множество различных значений от –(MAXINT+1) до +MAXINT. Элементу 4, например, предшествует 3, а следует за ним 5. Тип CHAR состоит из множества символов, упорядоченных согласно их порядковым номерам. Тип BOOLEAN состоит из множества [TRUE, FALSE]. Значение FALSE имеет порядковый номер 0, а TRUE – 1.

С другой стороны, вещественные числа не дискретны. Например, между любыми двумя вещественными числами есть другое вещественное число.

Перечисления

Иногда для облегчения чтения программы полезно определить дни недели как целые. При этом надо либо каждому дню присвоить номер, либо объявить их как именованные константы:

```
CONST
  Monday = 0;
  Tuesday = 1;
  Wednesday = 2;
  Thursday = 3;
  Friday = 4;
  Saturday = 5;
  Sunday = 6;
```

После таких объявлений можно ссылаться на любой из дней по имени:

```
IF Today = Sunday THEN
  WRITELN('Извините, по воскресеньям мы закрыты..');
```

В этом операторе проверяется, не принимает ли целочисленная переменная Today значение Sunday, иными словами, не содержит ли Today значение 6. Применяя такие объявления, программист затратит много усилий, особенно когда констант много (например, названия месяцев).

Перечислимые типы позволяют сократить ту же работу. Взгляните на следующее объявление:

```
VAR
  Day :(Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday);
```

В этом объявлении идентификаторы представляют дни недели, перечисленные в упорядоченной последовательности и разделенные запятыми. Так, Monday имеет внутренний код 0, а Sunday – 6. Другие дни представлены номерами от 0 до 6 согласно их положению в перечислении. Однако нельзя читать или выводить эти значения напрямую, как это делается с простыми типами (с помощью операторов WRITELN и READLN). С перечислениями можно использовать любой из следующих операторов:

- Можно присвоить любой из элементов перечисления переменной Day, например:

  ```
  Day := Friday;
  ```

  но нельзя переменной Day присвоить явное значение, например Day := 1. Таким образом обеспечивается присвоение перечислимым данным только разрешенных значений.

- С помощью функции ORD можно получить и использовать значение, соответствующее элементу перечисления. Например:

  ```
  WRITELN(ORD(Monday));   дает значение 0
  WRITELN(ORD(Tuesday));  дает значение 1
  ```

- Можно также посредством функций PRED и SUCC получить предшественника и последователя определенного элемента:

  ```
  WRITELN(PRED(Friday));  дает значение 3
  WRITELN(SUCC(Monday));  дает значение 1
  ```

- Можно сравнивать значения перечислимого типа с помощью логических операторов (простых или сложных), например:

```
IF (Day = Saturday) OR (Day = Sunday) THEN
  WRITELN('Это выходной день.');
```

Повторим, что нельзя использовать явные значения при сравнении, например `IF Day = 2`. Это приведет к ошибке.

В следующей программе в цикле FOR перечисление `Month` используется для вывода на экран соответствующих целых значений от 0 до 11.

```
{ ------------------------------ Пример 5-1 ------------------------------ }
PROGRAM Enumeration1(OUTPUT);
VAR
  Month :(Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec);
BEGIN
  WRITELN;
  FOR Month := Jan TO Dec DO
    WRITE(ORD(Month),' ')
END.
```

Вывод этой программы:

```
0 1 2 3 4 5 6 7 8 9 10 11
```

Как видите, значения, соответствующие 12 месяцам, – это интервал от 0 до 11. Для того чтобы увидеть интервал от 1 до 12, как в календаре, можно вместо выражения `ORD(Month)` использовать выражение `ORD(Month)+1`.

Поддиапазоны

Поддиапазон – это другой тип данных, определяемый пользователем, который помогает исключить данные, находящиеся вне диапазона.

Например, для представления номеров месяцев вместо использования типа INTEGER можно объявить переменную `Month` как поддиапазон:

```
VAR
  Month : 1..12;
```

Таким образом, любое значение вне интервала от 1 до 12 будет считаться ошибкой либо во время компиляции, либо во время выполнения. Другими словами, после такого объявления нельзя написать в программе оператор, подобный следующему:

```
Month := 13; ---> запрещенный оператор
```

Кроме того, если пользователь в ответ на запрос введет значение вне диапазона, то программа выдаст соответствующее сообщение об ошибке, хотя для того чтобы заставить некоторые компиляторы реагировать на ошибки значений вне диапазона, придется установить некоторый ключ.

В этом примере для представления значений месяца выбран тип INTEGER. Он называется *базовым типом* поддиапазона. В качестве базового типа можно использовать порядковый тип. Например, выбрав в качестве базового тип CHAR, можно объявить поддиапазоном буквы в верхнем регистре:

```
VAR
  Uppercase : 'A'..'Z'
```

В этом случае допустимыми данными для поддиапазона `Uppercase` будут только буквы в верхнем регистре. Следующий пример демонстрирует использование поддиапазона для представления месяцев, которые затем группируются по сезонам в операторе CASE. Программа просит ввести номер месяца и выводит на экран сезон, к которому относится данный месяц.

```
{ ------------------------------ Пример 5-2 ------------------------------ }
PROGRAM Subrange1(INPUT,OUTPUT);
VAR
  MonthNumber :1..12;
BEGIN
  WRITE('Пожалуйста, введите номер месяца: ');
  READLN(MonthNumber);
  CASE MonthNumber OF
     12, 1, 2  :WRITELN('Это зима.');
     3, 4, 5   :WRITELN('Это весна.');
     6, 7, 8   :WRITELN('Это лето.');
     9, 10, 11 :WRITELN('Это осень.')
  END
END.
```

Далее приведены результаты двух тестов программы. Второй выдает ошибку времени выполнения, поскольку в качестве номера месяца было введено число 14.

Тест 1:

```
Пожалуйста, введите номер месяца:2
Это зима.
```

Тест 2:

```
Пожалуйста, введите номер месяца: 14
Runtime error 201 at 0000:00BE.[1]
```

Вообще говоря, поддиапазон может быть подмножеством любой ранее определенной последовательности (порядкового типа). Поэтому если вы уже определили в программе перечисление `Day`, можно задать поддиапазон следующим образом:

---

[1] Данное сообщение об ошибке зависит от версии компилятора. – *Примеч. науч. ред.*

```
VAR
  WorkingDay : Monday..Friday;
```

Это допустимо, поскольку слова «Monday» и «Friday» компилятор уже знает.

Есть несколько ограничений на применение перечислений и поддиапазонов:

- Первый элемент поддиапазона должен быть меньше последнего.
- Хотя поддиапазон может быть подмножеством перечисления, но перечисление не может использовать элементы другого перечисления.
- Элементы перечисления не могут выступать в качестве идентификаторов других переменных. Это то же самое, что повторное объявление идентификатора переменной в программе.

Задание 5-1

Напишите объявление, определяющее следующие поддиапазоны:

A. Буквы в верхнем регистре

B. Буквы в нижнем регистре

C. Десятичные цифры

Опишите ввод переменных, соответствующих каждому поддиапазону, и вывод их на экран, предваряя его подходящим сообщением, например:

```
Буква в нижнем регистре  : r
Буква в верхнем регистре : T
Цифра                    : 5
```

5-2. Секция TYPE

Перечисления и поддиапазоны обычно связаны с оператором TYPE, который применяется для объявления новых, определенных пользователем типов и переименования предопределенных типов. Оператор TYPE находится в секции TYPE раздела объявлений и имеет вид:

```
TYPE
    имя-типа = определение-типа;
```

где `имя-типа` – это идентификатор типа, а `определение-типа` – это предопределенный тип или определение нового типа.

Переименование типов

Можно переименовать любой тип, даже простой тип, такой как INTEGER, например:

```
TYPE
  Day = INTEGER;
```

В этом объявлении типу INTEGER присвоено новое имя (Day). Таким образом в секции VAR можно объявить переменные типа Day, например:

```
VAR
  Holiday, Yesterday, Tomorrow : Day;
```

Тип Day – в действительности это тип INTEGER, но с другим именем (псевдонимом). В программе можно использовать любое из двух имен (INTEGER и Day), поскольку компилятор по-прежнему распознает тип INTEGER. Тем не менее это неправильное использование оператора TYPE.[1] Он предназначен для именования таких типов, как перечисления и поддиапазоны.[2]

Именование типов, определенных пользователем

Вместо объявления перечислений и поддиапазонов в секции VAR лучше объявить их как типы. Взгляните на следующие объявления:

```
TYPE
  Day = (Monday,Tuesday,Wednesday,Thursday,Friday,Saturday,Sunday);
  WorkingDay = Monday..Friday;
```

Здесь объявлены два новых типа: перечислимый тип Day и поддиапазон WorkingDay. Обратите внимание, что поддиапазон объявлен как подмножество перечисления Day. Нет необходимости говорить, что в этом случае объявление перечисления должно быть первым. Новые типы можно использовать для объявления переменных в секции VAR таким же образом, как вы использовали предопределенные типы языка. Например:

```
VAR
  Today, Yesterday, Tomorrow, Holiday :Day;
  DayOff :WorkingDay;
```

Применение оператора TYPE предохранит вас от написания длинных объявлений для перечислимых переменных `Today`, `Yesterday`, `Tomorrow` и `Holiday`. Все они просто имеют тип Day.

Теперь можно написать операторы присваивания, подобные следующим:

---

[1] Сейчас TYPE – это идентификатор секции для объявления типов. – *Примеч. науч. ред.*

[2] Данное высказывание на сегодняшний день можно считать устаревшим. Современный стиль программирования допускает свободное переименование типов для наилучшего понимания и читаемости кода программы. – *Примеч. науч. ред.*

```
Holiday := Friday;
DayOff := Tuesday;
Tomorrow := Sunday;
```

Для отображения содержимого переменных применяются следующие операторы вывода:

```
WRITELN(ORD(Holiday), ', ',ORD(DayOff),', ', ORD(Tomorrow));
```

В этом случае оператор выдаст значения 4, 1 и 6 соответственно.

В стандартном Паскале секция TYPE должна располагаться относительно других секций следующим образом:

Секция LABEL

Секция CONST

Секция TYPE

Секция VAR

Задание 5–2

Какие из следующих объявлений допустимы, если все они находятся в одной программе?

```
TYPE
{1}Football = (Saints, Cowboys);
{2}Games    = (Football, Baseball, Basketball)
{3}Week     = (Mon, Tue, Wed, Thu, Fri, Sat, Sun);
{4}Weekend  = Sat..Sun;
{5}Compiler = (C, Pascal, Fortran, Ada, Basic);
VAR
{6}WholeWeek  :Week;
{7}WorkingDay :(Mon, Tue, Wed, Fri);
{8}Weekday    :Mon..Fri;
{9}SW         :(Compiler, OperatingSystem, ApplicationProgram);
{10}DpTools    :(Hardware, Software, PeopleWare);
{11}DpTool     :(HW, SW, PW);
{12}C          :(TurboC, QuickC);
{13}Margin     : -10..+10;
```

Как упоминалось ранее, в Турбо Паскале порядок не важен, но секция TYPE должна предшествовать секции VAR, поскольку содержит определения пользовательских типов.[1]

[1] Это верно в случае определения пользовательских типов. Дело в том, что компилятор языка Паскаль является однопроходным, в отличие от компилятора языка C. Это означает, что компилятор собирает информацию о типах и переменных за один раз, «сверху вниз». – *Примеч. науч. ред.*

5–3. Массивы как структуры данных

Для того чтобы представить, например, имена игроков футбольной команды при помощи простых типов данных, понадобится переменная для каждого имени игрока. В этом случае потребуется слишком много переменных:

```
FirstPlayer
SecondPlayer
ThirdPlayer
...
```

Это не очень хорошая идея. Теперь представьте, что вы имеете дело с классом, в котором 100 учеников. Практически невозможно использовать 100 переменных для хранения имен. Удобным способом хранения такого набора однородных данных является использование структуры данных под названием массив. В случае футбольной команды необходимо было бы объявить только одну *индексированную* переменную и представлять данные следующим образом:

```
Player[1]
Player[2]
Player[3]
...
```

`Player` – это имя переменной, а номер в скобках называется *индексом*. Изменение индекса приводит к новой области памяти, в которой хранится новое имя. Этот тип структуры данных называется *одномерным массивом*. Он удобен для представления таких данных, как имена группы людей, отметки учащихся за несколько лет или любых подобных множеств однородных данных (см. табл. 5-1).

Таблица 5-1. Пример одномерного массива

Player[1]	Player[2]	Player[3]	Player[4]	Player[5]	
Able	Baker	Charlie	John	Sam	...

В главе 2 мы имели дело со специальным типом одномерного массива (PACKED ARRAY OF CHAR), служащего для хранения текстовой строки в стандартном Паскале, и вы уже знаете, что каждому элементу (символу) в этом массиве соответствует номер (индекс).

В другом приложении, возможно, понадобится *двумерный массив*, который позволяет манипулировать более сложными структурами. Предположим, что потребовалось хранить результаты группы учащихся различных классов так, как показано в табл. 5-2.

Каждый элемент в этой таблице относится к строке (номер ученика) и к столбцу (номер класса); это два измерения массива. Сами элементы данных представляют собой вещественные числа.

Таблица 5-2. Пример двумерного массива

№ ученика (первый индекс)	№ класса (второй индекс)				
	1	2	3	4	5
1	55.5	60.9	66.5	80.3	70.5
2	89.1	77.6	99.9	88.7	50.3
3	40.5	67.4	90.5	45.1	66.9
...	...	...	...	...	...
100	68.8	87.2	90.4	60.1	60.4

Переменные, представляющие данные в этой таблице, могут выглядеть так:

```
StudentScore[3][4]
```

Эта переменная представляет результат ученика №3 из класса №4; другими словами, число на пересечении строки №3 и столбца №4. Этой переменной можно присвоить численное значение, представляющее результат:

```
StudentScore[3][4] := 45.1;
```

Сравните теперь следующие операторы присваивания значений из таблицы:

```
StudentScore[1][1] := 55.5; { результат ученика №1 из класса №1 }
StudentScore[1][2] := 60.9; { результат ученика №1 из класса №2 }
StudentScore[3][5] := 66.9; { результат ученика №3 из класса №5 }
StudentScore[100][2] := 87.2; { результат ученика №100 из класса №2 }
```

Массивы относятся к структурным типам данных (в противоположность простым или *неструктурным* типам, которые мы рассматривали до сих пор). В Паскале много других структурных типов данных, полезных для различных приложений.

Фактически качество программы определяется двумя критериями:

1. Структурная эффективность программы, то есть легкость чтения и отладки, а также подверженность ошибкам.
2. Использование наиболее эффективных структур данных, имеющее целью сберечь время и позволить программе манипулировать данными наиболее эффективным образом.

Примечание по терминологии: *Переменную-массив можно назвать индексированной переменной. Элементы массива, соответствующие переменной-массиву, называются также компонентами массива. В математике одномерный массив называется вектором, а двумерный массив называется матрицей. Эти названия могут встретиться вам в математических приложениях.*

5–4. Одномерные массивы

Одномерные массивы объявляются следующим образом:

```
VAR
  имя-массива : ARRAY[диапазон-индекса] OF тип-элемента;
```

Например, массив для хранения результатов теста 10 учеников в виде вещественных чисел можно объявить так:

```
VAR
  Score : ARRAY[1..10] OF REAL;
```

Этот массив (с именем Score) может хранить до 10 вещественных чисел. Диапазон индекса [1..10] означает, что индексы элементов массива изменяются от 1 до 10. Диапазон индекса, который является поддиапазоном (целых чисел в этом примере), может быть любого порядкового типа, а элементы массива могут быть любого типа данных. Таким образом, приведенное выше объявление резервирует последовательность из 10 ячеек памяти, в которых хранятся 10 значений типа REAL, представляющих 10 элементов массива.

Пример: Результаты ученика

В этой программе массив Score используется для хранения результатов учеников в шести различных классах. Результаты вводятся с клавиатуры, затем сумма и среднее значение результатов выводятся на экран:

```
{ ------------------------------ Пример 5-3 ------------------------------ }
PROGRAM Scores1(INPUT,OUTPUT);
CONST
  NumberOfClasses = 6;
VAR
  Score :ARRAY[1..NumberOfClasses] OF REAL;
  Average, SumOfScores :REAL;
  Index                :INTEGER;
BEGIN
{ Чтение массива результатов }
{ ------------------------- }
  FOR Index := 1 TO NumberOfClasses DO
    BEGIN
      WRITE('Введите результат в классе #', Index,': ');
      READLN(Score[Index])
    END;
{ Вычисление суммы }
{ ---------------- }
  SumOfScores := 0;
  FOR Index := 1 TO NumberOfClasses DO
    SumOfScores := SumOfScores + Score[Index];
{ Вычисление среднего значения }
```

```
{ --------------------------- }
  Average := SumOfScores / NumberOfClasses;
{ Вывод результатов }
{ ----------------- }
  WRITELN;
  WRITELN('Сумма результатов = ', SumOfScores:0:2);
  WRITELN('Средний результат = ', Average:0:2);
  WRITELN;
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Результат выполнения программы:

```
Введите результат в классе #1: 90
Введите результат в классе #2: 80
Введите результат в классе #3: 85
Введите результат в классе #4: 75
Введите результат в классе #5: 89
Введите результат в классе #6: 91
Сумма результатов = 510.00
Средний результат = 85.00
Для продолжения нажмите ENTER..
```

В этой программе следует обратить внимание на следующие моменты:

- Длина массива объявлена в виде константы (`NumberOfClasses`).
- Диапазон индекса массива объявлен при помощи определенной ранее константы `NumberOfClasses` следующим образом:

  ```
  Score :ARRAY[1..NumberOfClasses] OF REAL;
  ```

  Это то же самое, что:

  ```
  Score :ARRAY[1..6] OF REAL;
  ```

Однако первое объявление намного лучше, потому что при необходимости обработать другое количество классов можно просто изменить значение константы `NumberOfClasses`, не изменяя при этом основное тело программы.

- Обратите внимание, что после того как программа прочитала результаты, они записаны в элементы массива и теперь доступны в памяти. Это означает, что сумму в программе можно вычислить позже. Раньше при необходимости вычислить сумму и среднее значение некоторых чисел (пример 4-4) надо было накапливать значения в переменной `Sum` во время ввода данных. Теперь у вас шесть переменных.
- Индекс массива выступает в качестве счетчика цикла FOR и для чтения данных, и для вычисления суммы. Индекс массива очень удобен для вывода результатов, особенно если вы хотите вывести результаты в виде таблицы.

Вывод результатов в табличной форме

Следующая программа решает ту же задачу, но выводит результаты в виде таблицы:

```
{ ------------------------------ Пример 5-4 ------------------------------ }
PROGRAM Scores2(INPUT,OUTPUT);
CONST
  NumberOfClasses = 6;
  Tab = ' '; { 9 spaces }
VAR
  Score :ARRAY[1..NumberOfClasses] OF REAL;
  Average, SumOfScores :REAL;
  Index                :INTEGER;
BEGIN
{ Чтение массива результатов }
{ ------------------------- }
  FOR Index := 1 TO NumberOfClasses DO
    BEGIN
      WRITE('Введите результат в классе #', Index,': ');
      READLN(Score[Index])
    END;
{ Вычисление суммы }
{ ---------------- }
  SumOfScores := 0;
  FOR Index := 1 TO NumberOfClasses DO
    SumOfScores := SumOfScores + Score[Index];
{ Вычисление среднего значения }
{ ---------------------------- }
  Average := SumOfScores / NumberOfClasses;
{ Вывод результатов }
{ ----------------- }
  WRITELN;
  WRITELN(Tab,'КЛАСС #');
  WRITE(' '); { 6 spaces }
  FOR Index := 1 TO NumberOfClasses DO
    WRITE(Index:7);
  WRITELN;
  WRITE(Tab);
  FOR Index := 1 TO NumberOfClasses DO
    WRITE('-------');
  WRITELN;
  WRITE('SCORES ');
  FOR Index := 1 TO NumberOfClasses DO
    WRITE(Score[Index]:7:2);
  WRITELN;
  WRITE(Tab);
  FOR Index := 1 TO NumberOfClasses DO
    WRITE('-------');
  WRITELN;
  WRITELN(Tab,'Сумма результатов ', SumOfScores:0:2);
```

```
  WRITELN(Tab,'Средний результат = ', Average:0:2);
  WRITELN;
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Пример выполнения программы:

```
Введите результат в классе #1: 90.5
Введите результат в классе #2: 80.5
Введите результат в классе #3: 86.2
Введите результат в классе #4: 90.3
Введите результат в классе #5: 74.8
Введите результат в классе #6: 98.5

           КЛАСС #
              1      2      3      4      5      6
            ----------------------------------------
РЕЗУЛЬТАТ    90.50  80.50  86.20  90.30  74.80  98.50
            ----------------------------------------
Сумма результатов = 520.80
Средний результат = 86.80
Для продолжения нажмите ENTER..
```

В этой программе для вывода на экран прерывистых линий, номеров классов и результатов был интенсивно использован цикл, что делает программу более настраиваемой. Например, прерывистая линия могла бы быть выведена следующим оператором:

```
WRITELN(' ----------------------------------------');
```

Это годится только для шести классов, а следующие операторы:

```
WRITE(Tab);
FOR Index := 1 TO NumberOfClasses DO
  WRITE('-------');
```

подходят для любого количества классов, поскольку для каждого класса выводится сегмент из семи черточек. Таким образом, для четырех классов вывод был бы таким:

```
        КЛАСС #
           1      2      3      4
         ---------------------------
SCORES    80.00  90.00  85.00  75.00
         ---------------------------
Сумма результатов = 330.00
Средний результат = 82.50
```

Заметьте, что количество черточек равно ширине поля, определенной в формате вывода переменных `Score` и `Index`:

```
WRITE(Index:7);
WRITE(Score[Index]:7:2);
```

Обратите внимание также на использование константы Tab (содержащей девять пробелов) для соответствующего выравнивания вывода.

Слабое место программы в том, что приходится повторять те же строчки кода всякий раз, когда надо нарисовать линию. Но подобные задачи могут быть запрограммированы отдельными процедурами, вызываемыми при необходимости. Они обсуждаются в книге позже.

Задание 5-3

Напишите программу на Паскале, которая читает и запоминает результаты тестов пяти учеников, а затем выводит информацию на экран в следующем виде:

```
Ученик #                  Результат
---------------------------------
   1                          90.00
   2                          88.00
   3                          91.00
   4                          78.00
   5                          75.00
---------------------------------
Средний результат = 84.40
```

Объявление массивов в секции TYPE

Предпочтительнее сочетать объявления массивов с оператором TYPE, как в этом примере:

```
TYPE
  AnArray = ARRAY[1..6] OF INTEGER;
VAR
  MyArray :AnArray;
```

В этом случае в секции VAR можно объявить более одного массива типа AnArray.

```
VAR
  YourArray, MyArray :AnArray;
```

Также можно использовать заранее объявленный поддиапазон в качестве диапазона индекса массива, например:

```
TYPE
  MyRange = 1..6;
  AnArray = ARRAY[MyRange] OF INTEGER;
VAR
MyArray :AnArray;
```

Индексация нашего массива начинается с 1, но это не обязательно. Диапазон индекса массива может быть любым допустимым поддиапазо-

ном, но вы всегда должны помнить о недопустимости выхода за границы диапазона индекса.

Пример: Сортировка массива

Если вы хотите отсортировать некоторые числа (или имена), лучше всего сохранить их в массиве, а затем использовать алгоритм сортировки. Простой (но отнюдь не наиболее эффективный) способ сортировки чисел по возрастанию известен под названием *пузырьковой сортировки* (*bubble sort*). Алгоритм следующий:

1. Сравнить первый и второй элементы. Если первый элемент больше, поменять их местами.
2. Повторить сравнение первого элемента со всеми элементами массива. Если он больше другого элемента, поменять их местами.
3. После всех сравнений первым элементом будет наименьший элемент массива.
4. Повторить предыдущие шаги для второго элемента, третьего элемента и так далее до предпоследнего элемента.

После завершения процесса массив будет отсортирован по возрастанию. Этот алгоритм продемонстрирован в следующей программе. Для выполнения сравнений необходимы два вложенных цикла. Внешний цикл (индекс I) начинается с первого элемента (I=1) и заканчивается перед последним элементом (I=ArraySize–1). Внутренний цикл (индекс J) начинается на шаг позже внешнего цикла (J=I+1) и продолжается вплоть до последнего элемента (J=ArraySize).

```
{ ------------------------------ Пример 5-5 ------------------------------ }
PROGRAM Sorting(INPUT,OUTPUT);
CONST
  ArraySize = 6;
TYPE
  Range = 1..ArraySize;
  NumbersArray = ARRAY[Range] OF INTEGER;
VAR
  Numbers    :NumbersArray;
  I, J , Pot :INTEGER;
BEGIN
{ Чтение массива }
{ -------------- }
  FOR I := 1 TO ArraySize DO
    BEGIN
      WRITE('Введите элемент #', I,': ');
      READLN(Numbers[I])
    END;
{ Сортировка массива }
{ ------------------ }
  FOR I := 1 TO ArraySize-1 DO { внешний цикл }
```

```
    BEGIN { необязательный блок }
      FOR J := I+1 TO ArraySize DO { внутренний цикл }
        BEGIN
          IF Numbers[I] > Numbers[J] THEN
            BEGIN { перестановка }
              Pot := Numbers[J];
              Numbers[J] := Numbers[I];
              Numbers[I] := Pot
            END
        END { конец внутреннего цикла }
    END; { конец внешнего цикла }
{ Display Results }
{ --------------- }
  WRITELN;
  WRITELN('Отсортированный массив:');
  FOR I := 1 TO ArraySize DO
    WRITELN(Numbers[I]);
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Результат выполнения:

```
Введите элемент #1: 6
Введите элемент #2: 33
Введите элемент #3: 4
Введите элемент #4: 2
Введите элемент #5: 55
Введите элемент #6: 9
Отсортированный массив:
2
4
6
9
33
55
Для продолжения нажмите ENTER..
```

Перестановка содержимого двух элементов выполняется с помощью третьей переменной (`Pot`) с целью временного хранения значения одной из переменных:

```
Pot := Numbers[J];
Numbers[J] := Numbers[I];
Numbers[I] := Pot
```

Этот процесс напоминает замену содержимого двух чашек, в одной из которых кофе, а в другой – чай; все, что вам надо, – это пустая чашка (`Pot`).

Для сортировки массива по убыванию достаточно заменить способ обработки «больше чем» на «меньше чем».

При сортировке массивов разной длины необходимо вводить значения элементов во время выполнения программы. Для этого объявите максимальную длину массива, например 100, и истинную длину, которая должна быть меньше максимальной. Истинную длину можно вводить одним из следующих способов:

- Установите счетчик, который увеличивается при каждом вводе элемента с клавиатуры. После окончания ввода всего массива значение счетчика будет равно числу элементов.
- Вводите число элементов с клавиатуры и используйте его как границу цикла FOR, в котором вводятся элементы массива.

Задание 5-4

Измените программу из задания 5-3 так, чтобы она выводила на экран лучший результат и номер ученика с лучшим результатом в классе. Вывод должен иметь вид:

```
Ученик #           Результат
-------------------------
   1                70.00
   2                88.00
   3                67.00
   4                90.00
   5                86.00
-------------------------
Средний результат = 80.20
Лучший результат  = 90.00
Лучший в классе ученик #4
```

Для определения наибольшего числа из массива результатов можете использовать следующий алгоритм:

1. Сохранить результат первого ученика в переменной, например `BestScore`, а индекс этого ученика – в переменной `BestOfClass`.
2. Начиная со второго элемента массива результатов и до конца повторять следующее сравнение: если любое число больше `BestScore`, сохранить его в `BestScore`, а его индекс – в `BestOfClass`. По окончании цикла переменная `BestScore` будет содержать наивысший результат, а соответствующий номер ученика будет находиться в `BestOfClass`.

5-5. Двумерные массивы

Для объявления двумерного массива используйте форму:

```
VAR
  имя-массива: ARRAY[диапазон-индекса-1, диапазон-индекса-2]
    OF тип-элемента;
```

Можно также объявить его в секции TYPE:

```
TYPE
  имя-типа = ARRAY[диапазон-индекса -1, диапазон-индекса -2]
    OF тип-элемента;
```

Где диапазон-индекса-1 и диапазон-индекса-2 — это диапазоны первого и второго измерений.

Взгляните на объявление:

```
TYPE
  Score = ARRAY[1..100, 1..6] OF INTEGER;
```

Этот оператор объявляет массив Score, который хранит результаты 100 учеников за шесть различных классов; вообще говоря, он может хранить до 600 целых значений. Как видите, каждое измерение представлено поддиапазоном.

Можно также объявить многомерный массив, используя общую форму:

```
TYPE
  имя-типа = ARRAY[диапазон-индекса-1, диапазон-индекса-2,
    ..., диапазон-индекса-n] OF тип-элемента;
```

Однако в большинстве приложений нет необходимости в более чем двух измерениях.

Пример: Результаты учеников

Следующая программа читает результаты учеников в различных классах, представленных в табл. 5-2. Для простоты демонстрации будут рассмотрены только четыре ученика и три класса. Однако количество учеников и классов можно изменить, просто поменяв значения двух констант NumberOfClasses и NumberOfStudents.

```
{ ------------------------------ Пример 5-6 ------------------------------ }
PROGRAM Scores3(INPUT,OUTPUT);
{ использование двумерного массива }
CONST
  NumberOfClasses = 3; { Измените это число для большего количества классов}
  NumberOfStudents = 4; { Измените это число для большего количества учеников }
  Tab = ' '; { 7 пробелов }
  Dash = '-';
  NumberOfDashes = 23;
TYPE
  ScoreArray = ARRAY[1..NumberOfStudents, 1..NumberOfClasses] OF REAL;
  AverageArray = ARRAY[1..NumberOfStudents] OF REAL;
VAR
  Score                                 :ScoreArray;
  Average                               :AverageArray;
  SumOfScores                           :REAL;
```

```
  StudentCount, ScoreCount, DashCount :INTEGER;
BEGIN
{ Чтение массива результатов }
{ ------------------------- }
  FOR StudentCount := 1 TO NumberOfStudents DO
    BEGIN
      WRITELN;
      WRITELN('Результат ученика #', StudentCount,': ');
      FOR ScoreCount := 1 TO NumberOfClasses DO
        BEGIN
          WRITE('Введите результат в классе #', ScoreCount,': ');
          READLN(Score[StudentCount, ScoreCount])
        END;
    END;
{ Вычисление среднего результата для каждого ученика }
{ ------------------------------------------------ }
  FOR StudentCount := 1 TO NumberOfStudents DO
    BEGIN
      SumOfScores := 0; { Инициализация для каждого ученика }
      FOR ScoreCount := 1 TO NumberOfClasses DO
        SumOfScores := SumOfScores + Score[StudentCount, ScoreCount];
      Average[StudentCount] := SumOfScores/NumberOfClasses
    END;
{ Вывод результатов }
{ ---------------- }
  WRITELN;
  WRITELN(Tab, 'Ученик #', Tab, 'Средний результат');
  WRITE(Tab);
  FOR DashCount := 1 TO NumberOfDashes DO
    WRITE(Dash);
  WRITELN;
  FOR StudentCount := 1 TO NumberOfStudents DO
    WRITELN(Tab, StudentCount:3, Tab, Average[StudentCount]:12:2);
  WRITE(Tab);
  FOR DashCount := 1 TO NumberOfDashes DO
    WRITE(Dash);
  WRITELN;
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Результат выполнения программы:

```
Результат ученика #1:
Введите результат в классе #1: 90
Введите результат в классе #2: 89
Введите результат в классе #3: 93
Результат ученика #2:
Введите результат в классе #1: 80
Введите результат в классе #2: 70
Введите результат в классе #3: 60
```

```
Результат ученика #3:
Введите результат в классе #1: 77
Введите результат в классе #2: 78
Введите результат в классе #3: 90
Результат ученика #4:
Введите результат в классе #1: 91
Введите результат в классе #2: 94
Введите результат в классе #3: 95

Ученик #           Средний результат
-----------------------------------
   1                     90.67
   2                     70.00
   3                     81.67
   4                     93.33
-----------------------------------
Для продолжения нажмите ENTER..
```

Обратите внимание на следующие моменты в этой программе:

- В секции TYPE были объявлены два типа массивов, двумерный массив `ScoreArray` и одномерный массив `AverageArray`. Эти идентификаторы типов используются в секции VAR для объявления двух массивов – `Score` и `Average`. Первый массив хранит результаты четырех учеников за три класса, а второй – средние результаты для четырех учеников (которые, конечно, представляют всего четыре значения).
- Данные вводятся с помощью двух циклов, при этом индекс `StudentCount` представляет собой счетчик учеников во внешнем цикле, а `ScoreCount` – счетчик результатов во внутреннем цикле. Каждое введенное с клавиатуры значение присваивается общей переменной-массиву:

  ```
  Score[StudentCount, ScoreCount]
  ```

  Точное расположение элементов массива определяется двумя индексами – `StudentCount` и `ScoreCount`.
- Среднее значение вычисляется для каждого ученика и запоминается в переменной-массиве:

  ```
  Average[StudentCount]
  ```

  Индекс `StudentCount` обозначает, какой ученик какой результат имеет.
- Обратите внимание на инициализацию переменной `SumOfScores` перед вычислением среднего результата. Это очень важный шаг, потому что если этого не сделать, то средний результат предыдущего ученика останется в переменной и добавится к новому среднему результату.

Инициализация массивов

Если вы присвоили значение только одному элементу инициализируемого массива, не ждите, что остальные элементы будут равны нулю.[1] В

таких приложениях надо инициализировать весь массив с помощью цикла, например:

```
FOR I := 1 TO N DO
  MyArray[I] := 0;
```

Если массив двумерный, то нужен второй цикл:

```
FOR I := 1 TO N DO
  FOR J := 1 TO M DO
    MyArray[I,J] := 0;
```

В последнем примере мы присвоили значение каждому элементу массива, поэтому необходимость в инициализации отпала.

Задание 5–5

Измените предыдущую программу так, чтобы она выводила имена учеников в соответствии с их результатами по убыванию, как в этом примере:

```
Имя ученика      Средний результат
-----------------------------------
Porter, Thomas          84.00
Dalton, Jack            83.33
Dixon, Jane             83.33
Bobbin, Dale            66.67
-----------------------------------
```

Заключение

В этой главе вашему вниманию был представлен обзор простых и структурных типов данных.

1. Теперь вы знаете, что простые типы данных делятся на вещественные и порядковые типы. Что касается порядковых типов, то вы научились использовать типы, определенные пользователем, перечисления и поддиапазоны.
2. Вы научились применять оператор TYPE для объявления новых типов или переименования предопределенных типов. Его общий вид:

   ```
   TYPE
       имя-типа = определение-типа;
   ```

   В стандартном Паскале расположение секции TYPE относительно других секций в разделе объявлений следующее:

---

[1] Современные компиляторы могут инициировать переменные значениями по умолчанию, но признаком хорошего стиля программирования является инициализация переменных вручную. – *Примеч. науч. ред.*

Секция LABEL
Секция CONST
Секция TYPE
Секция VAR

3. Вы изучили массив как предопределенный структурный тип данных, который можно объявить или в секции TYPE, или в секции VAR. Вы также научились объявлять и использовать одномерные и двумерные массивы. Основная форма объявления массива с любым числом измерений (в секции TYPE):

```
TYPE
  имя-типа = ARRAY[диапазон-индекса-1, диапазон-индекса -2,
    ..., диапазон-индекса -n] OF тип-элемента;
```

Упражнения

1. Определите, какие из следующих объявлений в секции TYPE не верны:

```
TYPE
{a}   Range1 = 1..100 DIV 2;
{b}   Range2 = 1..7;
{c}   Range3 = 1.0..7.0;
{d}   Range4 = 1..(100/2);
{e}   TVSystem = (SECAM, PAL, NTSC);
{f}   ProgrammingLanguage = (C, C++, CSharp, Pascal, Fortran, COBOL);
{g}   int = INTEGER;
{h}   float = REAL;
```

2. Даны следующие объявления типов:

```
TYPE
    Range = 1..7;
    Color = (Red, Green, Blue);
    int = INTEGER;
```

Определите, какие из следующих объявлений в секции VAR не верны:

```
VAR
{a}   MyArray   :ARRAY[Range];
{b}   HisArray  :ARRAY[1..Range] OF INTEGER;
{c}   YourArray :ARRAY[Color] OF INT;
{d}   HerArray  :ARRAY[1..Color] OF INTEGER;
{e}   Score     :ARRAY[1..5, Range] OF INTEGER;
{f}   Colors    :(Red, Green, Blue);
{g}   ColorComp :(Cyan, Magenta, Yellow);
{h}   Subrange1 :1..12 DIV 2;
```

3. Даны следующие объявления:

```
TYPE
    ColorComp = (Cyan, Magenta, Yellow);
VAR
    MyColor, YourColor :ColorComp;
```

Определите, какие из следующих инструкций не верны:

```
{a}   MyColor := Yellow;
{b}   YourColor := Cyan + 1;
{c}   WRITELN(ORD(Cyan), ' ', ORD(MyColor));
{d}   WRITELN(Magenta);
```

4. Что выводит каждый оператор WRITELN в следующем коде?

```
TYPE
    FamilyMember = (Sam, Camelia, Hazem, Craig, Sally);
BEGIN
{a}   WRITELN(Sam < Sally);
{b}   WRITELN(Sam > Sally);
{c}   WRITELN(Camelia > Craig);
{d}   WRITELN(ORD(Craig)-1 = ORD(Hazem));
{e}   WRITELN(ORD(Sam)+4 = ORD(Sally));
END.
```

5. Почему вызовет ошибку добавление в предыдущий код оператора:

```
Friend = (John, Craig);
```

Ответы

1. Неверные объявления: c, d и f.
2. Неверные объявления: a, b, d и f.
3. Неверные инструкции: b и d.
4. a. TRUE b. FALSE c. FALSE d. TRUE e. TRUE
5. Это повторное объявление идентификатора `Craig`.

6

Обработка текста

6–1. Работа с текстовыми данными

До сих пор мы имели дело в основном с числовыми значениями данных. В этой главе рассматривается использование символов и строк при работе с текстовыми данными, при этом особое внимание уделяется вводу с клавиатуры и выводу на экран символов и строк. Эти устройства трактуются как файлы и носят названия: стандартный файл INPUT (клавиатура) и стандартный файл OUTPUT (экран).

6–2. Советы по применению оператора OUTPUT

Для того чтобы вывести на экран много строк текста или числовые результаты в отдельных строчках, можно для каждой строки написать свой оператор WRITELN. Другой способ (требующий меньше усилий) состоит в использовании управляющих ASCII-кодов – 13 (возврат каретки) и 10 (перевод строки), – если нужна новая строка. Тогда, чтобы напечатать все результаты, достаточно одного оператора WRITE или WRITELN. В большинстве микрокомпьютеров пара «возврат каретки/перевод строки» считается признаком *конца строки*. В следующем примере управляющий символ CHR(10) объявляется как именованная константа LF (общепринятая аббревиатура перевода строки, linefeed), а управляющий символ CHR(13) как CR (общепринятая аббревиатура возврата строки, carriage return). Комбинация двух символов, CR и LF, производит тот же эффект, что и нажатие клавиши <Enter>.

```
{ ------------------------------ Пример 6-1 ------------------------------ }
PROGRAM Display1(INPUT,OUTPUT);
CONST
   LF = CHR(10);
   CR = CHR(13);
VAR
```

```
   X, Y, Z :INTEGER;
BEGIN
   WRITE('Введите три целых числа: ');
   READLN(X, Y, Z);
   WRITELN('X=', X, CR, LF, 'Y=', Y, CR, LF, 'Z=', Z)
END.
```

Результат выполнения примера:

```
Введите три целых числа: 11 22 33
X=11
Y=22
Z=33
```

Если в программе используется только LF, то результат будет таким:

```
X=11
    Y=22
        Z=33
```

А если только CR, то последний результат затирает два предыдущие. На выходе будет только один:

```
Z=33
```

6–3. Советы по применению оператора INPUT

В процессе последовательного чтения с помощью операторов READ или READLN можно попасть в некоторые ловушки, особенно при вводе символов. Поэтому важно понимать, как работают операторы ввода с различными типами данных.

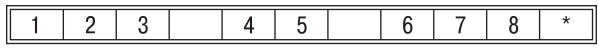
В процессе выполнения операторов READ или READLN значения заносятся в стандартный файл INPUT (клавиатура). Затем сохраненные значения читаются из файла и присваиваются переменным из списка ввода. При каждом нажатии клавиши <Enter> в файл INPUT записывается метка конца строки.

Ввод чисел при помощи оператора READLN

Предположим, что ввод содержит следующие числа:

```
123 45 678 <Enter>
```

Представьте, что числа записаны в файл INPUT, как изображено на следующем рисунке:



Метка конца строки показана в последней позиции и обозначена звездочкой (*). В первой позиции изображена маленькая стрелка (называемая *файловым указателем*), указывающая на начало файла. Будем считать, что эти значения читаются следующим оператором:

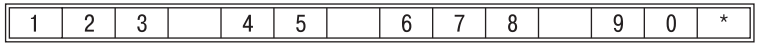
```
READLN(X, Y, Z);
```

После того как целое число (123) прочитано и присвоено переменной X, указатель передвигается к началу следующего численного значения (45). Затем второе значение читается и присваивается переменной Y, а указатель перемещается к третьему числу. Когда третье значение прочитано и присвоено Z, все переменные получают свои значения, и указатель передвигается за метку конца строки, после чего работа оператора READLN завершается. Если числовые значения будут разделены более чем одним пробелом, то лишние пробелы будут проигнорированы и результат останется верным.

Предположим теперь, что вы по ошибке ввели четыре числа:

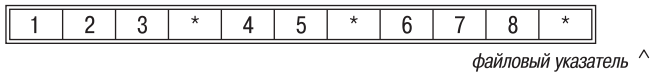
```
123 45 678 90 <Enter>
```

Программа игнорирует значение 90, поскольку после чтения третьего числа указатель переместится за метку конца строки, для того чтобы быть готовым к следующему чтению.



Эту возможность оператор READLN унаследовал с тех дней, когда данные вводили с перфокарт (каждая карта представляла строку данных). READLN читал определенное количество позиций и переходил к следующей карте.

Можно после ввода каждого числового значения нажимать клавишу <Enter>; в этом случае после каждого значения появится метка конца строки:



До тех пор пока всем трем переменным не присвоены значения, метка конца строки между значениями считается пробелом и поэтому игнорируется. Указатель перемещается от одной метки конца строки до другой, пока все три числа не будут прочитаны, затем указатель перемещается за следующую метку конца строки, завершая оператор READLN. Проверьте следующую программу (в ней два оператора READLN), используя значения, показанные в результатах теста:

```
{ ------------------------------ Пример 6-2 ------------------------------ }
PROGRAM ReadLnNumbers(INPUT,OUTPUT);
```

```
CONST
  CR = CHR(13);
  LF = CHR(10);
VAR
  A, C , D, E :INTEGER;
  B :REAL;
    BEGIN
      WRITE('Введите A, B, C: ');
{ Если вы введете более трех значений, прочитаны будут только три }
      READLN(A, B, C);
{ Теперь следующий READLN начнет чтение значений после метки конца
 строки, игнорируя все оставшиеся значения от предыдущего чтения }
      WRITE('Enter D, E: ');
      READLN(D, E);
      WRITELN('A=',A,', B=',B:0:2,', C=', C, CR, LF,
              'D=', D,', E= ',E)
END.
```

Результат выполнения теста:

```
Введите A, B, C: 1 2 3 4 5 6    ----> Ввод этих значений
Введите D, E: 7 8               ----> Ввод этих значений
A=1, B=2.00, C=3                ----> Ответ программы
D=7, E= 8
```

Обратите внимание, что лишние значения (4, 5, 6) в первой строке ввода были полностью игнорированы, а второе чтение началось со значения 7, которое следует за меткой конца строки.

Задание 6–1

Выполните приведенную выше программу со следующими входными значениями и изучите результаты:

```
1 2      <Enter>
3 4 5 6  <Enter>
7 8      <Enter>
```

Ввод чисел при помощи оператора READ

Процедура чтения с помощью оператора READ отличается, так как после окончания работы оператора READ файловый указатель не перемещается за метку конца строки, и поэтому следующий READ начнет чтение с того места, где остановился предыдущий READ. Замените оператор READLN на оператор READ и попробуйте следующий ввод:

```
1 2 3 4 5 6 7 <Enter>
```

После нажатия клавиши <Enter> программа не остановится на втором операторе ввода, поскольку файл INPUT содержит достаточное коли-

чество значений для переменных. В этом случае программа выведет на экран следующие результаты:

```
A=1, B=2.00, C=3
D=4, E= 5
```

Задание 6-2

Проверьте программу с оператором READ, используя следующий ввод:

1. `1 2 <Enter>`
   `3 4 5 6 7 <Enter>`
2. `1 2 3 4 <Enter>`
   `5 6 7 <Enter>`

Ввод символов при помощи оператора READ

При вводе символов операторы ввода работают иначе. Оператор READ последовательно читает символы из файла клавиатуры, включая метку конца строки (которая в действительности представлена двумя символами CR и LF), и присваивает каждый символ следующей переменной из списка ввода. Рассмотрим следующий оператор ввода:

```
READ(C1, C2, C3, C4);
```

где C1, C2, C3 и C4 – это переменные типа CHAR.

Если ввести четыре символа:

```
ABCD
```

то все они будут прочитаны и присвоены переменным, поэтому:

C1 содержит 'A'
C2 содержит 'B'
C3 содержит 'C'
C4 содержит 'D'

Теперь рассмотрим такой вариант ввода:

```
A B C D
```

Первые четыре символа (включая пробелы) будут присвоены четырем переменным, а остальные символы будут проигнорированы. В результате:

C1 содержит 'A'
C2 содержит ' ' (пробел)
C3 содержит 'B'
C4 содержит ' ' (пробел)

Выполните следующую программу с тестовыми значениями и посмотрите, что получилось. Обратите внимание, что программа выводит и содержимое переменных, и соответствующие ASCII-коды, которые помогут вам увидеть непечатные символы, такие как пробел, перевод строки и возврат каретки.

```
{ ------------------------------ Пример 6-3 ------------------------------ }
PROGRAM CharRead1(INPUT,OUTPUT);
CONST
  LF = CHR(10);
  CR = CHR(13);
VAR
  C1, C2, C3, C4 :CHAR;
BEGIN
  WRITE('Введите четыре символа: ');
  READ(C1, C2, C3, C4);
  WRITELN('Введенные вами значения присвоены переменным:', CR, LF,
          'C1= ', C1, CR, LF,
          'C2= ', C2, CR, LF,
          'C3= ', C3, CR, LF,
          'C4= ', C4);
  WRITELN('Соответствующие ASCII-коды:', CR, LF,
          ORD(C1),' ', ORD(C2),' ', ORD(C3),' ',ORD(C4))
END.
```

Результаты выполнения тестов:

Тест 1:

```
Введите четыре символа: A BCD
Введенные вами значения присвоены переменным:
C1= A
C2=     { пробел}
C3= B
C4= C
Соответствующие коды ASCII:
65 32 66 67
```

Второй переменной был присвоен ASCII-код 32, представляющий символ пробела.

Тест 2:

```
Введите четыре символа: ABCD
Введенные вами значения присвоены переменным:
C1= A
C2= B
C3= C
C4= D
Соответствующие ASCII-коды:
65 66 67 68
```

Во втором случае были прочитаны первые четыре символа, а последний проигнорирован. Если бы в программе был еще один оператор READ,

то он бы начал чтение с буквы «E». Метка конца строки рассматривается как нечисловой символ. Например, если запустить программу с таким вводом:

```
AB <Enter>
CD <Enter>
```

то она остановится после ввода двух символов, и на выходе получится следующий результат (см. ниже):

Тест 3:

```
C1= A
C2= B
C3=     { CR }
C4=     { LF }
Соответствующие ASCII-коды:
65 66 13 10
```

Третий и четвертый символы содержат CR и LF соответственно, поэтому при нажатии клавиши <Enter> в файл INPUT отправляются два символа – CR и LF. Обратите внимание, что CR появляется в виде пробела (он возвращает курсор в начало строки), в то время как LF перемещает курсор на новую строку.

То же самое произойдет, если вывести данные с помощью двух отдельных операторов READ:

```
READ(C1, C2);
READ(C3, C4);
```

Запустив программу, вы теперь заметите, что если напечатать два символа и нажать <Enter>, то программа остановится с тем же результатом, что и в тесте 3. Точно так же, если ввести символов больше, чем нужно, то будут прочитаны только четыре первых символа.

Ввод символов при помощи оператора READLN

Для того чтобы вводить символы таким образом:

```
AB <Enter>
CD <Enter>
```

необходимо избавиться от лишних символов, остающихся в файле (CR и LF) в случае применения оператора READLN.

В следующей программе два оператора READLN, поэтому можно ввести два символа (или более) с завершающим нажатием <Enter> и начать следующее чтение с чистым буфером.

```
{ ------------------------------ Пример 6-4 ------------------------------ }
PROGRAM CharReadln3(INPUT,OUTPUT);
CONST
  LF = CHR(10);
```

```
  CR = CHR(13);
VAR
  C1, C2, C3, C4 :CHAR;
BEGIN
  WRITE('Введите два символа: ');
  READLN(C1, C2);
  WRITE(' Введите два символа: ');
  READLN(C3, C4);
  WRITELN('Введенные вами значения присвоены переменным:', CR, LF,
          'C1= ', C1, CR, LF,
          'C2= ', C2, CR, LF,
          'C3= ', C3, CR, LF,
          'C4= ', C4);
  WRITELN('Соответствующие ASCII-коды:', CR, LF,
           ORD(C1),' ', ORD(C2),' ', ORD(C3),' ',ORD(C4))
END.
```

Результат выполнения примера:

```
Введите два символа: abcd <Enter>
Введите два символа: efgh <Enter>
Введенные вами значения присвоены переменным:
C1= a
C2= b
C3= e
C4= f
Соответствующие ASCII-коды:
97 98 101 102
```

Ввод разнородных типов

Разрешается использовать один оператор READ (или READLN) для совместного ввода числовых и символьных данных, но это требует дополнительного внимания. Лучше использовать отдельные операторы READLN для каждого типа, как показано в следующей программе. Такой способ не так чреват ошибками ввода данных.

```
{ ------------------------------ Пример 6-5 ------------------------------ }
PROGRAM CharNumRead(INPUT,OUTPUT);
CONST
  LF = CHR(10);
  CR = CHR(13);
VAR
  A, B :CHAR;
  X, Y :INTEGER;
BEGIN
  WRITE('Введите два символа: ');
  READLN(A, B);
  WRITE('Введите два целых числа: ');
  READLN(X, Y);
  WRITELN('Введенные вами значения присвоены переменным:', CR, LF,
          'A= ', A, CR, LF,
```

```
          'B= ', B, CR, LF,
          'X= ', X, CR, LF,
          'Y= ', Y)
END.
```

Тест 1:

```
Введите два символа: ABCD
Введите два целых числа: 3 4
Введенные вами значения присвоены переменным:
A= A
B= B
X= 3
Y= 4
```

Как видите, лишние символы («C» и «D») были пропущены первым оператором READLN. Тем не менее, запомните, что правила ввода символов остаются в силе; другими словами, если нажать клавишу <Enter> после ввода первой буквы, то переменной B будет присвоен код символа CR. Это результат выполнения теста (см. ниже):

Тест 2:

```
Введите два символа: A       <Enter>
B                            <Enter>
Введите два целых числа: 5 6
Введенные вами значения присвоены переменным:
A= A
B=       { B присвоен CR }
X= 5
Y= 6
```

Пример: Перестановка букв

Следующий пример дает хорошую практику по работе с символами и построению циклов. Программа просит ввести четыре символа, затем выводит на экран все возможные комбинации этих символов. Если программировать на Бэйсике, то для получения того же результата понадобится много операторов GOTO. Программа на Паскале устроена лучше.

```
{ ------------------------------ Пример 6-6 ------------------------------ }
PROGRAM Scrambling(INPUT,OUTPUT);
TYPE
  ScrambleArray = Array[1..4] OF CHAR;
VAR
  A              :ScrambleArray;
  I1, I2, I3, I4 :INTEGER;
BEGIN
  WRITE('Введите четыре символа: ');
  READ(A[1], A[2], A[3], A[4]);
  FOR I1 := 1 TO 4 DO
    BEGIN
      FOR I2 := 1 TO 4 DO
```

```
        BEGIN
          IF I2 <> I1 THEN
            FOR I3 := 1 TO 4 DO
              BEGIN
                IF I3 <> I1 THEN
                  IF I3 <> I2 THEN
                    BEGIN
                      I4 := 10 - (I1 + I2 + I3);
                      WRITELN(A[I1],' ',A[I2],' ',
                              A[I3],' ',A[I4]);
                    END { Конец IF }
              END { Конец цикла I3 }
        END { Конец цикла I2 }
    END { Конец цикла I1 }
END.
```

Результат выполнения примера:

```
Введите четыре символа: ABCD
A B C D
A B D C
A C B D
A C D B
A D B C
A D C B
B A C D
B A D C
B C A D
B C D A
B D A C
B D C A
C A B D
C A D B
C B A D
C B D A
C D A B
C D B A
D A B C
D A C B
D B A C
D B C A
D C A B
D C B A
```

Для хранения четырех символов отводится массив A из четырех элементов (типа CHAR), а с помощью трех вложенных циклов строятся различные комбинации элементов. Алгоритм основан на выборе четырех различных индексов, соответствующих четырем разным элементам массива.

Обратите внимание, что все блоки BEGIN-END (за исключением самого внутреннего) необязательны и используются только для ясности.

6–4. Чтение текстовой строки: EOLN

EOLN это логическая функция, используемая для обнаружения конца строки во время чтения из файла INPUT. Функция принимает значение FALSE, пока не будет обнаружена метка конца строки, затем принимает значение TRUE.

Эта функция полезна, когда не известно количество ожидаемых символов. Для того чтобы прочитать текстовую строку до метки конца строки (но не включая ее), можно использовать такой цикл:

```
WHILE NOT EOLN DO
  BEGIN
    READ(Ch);
    ...
  END;
```

Оператор READ будет продолжать чтение символов до обнаружения метки конца строки, завершая таким образом цикл WHILE. Однако заметьте, что метка конца строки остается в буфере и может быть прочитана следующим оператором READ, поэтому до следующего чтения необходимо очистить буфер с помощью оператора READLN.

Пример: Счетчик символов

Следующая программа читает текстовую строку с клавиатуры и выводит на экран количество символов в строке. Программа будет читать вводимые пользователем символы до тех пор, пока не будет нажата клавиша <Enter>, после чего выведет результат на экран:

```
{ ------------------------------ Пример 6-7 ----------------------------- }
PROGRAM CharCounter1(INPUT,OUTPUT);
VAR
  Ch      :CHAR;
  Counter :INTEGER;
BEGIN
  Counter := 0;
  WHILE NOT EOLN DO
    BEGIN
      READ(Ch);
      Counter := Counter + 1
    END;
  WRITELN;
  WRITELN('Число символов = ', Counter)
END.
```

Задание 6–3

Измените программу так, чтобы она считала только буквы.

6–5. Чтение текстового файла: EOF

Другая логическая функция, EOF, применяется для определения метки *конца файла*. Эта функция принимает значение FALSE, пока не будет обнаружена метка конца файла, затем принимает значение TRUE. При вводе с клавиатуры метка конца файла образуется при нажатии клавиш <Ctrl>+<Z> (ASCII 26). Эта функция удобна для чтения нескольких строк текста (файла). Для чтения и анализа нескольких строк текста можно использовать EOF совместно с EOLN, например:

```
WHILE NOT EOF DO
  BEGIN
    WHILE NOT EOLN DO
      BEGIN
        READ(Ch);
        ...             { Обработка данных }
      END;              { Конец строки }
    READLN              { Перемещение указателя }
  END;                  { Конец файла }
```

В этом коде файл читается строка за строкой. После прочтения всей строки функция EOLN принимает значение TRUE, и больше из этой строки не читается ни один символ. Затем оператор READLN перемещает курсор в начало следующей строки. Программа останавливается, когда обнаружена метка конца строки и внешний цикл завершен. Взгляните на следующий пример.

Пример: Счетчик частоты повторений

Следующая программа просит ввести букву. Затем она начинает читать символы, вводимые с клавиатуры. После нажатия <Ctrl>+<Z> программа заканчивает работу и выводит на экран количество повторений в файле определенной буквы.

```
{ ------------------------------ Пример 6-8 ------------------------------ }
PROGRAM FreqCounter1(INPUT,OUTPUT);
VAR
  Ch, SpecificChar     :CHAR;
  Counter, FreqCounter :INTEGER;
BEGIN
  Counter := 0;
  FreqCounter := 0;
  WRITE('Введите желаемую букву: ');
  READLN(SpecificChar);
  WRITELN('Начало печати. Для окончания нажмите Ctrl-Z.');
  WHILE NOT EOF DO
    BEGIN
      WHILE NOT EOLN DO
        BEGIN
          READ(Ch);
```

```
            IF (Ch >= 'A') AND (Ch <= 'Z') OR
               (Ch >= 'a') AND (Ch <= 'z') THEN
                 Counter := Counter + 1;
            IF Ch = SpecificChar THEN
                 Freqcounter := FreqCounter + 1;
          END;
        READLN
      END;
    WRITELN('Общее количество букв= ', Counter);
    WRITELN('Буква ''', SpecificChar, ''' встретилась ',
            FreqCounter, ' раз(а)');
    WRITELN('Частота повторения= ', freqCounter/Counter*100:2:2,'%')
END.
```

Определенная буква присваивается переменной `SpecificChar` и сравнивается с вводимой буквой (`Ch`). Если результат сравнения равен TRUE, то переменная `FreqCounter` увеличивается на единицу. Общее количество букв накапливается в переменной `Counter`. Частота повторений буквы определяется делением `FreqCounter` на `Counter` и умножением результата на 100.

Выполнение теста:

```
Введите желаемую букву: a
Начало печати. Для окончания нажмите Ctrl-Z.
Это тест для определения частоты повторения
буквы «a» в клавиатурном файле.
^Z
бщее количество букв= 67
Буква 'a' встретилась 4 раз(а)
Частота повторения= 5.97%
```

6–6. Обработка строк

В главе 2 вы научились объявлять, читать и выводить на экран переменные типа STRING, который был введен в современных реализациях (таких как Турбо Паскаль, UCSD и Макинтош). Вы также научились использовать функцию LENGTH для подсчета количества символов в строке. В этом разделе рассматриваются дополнительные строковые возможности, помогающие в обработке текста.

Советы по вводу/выводу строк

И при вводе и при выводе можно трактовать строковую переменную или как один элемент, или как массив, элементы-символы которого составляют строку. Взгляните на пример программы, которая читает строковую переменную и выводит ее на экран символ за символом, располагая каждый символ на отдельной строке (с помощью символа LF).

```
{ ------------------------------ Пример 6-9 ------------------------------ }
PROGRAM String1(INPUT,OUTPUT);
```

```
CONST
  LF = CHR(10);
VAR
  Name :STRING[30];
  I    :INTEGER;
BEGIN
  WRITE('Пожалуйста, введите имя: ');
  READLN(Name);
  FOR I := 1 TO LENGTH(Name) DO
    WRITE(Name[I],LF)
END.
```

Выполнение теста:

```
Пожалуйста, введите имя: PASCAL
P
 A
  S
   C
    A
     L
```

Пример: Сортировка имен

Можно построить массив типа STRING для хранения родственных элементов, таких как имена или адреса. Таким образом можно отсортировать имена в алфавитном порядке, по тому же алгоритму, который ранее использовался для сортировки чисел. Каждые две строки сравниваются символ за символом. Поэтому следующие выражения принимают значение TRUE:

```
'Able' < 'Baker'
'Baker' < 'Charlie'
'Charley' < 'Charlie'
```

Все буквы верхнего регистра по значению больше, чем буквы нижнего регистра. Также при сравнении учитываются лидирующие и завершающие пробелы. ASCII-код пробела (32) меньше, чем код любой буквы или цифры. В следующей программе массив из четырех имен читается, сортируется и выводится на экран.

```
{ ------------------------------ Пример 6-10 ----------------------------- }
PROGRAM SortStrings(INPUT,OUTPUT);
CONST
  Tab = ' ';
  NumOfElements = 4;
TYPE
  StringArray = ARRAY[1..NumOfElements] OF STRING[30];
VAR
  Name :StringArray;
  I, J :INTEGER;
  Temp :STRING[30];
```

```
BEGIN
{ Чтение элементов массива }
{ ----------------------- }
  FOR I := 1 TO NumOfElements DO
    BEGIN
      WRITE(Пожалуйста, введите имя #', I, ': ');
      READLN(Name[I])
    END;
{ Сортировка имен }
{ --------------- }
  FOR I := 1 TO NumOfElements-1 DO
    FOR J := I+1 TO NumOfElements DO
      IF Name[I] > Name[J] THEN
        BEGIN
          Temp := Name[I];
          Name[I] := Name[J];
          Name[J] := Temp
        END;
{ Конец внутреннего и внешнего циклов }
{ Вывод на экран отсортированных имен }
{ ---------------------------------- }
  WRITELN('Порядковый # Имя');
  WRITELN('----------------------------');
  FOR I := 1 TO NumOfElements DO
    WRITELN(I:2, Tab, Name[I])
END.
```

Пример выполнения:

```
Пожалуйста, введите имя #1: Rigby, Peter
Пожалуйста, введите имя #2: Berlin, Amy
Пожалуйста, введите имя #3: Sanders, Dale
Пожалуйста, введите имя #4: Brady, Clark

Порядковый #     Имя
-----------------------------------
 1                Berlin, Amy
 2                Brady, Clark
 3                Rigby, Peter
 4                Sanders, Dale
```

Задание 6–4

Напишите программу, которая меняет местами три строки. Это пример вывода для строк «WHO», «ARE» и «YOU»:

```
WHO ARE YOU
WHO YOU ARE
ARE WHO YOU
ARE YOU WHO
YOU WHO ARE
YOU ARE WHO
```

6–7. Строковые функции и процедуры

При работе с текстовыми редакторами иногда возникает необходимость вырезать, склеить, удалить или вставить в определенное место часть текста. Инструменты, делающие эти операции возможными, включены в современные реализации Паскаля для помощи программисту в обработке строк. Некоторые из них называются *функциями*, потому что они возвращают значение, замещающее вызов функции (например, LENGTH). Другие называются *процедурами*, поскольку они выполняют определенные операции, которые не обязательно возвращают значение (например, WRITELN). Они показаны в табл. 6-1.

В дополнение к этим инструментам можно найти множество функций, процедур или операторов в специальных реализациях, но здесь мы касаемся только наиболее используемых инструментов, которые практически стандартизованы.

Таблица 6-1. Строковые функции и процедуры

Вид	Назначение
Функции:	
LENGTH(str)	Возвращает количество символов в строке str
CONCAT(str1, str2,...)	Возвращает строку, образованную объединением str1, str2,...
COPY(str, pos, len)	Возвращает подстроку строки str, начиная с позиции pos, длиной len
POS(str1, str2)	Возвращает позицию первого вхождения первого символа строки str1 в строке str2. Если str1 не встречается в str2, возвращает ноль
Процедуры:	
INSERT(str1, str2, pos)	Вставляет строку str1 в строку str2в позиции pos
DELETE(str, pos, len)	Удаляет из строки str подстроку, начиная с позиции pos, длиной len

LENGTH

Можно определить динамическую[1] длину строки с помощью функции LENGTH.

Например, длину строки Name из последней программы определить так:

```
LENGTH(Name)
```

[1] На самом деле при любой манипуляции со строкой старое значение строки не меняется. Вместо этого программа заводит новую переменную строкового типа, записывает туда новое значение, а старую строку удаляет. Это связано с особенностями распределения памяти для строковых переменных и вопросами оптимизации скорости работы. – *Примеч. науч. ред.*

Выведя на экран значение этого выражения, получите точное количество символов, содержащихся в строковой переменной, включая пробелы. Если строковая переменная пуста, то динамическая длина равна нулю. В примере 6-1 пользователь вводит имя, а программа выводит на экран истинную длину переменной до и после присваивания.

CONCAT

В качестве примера использования функции CONCAT можно объединить три строки, «John», «M.» и «Smith», и присвоить результат строковой переменной Name:

```
Name := CONCAT('John ', 'M.', ' Smith');
```

Теперь переменная Name содержит полное имя: «John M. Smith». В Турбо Паскале строки можно также объединять при помощи оператора +:

```
Name := 'John ' + 'M.' + ' Smith';
```

Переменная Name получает то же значение, что и в первом варианте.

COPY

С помощью функции COPY можно извлечь подстроку из строки Name. Следующий оператор извлекает имя из строки Name и присваивает его переменной FirstName:

```
FirstName := COPY(Name, 1, 4);
```

Как видите, в число параметров необходимо включить начальную позицию извлекаемой подстроки (в данном случае 1) и длину подстроки (в данном случае 4).

POS

Функция POS возвращает целое значение, показывающее позицию первого вхождения подстроки в строку. Например, оператор:

```
Str1 := 'This is a test';
WRITELN(POS('is', Str1));
```

выводит в результате число 3, а это позиция буквы «i» в «This».

DELETE

Для удаления подстроки «Smith» из строки имени предназначена процедура DELETE:

```
DELETE(Name, 9, 5);
```

Обратите внимание, что строка «Smith» начинается в девятой позиции и содержит пять символов.

Применение процедуры изменяет изначальную переменную Name, а применение функции нет. Теперь переменная будет содержать значение «John M.».

INSERT

Не беспокойтесь, потому что подстроку можно вставить в нужное место. Для того чтобы поместить фамилию 'Smith' обратно в строку, обратимся к процедуре INSERT.

```
INSERT('Smith', Name, 9)
```

Теперь переменная Name содержит «John M. Smith».

Следующая программа демонстрирует применение строковых функций. Она принимает имя, отчество и фамилию и образует полное имя, включая завершающие пробелы. Кроме того, отчество преобразуется в инициал.

```
{ ------------------------------ Пример 6-11 ----------------------------- }
PROGRAM StringFunctions1(INPUT,OUTPUT);
VAR
  Name :STRING[30];
  First, Middle, Last :STRING[10];
BEGIN
  WRITE('Пожалуйста, введите ваше имя: ');
  READLN(First);
  First := CONCAT(First, ' ');
  WRITE('Пожалуйста, введите ваше отчество: ');
  READLN(Middle);
  Middle := COPY(Middle, 1, 1);
  Middle := CONCAT(Middle, '. ');
  WRITE('Пожалуйста, введите вашу фамилию: ');
  READLN(Last);
  Name := CONCAT(First, Middle, Last);
  WRITELN;
  WRITELN('Ваше полное имя: ',Name)
END.
```

Выполнение теста:

```
Пожалуйста, введите ваше имя: Sally
Пожалуйста, введите ваше отчество: Ann
Пожалуйста, введите вашу фамилию: Abolrous

Ваше полное имя: Sally A. Abolrous
```

Задание 6–5

Измените приведенную выше программу так, чтобы она превращала первую букву каждого имени в прописную, если она была введена в нижнем регистре.

Заключение

В этой главе вы изучили работу операторов READ и READLN с числовыми и символьными значениями. Вы также научились использовать функцию конца строки EOLN в процессе ввода текстовой строки с клавиатуры и функцию конца файла EOF при чтении текстового файла. Вы также изучили несколько важных функций и процедур обработки строк, которые доступны в современных реализациях Паскаля:

- CONCAT
- COPY
- POS
- INSERT
- DELETE

Но что более важно, на примерах и заданиях вы приобрели опыт в обработке текста – как строк, так и отдельных символов.

Упражнения

1. Напишите программу на Паскале, которая читает с клавиатуры строку текста и переводит все буквы из нижнего регистра в верхний. Это пример выполнения программы:

```
Введите строку текста и нажмите на <Enter>: Привет! Давно не виделись.
ПРИВЕТ! ДАВНО НЕ ВИДЕЛИСЬ.
```

2. Одно из интересных приложений текстовой обработки это кодирование и декодирование. Простой способ закодировать строку – это написать ее задом наперед. Напишите программу, которая читает с клавиатуры строку текста и выводит ее на экране наоборот. Ниже приведен пример выполнения требуемой программы:

```
Введите строку текста и нажмите <Enter>: Изучаем Паскаль за три дня
Обратный текст:
янд ирт аз ьлакаП меачузИ
```

3. Напишите программу, которая читает строку текста или текстовый файл и меняет регистр каждой буквы. Таким образом, все буквы верхнего регистра переходят в нижний, и наоборот. Ниже приведен пример выполнения требуемой программы:

```
Пожалуйста, введите имя: Mr. John Martin Smith
Преобразованное имя:
mR. jOHN mARTIN sMITH
```

4. Напишите программу, которая кодирует текстовую строку таким образом, что каждая буква заменяется стоящей за ней в алфавите. Также напишите программу декодирования, которая преобразует в

оригинальный текст перед кодированием. Ниже приведен пример выполнения требуемой программы:

```
Введите строку текста и нажмите <Enter>: This is a test
Закодированный текст:
Uijt!jt!b!uftu
```

Вы можете улучшить программу, специальным образом кодируя пробел и букву «Z», для того чтобы избежать неалфавитных символов в закодированной строке. Например, можно кодировать букву «Z» как «A». А символ пробела можно оставить без изменений.

Ответы

1. Напишите объявления переменных, а затем используйте следующий фрагмент кода, который реализует основной алгоритм вашей программы:

```
FOR I := 1 TO LENGTH(Name) DO
  BEGIN
    LowerCase := (ORD(Name[I]) > 96) AND (ORD(Name[I]) < 123);
    IF LowerCase THEN
      Name[I] := CHR(ORD(Name[I])-32);
    WRITE(Name[I])
  END;
```

2. Напишите объявления переменных, а затем используйте следующий фрагмент кода, который реализует основной алгоритм вашей программы:

```
WRITE(' Введите строку текста и нажмите на <ВВОД>: ');
READLN(Name);
WRITELN(' Обратный текст:');
FOR I := LENGTH(Name) DOWNTO 1 DO
  WRITE(Name[I]);
```

3. Измените код упражнения 1, добавив логическую переменную `UpperCase`, и используйте для замены регистра аналогичный алгоритм.

4. В следующей строке кода каждый символ строки Name заменяется его последователем.

```
FOR I := 1 TO LENGTH(Name) DO
  BEGIN
    Name[I] := SUCC(Name[I]);
    WRITE(Name[I])
  END;
```

Для обратной операции декодирующая функция использует функцию PRED.

7

Архитектура программы

7–1. Программы и подпрограммы

При работе с реальными приложениями встречаются более серьезные задачи, чем те, с которыми вы имели дело до сих пор, поэтому обычно следует разбивать главную задачу на несколько более простых подзадач, а программы, каждую индивидуально, на *подпрограммы*. Затем подпрограммы собираются вместе, образуя законченную программу. Разбив приложение на возможно меньшие модули, вы обнаружите, что многие из них решают общие задачи, такие как сортировка имен или чисел. Это означает, что вы сможете написать несколько общеупотребительных подпрограмм и использовать их в дальнейшем в различных приложениях. Другое преимущество подпрограмм заключается в том, что они позволяют избежать повторного написания операторов печати заголовка или вывода на экран меню; такие задачи оформляются в виде подпрограмм и вызываются при необходимости. В Паскале можно делить программу на подпрограммы, которые называются *процедурами* и *функциями*. Фактически сам язык Паскаль построен из предопределенных процедур и функций. Когда компилятор встречает в программе, например оператор WRITELN, то для выполнения поставленной задачи вызывается предопределенная процедура WRITELN.

7–2. Процедуры

Разделенное на процедуры главное тело программы Scores из главы 5 могло бы выглядеть примерно так:

```
BEGIN
  ReadScores;
  GetAverage;
  DisplayResults
END.
```

Главная программа состоит только из трех вызовов, каждый из которых представляет собой имя процедуры, выполняющей определенную задачу. Процедура ReadScores читает массив результатов, GetAverage вычисляет средний результат, а DisplayResults выводит результаты на экран. Как можно видеть, определенная пользователем процедура вызывается по ее имени, точно так же, как и любая стандартная процедура.

Определение процедуры

До вызова процедуру надо определить в *секции подпрограмм*, находящейся в конце раздела определений. Приведем полный список секций раздела определений:

Секция LABEL
Секция CONST
Секция TYPE
Секция VAR
Секция объявления процедур и функций

Определение процедуры очень похоже на определение программы; оно также состоит из заголовка, раздела объявлений и операторов. Начнем с простой процедуры, рисующей линию длиной 20 символов.

```
{ ------------------------------ Пример 7-1 ------------------------------ }
PROGRAM Procedures1(OUTPUT);
{ ------------ Начало процедуры ------------ }
PROCEDURE DrawLine;
CONST
  Dash = '-';
  LineLength = 20;
VAR
  Counter :INTEGER;
    BEGIN
      FOR Counter := 1 TO LineLength DO
        WRITE(Dash);
      WRITELN
    END;
{ --------------- Конец процедуры --------------- }
{ -------------- Главная программа -------------- }
BEGIN
  WRITELN;
  DrawLine;
  WRITELN('** ЭТО ТЕСТ **');
  Drawline
END.
```

Вывод:

```
--------------------
** ЭТО ТЕСТ **
--------------------
```

В главной программе нет переменных или констант, поэтому раздел объявлений содержит только определение процедуры. Определение начинается с заголовка процедуры:

```
PROCEDURE Drawline;
```

Заголовок включает имя процедуры, которое должно быть разрешенным идентификатором. Затем мы видим раздел объявлений:

```
CONST
  Dash = '-';
  LineLength = 20;
VAR
  Counter :INTEGER;
```

Раздел объявлений в процедуре состоит из тех же секций, что и раздел объявлений программы. В нашем примере были объявлены две именованные константы и переменная. Затем следует заключенный в блок оператор процедуры, который представляет решаемую задачу (рисование линии).

```
BEGIN
  FOR Counter := 1 TO LineLength DO
    WRITE(Dash);
  WRITELN
END;
```

Обратите внимание, что оператор END в процедуре заканчивается точкой с запятой, а не точкой.

В главной программе эта процедура вызывается дважды – до и после вывода текста.

Передача значений в процедуру

Процедура DrawLine рисует линию длиной 20 символов, что может не подходить для какого-нибудь другого приложения. В следующей программе процедура изменена так, что длина рисуемой линии меняется в соответствии с длиной выводимого текста. После запуска программа просит ввести предложение, затем выводит его на экран между двумя линиями той же длины, что и предложение. Сначала протестируйте программу, а затем познакомьтесь с комментариями.

```
{ ------------------------------ Пример 7-2 ------------------------------ }
PROGRAM Procedures2(OUTPUT);
VAR
  Len          :INTEGER;
  TestSentence :STRING;
{ ------------ Начало процедуры ------------ }
PROCEDURE DrawLine(LineLength :INTEGER);
CONST
```

```
    Dash = '-';
  VAR
    Counter :INTEGER;
  BEGIN
    FOR Counter := 1 TO LineLength DO
      WRITE(Dash);
    WRITELN
  END;
  { --------------- Конец процедуры --------------- }
  { -------------- Главная программа -------------- }
  BEGIN
    WRITE('Пожалуйста, введите предложение: ');
    READLN(TestSentence);
    Len := LENGTH(TestSentence);
    WRITELN;
    DrawLine(Len);
    WRITELN(TestSentence);
    Drawline(Len)
  END.
```

Выполнение примера:

```
Пожалуйста, введите предложение: Изучаем Паскаль за три дня
--------------------------
Изучаем Паскаль за три дня
--------------------------
```

Вместо определения количества черточек как константы, длина предложения объявляется в главной программе как переменная `Len`. После того как предложение введено, его длина вычисляется и передается в процедуру в качестве *параметра*. В этом случае вызов процедуры выглядит так:

```
DrawLine(Len);
```

Заголовок процедуры также должен включать принимаемый параметр:

```
PROCEDURE DrawLine(LineLength :INTEGER);
```

В скобках находится параметр `LineLength`, затем через двоеточие указан его тип (INTEGER).

Вызываемой процедуре передается значение переменной `Len` (из главной программы) и присваивается переменной `LineLength` для обработки. Переменная `Len` называется *фактическим параметром*, а переменная `LineLength` называется *формальным параметром*. После выполнения процедуры управление передается обратно в главную программу, выполнение которой продолжается с оператора, стоящего следом за вызовом процедуры. Значение формального параметра определено только во время выполнения процедуры. При вызове процедуры в качестве параметров могут выступать литеральные значения, например:

```
DrawLine(30);
```

В результате вызова будет нарисована линия длиной 30 символов.

Функции и процедуры могут передаваться в качестве параметров, но во многих реализациях это запрещено.

Когда в качестве параметра используется значение, то говорят, что параметр передается *по значению*; если параметр представляет собой переменную, то говорят, что параметр передается *по ссылке*.

Вызов процедуры может содержать более одного параметра, например:

```
Process(A, B, C);
```

Количество фактических параметров в вызове процедуры должно совпадать с количеством формальных параметров, это означает, что заголовок процедуры может выглядеть примерно так:

```
PROCEDURE Process(X, Y :INTEGER; Z :REAL);
```

Переменные `A` и `B` в вызывающей программе должны иметь тип INTEGER, поскольку они соответствуют `X` и `Y`, а переменная `C` – тип REAL, так как она поставлена в соответствие `Z`. Обратите внимание на точку с запятой, которая разделяет объявления в заголовке процедуры. Короче говоря, количество, тип и расположение фактических и формальных параметров должны совпадать.

Задание 7–1

Измените предыдущую программу так, чтобы можно было передавать в процедуру вид символа линии (– или * и т. д.), а вывод располагался по центру строки (предполагаем, что строка имеет длину 80 символов). Ниже приводится результат выполнения теста требуемой программы:

```
Пожалуйста, введите предложение: Изучаем C за три дня
Пожалуйста, введите символ линии: *
                    ********************
                    Изучаем C за три дня
                    ********************
```

Возврат значений из процедуры

Процедуру можно использовать для изменения значения переменной и возвращения его обратно в вызывающую программу. В этом случае перед формальными параметрами должно присутствовать ключевое слово VAR. Рассмотрим пример процедуры, которая принимает значения двух переменных, а возвращает их кубы. Заголовок процедуры может выглядеть примерно так:

```
PROCEDURE CubeThem(VAR X, Y :REAL);
```

Параметры в эту процедуру можно передать только по ссылке:

```
CubeThem(A, B);
```

Значения A и B будут переданы в процедуру, заместив, соответственно, X и Y, возведены в куб и возвращены обратно в вызывающую программу. В этом случае в качестве фактических параметров запрещено использовать литеральные значения или выражения.

Когда формальным параметрам предшествует слово VAR, они называются *параметрами-переменными*, в противном случае они называются *параметрами-значениями*.

Основная форма заголовка процедуры:

PROCEDURE имя;

или

PROCEDURE имя-процедуры(список-формальных-параметров);

Основная форма вызова процедуры:

имя-процедуры;

или

имя-процедуры(список-фактических-параметров);

Следующая программа представляет пример использования формальных параметров обоих типов. Она демонстрирует тот же алгоритм, что и программа PowerOperator (в главе 2), но принимает основание и степень и возвращает результат с помощью процедуры.

```
{ ------------------------------ Пример 7-3 -------------------------------}
PROGRAM VarParms(INPUT,OUTPUT);
VAR
  a, b, c :REAL;
{ ------------ Процедура Definition -------------- }
PROCEDURE PowerOperator(X, Y :REAL; VAR Z:REAL);
BEGIN
  Z := EXP(LN(X)*Y)
END;
{ ----------------- Главная программа ----------------- }
BEGIN
  WRITE('Введите основание и степень, разделенные пробелом:');
  READLN(a,b);
  PowerOperator(a, b, c);
  WRITELN('Значение ',a:0:2,' возведенное в степень ',b:0:2,' равно ',c:0:2)
END.
```

Выполнение теста:

```
Введите основание и степень, разделенные пробелом:2 5
Значение 2.00, возведенное в степень 5.00, равно 32.00
```

Обратите внимание, что X и Y объявлены в процедуре как параметры-значения, поскольку они принимают из главной программы только

значения, а Z объявлен параметром-переменной, так как он возвращает результат.

7–3. Глобальные и локальные переменные

И формальные параметры и переменные, объявленные в процедуре, называются *локальными переменными*, потому что они доступны только внутри этой процедуры; другими словами, они невидимы для главной программы и для любых других подпрограмм. С другой стороны, переменные, объявленные в главной программе, называются *глобальными переменными*, поскольку они доступны в любом разделе программы. В примере 7-2 переменная `TestSentence` является глобальной и доступна в процедуре DrawLine без передачи ее в качестве параметра. Любое присвоение этой переменной в процедуре изменит ее значение и в главной программе. Однако локальная переменная `Counter` не доступна в главной программе.

Теперь рассмотрим случай, когда объявлены две переменные с одним именем (например, `X`) – одна в главной программе, а другая – в процедуре. Повторное объявление глобальной переменной `X` в процедуре создаст локальную переменную с тем же именем и скроет глобальную переменную от процедуры. Это означает, что у вас будут две различные переменные, соответствующие двум различным областям памяти. После выхода из процедуры останется только одна глобальная переменная с именем «X». Эти ограничения помогают программисту предохранить глобальную переменную от случайного изменения в процедуре.

Переменные главной программы могут быть изменены другими процедурами, если они глобальные (и не объявлены повторно в процедуре) или передаются в процедуру по ссылке как параметры-переменные. Доступ к глобальным переменным из подпрограмм не рекомендуется, поскольку это нарушает модульность программы. Применение параметров защищает процедуры и делает их независимыми и удобными для различных программ.

Пример: Процедура сортировки

Вернемся к примеру 5-5 и разделим его на две основные процедуры. Эта программа читала, сортировала и выводила на экран массив из шести элементов. Теперь необходимо написать три процедуры, которые читают, сортируют и выводят на экран массив любого размера. Те же результаты будут достигнуты путем передачи в процедуру массива и количества его элементов. Главное тело программы будет состоять только из трех вызовов:

```
ReadNumbers(ArraySize, Numbers);
SortNumbers(ArraySize, Numbers);
PrintNumbers(ArraySize, Numbers);
```

Таким образом, любая из этих трех процедур может быть использована в любой программе. Важным моментом, о котором необходимо здесь упомянуть, является то, что передавая массив в процедуру или функцию, его необходимо объявить в секции TYPE. Тогда формальные параметры в заголовке процедуры будут выглядеть примерно так:

```
PROCEDURE ReadNumbers(L: INTEGER; VAR R :NumbersArray);
```

Параметр L соответствует ArraySize, а массив R соответствует массиву Numbers. Как можно видеть из объявления параметра, он имеет тип NumbersArray – тот же, что и тип массива Numbers. Приведем законченную программу:

```
{ ------------------------------ Пример 7-4 ------------------------------ }
PROGRAM Sorting(INPUT,OUTPUT);
CONST
  ArraySize = 6;
TYPE
  Range        = 1..ArraySize;
  NumbersArray = ARRAY[Range] OF INTEGER;
VAR
  Numbers :NumbersArray;
{ ----------------- Процедура чтения --------------- }
PROCEDURE ReadNumbers(L: INTEGER; VAR R :NumbersArray);
VAR
  I :INTEGER;
BEGIN
  FOR I := 1 TO L DO
    BEGIN
      WRITE('Введите элемент #', I,': ');
      READLN(R[I])
    END
END;
{ ----------------- Процедура сортировки --------------- }
PROCEDURE SortNumbers(M: INTEGER; VAR S :NumbersArray);
VAR
  I, J, Pot :INTEGER;
BEGIN
  FOR I := 1 TO M-1 DO
    FOR J := I+1 TO M DO
      IF S[I] > S[J] THEN
        BEGIN { Обмен содержимым }
          Pot := S[J];
          S[J] := S[I];
          S[I] := Pot
        END
END;
{ ---------------- Процедура печати --------------- }
PROCEDURE PrintNumbers(N: INTEGER; T :NumbersArray);
VAR
  I :INTEGER;
BEGIN
```

```
  WRITELN;
  WRITE('Отсортированный массив: ');
  FOR I := 1 TO N DO
    WRITE(T[I],' ');
  WRITELN;
END;
{ --------------- Главная программа ------------------- }
BEGIN
  ReadNumbers(ArraySize, Numbers);
  SortNumbers(ArraySize, Numbers);
  PrintNumbers(ArraySize, Numbers);
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Пример выполнения:

```
Введите элемент #1: 44
Введите элемент #2: 22
Введите элемент #3: 8
Введите элемент #4: 1
Введите элемент #5: 667
Введите элемент #6: 3
Отсортированный массив: 1 3 8 22 44 667
Для продолжения нажмите ENTER..
```

Обратите внимание, что параметр передается как параметр-переменная в процедуру, которая, как ожидается, изменит значение массива (т. е. `ReadNumbers` и `SortNumbers`), но в этом нет необходимости в случае процедуры PrintNumbers, которая выводит массив на экран и не возвращает никаких значений в главную программу. В последнем случае массив был передан как параметр-значение. Обратите также внимание на использование переменных в различных процедурах, что делает любой независимый блок программы. Если любую из этих процедур нужно использовать с массивами различных типов, необходимо только изменить тип `NumbersArray` или использовать то же имя типа для нового массива в главной программе. В этом примере можно применять процедуры совсем без параметров и обрабатывать глобальные переменные напрямую, но в этом случае необходимо будет дать одинаковые имена переменным во всех процедурах и в главной программе, что требует больших усилий и влечет за собой весь риск работы с глобальными переменными.

Подобно массивам, если в качестве формальных параметров в подпрограмму передаются перечислимые типы и поддиапазоны, то они должны быть объявлены в секции TYPE.

7–4. Функции

Функция – это подпрограмма, возвращающая значение, которое затем присваивается имени функции в вызывающей программе. Подобно

предопределенным функциям, функция, определенная пользователем, имеет один или более параметров. Определение функции находится в секции подпрограмм раздела объявлений и состоит из заголовка, раздела объявлений и операторов. Взгляните на этот заголовок функции, которая возвращает среднее значение трех чисел:

```
FUNCTION Avg(X, Y, Z :REAL) :REAL;
```

Заголовок напоминает заголовок процедуры – за исключением того, что после заголовка функции указан тип возвращаемого значения (:REAL). Функцию можно вызвать, например, с помощью следующих операторов:

```
D := Avg(A, B, C);
WRITELN(Avg(F, G, H):2:2);
WRITELN(Avg(94, 33.5, 45*1.2):2:2);
```

Как видите, параметр может быть литеральной константой, выражением или переменной.

Заголовок функции имеет следующий вид:

```
FUNCTION имя-функции(список-формальных-параметров) :возвращаемый-тип;
```

В этой программе демонстрируется функция Avg.

```
{ ------------------------------ Пример 7-5 ------------------------------ }
PROGRAM Functions1(INPUT, OUTPUT);
VAR
  A, B, C :REAL;
{ ------------ Начало функции ------------- }
FUNCTION Avg(X, Y, Z :REAL) :REAL;
BEGIN
  AVG := (X + Y + Z) / 3
END;
{ --------------- Конец функции ---------------- }
{ ------------- Главная программа -------------- }
BEGIN
  WRITE('Введите три числа: ');
  READLN(A, B, C);
  WRITELN('Среднее значение= ', Avg(A, B, C):0:2)
END.
```

Выполнение теста:

```
Введите три числа: 2 3 8
Среднее значение= 4.33
```

Функции, как и процедуры, представляют собой независимые подпрограммы. Все параметры, переменные и константы, объявленные внутри тела функции, являются локальными по отношению к ней и невидимы для других блоков. Подпрограмме-функции должно быть присвоено возвращаемое значение.

В подпрограмме-функции с именем функции нельзя обращаться как с переменной, то есть оно не может участвовать в выражениях. Ему можно только присвоить значение.[1]

Задание 7-2

Напишите функцию, определяющую максимальное число в одномерном массиве, и вставьте ее в программу. Можете использовать любые процедуры, написанные вами ранее.

7-5. Советы относительно области видимости переменных

Следующая программная структура состоит из трех программных блоков: процедуры Kid1, процедуры Kid2 и главной программы Parent. Согласно правилам области видимости переменных любая переменная, объявленная в Parent (глобальная переменная), доступна и для Kid1 и для Kid2, пока она не переобъявлена локально в одной из них. С другой стороны, любая локальная переменная, объявленная в Kid1, не видима и для Parent и для Kid2. То же самое относится и к переменным Kid2. Если считать главную программу родителем, а подпрограммы детьми, то отсюда следует: все, что принадлежит родителям, принадлежит и детям, но не наоборот. Другими словами, дети наследуют от родителей все, но каждый ребенок имеет свою собственность, которую не наследуют ни родитель, ни братья.

```
{ ------------------------------- Пример 7-6 --------------------------- }
PROGRAM Parent;
  { --------------- ПРОЦЕДУРА KID1 --------------- }
  PROCEDURE Kid1(...);
  ...
  BEGIN
  ...
  END; { --------- КОНЕЦ ПРОЦЕДУРЫ KID1 --------- }
  { ---------------- ПРОЦЕДУРА KID2 -------------- }
  PROCEDURE Kid2(...);
  ...
  BEGIN
  ...
  END; { --------- КОНЕЦ ПРОЦЕДУРЫ KID2 --------- }
{ ------------------------------ ГЛАВНАЯ ПРОГРАММА ---------------------- }
BEGIN
...
END.
```

[1] В современных версиях Паскаля это ограничение не действует, а правила присваивания возвращаемому значению отличаются от описанных. См. справку к используемой версии. – *Примеч. науч. ред.*

Любая из двух процедур может быть вызвана из главной программы. Процедура Kid1 может быть также вызвана из Kid2, поскольку она уже определена, но процедура Kid2 не может быть вызвана из Kid1, так как она еще не определена. Существует способ обойти это ограничение с помощью *упреждающего объявления* (*forward declaration*), поставив заголовок Kid2 после ключевого слова FORWARD в начале программы, например:

```
PROGRAM Parent;
{ Упреждающее объявление Kid2 }
  PROCEDURE Kid2(...); FORWARD;
{ Определение Kid1 }
  PROCEDURE Kid1(...);
  ...
{ Определение Kid2 }
  PROCEDURE Kid2(...);
    ...
{ Главная программа }
    ...
```

Теперь взгляните на новую программную структуру в следующем примере. Процедура GrandKid определена внутри процедуры Kid, то есть Kid стала родителем другой подпрограммы. В этом случае любая переменная в Kid является глобальной в GrandKid, как и переменные Parent (пока они не объявлены повторно в GrandKid). Тем не менее, локальные переменные GrandKid недоступны ни в Kid, ни в Parent.

```
{ ------------------------------- Пример 7-7 --------------------------- }
PROGRAM Parent;
    { ------------------- ПРОЦЕДУРА KID -------------------- }
    PROCEDURE Kid(...);
    ...
        { ------------ ПРОЦЕДУРА GRANDKID ----------- }
        PROCEDURE GrandKid(...);
        BEGIN
        ...
        END; { ----- КОНЕЦ ПРОЦЕДУРЫ GRANDKID ----- }
    BEGIN
    ...
    END; { ------------- КОНЕЦ ПРОЦЕДУРЫ KID -------------- }
{ ---------------------------- ГЛАВНАЯ ПРОГРАММА ------------------------ }
BEGIN
...
END.
```

Подведем итог:

- Область видимость переменной ограничена программным блоком, в котором она определена.
- Глобальная переменная доступна в любом программном блоке, если она не определена в нем повторно.

- Локальная переменная недоступна вне программного блока, в котором она определена. Однако она доступна любой подпрограмме, определенной внутри этого программного блока, если она не переопределяется внутри подпрограммы.
- Любая подпрограмма может быть вызвана из любого программного блока, если ее определение (или ее упреждающее объявление) предшествует вызову.

7–6. Рекурсия

Вызов функцией или процедурой самой себя называется *рекурсией*. Хороший пример рекурсии представляет собой функция вычисления факториала. Вы знаете (из главы 4), что факториал числа X может быть получен из соотношения:

factorial(X) = X * factorial(X–1)

Другими словами, чтобы получить факториал 4, надо умножить 4 на факториал 3; чтобы получить факториал 3, надо умножить 3 на факториал 2 и так далее до 1.

Приведем программу, которая содержит функцию факториала.

```
{ ------------------------------ Пример 7-8 ------------------------------ }
PROGRAM FunctionRecursion(INPUT, OUTPUT);
VAR
  A :INTEGER;
{ ------------- Определение функции--------------- }
FUNCTION Factorial(X :INTEGER) :REAL;
BEGIN
  IF X <= 1 THEN
    Factorial := 1
  ELSE
    Factorial := X * Factorial(X-1);
END;
{ --------------- Конец функции ----------------- }
{ ------------- Главная программа -------------- }
BEGIN
  WRITE('Введите число: ');
  READLN(A);
  WRITELN('Факториал ', A,' = ', Factorial(A):0:0)
END.
```

Выполнение теста:

```
Введите число: 6
Факториал 6 = 720
```

Обратите внимание на функцию в операторе:

```
Factorial := X * Factorial(X-1);
```

слева стоит имя функции, а справа – вызов функции для вычисления факториала X–1. Этот процесс будет продолжаться, пока не выполнится условие выхода из функции.

Задание 7–3

Напишите подпрограмму вычисления факториала в виде процедуры и сравните ее с функцией факториала.

Заключение

В этой главе вы изучили структуру программы на Паскале.

Вы знаете, как разделить программу на подпрограммы: функции или процедуры.

Важно помнить следующее:

1. Подпрограмма объявляется в последней секции раздела объявлений программы и состоит из заголовка, раздела объявлений и операторов.

   Заголовок процедуры имеет вид:

   PROCEDURE name;

   или

   PROCEDURE имя-процедуры(список-формальных-параметров);

   Заголовок функции имеет вид:

   FUNCTION имя-функции(список-формальных-параметров) :возвращаемый-тип;

2. Процедура вызывается по ее имени точно так же, как оператор. Если при вызове процедуры используются параметры, они должны совпадать с параметрами заголовка процедуры. Параметры процедуры – это или значения, или параметры-переменные. Параметр-переменная используется, когда требуется, чтобы процедура изменила его значение.[1]
3. Функция обычно вызывается как часть выражения; она возвращает единственное значение, которое замещает имя функции в этом выражении.[2]

---

[1] Условие обязательного изменения параметра вовсе не обязательно. – *Примеч. науч. ред.*

[2] Правило замещения имени функции относится к рассматриваемой версии Паскаля. – *Примеч. науч. ред.*

4. Вы знаете теперь, что каждая переменная имеет область видимости, и изучили правила, которые контролируют область видимости и связи между глобальными и локальными переменными.

Упражнения

1. Истинно или ложно:
   a. Параметры, используемые в объявлении процедуры или функции, называются формальными параметрами, а те, которые используются при вызове процедуры или функции, называются фактическими параметрами.
   b. Параметру-переменной предшествует ключевое слово VAR.
   c. Параметру-значению предшествует ключевое слово VAR.
   d. Имени функции может быть присвоено значение.
   e. Имя функции может участвовать в выражении.
   f. Функция всегда возвращает значение.
   g. Процедура не может возвратить значение.
   h. Массивы, перечисления и поддиапазоны должны объявляться в секции TYPE, если они используются в качестве формальных параметров в подпрограмме.
2. Истинно или ложно:
   a. Область видимости переменной ограничена программным блоком, в котором она объявлена.
   b. Глобальная переменная доступна в любом программном блоке, если она не переобъявлена в нем локально.
   c. Локальная переменная недоступна вне программного блока, в котором она объявлена. Однако она доступна любой подпрограмме, определенной внутри этого программного блока, если она не объявлена повторно внутри подпрограммы.
   d. Любая подпрограмма может быть вызвана из любого программного блока, если ее определение или ее упреждающее объявление предшествует вызову.
3. Напишите определение процедуры, которая принимает три целых параметра `x`, `y` и `z`. Параметр `x` изменяется процедурой и возвращается вызывающей программе. Также напишите вызов процедуры.
4. Последовательность Фибоначчи (названная по имени итальянского математика Леонардо Фибоначчи) – это последовательность чисел 1, 1, 2, 3, 5, 8, 13, ..., в которой каждое последующее число, после первых двух, равно сумме двух предыдущих чисел. Напишите функцию, которая принимает в качестве параметра номер в последовательности и возвращает соответствующее число Фибоначчи. Например, вызов Fibonacci(5) возвращает 8, а вызов Fibonacci(6) возвращает 13.

5. Напишите главную программу, которая вызывает функцию Фибоначчи (из упражнения 4) для печати определенного количества элементов последовательности Фибоначчи. Ниже приведен пример требуемого вывода:

```
Пожалуйста, введите максимальное число элементов: 9
Последовательность      Число Фибоначчи
0                1
1                1
2                2
3                3
4                5
5                8
6                13
7                21
8                34
```

Ответы

1. c, e и g ложно.
2. Все истинно.
3. Объявление процедуры:

   PROCEDURE MyProcedure(VAR X:REAL; Y, Z:REAL);

   Пример вызова процедуры:

   MyProcedure(a, b, c)

4. Определение функции Фибоначчи:

```
FUNCTION Fibonacci(I: INTEGER): LONGINT;
BEGIN
  IF I <= 1 THEN
    Fibonacci := 1
  ELSE
    Fibonacci := Fibonacci(i-1) + Fibonacci(i-2)
END;
```

5. Программа, которая вызывает функцию Фибоначчи; добавьте в нее объявление переменных.

```
WRITE('Пожалуйста, введите максимальное число элементов: ');
READLN(N);
WRITELN('Последовательность ', CHR(9), 'Число Фибоначчи');
I := 0;
WHILE I < N DO
  BEGIN
    WRITELN(I, CHR(9), CHR(9), Fibonacci(i));
    I := I + 1;
  END;
```

8

Множества и записи

8–1. Множества

Для того чтобы проверить, не принадлежит ли символ к буквам верхнего регистра, можно воспользоваться следующим условием:

```
READ(Character);
IF (Character >= 'A') AND (Character <= 'Z') THEN ...
```

Вот самый простой способ выразить то же самое условие. Взгляните на оператор:

```
IF Character IN ['A'..'Z'] THEN ...
```

Это выражение говорит само за себя. Это почти обычный английский. Здесь сказано: «Если символ принадлежит (IN) множеству заглавных букв ['A'..'Z'], то ...» .

Подобным образом можно определить любое множество элементов, например:

```
['a'..'z'] множество букв нижнего регистра
['A'..'Z', 'a'..'z'] множество всех букв
[0..9] множество цифр
```

Множество – это структурный тип данных, состоящий из неупорядоченных элементов (или *членов*).[1] Можно определить множество-константу с помощью заключенного в скобки списка его элементов, разделенных запятыми. В отличие от массива, порядок элементов во множестве не важен. Например, множество [1,3,5,7], которое представляет множество нечетных чисел между единицей и семеркой, это то же самое, что и множество [1,7,5,3]. Это указывает на другое отличие

[1] Иногда вместо термина «множество» употребляется термин «набор». – *Примеч. науч. ред.*

между множествами и массивами. В массиве можно получить доступ к элементу по его местоположению в массиве, но в множестве нельзя выбрать определенный элемент. Можно только проверить данные на принадлежность к множеству с помощью оператора IN. Если элементы множества образуют последовательный поддиапазон, можно использовать две точки (..); например, множество [1,2,3,4,6,8] может быть записано в виде [1..4,6,8]. Элементы множества могут быть любого порядкового типа, но все они должны иметь одинаковый тип, который называется *базовым типом*.

8–2. Объявление и присваивание множеств

Можно объявить множество-переменную, используя ключевые слова SET OF, как в следующем примере, где объявляется множество с базовым типом CHAR:

```
VAR
  LowerCase :SET OF CHAR;
```

После этого объявления переменной LowerCase можно присвоить множество-константу базового типа CHAR, например:

```
LowerCase := ['a'..'z'];
```

Можно также проверить переменную типа CHAR на принадлежность к этому множеству с помощью такого выражения:

```
IF Character IN LowerCase THEN ...
```

Заметьте, что если используется выражение IN ['a'..'z'], то нет необходимости в объявлениях.

Как и в случае других структурных типов, предпочтительно объявлять множества в секции TYPE; затем можно использовать этот тип в секции VAR для объявления переменных. Объявление имеет вид:

```
идентификатор-типа = SET OF базовый-тип;
```

Пример:

```
TYPE
  Days = (Monday,Tuesday,Wednesday,Thursday,Friday,Saturday,Sunday);
  Languages = (C,CPP,Pascal,Fortran,Basic,Cobol,Assembly);
  Digits = SET OF 0..9;
  Lowercase = SET OF 'a'..'z';
  Uppercase = SET OF 'A'..'Z';
  DaySet = SET OF Days;
  LanguageSet = SET OF Languages;
  CharacterSet = SET OF CHAR;
VAR
  WholeWeek, WorkingDays, WeekEnd :DaySet;
```

```
OddNum, EvenNum, Numbers        :Digits;
Small                           :Lowercase;
Capital                         :Uppercase;
ProgCodes, HLL, LLL, MLL        :LanguageSet;
Alphabet                        :CharacterSet;
```

В этом объявлении такие переменные, как `WeekEnd`, `WorkingDays` и `WholeWeek` – это множества с базовым типом Days. Любому из этих множеств-переменных может быть присвоен один или более элементов перечисления Days, например:

```
WeekEnd := [Saturday,Sunday];
WorkingDays := [Monday..Friday];
WholeWeek := [Monday..Sunday];
```

Последний оператор эквивалентен оператору:

```
WholeWeek := [Monday,Tuesday,Wednesday,Thursday,Friday,Saturday,Sunday];
```

Значение множества-переменной не определено, пока ему не присвоено значение. Когда вы присваиваете множество-константу множеству-переменной, их типы должны быть *совместимы*, то есть они должны иметь один тип, быть поддиапазонами одного типа или одно из них должно быть поддиапазоном другого. Пример присваиваний:

```
OddNum := [1,3,5,7,9];
EvenNum := [2,4,6,8];
ProgCodes := [C..Assembly];
LLL := [Assembly];
```

Пустое множество – это множество без элементов, и оно обозначается константой **[]**. Можно присвоить эту константу любому множеству-переменной любого базового типа, например:

```
OddNum := [];
```

Правила и ограничения

Вот основные правила и ограничения, контролирующие работу с множествами:

- Обычно максимальное количество элементов множества ограничено. Этот предел варьируется в различных реализациях Паскаля, например в Турбо Паскале он равен 255. Объявление SET OF INTEGER не разрешается, поскольку диапазон целых чисел превышает этот предел, но можно обойти это ограничение при помощи поддиапазона, например SET OF 0..99.

  В некоторых реализациях объявление SET OF CHAR также не разрешено, при этом можно использовать поддиапазон типа CHAR.

- Можно присвоить множество другому множеству:

  ```
  NewSet := OldSet;
  ```

В этом случае NewSet – это точная копия OldSet.

- Можно объявить массив множеств:

```
DaysArray = ARRAY[1..10] OF DaySet;
```

где DaySet – это объявленный ранее тип множества.

- Нельзя читать или записывать множества с помощью операторов ввода/вывода, но существуют некоторые приемы программирования, использующие операции над множествами, которые можно применить в данной ситуации.
- Множества можно передавать в качестве параметров в подпрограммы, в этом случае они должны быть объявлены как типы.

8–3. Множественные операторы и операции

Вместе с оператором IN к множествам соответствующих типов можно применять следующие операции:

Объединение (+)

Пересечение (*)

Разность (–)

Объединение

Объединение двух множеств S1 и S2 представляет собой множество, состоящее из элементов, которые принадлежат или множеству S1, или S2, или обоим. Для этой операции применяется оператор +, например (используем предыдущие объявления):

```
Alphabet := Small + Capital;
```

Пересечение

Пересечение двух множеств дает в результате множество, которое состоит из элементов, общих для обоих множеств. Так, результатом оператора:

```
MLL := [C,CPP,Cobol] * [Basic,Fortran,C,CPP]
```

будет множество MLL, состоящее из C и CPP.

Разность

Разность двух множеств S1 и S2 представляет собой множество элементов, которые принадлежат S1, но не принадлежат S2. Например, результатом оператора:

```
HLL := ProgCodes - LLL;
```

будет множество HLL, которое содержит все элементы множества ProgCodes за исключением элемента Assembly.

Советы по применению множественных операторов

Можно использовать оператор объединения для построения нового множества (чтение множества) путем чтения одного элемента за один раз и добавления его в множество, например:

```
Read(NewElement);
Set1 := Set1 + [NewElement];
```

Можно также использовать оператор разности для вывода на экран элементов множества. Это делается путем проверки переменной того же базового типа на принадлежность множеству. Если элемент – член множества, то он выводится на экран, вычитается из множества и заменяется своим последователем. Процесс продолжается, пока множество не станет пустым.

Задание 8–1

Вычислите следующие выражения:

(Ответы можно найти в файле *DRL8-1.TXT* на прилагаемой дискете.)

```
1. ['A','B','C','D'] + ['E','F']
2. ['A','B','C','D'] + ['B','C','E','F']
3. [1,3,7] + []
4. ['A','D','F'] * ['O','F']
5. [1,2,3,4] * [5,6,7]
6. [1,2,3,4] - [5,6,7]
7. [5,6,7] - []
8. [Able, Baker, Charlie] - [Able, Charlie]
```

Операторы отношений

Операторы отношений =, >=, <= и <> могут использоваться с множествами соответствующих типов.

Значения множественных операторов отношений показаны в табл. 8-1 на примере сравнения двух множеств – S1 и S2. В качестве примеров каждого оператора таблица содержит выражения, имеющие значение TRUE.

Операторы > и < не упомянуты в таблице, поскольку их нельзя использовать с множествами.

Таблица 8-1. Множественные операторы отношений

Выражение	Значение	Пример
S1 = S2	S1 и S2 состоят из одних и тех же элементов	[1,0] = [1,0]
S1 <> S2	S1 и S2 не содержат одних и тех же элементов	[1,0] <> [1,4]
S1 >= S2	Все элементы S2 входят в S1	[1,2,3,4] >= [1,2][1,2,3] >= [1,2,3]
S1 <= S2	Все элементы S1 входят в S2	[] <= [1,2,3][1,2,3] <= [1,2,3]

Относительный порядок старшинства операторов Паскаля (включая новый оператор IN) показан в табл. 8-2. Обратите внимание, что множественные операторы (+, –, *) используют те же символы, что и арифметические операторы. Операторы отношений также используются и с простыми типами данных, и с множествами.

Таблица 8-2. Старшинство операторов Паскаля

Оператор	Старшинство
NOT	Приоритет 1 (высший)
* / DIV MOD AND	Приоритет 2
+ – OR (XOR в Turbo Pascal)	Приоритет 3
= > < >= <= <> IN	Приоритет 4 (низший)

Можно объединять выражения отношений при помощи логических операторов AND, OR и NOT, но необходимо следить за приоритетом операторов, например:

```
IF (Ch IN Small) AND (Ch IN Capital) THEN ...
```

Скобки в этом выражении необходимы, потому что оператор IN имеет более низкий приоритет, чем оператор AND.

Задание 8–2

Напишите программу для проверки выражений, приведенных в табл. 8-1.

Пример: Анализатор текста

В следующей программе печатные символы делятся на следующие множества:

1. Буквы нижнего регистра
2. Буквы верхнего регистра

3. Алфавитные символы (представляющие собой объединение множеств 1 и 2)
4. Цифры
5. Символы пунктуации
6. Остальные символы

Программа читает текстовый файл символ за символом и проверяет каждый символ на принадлежность к какому-нибудь из этих множеств. Программа проста и состоит из четырех частей: объявления множеств, инициализации счетчиков, проверки принадлежности символов и вывода результатов.

```
{ ------------------------------ Пример 8-1 ------------------------------ }
PROGRAM TextAnalyzer(INPUT,OUTPUT);
TYPE
  LowerCase  = SET OF 'a'..'z';
  UpperCase  = SET OF 'A'..'Z';
  Digits     = SET OF '0'..'9';
  Characters = SET OF CHAR;
VAR
  Capital                       :UpperCase;
  Small                         :LowerCase;
  Numerals                      :Digits;
  Alphabet, Punctuation, Others :Characters;
  A, C, S, N, P, O, Counter     :INTEGER;
  Ch                            :CHAR;
BEGIN
  Counter := 0; { счетчик всех символов }
  A := 0;       { счетчик алфавитных символов }
  C := 0;       { счетчик прописных букв }
  S := 0;       { счетчик маленьких букв }
  N := 0;       { счетчик числовых символов }
  P := 0;       { счетчик символов пунктуации }
  O := 0;       { счетчик остальных символов }
  Small := ['a'..'z'];
  Capital := ['A'..'Z'];
  Alphabet := Small + Capital;
  Numerals := ['0'..'9'];
  Punctuation := [',',';','-','''',',','!','?',')','(','"',':','_'];
  WRITELN('Начинаем печатать ваш текстовый файл. Для завершения нажмите Ctrl-Z:');
  WHILE NOT EOF DO
    BEGIN
      WHILE NOT EOLN DO
        BEGIN
          READ(Ch);
          Counter := Counter + 1;
          IF Ch IN Alphabet THEN
            BEGIN
              A := A + 1;
```

```
            IF Ch IN Small THEN
              S := S + 1
            ELSE IF Ch IN Capital THEN
              C := C + 1
          END
        ELSE IF Ch IN Numerals THEN
          N := N + 1
        ELSE IF Ch IN Punctuation THEN
          P := P + 1
        ELSE
          O := O + 1
      END;
    READLN
  END;
  WRITELN('Общее количество символов = ', Counter);
  WRITELN('Количество алфавитных символов = ', A);
  WRITELN(' .Количество букв нижнего регистра: ', S);
  WRITELN(' .Количество букв верхнего регистра: ', C);
  WRITELN('Количество числовых символов = ', N);
  WRITELN('Количество символов пунктуации = ', P);
  WRITELN('Количество остальных символов = ', O)
END.
```

Пример выполнения:

```
Начинаем печатать ваш текстовый файл. Для завершения нажмите Ctrl-Z:
Стандартное множество операторов:
      1. Объединение (+).
      2. Пересечение (*).
      3. Разность (-).
^Z                          ----> Нажмите Ctrl+Z для завершения текста
Общее количество символов = 85
Количество алфавитных символов = 53
 .Количество букв нижнего регистра: 49
 .Количество букв верхнего регистра: 4
Количество числовых символов = 3
Количество символов пунктуации = 14
Количество остальных символов = 15
```

Множества удобны для проверки условий. Одно из общих применений множеств – перед оператором CASE для фильтрации нежелательных данных, которые не принадлежат ни одной альтернативе.

8–4. Записи

Запись – еще один структурный тип в Паскале – представляет собой набор связанных элементов данных, возможно, разных типов. Каждый элемент записи называется *полем*. Взгляните на эту запись, которая хранит информацию о каждом сотруднике компании:

Запись сотрудника

№ поля	Информация	Разрешенный тип данных
1.	Имя	STRING
2.	Адрес	TRING
3.	Номер телефона	TRING/INTEGER
4.	Почасовой тариф	REAL
5.	Семейное положение	CHAR/Enumeration

В отличие от массивов (которые содержат элементы одинакового типа), записи могут содержать поля любого типа, включая сам тип RECORD.

Объявление записи

Объявление записи имеет вид:

```
идентификатор-типа = RECORD
                       список-полей
                     END;
```

Список полей содержит имя и тип каждого поля, как в объявлении записи EmployeeRecord:

```
TYPE
EmployeeRecord = RECORD
                   Name          :STRING[25];
                   Address       :STRING[40];
                   Phone         :STRING[12];
                   Rate          :REAL;
                   MaritalStatus :CHAR;
                 END;
```

Если ваша реализация Паскаля не поддерживает тип STRING, можно заменить переменные типа STRING на переменные типа INTEGER или CHAR, при этом надо заменить переменную `Name` *другой переменной, например* `ID`.

Объявление записи должно завершаться ключевым словом END.

Затем в секции VAR объявляется запись типа EmployeeRecord:

```
VAR
  EmployeeRec :EmployeeRecord;
```

Как и в случае других структурных и определяемых пользователем типов данных, можно объявить запись прямо в секции VAR, но вы теперь знаете преимущества объявления структур данных как типов.

Доступ к полям

К каждому полю записи можно получить доступ, используя идентификатор записи вместе с идентификатором поля, разделенные точкой. Например, можно присвоить полю значение с помощью оператора:

```
EmployeeRec.Name := 'Charles A. Dixon';
EmployeeRec.Rate := 22.5;
```

Можно сделать то же самое в операторах ввода и вывода:

```
WRITELN('Имя сотрудника: ', EmployeeRec.Name);
```

Фактически область видимости идентификатора поля (такого как переменная `Name`) ограничена записью, в которой идентификатор был объявлен, и при желании он может быть использован где угодно в программе в качестве имени другой переменной. В следующем примере запись EmployeeRec заполняется, а затем выводится на экран.

Этот тип составной переменной называется *полевой переменной*.

```
{------------------------------ Пример 8-2 ------------------------------}
PROGRAM RecordExample1(OUTPUT);
TYPE
  EmployeeRecord = RECORD
                     Name          :STRING[25];
                     Address       :STRING[40];
                     Phone         :STRING[12];
                     Rate          :REAL;
                     MaritalStatus :CHAR;
                   END;
VAR
  EmployeeRec :EmployeeRecord;
BEGIN
{ Присвоение значений полевым переменным }
  EmployeeRec.Name := 'Diane J. Bedford';
  EmployeeRec.Address := '20 Carmen Avenue, New Orleans, LA 70112';
  EmployeeRec.Phone := '504-666-5043';
  EmployeeRec.Rate := 28.5;
  EmployeeRec.MaritalStatus := 'S';
{ Вывод на экран информации записи }
  WRITELN('Имя сотрудника: ', EmployeeRec.Name);
  WRITELN('Адрес: ', EmployeeRec.Address);
  WRITELN('Телефон #: ', EmployeeRec.Phone);
  WRITELN('Почасовой тариф: $', EmployeeRec.Rate:0:2);
  WRITELN('Семейное положение: ', EmployeeRec.MaritalStatus)
END.
```

Вывод:

```
Имя сотрудника:     Diane J. Bedford
Адрес:              20 Carmen Avenue, New Orleans, LA 70112
```

```
Телефон #:          504-666-5043
Почасовой тариф:    $28.50
Семейное положение: S
```

Оператор WITH

Однако существует более короткий путь выполнения этой задачи, с помощью оператора WITH. Причем в этом случае необходимость в полевых переменных отпадает. Взгляните на этот блок присваиваний с оператором WITH:

```
WITH EmployeeRec DO
  BEGIN
    Name := 'Charles A. Dixon';
    Address := '202 Greenwood, Gretna, LA 70088';
    Phone := '504-666-7574';
    Rate := 22.5;
    MaritalStatus := 'M'
  END;
```

Действие оператора WITH состоит в присоединении имени поля к имени записи. Если одна из переменных внутри блока не является полевой переменной, она не будет преобразована оператором WITH. Если за оператором WITH следует только один оператор, то, конечно, нет необходимости в блоке BEGIN-END.

Посредством оператора WITH можно вызывать процедуру, обрабатывающую поля записей, например:

```
WITH EmployeeRec DO
    DisplayResults(Name, Rate);
```

Этот оператор эквивалентен следующему:

```
DisplayResults(EmployeeRec.Name, EmployeeRec.Rate);
```

Основная форма оператора WITH:

```
WITH идентификатор-записи DO
    оператор;
```

Следующая программа демонстрирует алгоритм из примера 8-2, но разделена на три подпрограммы: GetData, DisplayInfo и DrawLine (которую вы написали ранее). Программа осуществляет вывод на экран в соответствующем формате, используя заголовок записи.

```
{------------------------------ Пример 8-3 ------------------------------}
PROGRAM RecordExample2(OUTPUT);
TYPE
  EmployeeRecord = RECORD
                     Name          :STRING[25];
                     Address       :STRING[40];
```

```
                    Phone         :STRING[12];
                    Rate          :REAL;
                    MaritalStatus :CHAR;
                END;
VAR
  EmployeeRec :EmployeeRecord;
{ ------------ Процедура DrawLine -------------- }
PROCEDURE DrawLine(LineLength, TabLength :INTEGER);
CONST
  Dash = '-';
VAR
  Counter :INTEGER;
BEGIN
  FOR Counter := 1 TO TabLength DO
    WRITE(' ');
  FOR Counter := 1 TO LineLength DO
    WRITE(Dash);
  WRITELN
END;
{ ------------- Процедура GetData -------------- }
PROCEDURE GetData(VAR Employee :EmployeeRecord);
{Name,Address,Phone,Rate,MaritalStatus);}
{ Присвоение значений полям }
BEGIN
  WITH Employee DO
    BEGIN
      Name := 'Diane J. Bedford';
      Address := '20 Carmen Avenue, New Orleans, LA 70112';
      Phone := '504-666-5043';
      Rate := 28.5;
      MaritalStatus := 'S'
    END
END;
{ ----------- Процедура DisplayInfo ------------ }
PROCEDURE DisplayInfo(Employee :EmployeeRecord);
{ Вывод на экран информации записи }
CONST
  Header ='Record of ';
VAR
  Len, Tab, Counter  :INTEGER;
  HeaderText, Status :STRING;
BEGIN
  WITH Employee DO
    BEGIN
      HeaderText := CONCAT(Header,Name);
      Len := LENGTH(HeaderText);
      Tab := (80-Len) DIV 2;
      DrawLine(Len, Tab);
      FOR Counter := 1 TO Tab DO
        WRITE(' ');
      WRITELN(HeaderText);
```

```
      DrawLine(Len, Tab);
      WRITELN('Адрес: ', Address);
      WRITELN('Телефон #: ', Phone);
      WRITELN('Почасовой тариф: $', Rate:0:2);
      IF MaritalStatus = 'M' THEN
        Status := 'Женат'
      ELSE
        Status := 'Одинокий';
      WRITELN('Семейное положение: ', Status)
    END
END;
{ --------------- Главная программа ----------------- }
BEGIN
  GetData(EmployeeRec);
  DisplayInfo(EmployeeRec)
END.
```

Вывод:

```
-------------------------
Record of Diane J. Bedford
-------------------------
Адрес:              20 Carmen Avenue, New Orleans, LA 70112
Телефон #:          504-666-5043
Почасовой тариф:    $28.50
Семейное положение: Одинокий
```

В этой программе надо обратить внимание на оператор WITH и передачу записи в подпрограмму в качестве параметра. Заметьте также, что запись передавалась один раз как параметр-переменная (с помощью VAR), когда она использовалась для возвращения значения поля, и один раз как параметр-значение, когда она была только приемником.

Реальная ценность такой программы проявляется, когда она читает информацию о сотруднике из файла, что мы вскоре рассмотрим.

8–5. Вложенные записи

В примере с записью EmployeeRecord можно разделить информацию поля адреса на название улицы, города, штата и почтовый индекс. Это означает, что поле адреса превращается в запись, вложенную в EmployeeRecord. Новая запись будет выглядеть так:

```
TYPE
  AddressRecord = RECORD
                    Street :STRING[18];
                    City   :STRING[15];
                    State  :STRING[2];
                    Zip    :String[5];
            END;
```

```
  EmployeeRecord = RECORD
                     Name          :STRING[25];
                     AddressRec    :AddressRecord;
                     Phone         :STRING[12];
                     Rate          :REAL;
                     MaritalStatus :CHAR;
             END;
VAR
  EmployeeRec :EmployeeRecord;
```

В этом объявлении два типа записи: AddressRecord и EmployeeRecord. Поле AddressRec в записи сотрудника имеет тип AddressRecord, который был определен ранее. Чтобы работать с полевой переменной в AddressRec, необходимо присоединить имена двух записей, EmployeeRec (которая является «дедушкой»), и AddressRec (которая является родителем). Несколько примеров присваивания:

```
EmployeeRec.AddressRec.Street := '15 Darell Street';
EmployeeRec.AddressRec.Zip := '60108';
```

При выводе на экран любого из этих полей применяется тот же метод:

```
WRITELN(EmployeeRec.AddressRec.Street);
WRITELN(EmployeeRec.AddressRec.City);
```

Законченная программа:

```
{ ------------------------------ Пример 8-4 ------------------------------ }
PROGRAM NestedRecord(OUTPUT);
TYPE
  AddressRecord = RECORD
                    Street :STRING[18];
                    City   :STRING[15];
                    State  :STRING[2];
                    Zip    :String[5];
             END;
  EmployeeRecord = RECORD
                     Name          :STRING[25];
                     AddressRec    :AddressRecord;
                     Phone         :STRING[12];
                     Rate          :REAL;
                     MaritalStatus :CHAR;
             END;
VAR
  EmployeeRec :EmployeeRecord;
BEGIN
  EmployeeRec.Name := 'Jean L. Krauss';
  EmployeeRec.AddressRec.Street := '15 Darell Street';
  EmployeeRec.AddressRec.City := 'Bloomingdale';
  EmployeeRec.AddressRec.State := 'IL';
  EmployeeRec.AddressRec.Zip := '60108';
  EmployeeRec.Phone := '312-987-5432';
```

```
  EmployeeRec.Rate := 27.5;
  EmployeeRec.MaritalStatus := 'M';
  WRITELN('Имя сотрудника: ', EmployeeRec.Name);
  WRITELN('Адрес: ', EmployeeRec.AddressRec.Street);
  WRITELN(' ', EmployeeRec.AddressRec.City);
  WRITE(' ', EmployeeRec.AddressRec.State);
  WRITELN(' ', EmployeeRec.AddressRec.Zip);
  WRITELN('Телефон #: ', EmployeeRec.Phone);
  WRITELN('Почасовой тариф: $', EmployeeRec.Rate:0:2);
  WRITELN('Семейное положение: ', EmployeeRec.MaritalStatus)
END.
```

Вывод:

```
Имя сотрудника:     Jean L. Krauss
Адрес:              15 Darell Street
                    Bloomingdale
                    IL 60108
Телефон #:          312-987-5432
Почасовой тариф:    $27.50
Семейное положение: M
```

Для того чтобы использовать оператор WITH с такой вложенной записью, необходимы два вложенных блока WITH, например:

```
WITH EmployeeRec DO
  WITH AddressRec DO
    BEGIN
      Name := 'Tammy M. Ockman';
      Street := '344 Temple Dr.';
      ...
    END;
```

Если какой-нибудь идентификатор поля принадлежит AddressRec, то он будет модифицировать и AddressRec, и EmployeeRec, но если он принадлежит непосредственно EmployeeRec, то он будет модифицировать только EmployeeRec. Если он представляет обычную переменную, то он не модифицирует ни одной записи.

Задание 8–3

Напишите полную программу, которая инициализирует и выводит на экран запись сотрудника, используя оператор WITH с вложенной записью адреса, показанной выше.

Заключение

В этой главе мы рассмотрели два структурных типа данных – множество и запись, и теперь:

1. Вы знаете, как объявлять множество определенного базового типа, используя форму:

```
идентификатор-типа = SET OF базовый-тип;
```

2. Вам также знакомы стандартные множественные операторы (объединение (+), пересечение (*) и разность (–)) и набор операторов отношений (= >=<=<>), и вы научились использовать эти операторы для работы с множествами.
3. Вам хорошо знакомы ограничения на множества, как и основные способы их применения в программировании.
4. Вы умеете объявлять типы записей, используя форму:

```
RECORD
   список-полей
END;
```

5. Вы можете получать доступ к полям с помощью полевых переменных или с помощью оператора WITH, который имеет вид:

```
WITH идентификатор-записи DO
   оператор;
```

6. Вы также знаете, как объявлять и использовать вложенные записи, а также как их обрабатывать.

Упражнения

1. Истинно или ложно:
   a. Множества, передаваемые в качестве параметров, должны быть объявлены в секции TYPE.
   b. Можно читать и запоминать записи с помощью операторов ввода и вывода.
   c. Можно объявить массив множеств.
   d. Главное различие между массивами и множествами в том, что множества могут содержать неупорядоченные элементы (члены).
   e. Главное различие между массивами и записями в том, что записи могут содержать элементы (поля) различных типов данных.
   f. Запись может содержать поля любого типа данных, за исключением типа RECORD.
   g. Доступ к элементам множеств и массивов может осуществляться по их относительному положению в множестве или массиве.
2. Даны следующие объявления:

```
VAR
  S1, S2           :SET OF CHAR;
  A, B, C, D, E, F :BOOLEAN;
```

Вычислите следующие выражения, предполагая, что S1 и S2 не пусты:

```
{a} A := S1*S2 <= S1;
{b} B := S1*S2 <= S2;
{c} C := S1+S2 >= S1;
{d} D := S1+S2 >= S2;
{e} E := S1-S2 <= S1+S2;
{f} F := S1-S2 >= S1+S2;
```

3. В следующих выражениях S1, S2 и S3 – целые, а S4 – символьная переменная.

```
{a} (S1 > 10) AND (S1 < 100);
{b} (S2=3) OR (S2=5) OR (S2=7);
{c} (S3 >= 2) AND (S3 < 9) OR (S3 = 11);
{d} (S4 >= 'a') AND (S4 <= 'z');
```

Напишите эквивалентные операторы, предполагая, что S1, S2, S3 и S4 это множества.

4. Даны следующие присвоения:

```
S1 := [1..9];
S2 := [1,3,5,7,9];
S3 := [2,4,6,8];
```

Вычислите следующие выражения:

```
{a} S2 + S3;
{b} S1 - S2;
{c} S1 - S3;
{d} S2 * S3;
{e} S3 * S1;
{f} S2 * S1;
```

5. Имеется следующий тип перечисления:

```
Days = (Mon, Tue, Wed, Thu, Fri, Sat, Sun);
```

Напишите фрагмент программы на Паскале, который делает следующее:

a. Объявляет множество DaySet базового типа Days.

b. Объявляет три переменные `WorkingDays`, `WeekEnd` и `WeekDays` базового типа DaySet.

c. Присваивает `WeekDays` все дни недели.

d. Присваивает Sat и Sun переменной `WeekEnd`.

e. Присваивает остальные дни переменной `WorkingDays`.

f. Проверяет следующие выражения (результат проверки приводится в виде комментария перед каждым выражением):

```
WeekDays = WorkingDays + WeekEnd; {TRUE}
WeekEnd = WeekDays - WorkingDays; {TRUE}
```

```
WorkingDays = WeekDays - WeekEnd; {TRUE}
WorkingDays * WeekEnd = [];       {TRUE}
Sat IN WorkingDays;               {FALSE}
Sun IN WeekEnd;                   {TRUE}
```

6. Напишите объявление записи, используя соответствующий тип данных для каждого из следующих элементов:
   a. Опись элементов, включающая имя, номер полки, инвентарный номер, имеющееся количество и цену.
   b. Запись сотрудника, которая включает имя, идентификационный номер, сетевой домен, псевдоним для электронной почты и номер кабинета.
   c. Запись сотрудника, которая включает те же поля, что и предыдущая запись сотрудника, а также дополнительную адресную запись. Адресная запись должна состоять из названия улицы, города, штата и почтового индекса.
   d. Запись новорожденного, которая включает имя, фамилию, имя отца, имя матери, вес, место рождения и дату рождения.
   e. Запись полета, которая состоит из авиалинии, номера рейса, времени прибытия, места вылета, времени вылета, места назначения.
   f. Запись книги, которая состоит из названия, имени автора, названия издательства и даты публикации.

Ответы

1. a. Истинно b. Ложно c. Истинно d. Истинно e. Истинно f. Ложно g. Ложно.
2. a. Истинно b. Истинно c. Истинно d. Истинно e. Истинно f. Ложно.
3. Эквивалентные множественные выражения:

```
{a} S1 := [11..99];
{b} S2 := [3,5,7];
{c} S3 := [2..8,11];
{d} S4 := ['a'..'z'];
```

4. a. S1 b. S3 c. S2 d. [] e. S3 f. S2
5. Требуемый код на Паскале:

```
TYPE
  Days   = (Mon, Tue, Wed, Thu, Fri, Sat, Sun);
  DaySet = SET OF Days;
VAR
  WorkingDays, WeekEnd, WeekDays :DaySet;
  A, B, C, D, E, F               :BOOLEAN;
```

```
BEGIN
  WorkingDays := [Mon..Fri];
  WeekEnd := [Sat..Sun];
  WeekDays := [Mon..Sun];
  A := WeekDays = WorkingDays + WeekEnd;
  B := WeekEnd = WeekDays - WorkingDays;
  C := WorkingDays = WeekDays - WeekEnd;
  D := WorkingDays * WeekEnd = [];
  E := Sat IN WorkingDays;
  F := Sun IN WeekEnd;
  WRITELN('A =',A);
  WRITELN('B =',B);
  WRITELN('C =',C);
  WRITELN('D =',D);
  WRITELN('E =',E);
  WRITELN('F =',F);
END.
```

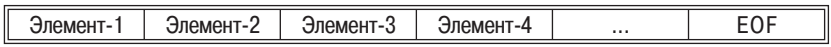
9

Файлы и приложения

9–1. Файлы данных

В предыдущей главе были рассмотрены различные структуры, предназначенные для хранения элементов данных, и теперь вы знаете, как организовать данные для максимально эффективной обработки. Однако если не сохранить каждый элемент данных на диске, то после выхода из программы данные исчезнут. Запись данных в файлы на диске обеспечивает возможность сохранить их и вызывать впоследствии для просмотра или дальнейшей обработки.

Тип FILE (являющийся структурным типом) определяется, в целом, как набор связанных элементов, записанных на диск или другой носитель в такой последовательности:



Элемент файла (также называемый файловым компонентом) может иметь простой или структурный тип, за исключением типа FILE.

Доступ к файлу может осуществляться с помощью любой из следующих операций:

- Чтение из файла (ввод)
- Запись в файл (вывод)

Файлы могут быть организованы как файлы *последовательного* доступа или как файлы *случайного* (*прямого*) доступа. В первом случае к элементу нельзя получить доступ, пока не будут прочитаны все предшествующие элементы. Примером файла последовательного доступа может служить список покупок, который приходится читать сверху вниз, чтобы найти определенную вещь. Файл случайного доступа организован подобно множеству почтовых ящиков, которые идентифицируются по номерам и доступны непосредственно, без необходимости заглядывать в каждый из них.

Несмотря на то что в стандартном Паскале разрешены только файлы последовательного доступа, многие реализации Паскаля (включая Турбо Паскаль и UCSD) поддерживают также и файлы случайного доступа. Здесь рассматриваются только файлы последовательного доступа.

9–2. Файлы типа TEXT

Стандартный Паскаль поддерживает два типа файлов – текстовые файлы и нетекстовые файлы (также называемые *двоичными* или *машинными* файлами).

Файл типа TEXT – это самая простая файловая структура, поскольку все ее элементы представляют собой символы (типа CHAR). Можно использовать стандартный файл INPUT (клавиатура) и стандартный файл OUTPUT (экран), которые относятся к TEXT-файлам. Файлы типа TEXT состоят из следующих друг за другом символьных строк, разделенных метками конца строки, и заканчиваются меткой конца файла, например:

```
Это текстовый файл. (EOLN)
Каждая строчка представляет собой последовательность символов. (EOLN)
Строки разделены метками конца строки. (EOLN)
Файл заканчивается меткой конца файла. (EOLN)
(EOF)
```

Файл на диске выглядит точно так же, как и файл на экране, введенный с клавиатуры. Символы в TEXT-файле хранятся в формате ASCII (в некоторых системах в EBCDIC), то есть если файл содержит, например, число 1234, то оно займет четыре байта, каждый из которых представляет собой ASCII-код цифры. Это не тот случай, когда число рассматривается как INTEGER и хранится в двоичном формате (0000010011010010) в двух байтах.

9–3. Чтение TEXT–файла

В примере 8-1 главы 8 рассматривалось чтение TEXT-файла с клавиатуры символ за символом и разбивка символов файла на группы. В данном разделе тот же самый алгоритм будет использоваться для чтения и анализа файла, предварительно сохраненного на диске. Чтобы заставить программу из примера 8-1 читать дисковый файл, в нее необходимо внести небольшие изменения. В процессе обсуждения этих изменений вы изучите протоколы, необходимые для извлечения информации из TEXT-файла.

Файловые переменные

Для того чтобы работать с TEXT-файлом, необходимо объявить *файловую переменную* типа TEXT. Если для нее выбрано имя «DiskFile», то объявление будет иметь вид:

```
VAR
  DiskFile :TEXT;
```

Файловые параметры

Для работы с файлом в стандартном Паскале надо указать файловую переменную (в нашем примере DiskFile) среди файловых параметров в заголовке программы (в других реализациях это не обязательно).

```
PROGRAM TextAnalyzer2(OUTPUT, DiskFile);
```

Здесь нет необходимости в параметре INPUT, поскольку никакие данные с клавиатуры вводиться не будут. Однако параметр OUTPUT нужен для отображения вывода на экране.

Открытие файла на ввод: RESET

Для того чтобы прочитать файл TEXT, его надо открыть с помощью процедуры RESET, например:

```
RESET(DiskFile);
```

Параметром процедуры является файловая переменная.

В Турбо Паскале необходимо использовать еще одну процедуру, ASSIGN, чтобы связать фактический файл данных на диске с файловой переменной. Прочитать текстовый файл C:\CONFIG.SYS (который уже существует в корневом каталоге диска C:)[1] можно, выполнив перед его открытием следующий оператор:

```
ASSIGN(DiskFile, 'C:\CONFIG.SYS');
```

Вместо файла CONFIG.SYS можно взять любой другой файл или создать новый текстовый файл в любом текстовом редакторе (таком как EDIT или EDLIN). В любом случае, если файл находится не в текущем каталоге или не на текущем диске, надо указать полное имя файла, как показано в предыдущем операторе (более подробно о полных именах рассказано в руководстве по вашей операционной системе).

[1] На вашей системе файл CONFIG.SYS может отсутствовать или располагаться по другому пути. – *Примеч. науч. ред.*

Поскольку программы этой книги компилировались с помощью компилятора Турбо Паскаля, то для открытия файлов были использованы следующие два оператора:

```
ASSIGN(DiskFile, 'CONFIG.SYS');
RESET(DiskFile);
```

Теперь файл CONFIG.SYS готов для ввода, и файловый указатель выставлен на первый элемент (символ) файла.

Некоторые реализации (такие как UCSD) связывают файловую переменную с именем файла в процедуре RESET, исключая таким образом необходимость в процедуре ASSIGN.

```
RESET(файловая-переменная, имя-файла);
```

Закрытие файла

По окончании чтения или записи в файл надо сделать последний шаг – закрыть файл с помощью процедуры CLOSE, иначе данные будут потеряны. В стандартном Паскале эта процедура отсутствует, и в ней нет необходимости, поскольку раньше использовались файлы на перфокартах и магнитных лентах.

Процедура CLOSE применяется для закрытия файла следующим образом:

```
CLOSE(DiskFile);
```

Некоторые версии включают в процедуру CLOSE больше параметров. Например, в UCSD файл закрывается оператором:

```
CLOSE(файловая-переменная, действие);
```

где «действие» заменяется или ключевым словом LOCK, если файл будет сохранен, или PURGE, если файл удаляется.

Подводя итог, приведем общие правила подготовки TEXT-файла к вводу:

- Заголовок программы:

  PROGRAM Program-name(список-файлов);

  (список-файлов в большинстве версий необязателен)

- Объявление файловой переменной:

  файловая-переменная :TEXT;

- Связывание файловой переменной с именем файла (только Турбо Паскаль):

  ASSIGN(файловая-переменная, имя-файла);

- Открытие файла на ввод:

  RESET(файловая-переменная);

- Закрытие файла (для всех версий, кроме стандартного Паскаля):

  CLOSE(файловая-переменная);
  CLOSE(файловая-переменная, действие); (UCSD)

Процедуры файлового ввода: READ, READLN

Операторы ввода/вывода, с которые мы имели дело до сих пор, в действительности представляют собой частный случай более общей формы. Полный вид процедуры READLN (или READ) следующий:

READLN(файловая-переменная, список-ввода);

READ(файловая-переменная, список-ввода);

Если файловая переменная не используется, то процедура принимает сокращенную форму, которую вы уже использовали:

READLN(список-ввода);

То же самое при использовании стандартного файла INPUT:

READLN(INPUT, список-ввода);

В нашем примере мы будем использовать файловую переменную `DiskFile`. Тогда оператор ввода примет вид:

```
READ(DiskFile,Ch);
```

где `Ch` — это символьная переменная, которую требуется прочитать.

Функции EOF и EOLN

Основная форма функции EOLN включает файловую переменную, например:

EOLN(файловая-переменная)

Если файловая переменная и скобки опущены, то предполагается стандартный файл INPUT (клавиатура).

То же самое относится к функции EOF:

EOF(файловая-переменная)

В примерах и заданиях этой главы встречаются текстовые файлы. В некоторых примерах файлы данных создаются, а в других читаются. Поэтому лучше изучать примеры последовательно. Выбирая примеры в произвольном порядке, убедитесь, что требуемый текстовый файл находится в текущем каталоге. Для удобства мы поместили все файлы данных в каталог примеров, дополнительно скопировав все файлы данных в каталог \TextFiles на прилагаемой дискете. Если вы случайно перезапишете один из файлов, то сможете скопировать его из этого каталога. При выполнении примера убедитесь, что и пример и используемые файлы данных находятся в текущем каталоге.

Пример: Анализатор дискового текстового файла

Итак, у вас в руках весь инструментарий для чтения файла, и вы можете протестировать следующую программу, которая предоставляет полный отчет по файлу CONFIG.SYS, находящемуся на прилагаемой дискете в том же каталоге, что и программа. Если файл CONFIG.SYS находится в корневом каталоге вашего диска, то замените оператор:

```
ASSIGN(DiskFile, 'CONFIG.SYS')
```

на оператор:

```
ASSIGN(DiskFile, 'C:\CONFIG.SYS')
```

[1]

Не ждите повторения результата, полученного при выполнении теста, поскольку разные компьютеры могут иметь разные конфигурационные файлы.

В соответствии с программой файл CONFIG.SYS распечатывается для проверки правильности отчета.

```
{ ------------------------------ Пример 9-1 ------------------------------ }
PROGRAM TextAnalyzer2(OUTPUT,DiskFile);
{ Чтение с диска текстового файла по одному символу }
TYPE
  LowerCase  = SET OF 'a'..'z';
  UpperCase  = SET OF 'A'..'Z';
  Digits     = SET OF '0'..'9';
  Characters = SET OF CHAR;
VAR
  DiskFile                    :TEXT; { Объявление переменной текстового файла }
  Capital                        :UpperCase;
  Small                          :LowerCase;
  Numerals                       :Digits;
  Alphabet, Punctuation, Others :Characters;
  A, C, S, N, P, O, Counter      :INTEGER;
  Ch                             :CHAR;
BEGIN
{ Привязка переменной к файлу 'CONFIG.SYS' в текущем каталоге }
  ASSIGN(DiskFile, 'CONFIG.SYS');
{ Открытие файла на ввод }
  RESET(DiskFile);
{ The program logic }
  Counter := 0; { счетчик всех символов }
  A := 0;       { счетчик алфавитных символов }
  C := 0;       { счетчик прописных букв }
  S := 0;       { счетчик маленьких букв }
  N := 0;       { счетчик числовых символов }
  P := 0;       { счетчик символов пунктуации }
```

[1] Проверьте перед этим наличие файла CONFIG.SYS по указанному пути. – *Примеч. науч. ред.*

```
  O := 0;       { счетчик остальных символов }
  Small := ['a'..'z'];
  Capital := ['A'..'Z'];
  Alphabet := Small + Capital;
  Numerals := ['0'..'9'];
  Punctuation := [',',';','-','''',',','!','?',')','(','"',':','_'];
{ Проверка конца дискового файла }
  WHILE NOT EOF(DiskFile) DO
    BEGIN
{ Проверка конца строки в дисковом файле}
      WHILE NOT EOLN(DiskFile) DO
        BEGIN
{ Чтение одного символа с дискового файла }
          READ(DiskFile,Ch);
          Counter := Counter + 1;
          IF Ch IN Alphabet THEN
            BEGIN
              A := A + 1;
              IF Ch IN Small THEN
                S := S + 1
              ELSE IF Ch IN Capital THEN
                C := C + 1
            END
          ELSE IF Ch IN Numerals THEN
            N := N + 1
          ELSE IF Ch IN Punctuation THEN
            P := P + 1
          ELSE
            O := O + 1
        END;
{ Перевод указателя на следующую строку }
      READLN(DiskFile)
    END;
{ Достигнут конец файла }
{ Закрытие файла }
  CLOSE(DiskFile);
{ Вывод отчета на экран }
  WRITELN;
  WRITELN('Общее количество символов = ', Counter);
  WRITELN('Количество алфавитных символов = ', A);
  WRITELN(' .Количество букв нижнего регистра: ', S);
  WRITELN(' .Количество букв верхнего регистра: ', C);
  WRITELN('Количество числовых символов = ', N);
  WRITELN('Количество символов пунктуации = ', P);
  WRITELN('Количество остальных символов = ', O);
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Распечатка файла CONFIG.SYS:

```
DEVICE=C:\SCSI\ASPI2DOS.SYS /D /Z /P140
DEVICE=C:\SCSI\ASPICD.SYS /D:ASPICD0
```

```
device=C:\WINDOWS\himem.sys
[common]
DEVICE=C:\CDROM\AOATAPI.SYS /D:IDECD000
```

Вывод программы:

```
Общее количество символов = 149
Количество алфавитных символов = 107
 .Количество букв нижнего регистра: 20
 .Количество букв верхнего регистра: 87
Количество числовых символов = 8
Количество символов пунктуации = 10
Количество остальных символов = 24
Для продолжения нажмите ENTER..
```

9–4. Вывод TEXT–файла на экран

Можно вывести на экран содержимое TEXT-файла по алгоритму предыдущей программы, добавляя после каждого чтения оператор WRITE:

```
READLN(DiskFile,Ch);
WRITE(Ch);
```

Кроме того, с помощью оператора WRITELN надо пропускать на экране одну строку всякий раз, когда встречается EOLN, иначе отдельные линии объединятся вместе.

Ниже приведена программа, которая читает тот же файл (CONFIG.SYS). Имя файла объявлено как константа, и его можно заменить на любое другое. Можно также использовать исходный файл самой программы (на прилагаемой дискете он называется 9-02.PAS), тогда программа прочитает сама себя.

```
{ ------------------------------ Пример 9-2 ------------------------------ }
PROGRAM ReadTextFile(INPUT,OUTPUT,DiskFile);
{ Чтение текстового файла, записанного на диске }
CONST
{ Можно заменить следующую константу именем любого существующего файла }
  FileName = 'C:\CONFIG.SYS';
VAR
  DiskFile :TEXT;
  Ch       :CHAR;
BEGIN
  ASSIGN(DiskFile, FileName);
  RESET(DiskFile);
  WHILE NOT EOF(DiskFile) DO
    BEGIN
      WHILE NOT EOLN(DiskFile) DO
        BEGIN
{ Чтение и вывод на экран одного символа из текстового файла }
          READ(DiskFile,Ch);
```

```
          WRITE(Ch)
        END;
{ Перевод указателя на следующую строку }
      READLN(DiskFile);
{ Пропуск на экране одной линии }
      WRITELN
    END;
  CLOSE(DiskFile);
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Вывод может выглядеть примерно так:

```
DEVICE=C:\SCSI\ASPI2DOS.SYS /D /Z /P140
DEVICE=C:\SCSI\ASPICD.SYS /D:ASPICD0
device=C:\WINDOWS\himem.sys
[common]
DEVICE=C:\CDROM\AOATAPI.SYS /D:IDECD000
Для продолжения нажмите ENTER..
```

Чтение TEXT-файла как множества строк

Если ваша версия Паскаля поддерживает тип STRING, то TEXT-файл можно прочитать по одной строчке.

Следующая программа работает с файлом, построенным построчно, а не посимвольно. Каждая длина имеет максимальную длину 80 символов, что является предполагаемой длиной строки. После чтения каждой строки указатель перемещается на следующую строку. Если какая-нибудь строка содержит менее 80 символов, в качестве динамической длины строки будет выбрана ее реальная длина. Если, напротив, строка содержит более 80 символов, то лишние символы будут проигнорированы. После старта программа просит ввести имя файла, который требуется вывести на экран, то есть программа работает подобно DOS-команде TYPE.

```
{ ------------------------------ Пример 9-3 ------------------------------ }
PROGRAM DisplayTextFile(OUTPUT,MyFile);
{ Чтение текстового файла, записанного на диск построчно }
VAR
  MyFile            :TEXT;
  OneLine, FileName :STRING[80];
BEGIN
  WRITE('Пожалуйста, введите имя файла, который надо вывести на экран: ');
  READLN(FileName);
  WRITELN;
  WRITELN('Содержимое файла ',FileName,' : ');
  ASSIGN(MyFile, FileName);
  RESET(MyFile);
{ Проверка конца текстового файла }
```

```
  WHILE NOT EOF(MyFile) DO
    BEGIN
{ Чтение и вывод на экран текстового файла по одной строке }
      READLN(MyFile,OneLine);
      WRITELN(OneLine);
    END;
  CLOSE(MyFile);
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Если файл не существует или его имя написано неверно, программа выдаст сообщение об ошибке, например:

```
Пожалуйста, введите имя файла, который надо вывести на экран: CNFIG.SYS
Содержимое файла CNFIG.SYS :
Runtime error 002 at 0000:00F2.[1]
```

Обратите внимание, что каждая строка была прочитана оператором READLN. При использовании оператора READ понадобился бы еще оператор READLN – для пропуска метки конца строки, которой завершается каждая строчка, и перемещения файлового указателя на начало следующей строки. Дело в том, что оператор READ читает символьную строку до метки конца строки, а затем останавливается. К тому же он не перемещает файловый указатель.

В этой программе можно проверять EOLN после чтения каждой строки, но необходимости в этом нет.

Чтение нескольких строк

Одним оператором READLN (READ) можно прочитать более одной строки, но это не всегда надежно. Чтобы понять возможные ловушки, взгляните на этот пример, читающий из текстового файла test.txt три строки, каждая из которых объявлена как STRING[5]. Файл содержит строку:

```
This is a test text file.
{ ------------------------------ Пример 9-4 ------------------------------ }
PROGRAM ReadMultipleStrings1(OUTPUT,F);
VAR
  F              :TEXT;
  Str1,Str2,Str3 :STRING[5];
BEGIN
  ASSIGN(F,'test.txt');
  RESET(F);
  READLN(F,Str1,Str2,Str3);
```

[1] Данное сообщение об ошибке зависит от версии компилятора. – *Примеч. науч. ред.*

```
  WRITELN('Str1= ', Str1);
  WRITELN('Str2= ', Str2);
  WRITELN('Str3= ', Str3);
  CLOSE(F);
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Вывод:

```
Str1= This
Str2= is a
Str3= test
```

Как видите, каждой строковой переменной присвоено пять символов (включая пробел). Теперь замените объявление строковой переменной на следующее:

```
Str1,Str2,Str3 :STRING;
```

После старта программы с таким объявлением длина каждой строки по умолчанию будет равна максимальной длине, поддерживаемой языком, и вы получите следующий результат:

```
Str1= This is a test text file.
Str2=
Str3=
```

Первой переменной была присвоена вся строка (вплоть до метки конца строки), а остальным двум ничего не осталось. Короче говоря, несколько строк можно читать без опасений, только если известна длина каждой из них.

9–5. Создание TEXT–файла: REWRITE

Для создания файла необходимо открыть файл на вывод. Этой цели служит процедура REWRITE (парная процедуре RESET). Она имеет вид:

```
REWRITE(файловая-переменная);
```

В Турбо Паскале необходимо связать файловую переменную с именем реального файла на диске посредством процедуры ASSIGN, как мы уже делали с входным файлом.

Некоторые реализации (такие как UCSD) вместо этого используют модифицированную форму процедуры REWRITE, где присутствуют и файловая переменная и имя файла:

```
REWRITE(файловая-переменная, имя-файла);    { UCSD }
```

Правила образования имени файла (действительного имени дискового файла) зависят от операционной системы. В DOS имя файла может иметь до восьми символов, а необязательное расширение – до трех (например, EMPLOYEE.DAT). В результате выполнения такого оператора пустой файл будет открыт и готов для записи.

Если открыть на запись существующий файл, то данные в нем будут потеряны и перезаписаны новыми данными.

Процедуры файлового вывода: WRITE, WRITELN

Для записи одного или нескольких элементов в файл используйте основную форму процедуры WRITELN (или WRITE):

WRITELN(файловая-переменная, список-вывода);

или

WRITE(файловая-переменная, список-вывода);

Если файловая переменная опущена, то подразумевается стандартный файл OUTPUT, а форма процедуры сокращается до той, которую вы уже использовали:

WRITELN(список-вывода);

что эквивалентно:

WRITELN(OUTPUT, список-вывода);

Как упоминалось ранее, по окончании записи дисковый файл необходимо закрыть с помощью процедуры CLOSE.

В следующем примере создается файл HELLO.TXT, а затем в него записывается константа «Привет, Паскаль».

```
{ ------------------------------ Пример 9-5 ---------------------------- }
PROGRAM CreateFile(F);
CONST
  TestSentence = 'Привет, Паскаль';
VAR
  F :TEXT;
BEGIN
  ASSIGN(F, 'HELLO.TXT'); { Только Турбо Паскаль }
  REWRITE(F); { открытие файла на запись }
  WRITELN(F, TestSentence);
  CLOSE(F)
END.
```

После выполнения этой программы в текущий каталог добавится файл HELLO.TXT. Для того чтобы убедиться в правильности записи

файла, можно вывести его на экран с помощью команды TYPE из MS DOS или с помощью примера 9-3 (который ее заменяет). В любом случае на экране появятся два слова: «Привет, Паскаль».

Как упоминалось ранее, текстовый файл может быть создан и записан с помощью любого текстового редактора, но создание файла в программе на Паскале становится важным, когда информация для нового файла основана на обработке данных других файлов.

Задание 9-1

Напишите программу, которая принимает с клавиатуры имя и/или ID-номер и рабочие часы за месяц для каждого сотрудника и записывает данные в файл с именем TIMSHEET.TXT. Программа должна обрабатывать данные любого количества сотрудников.

Пример: Файл сотрудника

В главе 8 мы создали запись сотрудника для хранения информации об имени, адресе, зарплате и т. д. каждого сотрудника. В следующей программе информация записи сотрудника сохраняется в файле EMPFILE.TXT с использованием структуры вложенной записи. Взгляните:

```
{ ------------------------------ Пример 9-6 ---------------------------- }
PROGRAM CreateEmpFile(INPUT,OUTPUT,F);
TYPE
  AddressRecord = RECORD
                    Street :STRING[18];
                    City   :STRING[15];
                    State  :STRING[2];
                    Zip    :String[5];
                  END;
  EmployeeRecord = RECORD
                     ID            :INTEGER;
                     Name          :STRING[20];
                     AddressRec    :AddressRecord;
                     Phone         :STRING[12];
                     Rate          :REAL;
                     MaritalStatus :CHAR;
                   END;
VAR
  F           :TEXT; { Файловая переменная }
  EmployeeRec :EmployeeRecord;
BEGIN
  ASSIGN(F, 'EMPFILE.TXT');
  REWRITE(F);
  WITH EmployeeRec DO
    WITH AddressRec DO
      BEGIN
        WRITE('Пожалуйста, введите ID сотрудника: '); READLN(ID);
```

```
        WRITE('Имя сотрудника: ');              READLN(Name);
        WRITE('Адрес: Улица:');                 READLN(Street);
        WRITE(' Город: ');                      READLN(City);
        WRITE(' Штат: ');                       READLN(State);
        WRITE(' Почтовый индекс: ');            READLN(Zip);
        WRITE('Номер телефона: ');              READLN(Phone);
        WRITE('Почасовой тариф: ');             READLN(Rate);
        WRITE('Семейное положение (S/M): ');    READLN(MaritalStatus);
{ Запись информации в файл }
        WRITELN(F, ID);
        WRITELN(F, Name);
        WRITELN(F, Street);
        WRITELN(F, City);
        WRITELN(F, State);
        WRITELN(F, Zip);
        WRITELN(F, Phone);
        WRITELN(F, Rate:0:2);
        WRITELN(F, MaritalStatus)
      END;
  CLOSE(F)
END.
```

Выполнение теста:

```
Пожалуйста, введите ID сотрудника: 122
Имя сотрудника: Tammy M. Ockman
Адрес:      Улица: 322 Temple Dr.
            Город: New Orleans
             Штат: LA
  Почтовый индекс: 70112
Номер телефона: 504-285-3434
Почасовой тариф: 22.45
Семейное положение (S/M): S
```

Далее следует вывод содержимого файла на экран:

```
122
Tammy M. Ockman
322 Temple Dr.
New Orleans
LA
70112
504-285-3434
22.45
S
```

Обратите внимание, что числовое поле ID, добавленное в запись, отличается от предыдущего (см. главу 8). Повторим, что если компьютер не поддерживает тип STRING (что маловероятно), то можно использовать только числовые или символьные поля. Результирующий файл содержит столько строк, сколько полей в записи. При желании можно записать все поля в одной строке, заменив все WRITELN на WRITE.

Задание 9–2

Измените предыдущую программу так, чтобы она могла сохранять более одной записи сотрудника. Ее можно построить в виде процедуры, вызываемой для каждого ввода данных одного сотрудника.

Пример: Платежная ведомость

Только что созданный файл содержит достаточный объем информации о сотрудниках и может быть использован для других целей. На основе информации из этого файла (всей или частично) можно создавать различные отчеты или другие файлы данных. Следующее приложение читает файл EMPFILE.TXT, но из каждой записи используются только три поля: ID, Name и HourlyRate. Сначала программа выводит на экран информацию о сотруднике, затем просит пользователя ввести HoursWorked для этого сотрудника. Затем путем перемножения HourlyRate и HoursWorked вычисляется зарплата. После обработки каждой записи ID, Name и Wages сохраняются в новом файле PAYFILE.TXT, на основе которого генерируется отчет за этот период выплат.

```
{ ------------------------------ Пример 9-7 ---------------------------- }
PROGRAM PayRoll(INPUT,OUTPUT,MasterFile,PayFile);
{ Эта программа читает файл EMPFILE.TXT, вычисляет зарплату и сохраняет
информацию в файле PAYFILE.TXT }
TYPE
  AddressRecord = RECORD
                    Street :STRING[18];
                    City   :STRING[15];
                    State  :STRING[2];
                    Zip    :STRING[5];
                  END;
  EmployeeRecord = RECORD
                     ID            :INTEGER;
                     Name          :STRING[20];
                     AddressRec    :AddressRecord;
                     Phone         :STRING[12];
                     Rate          :REAL;
                     MaritalStatus :CHAR;
                   END;
  PayRecord = RECORD
                ID    :INTEGER;
                Name  :STRING[20];
                Wages :REAL;
              END;
VAR
  MasterFile, PayFile :TEXT;
  EmployeeRec         :EmployeeRecord;
  PayRec              :PayRecord;
```

```
  HoursWorked, Wages  :REAL;
{ --------------- Процедура GetInfo ------------------ }
{ Эта процедура читает файл сотрудника EMPFILE.TXT и выводит на экран ID, имя
и почасовой тариф. }
PROCEDURE GetInfo(VAR F:TEXT);
BEGIN
  WITH EmployeeRec DO
    WITH AddressRec DO
      BEGIN
        READLN(F,ID); WRITELN('ID: ',ID);
        READLN(F,Name); WRITELN('Имя: ',Name);
        READLN(F,Street);
        READLN(F,City);
        READLN(F,State);
        READLN(F,Zip);
        READLN(F,Phone);
        READLN(F,Rate); WRITELN('Почасовой тариф: $', Rate:0:2);
        READLN(F,MaritalStatus);
      END;
END;
{ --------------- Процедура CalcWages ---------------- }
{ Эта процедура вычисляет зарплату. Результат возвращается в главную
программу }
PROCEDURE CalcWages(HoursWorked:REAL; VAR Wages:REAL);
BEGIN
  WITH EmployeeRec DO
    Wages := Hoursworked * Rate;
  Wages := ROUND(100 * Wages) / 100 { rounding cents }
END;
{ -------------- Процедура FilePayRoll --------------- }
{ Эта процедура записывает одну запись в выходной файл PAYFILE.TXT }
PROCEDURE FilePayRoll(VAR P :TEXT; Wages :REAL);
BEGIN
  WITH EmployeeRec DO
    BEGIN
      PayRec.ID := ID;
      PayRec.Name := Name;
      Payrec.Wages := Wages
    END;
  WITH PayRec DO
  WRITELN(P, ID:3, Name:20, Wages:10:2)
END;
{ --------------- Главная программа ------------------ }
BEGIN
  ASSIGN(MasterFile, 'EMPFILE.TXT'); RESET(MasterFile);
  ASSIGN(Payfile, 'PAYFILE.TXT'); REWRITE(PayFile);
  WHILE NOT EOF(MasterFile) DO
    BEGIN
      GetInfo(MasterFile);
      WRITE('Пожалуйста, введите рабочие часы за этот период оплаты: ');
      READLN(HoursWorked);
      CalcWages(HoursWorked, Wages);
```

```
        FilePayRoll(PayFile, Wages)
      END;
    CLOSE(MasterFile);
    CLOSE(PayFile)
  END.
```

Пример выполнения:

Предполагаем, что файл EMPFILE.TXT содержит три записи, с которыми и будет работать программа:

```
ID: 122                                                  ----> Информация из файла
Имя: Tammy M. Ockman                                     ----> Информация из файла
Почасовой тариф: $22.45                                  ----> Информация из файла
Пожалуйста, введите рабочие часы за этот период оплаты: 160  ----> Введено
                                                                   пользователем
ID: 123
Имя: Tara S. Strahan
Почасовой тариф: $15.24
Пожалуйста, введите рабочие часы за этот период оплаты: 160
ID: 125
Имя: John G. Trainer
Почасовой тариф: $28.55
Пожалуйста, введите рабочие часы за этот период оплаты: 140.5
```

Программа создает файл PAYFILE.TXT, содержащий следующие записи:

```
122      Tammy M. Ockman    3592.00
123      Tara S. Strahan    2438.40
125      John G. Trainer    4011.28
```

Программа состоит из трех процедур:

1. GetInfo, читающей одну запись из файла EMPFILE.TXT и выводящей на экран только выбранные поля. Заметьте, что необходимо прочитать все поля записи, даже если все они вам не нужны.
2. CalcWages, производящей вычисления.
3. FilePayRoll, сохраняющей запись PayRec в файле PAYFILE.TXT.

Некоторые важные моменты в этой программе:

- Когда файловые переменные (такие как MasterFile и PayFile) передаются в подпрограмму, они должны быть переданы как параметры-переменные (с помощью VAR). Тип TEXT используется с такими параметрами:

```
PROCEDURE FilePayRoll(VAR P :TEXT; Wages :REAL);
```

- Некоторые идентификаторы (например, Name и ID) фигурируют и в EmployeeRec и в PayRec. Это не вызывает конфликтов, поскольку это все – полевые переменные; помните, что область видимости полевых переменных ограничена их собственной записью. Кроме того, идентификатор Wages был объявлен и как глобальная перемен-

ная, и как полевая переменная (в записи PayRec) и использовался также как локальная переменная в процедуре FilePayroll.

- Взгляните на эти операторы присваивания в процедуре FilePayRoll:

```
WITH EmployeeRec DO
  BEGIN
    PayRec.ID := ID;
    PayRec.Name := Name;
    PayRec.Wages := Wages
```

Первые два оператора копируют значения полей ID и Name из EmployeeRec в соответствующие поля в PayRec. Оператор WITH модифицирует только переменные, принадлежащие записи EmployeeRec (ID и Name). А, например, на переменную `PayRec.ID` оператор WITH не оказывает влияния, поскольку ее явно модифицирует PayRec. В последнем операторе никакие переменные не подверглись изменению со стороны оператора WITH.

Задание 9–3

Добавьте в последнюю программу процедуру, которая выводит на экран сводный отчет платежной ведомости, как показано ниже:

```
----- Сводка платежной ведомости ------
ID ----------- Имя ---------- Оклад
122     Tammy M. Ockman     $3592.00
123     Tara S. Strahan     $2438.40
125     John G. Trainer     $4011.28
-------------------------------------
```

Можно, кроме того, изменить программу так, чтобы она читала HoursWorked из файла TIMSHEET.EMP, который создан в задании 9-1.

9–6. Нетекстовые файлы

TEXT-файл – это предопределенный тип файла, но, как упоминалось ранее, основное определение файла позволяет компонентам файла иметь любой тип, отличный от типа FILE.[1] Можно объявить файл любого предопределенного или определенного пользователем типа, используя форму:

идентификатор-типа = FILE OF тип-компонента;

Тип компонента может быть простым типом (таким как INTEGER), структурным типом (подобно массиву) или типом, определенным пользователем (например, запись).

[1] Такой файл иногда называют типизированным. – *Примеч. науч. ред.*

Приведем пример объявления файла, компонентами которого являются записи (здесь используется упрощенная форма EmployeeRecord, чтобы сделать программу короче):

```
TYPE
  EmployeeRecord = RECORD
                     ID   :INTEGER;
                     Name :STRING[20];
                     Rate :REAL;
                   END;
  EmpFileRec = FILE OF EmployeeRecord;
VAR
  F           :EmpFileRec;          { файловая переменная }
  EmployeeRec :EmployeeRecord;      { переменная типа запись }
```

Основные свойства нетекстовых файлов:

- Данные представлены во внутреннем двоичном формате. Это означает, что нельзя вывести на экран содержимое файла с помощью DOS-команды TYPE. Еще это ускоряет запись в файл и чтение из файла.
- Основное достоинство нетекстового файла проявляется при использовании структурных типов данных, таких как массивы или записи, поскольку они позволяют избежать ввода и вывода записи поле за полем. Например, после предыдущих объявлений вы можете читать и сохранять всю запись целиком, с помощью операторов:

```
READ(F, EmployeeRec);
WRITE(F, EmployeeRec);
```

- Поскольку нетекстовые файлы не состоят из строк, как TEXT-файлы, то процедуры READLN и WRITELN нельзя использовать с этими файлами.

Пример: Система расчета зарплаты

Это та же программа платежной ведомости, только лучше выглядит. Программа разделена на два самостоятельных модуля (программы). Первый модуль (пример 9-8) читает записи сотрудников с клавиатуры и сохраняет их в нетекстовом файле EMPFILE.BIN. Во втором модуле (пример 9-9) HoursWorked вводится с клавиатуры, а зарплата вычисляется и сохраняется в файле PAYFILE.TXT, который является TEXT-файлом. Первая программа может быть выполнена только однажды – для создания файла сотрудника, а вторая – каждый платежный период для создания PayFile. Ниже приводится программа:

```
{ ------------------------------ Пример 9-8 ------------------------------ }
PROGRAM EmpPayInfo(INPUT,OUTPUT,F);
{ Эта программа создает определенный пользователем файл EMPFILE.BIN,
элементами которого являются записи. }
TYPE
  EmployeeRecord = RECORD
                     ID   :INTEGER;
```

```
                    Name :STRING[20];
                    Rate :REAL;
                  END;
  EmpFileRec = FILE OF EmployeeRecord;
VAR
  F           :EmpFileRec;              { Файловая переменная }
  EmployeeRec :EmployeeRecord;
{ ----------- Процедура WriteRecord ---------- }
PROCEDURE WriteRecord;
BEGIN
{ Сохранение одной записи в файле }
  WRITE(F, EmployeeRec)
END;
{ ----------- Процедура GetData ---------- }
PROCEDURE getdata;
VAR
  Counter :INTEGER;
BEGIN
  Counter := 0;
  WITH EmployeeRec DO
    BEGIN
      WRITE('Пожалуйста, введите ID сотрудника (или 0 для завершения):');
      READLN(ID);
      WHILE ID <> 0 DO
        BEGIN
          Counter := Counter + 1;
          WRITE('Имя сотрудника: ');   READLN(Name);
          WRITE('Почасовой тариф: ');  READLN(Rate);
          WriteRecord;
          WRITE('Пожалуйста, введите ID сотрудника (или 0 для завершения):');
          READLN(ID)
        END
    END;
  WRITELN(Counter, ' Записи сотрудника занесены в файл.')
END;
{ ---------------- Главная программа -------------- }
{ Главная программа }
BEGIN
  ASSIGN(F, 'EMPFILE.BIN'); REWRITE(F);
  GetData;
  CLOSE(F);
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Второй модуль (PayRoll2) состоит из четырех процедур:

- GetInfo, которая читает запись из файла EMPFILE.BIN.
- CalcWages, выполняющей вычисления.
- FilePayRoll, сохраняющей запись в файле PAYFILE.TXT.

- **ReadPayRoll, читающей файл PAYFILE.TXT и выводящей на экран платежную ведомость в конце работы.**

```
{ ------------------------------ Пример 9-9 -------------------------- }
PROGRAM PayRoll2(INPUT,OUTPUT,MasterFile,PayFile);
{ Эта программа читает файл EMPFILE.BIN по одной записи, затем
вычисляет зарплату и сохраняет вывод в текстовом файле PAYFILE.TXT }
TYPE
  EmployeeRecord = RECORD
                     ID   :INTEGER;
                     Name :STRING[20];
                     Rate :REAL;
                   END;
  PayRecord = RECORD
                ID    :INTEGER;
                Name  :STRING[20];
                Wages :REAL;
              END;
  EmployeeFile = FILE OF EmployeeRecord;
VAR
  MasterFile         :EmployeeFile;
  PayFile            :TEXT;
  EmployeeRec        :EmployeeRecord;
  PayRec             :PayRecord;
  HoursWorked, Wages :REAL;
{ --------------- Процедура GetInfo ------------------ }
{ Эта процедура читает и выводит на экран запись из файла EMPFILE.BIN }
PROCEDURE GetInfo(VAR F :EmployeeFile);
BEGIN
  READ(F,EmployeeRec);
  WITH EmployeeRec DO
    BEGIN
      WRITELN('ID: ',ID);
      WRITELN('Имя: ',Name);
      WRITELN('Почасовой тариф: $', Rate:0:2);
    END;
END;
{ --------------- Процедура CalcWages ---------------- }
PROCEDURE CalcWages(HoursWorked :REAL; VAR Wages :REAL);
BEGIN
  WITH EmployeeRec DO
    Wages := Hoursworked * Rate;
  Wages := ROUND(100 * Wages) / 100       { округление центов }
END;
{ -------------- Процедура FilePayRoll --------------- }
{ Эта процедура сохраняет запись в PAYFILE.TXT }
PROCEDURE FilePayRoll(VAR P :TEXT; Wages :REAL);
BEGIN
  WITH EmployeeRec DO
    BEGIN
      PayRec.ID := ID;
      PayRec.Name := Name;
```

```
      Payrec.Wages := Wages
    END;
  WITH PayRec DO
    BEGIN
      WRITELN(P, ID);
      WRITELN(P, Name);
      WRITELN(P, Wages);
    END;
END;
{ -------------- Процедура ReadPayRoll --------------- }
{ Эта процедура читает и выводит на экран PAYFILE.TXT }
PROCEDURE ReadPayRoll(VAR P:TEXT);
VAR
  I :INTEGER;
BEGIN
  WITH PayRec DO
    BEGIN
      READLN(P, ID);
      READLN(P, Name);
      READLN(P, Wages);
      WRITE(ID:3,' ');
      WRITE(Name);
{ Заполнение остальных 20 позиций пробелами }
      FOR I := 1 TO 20-LENGTH(Name) DO
        WRITE(' ');
      WRITELN(' $',Wages:0:2)
    END;
END;
{ --------------- Главная программа ------------------ }
BEGIN
  ASSIGN(MasterFile, 'EMPFILE.BIN'); RESET(MasterFile);
  ASSIGN(Payfile, 'PAYFILE.TXT');    REWRITE(PayFile);
  WHILE NOT EOF(MasterFile) DO
    BEGIN
      GetInfo(MasterFile);
      WRITE('Пожалуйста, введите рабочие часы за этот период оплаты: ');
      READLN(HoursWorked);
      CalcWages(HoursWorked, Wages);
      FilePayRoll(PayFile, Wages)
    END;
  CLOSE(MasterFile);
  CLOSE(PayFile);
  RESET(PayFile);
  WRITELN('--------- Сводка платежной ведомости --------- ');
  WRITELN('ID # ------- Имя -------- Оклад');
  WHILE NOT EOF(PayFile) DO
    ReadPayroll(PayFile);
  WRITELN('----------------------------------- ');
  CLOSE(PayFile);
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Присоединение к файлу

Если потребуется добавить информацию о новом сотруднике в файл EMPFILE.BIN, то нельзя будет снова запустить пример 9-8, поскольку он удалит весь файл. Для этого есть другой путь.

Добавление данных в существующий файл называется *присоединением*, поскольку новые данные записываются в конец последовательного файла. В некоторых реализациях (включая Турбо Паскаль) файл может быть открыт для присоединения с помощью процедуры APPEND, которая имеет вид:

```
APPEND(файловая-переменная);
```

В то время как процедура REWRITE устанавливает файловый указатель в начало файла, APPEND позиционирует файловый указатель в конец файла, поэтому любые новые данные будут записаны сюда. Если ваша реализация не содержит процедуру APPEND, то для добавления новых элементов в файл придется применить следующие приемы:

- Открыть файл EMPFILE.BIN на чтение с помощью RESET.
- Открыть временный файл (то есть NEWFILE.TMP) на запись с помощью REWRITE.
- Скопировать каждый элемент файла EMPFILE.BIN в NEWFILE.TMP, затем принять с клавиатуры новые данные и записать их в NEWFILE.TMP.
- Открыть NEWFILE.TMP на чтение и EMPFILE.BIN на запись, затем скопировать содержимое файла NEWFILE.TMP обратно в EMPFILE.BIN.
- Удалить временный файл NEWFILE.TMP.

В стандартном Паскале, если файловая переменная не включена в заголовок программы, то файл считается *временным* и будет удален сразу после завершения программы. В UCSD вы получите тот же результат, если закроете файл с помощью ключевого слова PURGE.

В Турбо Паскале можно удалить файл после его закрытия, используя процедуру ERASE, которая имеет вид:

```
ERASE(файловая-переменная);
```

Если информация в файле должна быть изменена (как в случае повышения зарплаты сотрудников), то подобный алгоритм можно использовать для обновления последовательного файла, что будет показано позже в следующей главе.

Задание 9-4

Взяв за основу приложение платежной ведомости, напишите программу, объединяющую в одном меню все инструменты, изученные вами к этому моменту. Меню должно содержать следующие опции:

- Вывести на экран файл сотрудника.
- Вывести на экран запись сотрудника.
- Добавить нового сотрудника.

В этой программе может быть использована следующая процедура Menu:

```
{ --------------- Процедура Menu ------------------ }
PROCEDURE Menu;
VAR
  Option :INTEGER;
BEGIN
  WRITELN(Header);
  WRITELN;
  WRITELN('1. Вывести на экран файл сотрудника.');
  WRITELN('2. Вывести на экран запись сотрудника.');
  WRITELN('3. Добавить нового сотрудника.');
  WRITELN('4. Выход.');
  WRITELN(Separator);
  WRITE('Сделайте выбор и нажмите цифру: ');
  READLN(Option);
  CASE Option OF
     1 : Readit(DbFile);
     2 : ReadRec(DbFile, EmployeeRec);
     3 : AddRec(NewFile, DbFile, EmployeeRec);
     4 : Exit
  END;
  Menu
END;
```

Как видите, в процедуре Menu опции 1 и 3 соответствуют процедурам, которые вы должны разработать. Четвертую опцию можно реализовать посредством процедур Турбо Паскаля EXIT, GOTO или любого подходящего оператора в вашем компиляторе, который позволяет выходить из меню. Заметьте, что в этом примере для добавления нового сотрудника в файл (опция 3) создавался временный файл NewFile, но если в вашем компиляторе есть процедура APPEND, обратитесь к ней – это поможет вам сэкономить силы. Эта программа составляет ядро базы данных и может быть изменена для включения новых возможностей, таких как добавление информации о сотруднике и удаление ненужных записей из базы данных.

9–7. Использование переменной файлового буфера

В стандартном Паскале при объявлении файла автоматически создается *файловое окно* или *переменная файлового буфера*. Этот буфер помогает компенсировать дисковые транзакции ввода/вывода, которые занимают значительное время по сравнению с транзакциями в памяти. Файловое окно представляет собой область памяти, предназначенную для хранения элементов файла и переноса их на внешний носитель (диск) на время выполнения других операторов программы. Однако в настоящее время на современном оборудовании эта проблема практически не актуальна.

Так, если объявить файловую переменную DiskFile, то будет создана следующая переменная файлового буфера:

```
DiskFile^
```

Для работы с этим буфером надо вызывать предопределенную процедуру GET для чтения из файла и PUT для записи в файл. Эти процедуры применяются вместо READ и WRITE. Рассмотрим следующие объявления:

```
VAR
DiskFile: TEXT;
Ch: CHAR;
```

Оператор WRITE(DiskFile, Ch), применяемый для записи данных в файл, эквивалентен следующим операторам:

```
DiskFile^ := Ch;
PUT(DiskFile)
```

Первый оператор копирует данные, хранимые в Ch, в буфер, а второй оператор перемещает данные из буфера в файловую переменную.

Аналогично оператор READ(DiskFile, Ch), читающий данные из файла, эквивалентен следующим операторам:

```
Ch := DiskFile^;
GET(DiskFile)
```

Первый оператор копирует значение буфера (который всегда содержит следующий компонент файла) в переменную Ch, тогда как второй оператор читает следующий компонент файла в файловую переменную.

Можете протестировать любой из предыдущих примеров, используя процедуры PUT и GET. Однако помните, что некоторые версии Паскаля, включая Турбо Паскаль, не поддерживают переменную файлового буфера и, следовательно, не поддерживают GET и PUT.

Заключение

В этой главе мы рассмотрели основные инструменты обработки файлов данных:

1. Вы знаете, что стандартный Паскаль поддерживает текстовые и нетекстовые последовательные файлы, в то время как современные версии также поддерживают файлы произвольного/прямого доступа.
2. Во время путешествия по миру последовательных файлов вы научились объявлять, создавать, записывать в файл и читать из файла.
3. TEXT-файл объявляется следующим образом:

   ```
   файловая-переменная :TEXT;
   ```

4. Другие файлы объявляются, используя форму:

   ```
   TYPE
      идентификатор-типа = FILE OF тип-компонента;
   VAR
      файловая-переменная :идентификатор-типа;
   ```

   В стандартном Паскале файловая переменная должна быть включена в число файловых параметров, иначе файл будет считаться временным и будет автоматически удален после завершения программы. В UCSD для удаления этого временного файла нужно использовать ключевое слово PURGE, а в Турбо Паскале – процедуру ERASE.

5. Процедуры, используемые для открытия файла и на ввод, и на вывод:

   ```
   RESET(файловая-переменная);      (для ввода)
   REWRITE(файловая-переменная);    (для вывода)
   ```

   В современной версии Паскаля (такой как Турбо) можно также открыть последовательный файл на присоединение с помощью процедуры APPEND, которая имеет форму, подобную приведенным выше.

6. В версиях, отличных от стандартного Паскаля, файловая переменная должна быть связана с именем реального файла на диске. В Турбо Паскале это выполняется процедурой ASSIGN; в UCSD имя реального файла выступает в качестве второго параметра процедуры RESET или REWRITE. В этих реализациях файл должен быть закрыт после обработки с помощью процедуры CLOSE.
7. Вы изучили основную форму следующих операторов ввода/вывода:

   ```
   READLN(файловая-переменная, список-ввода);
   READ(файловая-переменная, список-ввода);
   WRITELN(файловая-переменная, список-вывода);
   WRITE(файловая-переменная, список-вывода);
   ```

Вы также знаете, что процедуры READLN и WRITELN нельзя использовать с нетекстовыми файлами. Вы также изучили процедуры стандартного Паскаля GET и PUT, которые используются во взаимодействии с переменными файлового буфера для чтения из файлов и записи в файлы данных. Эти процедуры не входят в некоторые версии Паскаля.

8. Вы также изучили основную форму функций EOF и EOLN:

   EOF(файловая-переменная)

   EOLN(файловая-переменная)

   Наконец, вы получили достаточные практические навыки для создания и манипулирования файлами в различных приложениях.

Упражнения

1. Даны следующие множества типа CHAR:

```
Small := ['a'..'z'];
Capital := ['A'..'Z'];
```

   Напишите операторы присваивания, определяющие множество алфавитных символов Alphabet и множество гласных Vowels. Затем напишите оператор IF для подсчета каждой категории и увеличения соответствующего счетчика: A для Alphabet, S для Small, C для Capital и V для Vowels.

2. Напишите программу, которая открывает текстовый файл и копирует его содержимое в другой текстовый файл. Добавьте номер строки в каждую строку нового файла. Имя входного файла дается во время выполнения программы. В качестве входного файла можно взять любой файл с текстом на Паскале из книги (или напечатать следующий файл и сохранить его в текущем каталоге под именем Hello.PAS).

```
{ ------- Hello.PAS ------- }
PROGRAM FirstProgram(OUTPUT);
BEGIN
  WRITELN('Привет, Мир!')
END.
```

   После запуска программа должна попросить ввести имя файла:

```
Пожалуйста, введите имя исходного файла: Hello.PAS -----> Введите это имя
```

   Затем должно быть выведено содержимое нового файла (NewFile.TXT):

```
Содержимое файла NewFile.TXT :
1       : { ------- Hello.PAS ------- }
2       : PROGRAM FirstProgram(OUTPUT);
```

```
3       : BEGIN
4       :   WRITELN('Привет, Мир!')
5       : END.
```

3. В задании 9-3 вы выводили итоговую сводку платежной ведомости в виде следующего текста:

```
122      Tammy M. Ockman     $898.00
123      Tara S. Strahan     $609.60
125      John G. Trainer     $1142.00
```

Измените программу, чтобы она помечала записи сотрудников, получивших более $1000.00 за этот платежный период. Сделайте это, добавив звездочку (*) в конец записи. Приведем пример требуемого выходного файла (PAYFILE.TXT):

```
122      Tammy M. Ockman     $898.00
123      Tara S. Strahan     $609.60
125      John G. Trainer     $1142.00*
```

Ответы

1. Объявите множества Vowels и Alphabet для хранения гласных и алфавитных символов, соответственно. Также объявите целый счетчик для подсчета каждого типа. Присвойте значения Vowels и Alphabet:

```
Alphabet := Small + Capital;
Vowels := ['a','e','i','o','u','A','E','I','O','U'];
```

Используйте следующий оператор IF для подсчета каждой категории:

```
IF Ch IN Alphabet THEN
  BEGIN
    A := A + 1;
    IF Ch IN Vowels THEN
      V := V + 1; { Счетчик гласных }
    IF Ch IN Small THEN
      S := S + 1; { Счетчик маленьких букв }
    IF Ch IN Capital THEN
      C := C + 1; { Счетчик прописных букв }
  END
```

2. Копирование MyFile в NewFile и добавление в него номера строки можно выполнить посредством следующего фрагмента кода:

```
ASSIGN(MyFile, ReadFileName);
RESET(MyFile);
ASSIGN(NewFile, WriteFileName);
REWRITE(NewFile);
WHILE NOT EOF(MyFile) DO
```

```
  BEGIN
    READLN(MyFile,OneLine);
    Counter := Counter +1;
    WRITELN(NewFile, Counter:3,': ', OneLine);
  END;
CLOSE(MyFile);
CLOSE(NewFile);
```

Обратите внимание, что `ReadFileName` и `WriteFileName` – это строковые переменные, которые хранят имена реальных файлов. Имена файлов можно также принимать как ввод пользователя.

3. Ниже приведены требуемые изменения программы из задания 9-3:

```
WITH PayRec DO
  IF Wages >= 1000.0 THEN
    WRITELN(P, ID:3, Name:20, Wages:10:2, '*')
  ELSE
    WRITELN(P, ID:3, Name:20, Wages:10:2);
```

Кроме того, необходимо немного изменить формат вывода – для того, чтобы правильно вывести файл на экран.

10

Использование вариантных записей

В этой главе мы постепенно разработаем систему расчета зарплаты. При этом вы будете все больше узнавать о *вариантных* (*variant*) записях – усовершенствованном типе записей, облегчающем управление программой. К концу главы система расчета зарплаты будет закончена.

10–1. Вариантные записи

В реальных приложениях сотрудников одной и той же компании можно разделить на различные категории. Некоторые сотрудники получают твердый оклад, некоторые – почасовую плату, а другие – комиссионные. Зарплата для каждой из этих категорий рассчитывается поразному. Вот пример записи для сотрудника на окладе:

```
SalariedEmployee = RECORD
                     ID            :STRING[5];
                     Name          :STRING[20];
                     Position      :STRING[20];
                     SSN           :STRING[11];
                     MonthlySalary :REAL
                   END;
```

Это пример записи сотрудника, получающего еженедельное вознаграждение:

```
HourlyEmployee = RECORD
                   ID         :STRING[5];
                   Name       :STRING[20];
                   Position   :STRING[20];
                   SSN        :STRING[11];
                   HourlyRate :REAL
                 END;
```

А это пример записи продавца, получающего комиссионные:

```
CommissionEmployee = RECORD
                       ID          :STRING[5];
                       Name        :STRING[20];
                       Position    :STRING[20]
                       SSN         :STRING[11];
                       Commission  :REAL;
                       BasicSalary :REAL;
                       Area        :STRING[20]
                     END;
```

Использовать в одной программе три различные записи для информации о сотруднике – не очень хорошая идея. В Паскале вариантные записи позволяют программисту хранить различные типы данных в одной и той же области памяти. В этом примере вариантная запись имеет фиксированную часть, содержащую поля, одинаковые для всех сотрудников (такие как ID, Name и SSN), и вариантную часть, которая меняется от категории к категории (например, детали оплаты). Для того чтобы иметь возможность отличать вариантную запись по типу, ее необходимо объявить с помощью структуры CASE с одним полем в виде CASE-выражения. Это поле называется *полем метки*.

Пример поля метки, которое можно добавить в запись, – символьная переменная, содержащая, например, значения 1, 2 или 3 для представления следующих категорий:

1 = сотрудник на окладе
2 = сотрудник на почасовой оплате
3 = сотрудник, получающий комиссионные

Вариантная запись сотрудника:

```
SalariedEmployee = RECORD
                     ID             :STRING[5];
                     Name, Position :STRING[20];
                     SSN            :STRING[11];
                     CASE Category: CHAR OF
                       '1':(MonthlySalary :REAL);
                       '2':(HourlyRate    :REAL);
                       '3':(Commission,
                            BasicSalary   :REAL;
                            Area          :STRING[20])
                   END;
```

Здесь поле метки – Category. Если значение поля метки равно 1, то управление передается на строку, содержащую переменную `MonthlySalary` (ежемесячный оклад); если 2, то на строку с переменной `HourlyRate` (почасовая оплата); если 3, то в действие приводятся три переменные – `Commission`, `BasicSalary` и `Area`.

Вариантная запись может содержать фиксированную часть, предшествующую вариантной части, или только вариантную часть. Объявление имеет следующий основной вид:

```
имя-типа = RECORD
    фиксированный список-полей
    вариантный список-полей
END;
```

Вариантный список полей имеет вид:

```
CASE поле-метки: определение-типа OF
    метка-1: (список-полей: определение-типа);
    метка-2: (список-полей: определение-типа);
    ...
    метка-n: (список-полей: определение-типа);
```

Обратите внимание, что список полей в каждом случае заключен в скобки, а структура CASE не содержит оператор END.

10–2. Пример: Улучшенная система расчета зарплаты

В этой программе осуществляется чтение записей разных сотрудников из платежной ведомости. Требуемая запись извлекается по номеру социального страхования (SSN), который вводится с клавиатуры. Перед запуском этой программы необходимо создать текстовый файл payroll.txt с записями сотрудников. Это можно сделать в любом текстовом редакторе. Записи в файле должны следовать друг за другом подряд, без пропусков и соответствовать описанию записи. Создав такой файл (даже с одной записью), можно с помощью этой программы добавить в него новые записи. Для тестирования программы можно взять файл payroll.txt с прилагаемой дискеты. Вот этот файл:

```
1MGT5
Tammy M. Ockman
Business Manager
232-65-1567

1                       -----------------> Поле метки
3333.33
2STF1
Tara S. Strahan
Secretary II
404-38-1132

2                       -----------------> Поле метки
8.24
3SAL4
John G. Trainer
```

```
Sales Representative
334-88-1234

3                    ---------------> Поле метки (обратите внимание
                                        на дополнительные поля в категории 3)
0.25
500.0
Baton Rouge, LA
1MGT4
Sally A. Abolrous
Technical Editor
434-65-6052
4343.88
1MGT1
James A. Abolrous
President
434-55-6666

1                    ---------------> Поле метки
4343.88
```

Перед там как перейти к обсуждению, рассмотрим следующую программу:

```
{ ------------------------- Пример 10-1 ----------------------------- }
PROGRAM EmployeeDataBase2(INPUT, OUTPUT, PayrollFile, NewFile);
CONST
  FileName = 'payroll.txt';
  TempFile = 'temp.txt';
  Header = '------------- Главное меню --------------';
  Header1 = '------- База данных сотрудников ---------';
  Header2 = '---------- Запись сотрудника ------------';
  Separator = '----------------------------------------';
TYPE
  EmployeeRecord = RECORD
                     ID                   :STRING[5];
                     Name, Position       :STRING[20];
                     SSN                  :STRING[11];
                     CASE Category        :CHAR OF
                       '1': (MonthlySalary :REAL);
                       '2': (HourlyRate    :REAL);
                       '3': (Commission,
                             BasicSalary   :REAL;
                     Area                 :STRING[20])
                   END;
VAR
  NewFile, PayrollFile :TEXT;
  EmployeeRec          :EmployeeRecord;
  Title                :ARRAY [1..9] OF STRING[20];
  OneLine              :STRING[80];
{ ------------------------ Процедура ReadRec ----------------------- }
PROCEDURE ReadRec(VAR PayrollFile: TEXT; Employee: EmployeeRecord);
```

```
VAR
  SSNumber :STRING[11];
  Found    :INTEGER;
BEGIN
  Found := 0; {Reset the flag}
  ASSIGN(PayrollFile, FileName);
  RESET(PayrollFile);
  WRITELN;
  WRITE('Пожалуйста, введите номер социального страхования сотрудника: ');
  READLN(SSNumber);
  WHILE NOT EOF(PayrollFile) DO
    BEGIN
      WITH Employee DO
        BEGIN
          READLN(PayrollFile, ID);
          READLN(PayrollFile, Name);
          READLN(PayrollFile, Position);
          READLN(PayrollFile, SSN);
          READLN(PayrollFile, Category);
          CASE Category OF
            '1': READLN(PayrollFile, MonthlySalary);
            '2': READLN(PayrollFile, HourlyRate);
            '3': BEGIN
                   READLN(PayrollFile, Commission);
                   READLN(PayrollFile, BasicSalary);
                   READLN(PayrollFile, Area)
                 END
          END; { Конец структуры CASE }
          IF SSNumber = SSN THEN
            BEGIN
              WRITELN(Header2);
              WRITELN(Title[1], ID);
              WRITELN(Title[2], Name);
              WRITELN(Title[3], Position);
              WRITELN(Title[4], SSN);
              CASE Category OF
                '1': WRITELN(Title[5], MonthlySalary:0:2);
                '2': WRITELN(Title[6], HourlyRate:0:2);
                '3': BEGIN
                       WRITELN(Title[7], Commission:0:2);
                       WRITELN(Title[8], BasicSalary:0:2);
                       WRITELN(Title[9], Area)
                     END
              END; { Конец структуры CASE }
              Found := 1
            END
        END { Конец блока WITH }
    END;
  CLOSE(PayrollFile);
  IF Found <> 1 THEN
    BEGIN
```

```
      WRITELN('SSN не найден в файле.');
      WRITELN('Пожалуйста, попытайтесь снова.');
      WRITELN
    END
END;
{ ------------------------ Процедура AddRec ------------------------ }
PROCEDURE AddRec(VAR NewFile, PayrollFile: TEXT;
                 Employee: EmployeeRecord);
BEGIN
  ASSIGN(PayrollFile, FileName);
  RESET(PayrollFile);
  ASSIGN(NewFile, TempFile);
  REWRITE(NewFile);
{ Проверка конца текстового файла }
  WHILE NOT EOF(PayrollFile) DO
    BEGIN
{ Копирование каждой записи из PayrollFile в NewFile }
      READLN(PayrollFile, OneLine);
      WRITELN(NewFile, OneLine)
    END;
{ Ввод новой записи с клавиатуры}
  WITH Employee DO
    BEGIN
      WRITE('Пожалуйста, введите ID сотрудника: ');
      READLN(ID);
      WRITE('Имя: '); READLN(Name);
      WRITE('Должность: '); READLN(Position);
      WRITE('Номер социального страхования (xxx-xx-xxxx): '); READLN(SSN);
      WRITE('Категория оплаты: '); READLN(Category);
      CASE Category OF
        '1': BEGIN
               WRITE('Месячный оклад: ');
               READLN(MonthlySalary);
             END;
        '2': BEGIN
               WRITE('Тариф: ');
               READLN(HourlyRate);
             END;
        '3': BEGIN
               WRITE('Комиссионные: ');
               READLN(Commission);
               WRITE('Основной оклад: ');
               READLN(BasicSalary);
               WRITE('Регион: ');
               READLN(Area)
             END
      END;
{ Запись информации в NewFile }
      WRITELN(NewFile, ID);
      WRITELN(NewFile, Name);
      WRITELN(NewFile, Position);
```

```
      WRITELN(NewFile, SSN);
      WRITELN(NewFile, Category);
      CASE Category OF
        '1': WRITELN(NewFile, MonthlySalary:0:2);
        '2': WRITELN(NewFile, HourlyRate:0:2);
        '3': BEGIN
               WRITELN(NewFile, Commission:0:2);
               WRITELN(NewFile, BasicSalary:0:2);
               WRITELN(NewFile, Area)
             END
      END
    END;
  CLOSE(NewFile);
  CLOSE(PayrollFile);
{ Копирование NewFile обратно в файл платежной ведомости }
  ASSIGN(PayrollFile, FileName);
  REWRITE(PayrollFile);
  ASSIGN(NewFile, TempFile);
  RESET(NewFile);
  WHILE NOT EOF(NewFile) DO
    BEGIN
      READLN(NewFile, OneLine);
      WRITELN(PayrollFile, OneLine)
    END;
  CLOSE(NewFile);
  ERASE(NewFile); { Удаление временного файла }
  CLOSE(PayrollFile)
END;
{ ------------------------ Процедура Menu -------------------------- }
PROCEDURE Menu;
VAR
  Option: INTEGER;
BEGIN
  WRITELN(Header);
  WRITELN;
  WRITELN('1. Показать запись сотрудника.');
  WRITELN('2. Добавить нового сотрудника.');
  WRITELN('3. Выход.');
  WRITELN(Separator);
  WRITE('Сделайте выбор и нажмите цифру: ');
  READLN(Option);
  CASE Option OF
    1: ReadRec(PayrollFile, EmployeeRec);
    2: AddRec(NewFile, PayrollFile, EmployeeRec);
    3: Exit
  END;
  Menu
END;
{ -------------------------- Главная программа ------------------------- }
BEGIN
{ Присваивание заголовков }
```

```
  Title[1] := 'ID: ';
  Title[2] := 'Имя: ';
  Title[3] := 'Должность: ';
  Title[4] := 'SSN: ';
  Title[5] := 'Оклад: ';
  Title[6] := 'Тариф: ';
  Title[7] := 'Комиссионные: ';
  Title[8] := 'Основной оклад: ';
  Title[9] := 'Регион: ';
  Menu
END.
```

Выполнение теста:

(A) Чтение записей из файла. **При выполнении теста из файла читаются три записи трех различных категорий сотрудников. При попытке прочитать четвертую запись мы получили сообщение «SSN не найден в файле» и ввели с клавиатуры номер социального страхования, отсутствующий в файле. Обратите внимание, что введенные с клавиатуры данные для ясности выделены полужирным курсивным шрифтом.**

```
---------- Главное меню -------------
1. Показать запись сотрудника.
2. Добавить нового сотрудника.
3. Выход.
-------------------------------------
Сделайте выбор и нажмите цифру: 1
Пожалуйста, введите номер социального страхования сотрудника: 434-55-6666

-------- Запись сотрудника -----------
ID: 1MGT1
Имя: James A. Abolrous
Должность: President
SSN:434-55-6666
Зарплата: 4343.88
---------- Главное меню -------------
1. Показать запись сотрудника.
2. Добавить нового сотрудника.
3. Выход.
-------------------------------------
Сделайте выбор и нажмите цифру: 1
Пожалуйста, введите номер социального страхования сотрудника: 404-38-1132

-------- Запись сотрудника -----------
ID: 2STF1
Имя: Tara S. Strahan
Должность: Secretary II
SSN:404-38-1132
Тариф: 8.24
---------- Главное меню -------------
1. Показать запись сотрудника.
2. Добавить нового сотрудника.
```

```
3. Выход.
-------------------------------------
Сделайте выбор и нажмите цифру: 1
Пожалуйста, введите номер социального страхования сотрудника: 334-88-1234

-------- Запись сотрудника -----------
ID: 3SAL4
Имя: John G. Trainer
Должность: Sales Representative
SSN: 334-88-1234
Комиссионные: 0.25
Основной оклад: 500.00
Регион: Baton Rouge, LA
----------- Главное меню -------------
1. Показать запись сотрудника.
2. Добавить нового сотрудника.
3. Выход.
-------------------------------------
Сделайте выбор и нажмите цифру: 1
Пожалуйста, введите номер социального страхования сотрудника: 555-55-5555

SSN не найден в файле.
Пожалуйста, попытайтесь снова.
----------- Главное меню -------------
1. Показать запись сотрудника.
2. Добавить нового сотрудника.
3. Выход.
-------------------------------------
Сделайте выбор и нажмите цифру: 3
```

(B) Добавление записей в файл. При дальнейшем выполнении в платежную ведомость добавляется новая запись:

```
------------ Главное меню ------------
1. Показать запись сотрудника.
2. Добавить нового сотрудника.
3. Выход.
-------------------------------------
Сделайте выбор и нажмите цифру: 2
Пожалуйста, введите ID сотрудника: 3SAL6
Имя: Barbara Ortiz
Должность: Sales Representative
SSN (xxx-xx-xxxx): 347-12-3456
Категория оплаты: 3
Комиссионные: .15
Основной оклад: 450.0
Регион: New Orleans, LA
----------- Главное меню -------------
1. Показать запись сотрудника.
2. Добавить нового сотрудника.
3. Выход.
```

```
---------------------------------------
Сделайте выбор и нажмите цифру: 3
```

В этом месте в файл платежной ведомости была добавлена запись о сотруднике по имени Barbara Ortiz, и теперь запись можно вывести на экран, воспользовавшись опцией 2. Нет необходимости говорить, что можно также добавить запись в конец файла при помощи любого текстового редактора. Вот некоторые важные моменты в этой программе:

- Программа разделена на три процедуры:
  - Menu: выводит пункты меню на экран и обеспечивает реакцию на их выбор.
  - ReadRec: читает из файла.
  - AddRec: добавляет запись в файл.
- При использовании вариантной записи для чтения или для записи необходимо применять структуру CASE.
- Для проверки существования в файле требуемой записи и выдачи соответствующего сообщения в процедуре ReadRec используется флаг Found. Обратите внимание, что флаг сбрасывается в начале процедуры (Found := 0). Когда требуемая запись найдена, флагу присваивается значение 1; в противном случае он сохраняет нулевое значение, и после достижения метки конца файла выводится сообщение «SSN не найден...».

Задание 10–1

Напишите программу, которая бы создавала файл платежной ведомости для предыдущего примера. Эту программу можно запустить только однажды, поскольку при каждом запуске создается новый файл. На прилагаемой дискете можно найти программу под именем DRL10-1.PAS. Она создает файл с именем pr.txt, что позволяет избежать перезаписи файла payroll.txt.

10–3. Удаление записей из файла

Алгоритм удаления записей сотрудников из файла платежной ведомости следующий:

1. Ввести номер социального страхования удаляемого сотрудника.
2. Открыть файл платежной ведомости на чтение, а временный файл на запись.
3. Прочитать файл платежной ведомости до метки конца файла. Сравнить поле SSN каждой записи с введенным номером социального страхования.

4. Скопировать во временный файл все записи, за исключением тех, для которых произошло совпадение.
5. Скопировать временный файл в исходный.
6. Удалить временный файл.

Для удаления временного файла в предыдущую программу необходимо добавить следующую процедуру (исходный код этой процедуры находится на прилагаемой дискете под именем Del-Proc.PAS).

```
{ ------------------------  Процедура DelRec ------------------------ }
PROCEDURE DelRec(VAR NewFile, PayrollFile: TEXT;
                 Employee: EmployeeRecord);
VAR
  SSNumber: STRING[11];
BEGIN
  ASSIGN(PayrollFile, FileName);
  RESET(PayrollFile);
  ASSIGN(NewFile, TempFile);
  REWRITE(NewFile);
  WRITE('Пожалуйста, введите номер социального страхования удаляемого
сотрудника: ');
  READLN(SSNumber);
  WHILE NOT EOF(PayrollFile) DO
    BEGIN
      WITH Employee DO
        BEGIN
          READLN(PayrollFile, ID);
          READLN(PayrollFile, Name);
          READLN(PayrollFile, Position);
          READLN(PayrollFile, SSN);
          READLN(PayrollFile, Category);
          CASE Category OF
            '1': READLN(PayrollFile, MonthlySalary);
            '2': READLN(PayrollFile, HourlyRate);
            '3': BEGIN
                   READLN(PayrollFile, Commission);
                   READLN(PayrollFile, BasicSalary);
                   READLN(PayrollFile, Area)
                 END
          END; { Конец структуры CASE }
          IF SSNumber <> SSN THEN
            BEGIN
              WRITELN(NewFile, ID);
              WRITELN(NewFile, Name);
              WRITELN(NewFile, Position);
              WRITELN(NewFile, SSN);
              WRITELN(NewFile, Category);
              CASE Category OF
                '1': WRITELN(NewFile, MonthlySalary:0:2);
                '2': WRITELN(NewFile, HourlyRate:0:2);
```

```
                '3': BEGIN
                       WRITELN(NewFile, Commission:0:2);
                       WRITELN(NewFile, BasicSalary:0:2);
                       WRITELN(NewFile, Area)
                     END
              END; { Конец структуры CASE }
            END
        END { Конец блока WITH }
    END;
  CLOSE(NewFile);
  CLOSE(PayrollFile);
{ Копирование NewFile обратно в файл платежной ведомости }
  ASSIGN(PayrollFile, FileName);
  REWRITE(PayrollFile);
  ASSIGN(NewFile, TempFile);
  RESET(NewFile);
  WHILE NOT EOF(NewFile) DO
    BEGIN
      READLN(NewFile, OneLine);
      WRITELN(PayrollFile, OneLine)
    END;
  CLOSE(NewFile);
  ERASE(NewFile); { Удаление временного файла }
  CLOSE(PayrollFile);
  WRITELN('Сотрудник ', SSNumber, ' удален из файла.')
END;
```

Для того чтобы среди пунктов меню появилась опция «Удалить запись», необходимо модифицировать процедуру Menu. Она может выглядеть, как процедура, приведенная ниже (она есть на прилагаемой дискете под именем Mnu-Proc.PAS).

```
{ ------------------------ Процедура Menu -------------------------- }
PROCEDURE Menu;
VAR
  Option: INTEGER;
BEGIN
  WRITELN(Header);
  WRITELN;
  WRITELN('1. Показать запись сотрудника.');
  WRITELN('2. Добавить нового сотрудника.');
  WRITELN('3. Удалить сотрудника.');
  WRITELN('4. Выход.');
  WRITELN(Separator);
  WRITE(' Сделайте выбор и нажмите цифру: ');
  READLN(Option);
  CASE Option OF
    1: ReadRec(PayrollFile, EmployeeRec);
    2: AddRec(NewFile, PayrollFile, EmployeeRec);
    3: DelRec(NewFile, PayrollFile, EmployeeRec);
    4: Exit
```

```
  END;
  Menu
END;
```

Далее приведены все модули программы, собранные вместе:

```
{ ------------------------ Пример 10-2 ---------------------------- }
PROGRAM EmployeeDataBase2(INPUT, OUTPUT, PayrollFile, NewFile);
CONST
  FileName = 'payroll.txt';
  TempFile = 'temp.txt';
  Header = '------------- Главное меню --------------';
  Header1 = '------- База данных сотрудников ----------';
  Header2 = '---------- Запись сотрудника -------------';
  Separator = '----------------------------------------';
TYPE
  EmployeeRecord = RECORD
                     ID                    :STRING[5];
                     Name, Position        :STRING[20];
                     SSN                   :STRING[11];
                     CASE Category         :CHAR OF
                       '1': (MonthlySalary :REAL);
                       '2': (HourlyRate    :REAL);
                       '3': (Commission,
                             BasicSalary   :REAL;
                             Area          :STRING[20])
                     END;
VAR
  NewFile, PayrollFile :TEXT;
  EmployeeRec          :EmployeeRecord;
  Title                :ARRAY [1..9] OF STRING[20];
  OneLine              :STRING[80];

{ ------------------------ Процедура ReadRec ----------------------- }
PROCEDURE ReadRec(VAR PayrollFile: TEXT; Employee: EmployeeRecord);
VAR
  SSNumber :STRING[11];
  Found    :INTEGER;
BEGIN
  Found := 0; {Reset the flag}
  ASSIGN(PayrollFile, FileName);
  RESET(PayrollFile);
  WRITELN;
  WRITE('Пожалуйста, введите номер социального страхования сотрудника: ');
  READLN(SSNumber);
  WHILE NOT EOF(PayrollFile) DO
    BEGIN
      WITH Employee DO
        BEGIN
          READLN(PayrollFile, ID);
          READLN(PayrollFile, Name);
          READLN(PayrollFile, Position);
```

```
            READLN(PayrollFile, SSN);
            READLN(PayrollFile, Category);
            CASE Category OF
              '1': READLN(PayrollFile, MonthlySalary);
              '2': READLN(PayrollFile, HourlyRate);
              '3': BEGIN
                     READLN(PayrollFile, Commission);
                     READLN(PayrollFile, BasicSalary);
                     READLN(PayrollFile, Area)
                   END
            END; { Конец структуры CASE }
            IF SSNumber = SSN THEN
              BEGIN
                WRITELN(Header2);
                WRITELN(Title[1], ID);
                WRITELN(Title[2], Name);
                WRITELN(Title[3], Position);
                WRITELN(Title[4], SSN);
                CASE Category OF
                  '1': WRITELN(Title[5], MonthlySalary:0:2);
                  '2': WRITELN(Title[6], HourlyRate:0:2);
                  '3': BEGIN
                         WRITELN(Title[7], Commission:0:2);
                         WRITELN(Title[8], BasicSalary:0:2);
                         WRITELN(Title[9], Area)
                       END
                END; { Конец структуры CASE }
                Found := 1
              END
          END { Конец блока WITH }
      END;
    CLOSE(PayrollFile);
    IF Found <> 1 THEN
      BEGIN
        WRITELN('SSN не найден в файле.');
        WRITELN('Пожалуйста, попытайтесь снова.');
        WRITELN
      END
END;

{ ------------------------ Процедура AddRec ------------------------ }
PROCEDURE AddRec(VAR NewFile, PayrollFile: TEXT;
                 Employee: EmployeeRecord);
BEGIN
  ASSIGN(PayrollFile, FileName);
  RESET(PayrollFile);
  ASSIGN(NewFile, TempFile);
  REWRITE(NewFile);
{ Проверка конца текстового файла }
  WHILE NOT EOF(PayrollFile) DO
    BEGIN
```

```
{ Копирование каждой записи из PayrollFile в NewFile }
      READLN(PayrollFile, OneLine);
      WRITELN(NewFile, OneLine)
    END;
{ Ввод новой записи с клавиатуры }
  WITH Employee DO
    BEGIN
      WRITE('Пожалуйста, введите ID сотрудника: ');
      READLN(ID);
      WRITE('Имя: '); READLN(Name);
      WRITE('Должность: '); READLN(Position);
      WRITE('Номер социального страхования (xxx-xx-xxxx): '); READLN(SSN);
      WRITE('Категория оплаты: '); READLN(Category);
      CASE Category OF
        '1': BEGIN
               WRITE('Месячный оклад: ');
               READLN(MonthlySalary);
             END;
        '2': BEGIN
               WRITE('Тариф: ');
               READLN(HourlyRate);
             END;
        '3': BEGIN
               WRITE('Комиссионные: ');
               READLN(Commission);
               WRITE('Основной оклад: ');
               READLN(BasicSalary);
               WRITE('Регион: ');
               READLN(Area)
             END
      END;
{ Запись информации в новый файл }
      WRITELN(NewFile, ID);
      WRITELN(NewFile, Name);
      WRITELN(NewFile, Position);
      WRITELN(NewFile, SSN);
      WRITELN(NewFile, Category);
      CASE Category OF
        '1': WRITELN(NewFile, MonthlySalary:0:2);
        '2': WRITELN(NewFile, HourlyRate:0:2);
        '3': BEGIN
               WRITELN(NewFile, Commission:0:2);
               WRITELN(NewFile, BasicSalary:0:2);
               WRITELN(NewFile, Area)
             END
      END
    END;
  CLOSE(NewFile);
  CLOSE(PayrollFile);
{ Копирование NewFile обратно в файл платежной ведомости }
  ASSIGN(PayrollFile, FileName);
```

```
  REWRITE(PayrollFile);
  ASSIGN(NewFile, TempFile);
  RESET(NewFile);
  WHILE NOT EOF(NewFile) DO
    BEGIN
      READLN(NewFile, OneLine);
      WRITELN(PayrollFile, OneLine)
    END;
  CLOSE(NewFile);
  ERASE(NewFile); { Удаление временного файла }
  CLOSE(PayrollFile)
END;

{ ------------------ Процедура DelRec ---------------------- }
PROCEDURE DelRec(VAR NewFile, PayrollFile: TEXT;
                 Employee: EmployeeRecord);
VAR
  SSNumber: STRING[11];
BEGIN
  ASSIGN(PayrollFile, FileName);
  RESET(PayrollFile);
  ASSIGN(NewFile, TempFile);
  REWRITE(NewFile);
  WRITE('Пожалуйста, введите номер социального страхования удаляемого
сотрудника: ');
  READLN(SSNumber);
  WHILE NOT EOF(PayrollFile) DO
    BEGIN
      WITH Employee DO
        BEGIN
          READLN(PayrollFile, ID);
          READLN(PayrollFile, Name);
          READLN(PayrollFile, Position);
          READLN(PayrollFile, SSN);
          READLN(PayrollFile, Category);
          CASE Category OF
            '1': READLN(PayrollFile, MonthlySalary);
            '2': READLN(PayrollFile, HourlyRate);
            '3': BEGIN
                   READLN(PayrollFile, Commission);
                   READLN(PayrollFile, BasicSalary);
                   READLN(PayrollFile, Area)
                 END
          END; { Конец структуры CASE }
          IF SSNumber <> SSN THEN
            BEGIN
              WRITELN(NewFile,ID);
              WRITELN(NewFile,Name);
              WRITELN(NewFile,Position);
              WRITELN(NewFile,SSN);
              WRITELN(NewFile,Category);
```

```
              CASE Category OF
                '1': WRITELN(NewFile,MonthlySalary:0:2);
                '2': WRITELN(NewFile,HourlyRate:0:2);
                '3': BEGIN
                       WRITELN(NewFile,Commission:0:2);
                       WRITELN(NewFile,BasicSalary:0:2);
                       WRITELN(NewFile,Area)
                     END
              END; { Конец структуры CASE }
            END
        END { Конец блока WITH }
    END;
  CLOSE(NewFile);
  CLOSE(PayrollFile);
{ Копирование NewFile обратно в файл платежной ведомости }
  ASSIGN(PayrollFile, FileName);
  REWRITE(PayrollFile);
  ASSIGN(NewFile, TempFile);
  RESET(NewFile);
  WHILE NOT EOF(NewFile) DO
    BEGIN
      READLN(NewFile,OneLine);
      WRITELN(PayrollFile,OneLine)
    END;
  CLOSE(NewFile);
  ERASE(NewFile); { Удаление временного файла }
  CLOSE(PayrollFile);
  WRITELN('Сотрудник ', SSNumber, ' удален из файла.')
END;

{ ------------------------ Процедура Menu --------------------------- }
PROCEDURE Menu;
VAR
  Option: INTEGER;
BEGIN
  WRITELN(Header);
  WRITELN;
  WRITELN('1. Показать запись сотрудника.');
  WRITELN('2. Добавить нового сотрудника.');
  WRITELN('3. Удалить сотрудника.');
  WRITELN('4. Выход.');
  WRITELN(Separator);
  WRITE('Сделайте выбор и нажмите цифру: ');
  READLN(Option);
  CASE Option OF
    1: ReadRec(PayrollFile, EmployeeRec);
    2: AddRec(NewFile, PayrollFile, EmployeeRec);
    3: DelRec(NewFile, PayrollFile, EmployeeRec);
    4: Exit
  END;
  Menu
```

```
END;
{ -------------------------- Главная программа ------------------------- }
BEGIN
{ Присваивание заголовков }
  Title[1] := 'ID: ';
  Title[2] := 'Имя: ';
  Title[3] := 'Должность: ';
  Title[4] := 'SSN: ';
  Title[5] := 'Зарплата: ';
  Title[6] := 'Тариф: ';
  Title[7] := 'Комиссионные: ';
  Title[8] := 'Основной оклад: ';
  Title[9] := 'Регион: ';
  Menu
END.
```

Пример выполнения:

Удаляется сотрудник с номером социального страхования 347-12-3456. Ввод пользователя для ясности показан полужирным курсивным шрифтом.

```
----------- Главное меню -------------
1. Показать запись сотрудника.
2. Добавить нового сотрудника.
3. Удалить сотрудника.
4. Выход.
--------------------------------------
Сделайте выбор и нажмите цифру: 3
Пожалуйста, введите номер социального страхования удаляемого сотрудника: 347-
12-3456
Сотрудник 347-12-3456 удален из файла.

----------- Главное меню -------------
1. Показать запись сотрудника.
2. Добавить нового сотрудника.
3. Удалить сотрудника.
4. Выход.
--------------------------------------
Сделайте выбор и нажмите цифру: 4
```

Задание 10–2

Предыдущая программа выдаст сообщение «Сотрудник ... удален из файла» независимо от того, присутствует ли в файле указанный номер социального страхования.

Добавьте в программу необходимый код, чтобы программа в каждом случае посылала надлежащее сообщение.

10–4. Обновление записей

Алгоритм обновления записей в файле следующий:

1. Ввести номер социального страхования сотрудника, чью запись требуется обновить.
2. Открыть файл платежной ведомости на чтение, а временный файл – на запись.
3. Читать файл платежной ведомости до метки конца файла. Сравнить поле SSN каждой записи с введенным номером социального страхования.
4. Копировать каждую запись во временный файл, пока не встретится запись, которую надо обновить.
5. Прочитать с клавиатуры новые данные для обновляемой записи и записать их во временный файл.
6. Скопировать остальные записи во временный файл.
7. Скопировать временный файл в исходный файл платежной ведомости.
8. Удалить временный файл.

Можете добавить в программу следующую процедуру (эта процедура находится на прилагаемой дискете под именем Upd-Proc.PAS):

```
{ ---------------------- Процедура UpdateRec ------------------------- }
PROCEDURE UpdateRec(VAR NewFile, PayrollFile: TEXT;
                    Employee: EmployeeRecord);
VAR
  SSNumber :STRING[11];
  Found    :INTEGER;
BEGIN
  Found := 0;
  ASSIGN(PayrollFile, FileName);
  RESET(PayrollFile);
  ASSIGN(NewFile, TempFile);
  REWRITE(NewFile);
  WRITE('Пожалуйста, введите номер социального страхования сотрудника, запись
         которого нужно обновить: ');
  READLN(SSNumber);
  WHILE NOT EOF(PayrollFile) DO
    BEGIN
      WITH Employee DO
        BEGIN
          READLN(PayrollFile, ID);
          READLN(PayrollFile, Name);
          READLN(PayrollFile, Position);
          READLN(PayrollFile, SSN);
          READLN(PayrollFile, Category);
          CASE Category OF
```

```
    '1': READLN(PayrollFile, MonthlySalary);
    '2': READLN(PayrollFile, HourlyRate);
    '3': BEGIN
           READLN(PayrollFile, Commission);
           READLN(PayrollFile, BasicSalary);
           READLN(PayrollFile, Area)
         END
  END; { Конец структуры CASE }
  IF SSNumber <> SSN THEN
    BEGIN
      WRITELN(NewFile, ID);
      WRITELN(NewFile, Name);
      WRITELN(NewFile, Position);
      WRITELN(NewFile, SSN);
      WRITELN(NewFile, Category);
      CASE Category OF
        '1': WRITELN(NewFile, MonthlySalary:0:2);
        '2': WRITELN(NewFile, HourlyRate:0:2);
        '3': BEGIN
               WRITELN(NewFile, Commission:0:2);
               WRITELN(NewFile, BasicSalary:0:2);
               WRITELN(NewFile, Area);
             END
      END; { Конец структуры CASE }
    END
  ELSE
    BEGIN
      Found := 1;
      WRITELN('Пожалуйста, введите новую информацию:');
      WRITE('ID: '); READLN(ID);
      WRITELN(NewFile, ID);
      WRITE('Имя: '); READLN(Name);
      WRITELN(NewFile, Name);
      WRITE('Должность: '); READLN(Position);
      WRITELN(NewFile, Position);
      WRITELN(NewFile, SSN);
      WRITE('Категория: '); READLN(Category);
      WRITELN(NewFile, Category);
      CASE Category OF
        '1': BEGIN
               WRITE('Зарплата: ');
               READLN(MonthlySalary);
               WRITELN(NewFile, MonthlySalary:0:2)
             END;
        '2': BEGIN
               WRITE('Почасовой Тариф: ');
               READLN(HourlyRate);
               WRITELN(NewFile, HourlyRate:0:2)
             END;
        '3': BEGIN
               WRITE('Комиссионные: ');
```

```
                        READLN(Commission);
                        WRITELN(NewFile, Commission:0:2);
                        WRITE('Основной оклад: ');
                        READLN(BasicSalary);
                        WRITELN(NewFile, BasicSalary:0:2);
                        WRITE('Регион: ');
                        READLN(Area);
                        WRITELN(NewFile, Area)
                      END
                END; { Конец структуры CASE }
              END
          END { Конец блока WITH }
      END;
    CLOSE(NewFile);
    CLOSE(PayrollFile);
  { Копирование NewFile обратно в файл платежной ведомости }
    ASSIGN(PayrollFile, FileName);
    REWRITE(PayrollFile);
    ASSIGN(NewFile, TempFile);
    RESET(NewFile);
    WHILE NOT EOF(NewFile) DO
      BEGIN
        READLN(NewFile, OneLine);
        WRITELN(PayrollFile, OneLine)
      END;
    CLOSE(NewFile);
    ERASE(NewFile); { Удаление временного файла }
    CLOSE(PayrollFile);
  { Сообщения пользователя }
    IF Found =1 THEN
      WRITELN('Запись сотрудника ', SSNumber, ' обновлена.')
    ELSE
      BEGIN
        WRITELN('Номер социального страхования ', SSNumber, ' не найден.');
        WRITELN('Проверьте номер и попытайтесь снова.');
        WRITELN
      END
  END;
```

Флаг Found имеет тип INTEGER. Однако можно выбрать любой другой тип, например BOOLEAN, что облегчит чтение программы. Флаг типа BOOLEAN позволяет применять такие операторы, как IF Found и IF NOT Found.

Для того чтобы включить опцию обновления, необходимо модифицировать процедуру меню, как показано ниже (эта процедура – Mnu-Pro2.PAS – находится на прилагаемой дискете).

```
{ ------------------------ Процедура Menu --------------------------- }
PROCEDURE Menu;
VAR
  Option: INTEGER;
```

```
BEGIN
  WRITELN(Header);
  WRITELN;
  WRITELN('1. Показать запись сотрудника.');
  WRITELN('2. Добавить нового сотрудника.');
  WRITELN('3. Удалить сотрудника.');
  WRITELN('4. Обновить  запись сотрудника.');
  WRITELN('5. Выход.');
  WRITELN(Separator);
  WRITE('Сделайте выбор и нажмите цифру: ');
  READLN(Option);
  CASE Option OF
    1: ReadRec(PayrollFile, EmployeeRec);
    2: AddRec(NewFile, PayrollFile, EmployeeRec);
    3: DelRec(NewFile, PayrollFile, EmployeeRec);
    4: UpdateRec(NewFile, PayrollFile, EmployeeRec);
    5: Exit
  END;
  Menu
END;
```

Задание 10–3

Объедините процедуры вместе с целью построения завершенной программы, которая может выводить на экран, добавлять, удалять и обновлять записи сотрудников.

10–5. Повышение модульности программы

Итак, в программу добавлены дополнительные процедуры, и необходимо еще раз проверить модульность программы. Один из недостатков программы состоит в том, что для проверки существования требуемой записи в трех процедурах используется флаг Found. Другой недостаток в том, что файл payroll.txt копируется во временный файл независимо от существования требуемой записи. Этой избыточности можно избежать, если написать процедуру, которая просматривает файл и устанавливает (или сбрасывает) флаг Found. В этом случае при входе в любую из других процедур новая процедура будет заранее «знать», существует запись или нет. Таким образом, все необходимые шаги можно включить в блок IF:

```
READLN(SSNumber);
SearchRec(PayrollFile, EmployeeRec, SSNumber, Found);
IF Found =1 THEN
  BEGIN
    ...
{ открытие файла и выполнение необходимых операций }
    ...
```

```
  END
ELSE
{ посылка соответствующего сообщения }
END;
```

Новая процедура SearchRec вызывается после ввода SSNumber с клавиатуры. Процедура открывает файл, ищет требуемого сотрудника и возвращает соответствующее значение флага. Если запись найдена, то другие процедуры выполняют стандартные действия (обновление, удаление или чтение); в противном случае посылается соответствующее сообщение и переоткрывать файлы не надо.

Рассмотрим некоторые важные моменты процедуры SearchRec:

- SSNumber и флаг передаются в процедуру как параметры.
- Поскольку ожидается изменение значения флага, то его надо передавать, используя ключевое слово VAR.
- Для передачи строковой переменной параметр должен быть типизирован; поэтому в секции TYPE объявляется новый тип:

```
SSNstring = STRING[11];
```

Ниже приводится программа в окончательном виде:

```
{ ------------------------- Пример 10-3 ---------------------------- }
PROGRAM EmployeeDataBase2(INPUT, OUTPUT, PayrollFile, NewFile);
CONST
  FileName = 'payroll.txt';
  TempFile = 'temp.txt';
  Header = '------------- Главное меню --------------';
  Header1 = '------- База данных сотрудников ----------';
  Header2 = '---------- Запись сотрудника -------------';
  Separator = '-----------------------------------------';
TYPE
  EmployeeRecord = RECORD
                     ID                    :STRING[5];
                     Name, Position        :STRING[20];
                     SSN                   :STRING[11];
                     CASE Category         :CHAR OF
                       '1': (MonthlySalary :REAL);
                       '2': (HourlyRate    :REAL);
                       '3': (Commission,
                             BasicSalary   :REAL;
                             Area          :STRING[20])
                   END;
  SSNstring = STRING[11];
VAR
  NewFile, PayrollFile :TEXT;
  EmployeeRec          :EmployeeRecord;
  Title                :ARRAY [1..9] OF STRING[20];
  OneLine              :STRING[80];
{ ------------------------ Процедура SearchRec --------------------- }
```

```
PROCEDURE SearchRec(VAR PayrollFile: TEXT;
                    Employee: EmployeeRecord;
                    SSNumber: SSNstring;
                    VAR Found: INTEGER);
BEGIN
  Found := 0;
  ASSIGN(PayrollFile, FileName);
  RESET(PayrollFile);
  WHILE NOT EOF(PayrollFile) DO
    BEGIN
      WITH Employee DO
        BEGIN
          READLN(PayrollFile, ID);
          READLN(PayrollFile, Name);
          READLN(PayrollFile, Position);
          READLN(PayrollFile, SSN);
          READLN(PayrollFile, Category);
          CASE Category OF
            '1': READLN(PayrollFile, MonthlySalary);
            '2': READLN(PayrollFile, HourlyRate);
            '3': BEGIN
                   READLN(PayrollFile, Commission);
                   READLN(PayrollFile, BasicSalary);
                   READLN(PayrollFile, Area)
                 END
          END; { Конец структуры CASE }
          IF SSNumber = SSN THEN
            Found := 1;
        END { Конец блока WITH }
    END;
  CLOSE(PayrollFile);
END;
{ ------------------------ Процедура ReadRec ----------------------- }
PROCEDURE ReadRec(VAR PayrollFile: TEXT;
                  Employee: EmployeeRecord);
VAR
  SSNumber :STRING[11];
  Found    :INTEGER;
BEGIN
  WRITELN;
  WRITE('Пожалуйста, введите номер социального страхования сотрудника: ');
  READLN(SSNumber);
  SearchRec(PayrollFile, EmployeeRec, SSNumber, Found);
  IF Found =1 THEN
    BEGIN
      ASSIGN(PayrollFile, FileName);
      RESET(PayrollFile);
      WHILE NOT EOF(PayrollFile) DO
        BEGIN
          WITH Employee DO
            BEGIN
```

```
                READLN(PayrollFile, ID);
                READLN(PayrollFile, Name);
                READLN(PayrollFile, Position);
                READLN(PayrollFile, SSN);
                READLN(PayrollFile, Category);
                CASE Category OF
                  '1': READLN(PayrollFile, MonthlySalary);
                  '2': READLN(PayrollFile, HourlyRate);
                  '3': BEGIN
                         READLN(PayrollFile, Commission);
                         READLN(PayrollFile, BasicSalary);
                         READLN(PayrollFile, Area)
                       END
                END; { Конец структуры CASE }
                IF SSNumber = SSN THEN
                  BEGIN
                    WRITELN(Header2);
                    WRITELN(Title[1],ID);
                    WRITELN(Title[2],Name);
                    WRITELN(Title[3],Position);
                    WRITELN(Title[4], SSN);
                    CASE Category OF
                      '1': WRITELN(Title[5], MonthlySalary:0:2);
                      '2': WRITELN(Title[6], HourlyRate:0:2);
                      '3': BEGIN
                             WRITELN(Title[7], Commission:0:2);
                             WRITELN(Title[8], BasicSalary:0:2);
                             WRITELN(Title[9], Area)
                           END
                    END; { Конец структуры CASE }
                  END
              END { Конец блока WITH }
          END;
        CLOSE(PayrollFile)
      END
    ELSE {Если не найден }
      BEGIN
        WRITELN('SSN не найден в файле.');
        WRITELN('Пожалуйста, попытайтесь снова.');
        WRITELN
      END
END;
{ ------------------------ Процедура DelRec ------------------------- }
PROCEDURE DelRec(VAR NewFile, PayrollFile: TEXT;
                 Employee: EmployeeRecord);
VAR
  SSNumber: STRING[11];
  Found   :INTEGER;
BEGIN
  WRITE('Пожалуйста, введите номер социального страхования удаляемого
сотрудника: ');
```

```
  READLN(SSNumber);
  SearchRec(PayrollFile, EmployeeRec, SSNumber, Found);
  IF Found =1 THEN
    BEGIN
      ASSIGN(NewFile, TempFile);
      REWRITE(NewFile);
      ASSIGN(PayrollFile, FileName);
      RESET(PayrollFile);
      WHILE NOT EOF(PayrollFile) DO
        BEGIN
          WITH Employee DO
            BEGIN
              READLN(PayrollFile, ID);
              READLN(PayrollFile, Name);
              READLN(PayrollFile, Position);
              READLN(PayrollFile, SSN);
              READLN(PayrollFile, Category);
              CASE Category OF
                '1': READLN(PayrollFile, MonthlySalary);
                '2': READLN(PayrollFile, HourlyRate);
                '3': BEGIN
                       READLN(PayrollFile, Commission);
                       READLN(PayrollFile, BasicSalary);
                       READLN(PayrollFile, Area)
                     END
              END; { Конец структуры CASE }
              IF SSNumber <> SSN THEN
                BEGIN
                  WRITELN(NewFile, ID);
                  WRITELN(NewFile, Name);
                  WRITELN(NewFile, Position);
                  WRITELN(NewFile, SSN);
                  WRITELN(NewFile, Category);
                  CASE Category OF
                    '1': WRITELN(NewFile, MonthlySalary:0:2);
                    '2': WRITELN(NewFile, HourlyRate:0:2);
                    '3': BEGIN
                           WRITELN(NewFile, Commission:0:2);
                           WRITELN(NewFile, BasicSalary:0:2);
                           WRITELN(NewFile, Area)
                         END
                  END; { Конец структуры CASE }
                END;
            END { Конец блока WITH }
        END; {Конец DO }
      CLOSE(NewFile);
      CLOSE(PayrollFile);
{ Копирование NewFile обратно в файл платежной ведомости }
      ASSIGN(PayrollFile, FileName);
      REWRITE(PayrollFile);
      ASSIGN(NewFile, TempFile);
```

```
      RESET(NewFile);
      WHILE NOT EOF(NewFile) DO
        BEGIN
          READLN(NewFile, OneLine);
          WRITELN(PayrollFile, OneLine)
        END;
      CLOSE(NewFile);
      ERASE(NewFile); { Удаление временного файла }
      CLOSE(PayrollFile);
{ Сообщения пользователя }
      WRITELN('Сотрудник ', SSNumber,' удален из файла.')
    END { Конец блока «IF Found..» }
  ELSE {Если не найден }
    BEGIN
      WRITELN('Номер социального страхования ', SSNumber, ' не найден.');
      WRITELN('Проверьте номер и попытайтесь снова.');
      WRITELN
    END
END;

{ -------------------------- Процедура AddRec ----------------------- }
PROCEDURE AddRec(VAR NewFile, PayrollFile: TEXT;
                 Employee: EmployeeRecord);
BEGIN
  ASSIGN(PayrollFile, FileName);
  RESET(PayrollFile);
  ASSIGN(NewFile, TempFile);
  REWRITE(NewFile);
  WHILE NOT EOF(PayrollFile) DO
    BEGIN
{ Копирование каждой записи из PayrollFile в NewFile }
      READLN(PayrollFile, OneLine);
      WRITELN(NewFile, OneLine)
    END;
{Ввод новой записи с клавиатуры }
  WITH Employee DO
    BEGIN
      WRITE('Пожалуйста, введите ID сотрудника: ');
      READLN(ID);
      WRITE('Имя: '); READLN(Name);
      WRITE('Должность: '); READLN(Position);
      WRITE('Номер социального страхования (xxx-xx-xxxx): '); READLN(SSN);
      WRITE('Категория оплаты: '); READLN(Category);
      CASE Category OF
        '1': BEGIN
               WRITE('Месячный оклад: ');
               READLN(MonthlySalary)
             END;
        '2': BEGIN
               WRITE('Тариф: ');
               READLN(HourlyRate)
```

```
            END;
        '3': BEGIN
               WRITE('Комиссионные: ');
               READLN(Commission);
               WRITE('Основной оклад: ');
               READLN(BasicSalary);
               WRITE('Регион: ');
               READLN(Area)
             END
      END;
{ Запись информации в новый файл }
      WRITELN(NewFile, ID);
      WRITELN(NewFile, Name);
      WRITELN(NewFile, Position);
      WRITELN(NewFile, SSN);
      WRITELN(NewFile, Category);
      CASE Category OF
        '1': WRITELN(NewFile, MonthlySalary:0:2);
        '2': WRITELN(NewFile, HourlyRate:0:2);
        '3': BEGIN
               WRITELN(NewFile, Commission:0:2);
               WRITELN(NewFile, BasicSalary:0:2);
               WRITELN(NewFile, Area)
             END
      END
    END;
  CLOSE(NewFile);
  CLOSE(PayrollFile);
{ Копирование NewFile обратно в файл платежной ведомости }
  ASSIGN(PayrollFile, FileName);
  REWRITE(PayrollFile);
  ASSIGN(NewFile, TempFile);
  RESET(NewFile);
  WHILE NOT EOF(NewFile) DO
    BEGIN
      READLN(NewFile, OneLine);
      WRITELN(PayrollFile, OneLine)
    END;
  CLOSE(NewFile);
  ERASE(NewFile); { Удаление временного файла }
  CLOSE(PayrollFile)
END;

{ ---------------------- Процедура UpdateRec ------------------------- }
PROCEDURE UpdateRec(VAR NewFile, PayrollFile: TEXT;
                    Employee: EmployeeRecord);
VAR
  SSNumber :STRING[11];
  Found    :INTEGER;
BEGIN
```

```
  WRITE('Пожалуйста, введите номер социального страхования обновляемого
сотрудника: ');
  READLN(SSNumber);
  SearchRec(PayrollFile, EmployeeRec, SSNumber, Found);
  IF Found =1 THEN
    BEGIN
      ASSIGN(PayrollFile, FileName);
      RESET(PayrollFile);
      ASSIGN(NewFile, TempFile);
      REWRITE(NewFile);
      WHILE NOT EOF(PayrollFile) DO
        BEGIN
          WITH Employee DO
            BEGIN
              READLN(PayrollFile, ID);
              READLN(PayrollFile, Name);
              READLN(PayrollFile, Position);
              READLN(PayrollFile, SSN);
              READLN(PayrollFile, Category);
              CASE Category OF
                '1': READLN(PayrollFile, MonthlySalary);
                '2': READLN(PayrollFile, HourlyRate);
                '3': BEGIN
                       READLN(PayrollFile, Commission);
                       READLN(PayrollFile, BasicSalary);
                       READLN(PayrollFile, Area)
                     END
              END; { Конец структуры CASE }
              IF SSNumber <> SSN THEN
                BEGIN
                  WRITELN(NewFile, ID);
                  WRITELN(NewFile, Name);
                  WRITELN(NewFile, Position);
                  WRITELN(NewFile, SSN);
                  WRITELN(NewFile, Category);
                  CASE Category OF
                    '1': WRITELN(NewFile, MonthlySalary:0:2);
                    '2': WRITELN(NewFile, HourlyRate:0:2);
                    '3': BEGIN
                           WRITELN(NewFile, Commission:0:2);
                           WRITELN(NewFile, BasicSalary:0:2);
                           WRITELN(NewFile, Area)
                         END
                  END { Конец структуры CASE }
                END { Конец блока IF }
              ELSE
                BEGIN
                  WRITELN('Пожалуйста, введите новую информацию:');
                  WRITE('ID: '); READLN(ID);
                  WRITELN(NewFile, ID);
                  WRITE('Имя: '); READLN(Name);
```

```
                    WRITELN(NewFile, Name);
                    WRITE('Должность: '); READLN(Position);
                    WRITELN(NewFile, Position);
                    WRITELN(NewFile, SSN);
                    WRITE('Категория: '); READLN(Category);
                    WRITELN(NewFile, Category);
                    CASE Category OF
                      '1': BEGIN
                             WRITE('Зарплата: ');
                             READLN(MonthlySalary);
                             WRITELN(NewFile, MonthlySalary:0:2)
                           END;
                      '2': BEGIN
                             WRITE('Почасовой тариф: ');
                             READLN(HourlyRate);
                             WRITELN(NewFile, HourlyRate:0:2)
                           END;
                      '3': BEGIN
                             WRITE('Комиссионные: ');
                             READLN(Commission);
                             WRITELN(NewFile, Commission:0:2);
                             WRITE('Основной оклад: ');
                             READLN(BasicSalary);
                             WRITELN(NewFile, BasicSalary:0:2);
                             WRITE('Регион: ');
                             READLN(Area);
                             WRITELN(NewFile, Area)
                           END
                    END { Конец структуры CASE }
                  END { Конец блока ELSE }
              END { Конец блока WITH }
          END; { Конец DO }
        CLOSE(NewFile);
        CLOSE(PayrollFile);
{ Копирование NewFile обратно в файл платежной ведомости }
        ASSIGN(PayrollFile, FileName);
        REWRITE(PayrollFile);
        ASSIGN(NewFile, TempFile);
        RESET(NewFile);
        WHILE NOT EOF(NewFile) DO
          BEGIN
            READLN(NewFile, OneLine);
            WRITELN(PayrollFile, OneLine)
          END;
        CLOSE(NewFile);
        ERASE(NewFile); { Удаление временного файла }
        CLOSE(PayrollFile);
{ Сообщения пользователя }
        WRITELN('Запись сотрудника ', SSNumber, ' обновлена.')
      END { Конец блока IF }
    ELSE
```

```
    BEGIN
      WRITELN('Номер социального страхования ', SSNumber, ' не найден.');
      WRITELN('Проверьте номер и попытайтесь снова.');
      WRITELN
    END
END;

{ -------------------------- Процедура Menu ------------------------- }
PROCEDURE Menu;
VAR
  Option: INTEGER;
BEGIN
  WRITELN(Header);
  WRITELN;
  WRITELN('1. Показать запись сотрудника.');
  WRITELN('2. Добавить нового сотрудника.');
  WRITELN('3. Удалить сотрудника.');
  WRITELN('4. Обновить запись сотрудника.');
  WRITELN('5. Выход.');
  WRITELN(Separator);
  WRITE('Сделайте выбор и нажмите цифру: ');
  READLN(Option);
  CASE Option OF
    1: ReadRec(PayrollFile, EmployeeRec);
    2: AddRec(NewFile, PayrollFile, EmployeeRec);
    3: DelRec(NewFile, PayrollFile, EmployeeRec);
    4: UpdateRec(NewFile, PayrollFile, EmployeeRec);
    5: Exit
  END;
  Menu
END;

{ -------------------------- Главная программа ------------------------- }
BEGIN
{ Присваивание заголовков }
  Title[1] := 'ID: ';
  Title[2] := 'Имя: ';
  Title[3] := 'Должность: ';
  Title[4] := 'SSN: ';
  Title[5] := 'Зарплата: ';
  Title[6] := 'Тариф: ';
  Title[7] := 'Комиссионные: ';
  Title[8] := 'Основной оклад: ';
  Title[9] := 'Регион: ';
  Menu
END.
```

Предложения

Для того чтобы сделать программу более надежной, можно добавить следующие функции:

- При вводе номера социального страхования нового сотрудника программа не проверяет формат данных; таким образом, может быть принят неверный номер, такой как «12345-678». Можно добавить новые операторы для проверки как цифр, так и расположения дефисов.
- При вводе значений, отличных от 1, 2 или 3, они будут приняты как Category. Соответственно, зарплата не будет обработана структурой CASE. Можно добавить необходимые шаги для проверки правильности значений переменной Category.
- Если добавить какого-то сотрудника в файл дважды, то программа не будет знать, что запись уже существует. Поэтому перед добавлением новой записи следует проверять номер социального страхования.
- Работая в Турбо Паскале, лучше работать с модулями (*units*). Тогда каждую процедуру можно поместить в отдельный файл, называемый unit. Эти файлы могут быть использованы несколькими программами.

Эти усовершенствования оставлены вам в качестве заданий.

Заключение

1. В этой главе вы научились использовать вариантную запись, чтобы хранить данные в эффективной структуре.
2. Вариантную запись можно объявлять в следующем формате:

```
имя-типа = RECORD
    фиксированный список-полей
    вариантный список-полей
END;
```

   Вариантный список полей имеет вид:

```
CASE поле-метки: определение-типа OF
    метка-1: (список-полей: определение-типа);
    метка-2: (список-полей: определение-типа);
    ...
метка-n: (список-полей: определение-типа);
```

3. Вариантная запись может включать фиксированную часть, стоящую перед вариантной частью, или только вариантную часть.
4. Вы научились извлекать, заносить, обновлять и удалять с помощью структуры CASE хранимые в файле вариантные записи.

Упражнения

1. Напишите объявление вариантной записи для пациента, которая содержит следующую информацию:

- ID пациента
- Имя пациента
- Номер социального страхования
- Статус курильщика, который включает три случая:
  - курит (Y): требуется информация о виде (сигареты, сигары, трубка и т. д.) и количестве лет курения;
  - не курит (N): дополнительная информация не требуется;
  - бросает курить (Q): требуется информация о количестве лет курения и количестве лет с момента, как начал бросать.

2. Напишите объявление вариантной записи геометрического тела, которая может быть использована для вычисления площади поверхности и/или объема следующих тел:

Форма	Требуемая информация
Окружность	Радиус
Цилиндр	Радиус и высота
Сфера	Радиус
Куб	Сторона

Ответы

1.
```
PatientRecord = RECORD
                  PatientID            :STRING[5];
                  Name                 :STRING[20];
                  SSN                  :STRING[11];
                  CASE Smoking         : CHAR OF
                    'Y': (TypeOfSmoke  : STRING[10];
                          HowLong      : INTEGER);
                    'N': ();
                    'Q': (YearsQuitting : INTEGER;
                          YearsSmoking  : INTEGER)
                END;
```

2.
```
FigureName = (Circle, Cylinder, Sphere, Cube);
ShapeInformation = RECORD
                     CASE    Figure   : FigureName OF
                             Circle   :(Radius : REAL);
                             Cylinder :(CylRadius: REAL;
                                        Height: REAL );
                             Sphere   :(SphRadius: REAL);
                             Cube     :(Side: REAL)
                     END;
```

11

Указатели и связанные списки

11–1. Динамическое выделение памяти

До сих пор мы имели дело с переменными, называемыми *статическими*. Связь между статической переменной и ее содержимым в памяти устанавливается во время компиляции и не меняется в течение выполнения программы. Напротив, *динамическая переменная* создается или размещается во время выполнения программы. Другими словами, необходимое пространство в памяти для динамической переменной выделяется во время работы программы и может быть освобождено и передано другой переменной. В Паскале можно создавать простые динамические переменные и сложные *динамические структуры данных*, такие как *связанные списки*. Связанные списки могут понадобиться, когда невозможно заранее определить требования к памяти. Противоположностью связанного списка выступает массив как пример *статической структуры данных*. Память для соответствующих элементов массива выделяется во время компиляции. Недостаток массива состоит в необходимости выделять достаточное пространство для его элементов. Определение огромного массива, превосходящего действительные потребности, ведет к напрасному расходованию памяти, а небольшой массив ограничит программу определенным количеством элементов. Затруднение возникает, когда требуется вставить в массив новый элемент. Динамические структуры данных могут расширяться и сжиматься во время выполнения программы, так же ведет себя и связанная с ними область памяти. Динамическое выделение памяти выполняется с помощью указателей. В следующем разделе вы научитесь объявлять и использовать указатели.

11–2. Указатели

Указатель – это специальный тип переменной, которая не содержит данных, а хранит их адреса в памяти. Поэтому говорят, что указатель

«ссылается» на местоположение данных. В программе можно переопределить указатель, чтобы он ссылался на другую область памяти или в никуда. Также можно освободить область памяти, связанную с указателем, и передать ее другой переменной. Указатели могут ссылаться на любой тип данных от CHAR и INTEGER до сложных структур данных, таких как записи и связанные списки.

Указатель на целое объявляется следующим образом:

```
VAR
  PtrVariable : ^INTEGER;
```

Для того чтобы использовать указатель, надо выделить память с помощью процедуры NEW:

```
NEW(PtrVariable);
```

Она присваивает адрес памяти переменной-указателю `PtrVariable` (например, 709Ch). Значение, хранимое по этому адресу, представляет собственно данные. К области памяти, на которую ссылается указатель `PtrVariable`, можно обращаться с помощью имени переменной:

```
PtrVariable^
```

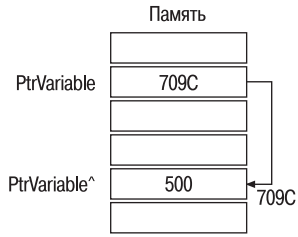
трактуемой как обычная переменная. Например, можно присвоить ей численное значение:

```
PtrVariable^ := 500;
```

Или присвоить ее другой статической переменной:

```
AnotherVariable := PtrVariable^ ;
```

Это демонстрирует следующая диаграмма:



Память, выделенная указателю, может быть освобождена с помощью процедуры DISPOSE:

```
DISPOSE(PtrVariable);
```

После освобождения памяти указатель становится неопределенным.

Все эти особенности показаны в следующей программе. Объявляются целая переменная (`MyInteger`) и целый указатель (`MyIntegerPointer`). Со-

держимое памяти MyIntegerPointer^ присваивается переменной MyInteger. Оба объекта при выводе на печать должны выдать одно и то же значение (500).

```
{ ---------------------------- Пример 11-1 ------------------------------- }
PROGRAM PointerExample(OUTPUT);
VAR
  MyIntegerPointer :^INTEGER;
  MyInteger :INTEGER;
BEGIN
  MyInteger := 50;
  NEW(MyIntegerPointer);
  MyIntegerPointer^ := 500;
  MyInteger := MyIntegerPointer^;
  WRITELN('Значение MyIntege: ', MyInteger);
  WRITELN('Значение, на которое указывает MyIntegerPointer:
',MyIntegerPointer^);
  DISPOSE(MyIntegerPointer);
  WRITELN('Для продолжения нажмите любую клавишу...');
  READLN
END.
```

Заметим, что в этой программе нет необходимости в процедуре DISPOSE, поскольку память автоматически освобождается при выходе из программы. Процедура нужна только для демонстрации.

Таким же образом можно объявлять указатели на другие типы данных, например:

```
VAR
  MyCharPointer :^CHAR;
  MyStringPointer :^STRING;
  MyRealPointer :^REAL;
```

Перед тем как использовать любой из этих указателей, не забудьте применить процедуру NEW для размещения памяти для каждого указателя:

```
NEW(MyCharPointer);
NEW(MyStringPointer);
NEW(MyRealPointer);
```

После окончания работы можно освободить связанную с указателями память с помощью DISPOSE:

```
DISPOSE(MyCharPointer);
DISPOSE(MyStringPointer);
DISPOSE(MyRealPointer);
```

Операции с указателями

Операции над указателями ограничены присваиванием и сравнением.

Присваивание

Если два указателя ptr1 и ptr2 имеют одинаковый тип, то следующий оператор имеет силу:

```
ptr1 := ptr2;
```

Действие этого оператора заключается в перенаправлении указателя ptr1 на ту же область памяти, на которую ссылается указатель ptr2. Область памяти, на которую перед присваиванием ссылался указатель ptr1, теперь недоступна (если на нее не ссылается другой указатель).

При освождении этих указателей надо освободить только один из них, поскольку они ссылаются на одну и ту же область. Второй оператор DISPOSE вызовет сообщение об ошибке.

Очевидно, что предыдущее присваивание совершенно отличается от следующего:

```
ptr1^ := ptr2^;
```

которое означает копирование содержимого области, на которую указывает ptr2, в область, на которую указывает ptr1. Тем не менее оба указателя могут продолжать ссылаться на ту же самую исходную область.

Сравнение

Два указателя можно сравнить при помощи логических операторов = или <>, например:

```
IF ptr1 = ptr2 THEN ..
```

Логическое выражение ptr1 = ptr2 принимает значение TRUE, если два указателя ссылаются на одну и ту же область памяти.

Эти особенности демонстрирует следующая программа:

```
{ -------------------- Пример 11-2 ----------------------- }
PROGRAM PointerExample2(OUTPUT);
TYPE
  intptr = ^INTEGER;
  realptr = ^REAL;
VAR
  MyIntegerPointer, AnotherIntPointer :intptr;
  MyRealPointer                       :realptr;
BEGIN
  NEW(MyIntegerPointer);
  NEW(MyRealPointer);
  NEW(AnotherIntPointer);
  MyRealPointer^ := 2.25;
  MyIntegerPointer^ := 500;
  AnotherIntPointer^ := 400;
{ Копирование содержимого памяти:}
  MyRealPointer^ := MyIntegerPointer^;
```

```
  { Перенаправление MyIntegerPointer:}
  MyIntegerPointer := AnotherIntPointer;
  WRITELN('MyRealPointer указывает на: ', MyRealPointer^:2:2);
  WRITELN;
{Проверка ссылки двух указателей на одну область:}
  IF (MyIntegerPointer = AnotherIntPointer) THEN
    WRITELN('Да, два указателя ссылаются на ту же самую область.');
  WRITELN('MyIntegerPointer указывает на: ', MyIntegerPointer^);
  WRITELN('AnotherIntPointer указывает на: ', AnotherIntPointer^);
  WRITELN;
{ Примечание: Для любого указателя данной программы процедура DISPOSE
необязательна. }
  DISPOSE(MyIntegerPointer);
  DISPOSE(MyRealPointer);
  {DISPOSE(AnotherIntPointer);} {теперь это неверно..}
  WRITELN('Для продолжения нажмите любую клавишу...');
  READLN
END.
```

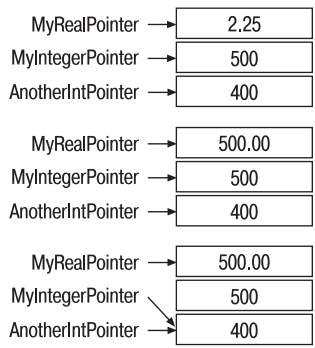
После запуска программа выведет на экран следующие сообщения:

```
MyRealPointer указывает на: 500.00
Да, два указателя ссылаются на ту же самую область.
MyIntegerPointer указывает на: 400
AnotherIntPointer указывает на: 400

Для продолжения нажмите любую клавишу...
```

Далее приведены операции, выполняемые в программе (см. также рисунок).

1. Переменной `MyRealPointer^` присваивается значение 2.25.
2. Переменной `MyIntegerPointer^` присваивается значение 500.
3. Переменной `AnotherIntPointer^` присваивается значение 400.
4. Содержимое `MyIntegerPointer^` копируется в `MyRealPointer^`; поэтому оно становится равным 500.00.
5. Указатель `MyIntegerPointer` перенаправляется на область, на которую указывает `AnotherIntPointer` и которая хранит значение 400.



Нельзя читать значение указателя. Разрешено читать только содержимое, на которое ссылается указатель.

Указатели на записи

Можно объявить указатель на запись таким же образом, как и для других типов данных. Предпочтительнее сначала определить тип указателя в секции TYPE:

```
TYPE
  emprec = RECORD
             ID   :INTEGER;
             Wage : REAL;
           END;
  empptr = ^emprec;
```

а затем объявить переменную-указатель в секции VAR:

```
VAR
  ptr1, ptr2 :empptr;
```

В случае записей разрешены любые операции над указателями, как и для других типов данных. Но есть некоторые ограничения:

- Будучи созданным, указатель на запись связывается с определенным типом записи и не может быть использован с другим типом записи.
- Выражение отношения:

  ```
  ptr1^ = ptr2^
  ```

  неверно, поскольку записи нельзя сравнивать с помощью выражений отношения. Однако можно сравнивать два указателя, проверяя, не ссылаются ли они на одну и ту же запись.

  ```
  ptr1 = ptr2
  ```

  или

  ```
  ptr1 <> ptr2
  ```

- Доступ к содержимому полей производится с помощью полевых переменных:

  ```
  tr1^.ID
  ptr1^.Wage
  ptr2^.ID
  ptr2^.Wage
  ```

  или с помощью оператора WITH:

  ```
  WITH ptr1^ DO
    ID := 123;
    Wage := 22.5;
    ...
  ```

В следующей программе для доступа к полям записи используются два указателя ptr1 и ptr2, затем один из указателей перенаправляется в точку записи, на которую ссылается другой указатель.

```
{ -------------------- Пример 11-3 ----------------------- }
PROGRAM PointersToRecords(OUTPUT);
TYPE
  emprec = RECORD
             ID  :INTEGER;
             Wage: REAL;
           END;
  empptr = ^emprec;
VAR
  ptr1, ptr2 : empptr;
BEGIN
  NEW(ptr1);
  NEW(ptr2);
{ Присвоение значений полям }
  ptr1^.ID := 123;
  ptr1^.Wage := 25.5;
  ptr2^.ID := 456;
  ptr2^.Wage := 33.25;
{Печать содержимого:}
  WRITELN('До перенаправления ptr1:');
  WRITELN('Ptr1 указывает на ID:', ptr1^.ID,', и Wage: $', ptr1^.Wage:2:2);
  WRITELN('Ptr2 указывает на ID:', ptr2^.ID,', и Wage: $', ptr2^.Wage:2:2);
{Перенаправление ptr1:}
  ptr1 := ptr2;
{Печать содержимого:}
  WRITELN;
  WRITELN('После перенаправления ptr1:');
  WRITELN('Ptr1 указывает на ID:', ptr1^.ID,', и Wage: $', ptr1^.Wage:2:2);
  WRITELN('Ptr2 указывает на ID:', ptr2^.ID,', и Wage: $', ptr2^.Wage:2:2);
  WRITELN('Для продолжения нажмите любую клавишу...');
  READLN
END.
```

В результате выполнения программы на экране появятся следующие результаты:

```
До перенаправления ptr1:
Ptr1 указывает на ID: 123, и Wage: $25.50
Ptr2 указывает на ID: 456, и Wage: $33.25

После перенаправления ptr1:
Ptr1 указывает на ID: 456, и Wage: $33.25
Ptr2 указывает на ID: 456, и Wage: $33.25
Для продолжения нажмите любую клавишу...
```

Задание 11-1

Напишите программу, которая с помощью указателя на запись сотрудника создавала бы файл платежной ведомости, с которым мы работали в главе 10. Можете модифицировать файл Drl10-1.pas, находящийся на прилагаемой дискете.

Передача указателей в качестве параметров

Можно передавать указатели как параметры в функции и процедуры таким же образом, как и статические переменные. Можно передавать указатели или по значению, или как переменные параметры. Помните, что и фактические и формальные параметры должны иметь один и тот же тип; если это указатели на записи, то они должны ссылаться на тот же тип записи.

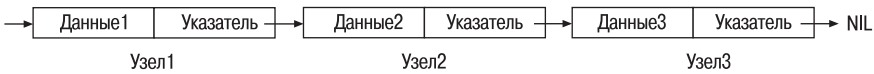
Задание 11-2

Для того чтобы применить указатели к процедурам, внесите необходимые изменения в программу 10-02.pas на прилагаемой дискете.

11-3. Основы связанных списков

Связанный список – это набор элементов данных, называемых *узлами*. Каждый узел содержит указатель на следующий узел. Связанный список может хранить любой тип данных, но обычно используется для хранения записей. Следующая диаграмма демонстрирует связанный список.

Как видно из диаграммы, каждый узел в связанном списке содержит два элемента, данные и указатель на следующий узел. Указатель последнего узла ссылается на константу NIL, которая обозначает конец списка.



Объявление списка

Приведем объявление связанного списка, содержащего поле данных целого типа:

```
TYPE
  ListPointer = ^ListRecord;
  ListRecord = RECORD
```

```
    DataField :INTEGER;
    NextField :ListPointer;
  END;
```

Каждый узел в связанном списке (ListRecord) имеет структуру записи. Он содержит *поле данных* (DataField), содержащее фактические данные, и *поле указателя* (NextField), также известное как *поле связи*, которое хранит порядок следования в списке. Обратите внимание, что определение указателя (ListPointer) предшествует определению записи (ListRecord). Это единственный случай, когда разрешено использовать идентификатор до того, как он определен. Точно так же можно объявить связанный список для хранения записей, но лучше начать с более простых списков.

Построение списка

Для построения связанного списка необходимо объявить два указателя – один для ссылки на первый узел (то есть `FirstPointer`), а второй для временного использования в процессе построения (то есть `ToolPointer`). Очевидно, оба указателя имеют тип ListPointer.

```
VAR
  FirstPointer, ToolPointer :ListPointer;
```

Этапы построения списка:

1. Инициализировать пустой список путем присвоения указателю `FirstPointer` значения NIL:

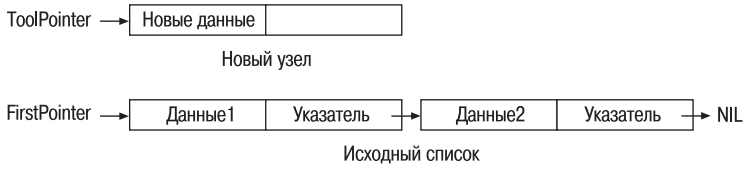
```
FirstPointer := NIL;
```

2. Создать узел с помощью временного указателя:

```
NEW(ToolPointer);
```

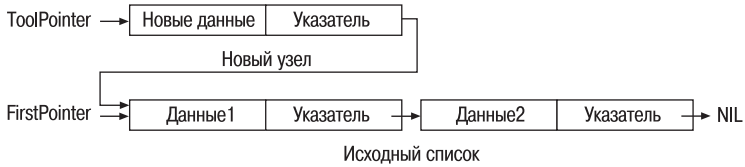
3. Ввести целое значение с клавиатуры (или другого носителя) и занести его в поле данных этого узла:

```
READLN(ToolPointer^.DataField);
```



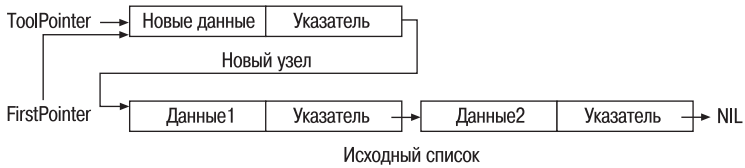
4. Добавить узел в список, установив поле указателя так, чтобы оно ссылалось туда же, куда и `FirstPointer`:

```
ToolPointer^.NextField := FirstPointer;
```



5. Перенаправить `FirstPointer` на новый узел (который является началом списка):

```
FirstPointer := ToolPointer;
```



Повторяйте предыдущие шаги до тех пор, пока все данные не будут прочитаны. Последний прочитанный элемент данных будет находиться в первом узле связанного списка.

Добавление нового узла в существующий список выполняется в том же порядке, за исключением первого шага (поскольку не надо создавать пустой список).

Ниже приводится фрагмент процедуры создания списка:

```
FirstPointer := NIL;
WHILE { Логическое выражение } DO
  BEGIN
    NEW(ToolPointer);
    READLN(ToolPointer^.DataField);
    ToolPointer^.NextField := FirstPointer;
    FirstPointer := ToolPointer;
  END;
```

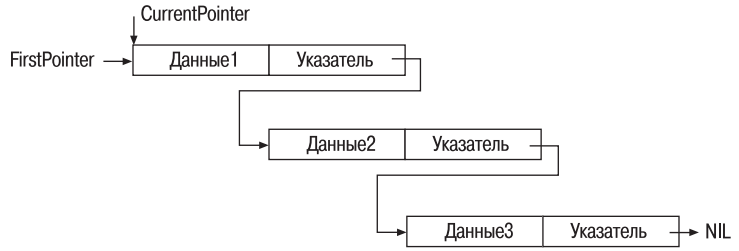
Чтение списка

Для чтения связанного списка необходимы два указателя: `FirstPointer` – указывает на первый узел списка и `CurrentPointer`, который перемещается от узла к узлу вдоль всего списка.

Этапы чтения и вывода на экран содержимого списка:

1. Направить `CurrentPointer` на первый узел путем присвоения ему адреса, содержащегося в `FirstPointer`:

```
CurrentPointer := FirstPointer;
```

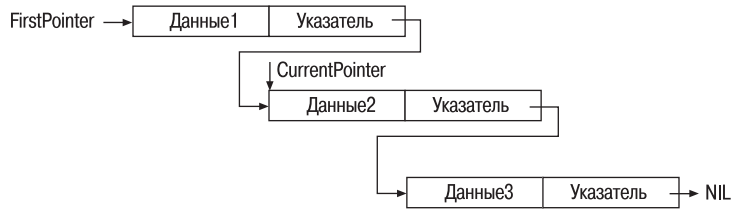


2. Использовать `CurrentPointer` для доступа к полю данных и вывода на экран его содержимого:

```
WRITELN(CurrentPointer^.DataField);
```

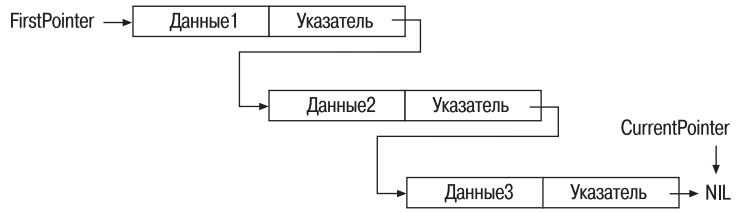
3. Переместить `CurrentPointer` на следующий узел, присвоив ему адрес, содержащийся в поле указателя (NextField) текущего узла:

```
CurrentPointer := CurrentPointer^.NextField;
```



4. Повторять шаги 2 и 3, пока не доберетесь до последнего узла. Это произойдет, когда следующее условие примет значение TRUE:

```
CurrentPointer = NIL
```



Фрагмент программы чтения списка:

```
VAR
  CurrentPointer :ListPointer;
BEGIN
  CurrentPointer := FirstPointer;
  WHILE CurrentPointer <> NIL DO
    BEGIN
      WRITELN(CurrentPointer^.DataField);
      CurrentPointer := CurrentPointer^.NextField
```

```
    END;
  WRITELN
END
```

Пример: Демонстрация связанного списка

В следующей программе вы построите связанный список для хранения имен людей, а затем прочитаете и выведете на экран его содержимое. Программа содержит следующие процедуры:

- Menu: для отображения и ввода опций пользователя
- GetData: для приема данных с клавиатуры
- BuildList: для чтения содержимого списка
- DisplayInfo: для вывода списка на экран

```
{ ---------------------------- Пример 11-4 ------------------------------- }
PROGRAM LinkedListDemo(INPUT, OUTPUT);
CONST
  Header = '------------- Главное меню -------------';
  Separator = '----------------------------------------';
TYPE
  DataString = STRING[30];
  ListPointer = ^ListRecord;
  ListRecord = RECORD
                 DataField :DataString;
                 NextField :ListPointer
               END;
VAR
  FirstPointer :ListPointer;

{ --------------------------- Процедура BuildList ----------------------- }
PROCEDURE BuildList(VAR FirstPointer :ListPointer;
                    DataItem :DataString);
{Примечание: FirstPointer передается с помощью ключевого
слова VAR, поскольку он будет изменен этой процедурой.}
VAR
  ToolPointer :ListPointer;
BEGIN
  NEW(ToolPointer);
  ToolPointer^.DataField := DataItem;
  ToolPointer^.NextField := FirstPointer;
  FirstPointer := ToolPointer
END;

{ -------------------------- Процедура ReadList ------------------------- }
PROCEDURE ReadList(FirstPointer :ListPointer);
VAR
  CurrentPointer :ListPointer;
BEGIN
  CurrentPointer := FirstPointer;
  WHILE CurrentPointer <> NIL DO
    BEGIN
```

```
      WRITELN(CurrentPointer^.DataField);
      CurrentPointer := CurrentPointer^.NextField
    END;
  WRITELN
END;

{ ------------------------- Процедура GetData--------------------------- }
PROCEDURE GetData(VAR FirstPointer :ListPointer);
{ Примечание: FirstPointer передается с помощью ключевого
слова VAR, поскольку он будет изменен после передачи в процедуру BuildList.}
VAR
  Name :DataString;
BEGIN
  WRITELN('Введите имена, добавляемые в список,', по окончании нажмите
ENTER.');
{ Чтение первого элемента данных }
  READLN(Name);
{ Проверка конца данных }
  WHILE LENGTH(Name) <> 0 DO
    BEGIN
      BuildList(FirstPointer, Name);
      READLN(Name)
    END
END;

{ ------------------------ Процедура DisplayInfo ----------------------- }
PROCEDURE DisplayInfo(FirstPointer :ListPointer);
BEGIN
  WRITELN(Separator);
  WRITELN('Содержимое списка: ');
  ReadList(FirstPointer);
  WRITE('Для продолжения нажмите любую клавишу...');
  READLN
END;

{ -------------------------- Процедура Menu ---------------------------- }
PROCEDURE Menu;
VAR
  Option :INTEGER;
BEGIN
  WRITELN(Header);
  WRITELN('1. Занести данные в список.');
  WRITELN('2. Вывести список на экран.');
  WRITELN('3. Выход.');
  WRITELN(Separator);
  WRITE('Сделайте выбор и нажмите цифру: ');
  READLN(Option);
  CASE Option OF
    1 : GetData(FirstPointer);
    2 : DisplayInfo(FirstPointer);
    3 : Exit
  END;
```

```
  Menu
END;

{ ---------------------------- Главная программа ------------------------}
BEGIN
{ Инициализация пустого списка }
  FirstPointer := NIL;
  menu
END.
```

Запустив программу и решив занести данные в список (опция 1), вы должны будете ввести некоторые имена, а затем закончить ввод, нажав клавишу <Enter> (не вводя никакого текста). В этот момент список создается в памяти и может быть выведен на экран. Обратите внимание, что имя, введенное с клавиатуры последним, на экране появится первым, поскольку узел всегда вставляется в начало списка. В следующем тесте данные, введенные пользователем, для ясности показаны выделенным шрифтом с наклоном.

Пример выполнения:

```
----------- Главное меню -------------
1. Занести данные в список.
2. Вывести список на экран.
3. Выход.
--------------------------------------
Сделайте выбор и нажмите цифру: 1
Введите имена, добавляемые в список, по окончании нажмите ENTER.
John Smith          <ENTER> -----> Имена, введенные с клавиатуры
Jean Murdock        <ENTER>
Sally Bedford       <ENTER>
Deanna Loerwold     <ENTER>
<ENTER>
----------- Главное меню -------------
1. Занести данные в список.
2. Вывести список на экран.
3. Выход.
--------------------------------------
Сделайте выбор и нажмите цифру: 2
--------------------------------------
Содержимое списка:
Deanna Loerwold       -----> Обратите внимание на последовательность имен
Sally Bedford
Jean Murdock
John Smith

Для продолжения нажмите любую клавишу...
----------- Главное меню -------------
1. Занести данные в список.
2. Вывести список на экран.
3. Выход.
--------------------------------------
Сделайте выбор и нажмите цифру: 3
```

В этой программе обратите внимание на следующее:

- Ключевое слово VAR используется в процедуре BuildList, поскольку она изменяет адрес в указателе с помощью оператора:

```
FirstPointer := ToolPointer;
```

  Процедура GetData не обновляет `FirstPointer` явно, а передает его процедуре BuildList; поэтому ключевое слово VAR все же необходимо.

- Заметьте, что пустой список инициализируется только один раз в главной программе:

```
FirstPointer := NIL;
```

  Это означает, что можно продолжать вводить элементы данных в тот же список, если еще раз выбрать опцию 1. Список повторно инициализируется, если выйти из программы и запустить ее снова. Для того чтобы пустой список инициализировался при выборе опции 1, надо переместить оператор FirstPointer := NIL в процедуру GetData.

- В программе были задействованы три указателя, но процедура NEW использует только `ToolPointer`. Эта процедура необходима только для выделения памяти при создании узлов.

Хранение списков в файлах

Для сохранения связанного списка в файле надо выполнить следующие шаги:

1. Открыть файл на запись.
2. Направить `CurrentPointer` на первый узел:

```
CurrentPointer := FirstPointer;
```

3. Прочитать поле данных (CurrentPointer^.DataField) и записать его в файл:

```
WRITE(MyListFile, CurrentPointer^.DataField);
```

4. Переместить `CurrentPointer` на следующий узел, заменив его адрес содержимым поля указателя (NextField):

```
CurrentPointer := CurrentPointer^.NextField;
```

5. Повторять шаги 3 и 4, пока не будет достигнут конец списка. В этот момент CurrentPointer примет значение NIL.
6. Закрыть файл.

Следующий фрагмент программы объединяет приведенные выше шаги:

```
VAR
  CurrentPointer :ListPointer;
```

```
BEGIN
  ASSIGN(MyListFile, FileName);
  REWRITE(MyListFile);
  CurrentPointer := FirstPointer;
  WHILE CurrentPointer <> NIL DO
    BEGIN
      WRITE(MyListFile, CurrentPointer^.DataField);
      CurrentPointer := CurrentPointer^.NextField
    END;
  CLOSE(MyListFile)
END;
```

Чтение списков из файлов

При записи связанного списка в файл сохраняются только данные. Указатели списка существуют только в памяти для управления списком. Поэтому после записи файла на диск он становится обычным файлом данных и может быть прочитан в соответствии с обычной процедурой. После чтения файла необходимо построить данные в виде связанного списка. Для занесения прочитанных из файла данных в связанный список надо сделать следующее:

1. Открыть файл на чтение.
2. Прочитать элемент данных из файла.
3. Добавить элемент данных в список с помощью процедуры BuildList, приведенной ранее.

   В следующем фрагменте элемент данных Name читается из файла MyListFile и добавляется в список:

   ```
   WHILE NOT EOF (MyListFile) DO
     BEGIN
       READ(MyListFile, Name);
       BuildList(FirstPointer, Name);
     END;
   ```

4. Повторять шаги 2 и 3 до конца файла.

Задание 11–3

Измените пример 11-4, добавив следующие две опции:

- Записать список в файл
- Добавить данные из файла

Что касается типа данных, то можно использовать как текстовый, так и нетекстовый файл.

Пример: Список записей

В этом разделе мы будем работать со связанным списком, имеющим большее практическое значение, – списком записей сотрудников. Взгляните на эти типы:

```
TYPE
{Объявление типа данных }
  SSNstring  = STRING[11];
  DataRecord = RECORD
                 ID             :STRING[5];
                 Name, Position :STRING[20];
                 SSN            :SSNstring;
                 Rate           :REAL
               END;
{Объявление списка }
ListPointer = ^ListRecord;
ListRecord  = RECORD
                DataField :DataRecord;
                NextField :ListPointer
              END;
EmpFile = FILE OF DataRecord;
```

Эти объявления делятся на две основные части:

1. Объявление типа данных, выступающих в качестве поля данных связанного списка (запись).
2. Определение связанного списка.

Обратите внимание, что в этих объявлениях тип SSNstring стоит первым, поскольку он используется в определении записи сотрудника (DataRecord). Заметьте также, что поле данных DataField в связанном списке имеет тип DataRecord.

Также определяется файл DataRecords, в котором будут храниться данные. Файл записей значительно облегчает файловую обработку.

В качестве глобальных переменных будут использованы указатель списка, файловая переменная и переменная-запись.

```
VAR
  FirstPointer :ListPointer;
  MyListFile   :EmpFile;
  EmpRecord    :DataRecord;
```

При работе со списком записей применяются те же приемы, что и в случае простых списков, так как мы по-прежнему имеем дело с узлами. Не забудьте использовать полевые переменные для чтения полей. Например, в списке строк ссылайтесь на каждую строку с помощью переменной:

```
CurrentPointer^.DataField
```

В списке записей ссылайтесь на поле SSN (как пример) посредством переменной:

```
CurrentPointer^.DataField.SSN
```

то же самое можно сделать при помощи оператора WITH:

```
WITH CurrentPointer^.DataField DO
  BEGIN
    WRITE(ID :7);
    WRITE(Name :22); ...
```

11–4. Поиск в списке

В реальных приложениях вывод на экран всего списка неудобен, поскольку список может быть очень длинный. Вместо этого для вывода на экран определенной записи надо осуществлять поиск в списке по ключевому полю, такому как номер социального страхования (SSN). Для поиска в списке необходимо сделать следующее:

1. Начать с первого узла, установив `CurrentPointer` по адресу первого узла:

   ```
   CurrentPointer := FirstPointer;
   ```

2. Сравнить номер социального страхования, введенный с клавиатуры (`SSNumber`), с полем SSN этого узла. В случае совпадения установить флаг, например `Found`:

   ```
   IF CurrentPointer^.DataField.SSN = SSNumber THEN
     Found := TRUE
   ```

   В этом случае `CurrentPointer` уже указывает на требуемый узел и может быть использован для чтения информации.

3. Если требуемая запись не найдена, переместить `CurrentPointer` на следующий узел.

   ```
   CurrentPointer := CurrentPointer^.NextField;
   ```

4. Повторять шаги 2 и 3, пока или не найдется соответствующая запись (Found = TRUE), или не будет достигнут конец списка (CurrentPointer = NIL). Таким образом, в цикле WHILE будут проверяться следующие два условия:

   ```
   WHILE (CurrentPointer <> NIL) AND (NOT Found) DO
       ....
   ```

Следующий фрагмент программы включает описанные выше шаги:

```
CurrentPointer := FirstPointer;
WHILE (CurrentPointer <> NIL) AND (NOT Found) DO
  IF CurrentPointer^.DataField.SSN = SSNumber THEN
```

```
    Found := TRUE
  ELSE
    CurrentPointer := CurrentPointer^.NextField;
```

Ниже представлен фрагмент, выводящий на экран информацию нужного узла:

```
WITH CurrentPointer^.DataField DO
  BEGIN
    WRITELN('ID: ',ID);
    WRITELN('Имя: ',Name);
    WRITELN('Должность: ', Position);
    WRITELN('Номер социального страхования: ',SSN);
    WRITELN('Почасовой тариф: ',Rate :2:2)
  END;
```

Следующая программа представляет собой версию базы данных сотрудников. Вместе с улучшенной файловой обработкой она имеет опции поиска и вывода на экран определенной записи.

Программа включает в себя следующие процедуры:

- SearchList: поиск определенной записи
- BuildList: добавление записи в список
- ReadList: вывод на экран полного списка
- GetData: прием данных с клавиатуры
- DisplayRec: вывод на экран определенной записи
- DisplayItAll: вывод на экран заголовков полей и вызов ReadList
- ReadFile: чтение записей из файла данных и вызов BuildList
- SaveList: сохранение списка в файле
- Menu: вывод на экран меню пользователя

```
{ ---------------------------- Пример 11-5 -------------------------- }
PROGRAM LinkedListDB(INPUT, OUTPUT, MyListFile);
{ Эта программа работает с базой данных сотрудников как со списком записей. }
CONST
  FileName = 'emplist.bin';
  Header = '------------- Главное меню -------------';
  Separator = '----------------------------------------';
TYPE
{Объявление типа данных }
  SSNstring  = STRING[11];
  DataRecord = RECORD
                 ID             :STRING[5];
                 Name, Position :STRING[20];
                 SSN            :SSNstring;
                 Rate           :REAL
               END;
```

```
{Объявление списка }
  ListPointer = ^ListRecord;
  ListRecord  = RECORD
                  DataField :DataRecord;
                  NextField :ListPointer
                END;
  EmpFile = FILE OF DataRecord;
VAR
  FirstPointer :ListPointer;
  MyListFile :EmpFile;
  EmpRecord :DataRecord;

{ -------------------------- Процедура SearchList ---------------------- }
PROCEDURE SearchList(FirstPointer :ListPointer;
                     VAR CurrentPointer :ListPointer;
                     SSNumber :SSNstring;
                     VAR Found :BOOLEAN);
{ Эта процедура осуществляет поиск в связанном списке по номеру социального
страхования сотрудника. Если находит, то значение логического флага Found
становится TRUE, а CurrentPointer указывает на требуемый узел. }
BEGIN
  CurrentPointer := FirstPointer;
  WHILE (CurrentPointer <> NIL) AND (NOT Found) DO
    IF CurrentPointer^.DataField.SSN = SSNumber THEN
      Found := TRUE
    ELSE
      CurrentPointer := CurrentPointer^.NextField;
END;

{ -------------------------- Процедура BuildList ----------------------- }
PROCEDURE BuildList(VAR FirstPointer :ListPointer;
                    DataItem :DataRecord);
{ Эта процедура создает связанный список или добавляет в него узлы.}
{ Примечание: FirstPointer передается с помощью ключевого
слова VAR, поскольку он будет обновлен этой процедурой. }
VAR
  ToolPointer :ListPointer;
BEGIN
  NEW(ToolPointer);
  ToolPointer^.DataField := DataItem;
  ToolPointer^.NextField := FirstPointer;
  FirstPointer := ToolPointer
END;

{ -------------------------- Процедура ReadList ------------------------ }
PROCEDURE ReadList(FirstPointer :ListPointer);
{ Эта процедура читает и выводит на экран содержимое списка. }
VAR
  CurrentPointer :ListPointer;
BEGIN
  CurrentPointer := FirstPointer;
  WHILE CurrentPointer <> NIL DO
    BEGIN
```

```
      WITH CurrentPointer^.DataField DO
        BEGIN
          WRITE(ID :7);
          WRITE(Name :22);
          WRITE(Position :22);
          WRITE(SSN :13);
          WRITELN(' $' ,Rate :0:2)
        END;
      CurrentPointer := CurrentPointer^.NextField
    END;
  WRITELN
END;

{ ------------------------- Процедура GetData ----------------------- }
PROCEDURE GetData(VAR FirstPointer :ListPointer);
{ Эта процедура принимает с клавиатуры данные о сотруднике и передает
процедуре BuildList информацию записи для добавления ее в связанный список. }
VAR
  Item :DataRecord;
BEGIN
  WRITELN('Пожалуйста, введите информацию записи,',
' по окончании нажмите ENTER.');
{ Чтение первого элемента данных }
  WITH Item DO
    BEGIN
      WRITE('ID: '); READLN(ID);
      WRITE('Имя: '); READLN(Name);
      WRITE('Должность: '); READLN(Position);
      WRITE('SSN: '); READLN(SSN);
      WRITE('Тариф: '); READLN(Rate);
      WRITE(Separator)
    END;
  BuildList(FirstPointer, Item);
END;

{ ----------------------- Процедура DisplayItAll -------------------- }
PROCEDURE DisplayItAll(FirstPointer :ListPointer);
{ Эта процедура выводит на экран заголовки полей в соответствующем формате
и вызывает процедуру ReadList для вывода на экран содержимого списка. }
BEGIN
  WRITELN(Separator);
  WRITELN('Содержимое списка: ');
  WRITELN('ID' :7, 'Имя' :22, 'Должность' :22, 'SSN' :13,'Тариф' :7);
  WRITELN;
  ReadList(FirstPointer);
  WRITE('Для продолжения нажмите любую клавишу...');
  READLN
END;

{ ------------------------- Процедура DisplayRec ---------------------- }
PROCEDURE DisplayRec(FirstPointer :ListPointer);
```

```
{ Эта процедура выводит на экран информацию об определенном сотруднике.
Она вызывает процедуру SearchList для поиска в списке по номеру социального
страхования сотрудника. Если найдена, информация выводится на экран,
в противном случае выдается сообщение "не найден". }
VAR
  CurrentPointer :ListPointer;
  SSNumber       :SSNstring;
  Found          :BOOLEAN;
BEGIN
  Found := FALSE;
  WRITELN(Separator);
  WRITE('Введите номер социального страхования сотрудника:');
READLN(SSNumber);
  SearchList(FirstPointer, CurrentPointer, SSNumber, Found);
  IF NOT Found THEN
    WRITELN('SSN: ', SSNumber, ' Не найден')
  ELSE
    WITH CurrentPointer^.DataField DO
      BEGIN
        WRITELN('ID: ',ID);
        WRITELN('Имя: ',Name);
        WRITELN('Должность: ', Position);
        WRITELN('Номер социального страхования: ',SSN);
        WRITELN('Почасовой тариф: ',Rate :2:2)
      END;
  WRITE('Для продолжения нажмите любую клавишу...');
  READLN
END;

{ --------------------------- Процедура SaveList ---------------------- }
PROCEDURE SaveList(FirstPointer :ListPointer;
                   VAR MyListFile: EmpFile);
{Эта процедура сохраняет поля данных связанного списка в файле типа RECORD. }
VAR
  CurrentPointer :ListPointer;
BEGIN
  ASSIGN(MyListFile, FileName);
  REWRITE(MyListFile);
  CurrentPointer := FirstPointer;
  WHILE CurrentPointer <> NIL DO
    BEGIN
      WRITE(MyListFile, CurrentPointer^.DataField);
      CurrentPointer := CurrentPointer^.NextField
    END;
  CLOSE(MyListFile)
END;

{ --------------------------- Процедура ReadFile ----------------------}
PROCEDURE ReadFile(VAR FirstPointer :ListPointer;
                   VAR MyListFile: EmpFile);
{Эта процедура читает данные из файла emplist.bin и добавляет данные в
связанный список. }
```

```
VAR
  Item :DataRecord;
BEGIN
  ASSIGN(MyListFile, FileName);
  RESET(MyListFile);
  WHILE NOT EOF (MyListFile) DO
    BEGIN
      READ(MyListFile, Item);
      BuildList(FirstPointer, Item);
    END;
  CLOSE(MyListFile)
END;

{ ---------------------------- Процедура Menu ------------------------- }
PROCEDURE Menu;
VAR
  Option :INTEGER;
BEGIN
  WRITELN(Header);
  WRITELN('1. Добавить записи в список.');
  WRITELN('2. Вывести на экран весь список.');
  WRITELN('3. Вывести на экран запись о сотруднике.');
  WRITELN('4. Добавить записи из файла.');
  WRITELN('5. Сохранить список в файле.');
  WRITELN('6. Выход.');
  WRITELN(Separator);
  WRITE('Сделайте выбор и нажмите цифру: ');
  READLN(Option);
  CASE Option OF
    1 : GetData(FirstPointer);
    2 : DisplayItAll(FirstPointer);
    3 : DisplayRec(FirstPointer);
    4 : ReadFile(FirstPointer, MyListFile);
    5 : SaveList(FirstPointer, MyListFile);
    6 : Exit
  END;
  Menu
END;

{ ------------------------------ Главная программа ---------------------- }
BEGIN
{ Инициализация пустого списка. }
  FirstPointer := NIL;
  menu
END.
```

Пример файла emplist.bin находится на прилагаемой дискете. После запуска программы можно начать загрузку записей из файла, выбрав опцию 4, затем вывести список на экран с помощью опции 2. В следующем тесте ввод пользователя для ясности выделен полужирным шрифтом с курсивом.

Выполнение теста:

```
---------- Главное меню -------------
1. Добавить записи в список.
2. Вывести на экран весь список.
3. Вывести на экран запись о сотруднике.
4. Добавить записи из файла.
5. Сохранить список в файле.
6. Выход.
-------------------------------------
Сделайте выбор и нажмите цифру: 4 -----> В этом месте список загружается в
память.
---------- Главное меню -------------
1. Добавить записи в список.
2. Вывести на экран весь список.
3. Вывести на экран запись о сотруднике.
4. Добавить записи из файла.
5. Сохранить список в файле.
6. Выход.
-------------------------------------
Сделайте выбор и нажмите цифру: 2
-------------------------------------
Содержимое списка:
    ID               Имя             Должность            SSN            Тариф
   456        Mark Poche       Staff Assistant    999-99-9999          $23.00
   345     Deanna Bedford          Secretary I    444-44-4444          $12.55
   123  John Martin Smith        Sales Manager    111-11-1111          $22.50
   234      James Strahan  Sales Representative    222-22-2222          $11.50
   987     Charles Berlin            President    333-33-3333          $60.50
Для продолжения нажмите любую клавишу...
```

Для удобства номера социального страхования подобраны так, чтобы в процессе поиска сотрудника их было легче запомнить. Пример:

```
---------- Главное меню -------------
1. Добавить записи в список.
2. Вывести на экран весь список.
3. Вывести на экран запись о сотруднике.
4. Добавить записи из файла.
5. Сохранить список в файле.
6. Выход.
-------------------------------------
Сделайте выбор и нажмите цифру: 3
-------------------------------------
Введите номер социального страхования сотрудника: 111-11-1111
ID: 123
Имя: John Martin Smith
Должность: Sales Manager
Номер социального страхования: 111-11-1111
Почасовой тариф: 22.50

Для продолжения нажмите любую клавишу...
```

```
---------- Главное меню -------------
1. Добавить записи в список.
2. Вывести на экран весь список.
3. Вывести на экран запись о сотруднике.
4. Добавить записи из файла.
5. Сохранить список в файле.
6. Выход.
-------------------------------------
Сделайте выбор и нажмите цифру: 6
```

Обратите внимание на некоторые моменты в процедуре поиска:

- Вызов процедуры поиска имеет вид:

```
SearchList(FirstPointer, CurrentPointer, SSNumber, Found);
```

  Где SSNumber – это номер социального страхования, который должен совпасть с полем SSN.

- И CurrentPointer, и флаг Found передаются с помощью ключевого слова VAR, так как ожидается, что их значения изменятся в результате поиска.

```
PROCEDURE SearchList(FirstPointer :ListPointer;
                     VAR CurrentPointer :ListPointer;
                     SSNumber :SSNstring;
                     VAR Found :BOOLEAN);
```

Задание 11–4

Добавьте в предыдущую программу процедуру, которая включает в ваше меню пункт «Обновить запись». Для обновления записи найдите ее, примите новые данные с клавиатуры и внесите запись в поле данных текущего узла. Не забудьте также обновить пункты меню, добавив пункт «Обновить запись».

11–5. Удаление узлов из списка

Для удаления узла из связанного списка необходимо объявить три указателя:

- FirstPointer: указывает на первый узел
- CurrentPointer: указывает на текущий узел
- PreviousPointer: указывает на предыдущий узел

Алгоритм удаления узла зависит от его относительного положения в цепочке. Рассмотрим два случая:

(A) Если это первый узел списка. В этом случае процедура удаления проста, и достаточно двух указателей – FirstPointer и CurrentPointer:

1. Установить `CurrentPointer` на узел, который надо удалить (первый узел).
2. Установить `FirstPointer` на второй узел в списке.

```
FirstPointer := FirstPointer^.NextField;
```

3. Освободить CurrentPointer.

```
DISPOSE(CurrentPointer);
```

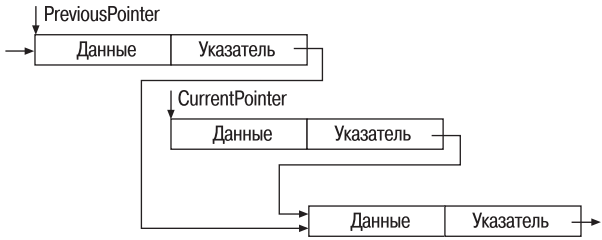
(B) Если узел имеет предшественника. В этом случае необходим третий указатель, который смотрит на предыдущий узел. Алгоритм удаления узла:

4. Установить `CurrentPointer` на узел, который надо удалить.
5. Установить `PreviousPointer` на следующий узел после текущего:

```
PreviousPointer^.NextField := CurrentPointer^.NextField;
```

6. Освободить `CurrentPointer`:

```
DISPOSE(CurrentPointer);
```



Описанные выше шаги подразумевают такое изменение алгоритма процедуры поиска, чтобы `PreviousPointer` следовал за `CurrentPointer` шаг за шагом вдоль всего списка.

Новая версия процедуры SearchList:

```
{ ------------------------- Процедура SearchList ----------------------- }
PROCEDURE SearchList(FirstPointer :ListPointer;
                     VAR CurrentPointer :ListPointer;
                     VAR PreviousPointer :ListPointer;
                     SSNumber :SSNstring;
                     VAR Found :BOOLEAN);
BEGIN
  PreviousPointer := NIL;
  CurrentPointer := FirstPointer;
```

```
  WHILE (CurrentPointer <> NIL) AND (NOT Found) DO
    IF CurrentPointer^.DataField.SSN = SSNumber THEN
      Found := TRUE
    ELSE
      BEGIN
        PreviousPointer := CurrentPointer;
        CurrentPointer := CurrentPointer^.NextField
      END
END;
```

Заголовок процедуры изменен на:

```
PROCEDURE SearchList(FirstPointer :ListPointer;
                     VAR CurrentPointer :ListPointer;
                     VAR PreviousPointer :ListPointer;
                     SSNumber :SSNstring;
                     VAR Found :BOOLEAN);
```

Поскольку `PreviousPointer` **следует за** `CurrentPointer`**, и предполагается, что его значение будет меняться, то ему должно предшествовать ключевое слово VAR.**

Ниже приведена процедура DelRecord:

```
{ ------------------------- Процедура DelRecord ------------------------- }
PROCEDURE DelRecord(VAR FirstPointer :ListPointer);
VAR
  CurrentPointer, PreviousPointer :ListPointer;
  Found :BOOLEAN;
  SSNumber: SSNstring;
BEGIN
  Found := FALSE;
  WRITELN(Separator);
  WRITE('Введите номер социального страхования удаляемого сотрудника:');
  READLN(SSNumber);
  SearchList(FirstPointer, CurrentPointer, PreviousPointer, SSNumber, Found);
  IF NOT Found THEN
    WRITELN('SSN: ', SSNumber, ' Не найден')
  ELSE
    BEGIN
      IF PreviousPointer = NIL THEN
{ Должен быть удален первый узел. }
        FirstPointer := FirstPointer^.NextField
      ELSE
{ Удаляемый узел имеет предшественника. }
        PreviousPointer^.NextField := CurrentPointer^.NextField;
      DISPOSE(CurrentPointer);
      WRITELN('Запись удалена из списка.')
    END;
  WRITE('Для продолжения нажмите любую клавишу...');
  READLN
END;
```

Изменения в процедуре SearchList повлияют на другие процедуры. Любой вызов SearchList должен включать в себя новый параметр – указатель PreviousPointer, как показано в следующем вызове:

```
SearchList(FirstPointer, CurrentPointer, PreviousPointer,
           SSNumber, Found);
```

В примере 11-5 процедура DisplayRec вызывает SearchList, используя только два параметра (два указателя). Это и понятно, ведь PreviousPointer не задействован в процедуре DisplayRec. Для того чтобы включить процедуру SearchList в программу, необходимо изменить вызовы процедуры SearchList. В процедуре DisplayRec можно объявить фиктивный указатель, который не имеет никаких обязанностей, за исключением передачи в качестве параметра в процедуру поиска. Пример нового вызова:

```
SearchList(FirstPointer, CurrentPointer, DummyPointer,
           SSNumber, Found);
```

В следующей программе база данных сотрудников почти заполнена: Основные особенности программы:

- Процедура SearchList вызывается до ввода данных о новом сотруднике. Проверяется существование номера социального страхования. Если номер существует, то никакие данные не вводятся, и выдается соответствующее сообщение.
- Добавлена процедура DelRecord.
- Добавлена процедура UpdateRec.
- Если номер социального страхования не найден, то для отмены всех операций (то есть удаления, обновления или вывода на экран) вызывается процедура SearchList.

```
{ ---------------------------- Пример 11-6 ---------------------------- }
PROGRAM LinkedListDB(INPUT, OUTPUT, MyListFile);
CONST
  FileName = 'emplist.bin';
  Header = '------------- Главное меню -------------';
  Separator = '----------------------------------------';
TYPE
{Объявление типа данных }
  SSNstring  = STRING[11];
  DataRecord = RECORD
                 ID             :STRING[5];
                 Name, Position :STRING[20];
                 SSN            :SSNstring;
                 Rate           :REAL
               END;
{Объявление списка }
  ListPointer = ^ListRecord;
  ListRecord  = RECORD
```

```
                DataField :DataRecord;
                NextField :ListPointer
              END;
  EmpFile = FILE OF DataRecord;
VAR
  FirstPointer :ListPointer;
  MyListFile   :EmpFile;
  EmpRecord    :DataRecord;

{ ------------------------- Процедура SearchList ----------------------- }
PROCEDURE SearchList(FirstPointer :ListPointer;
                     VAR CurrentPointer :ListPointer;
                     VAR PreviousPointer :ListPointer;
                     SSNumber :SSNstring;
                     VAR Found :BOOLEAN);
{ Эта процедура осуществляет поиск в связанном списке по номеру социального
страхования сотрудника. В случае нахождения значение логического флага Found
становится TRUE. }
BEGIN
  PreviousPointer := NIL;
  CurrentPointer := FirstPointer;
  WHILE (CurrentPointer <> NIL) AND (NOT Found) DO
    IF CurrentPointer^.DataField.SSN = SSNumber THEN
      Found := TRUE
    ELSE
      BEGIN
        PreviousPointer := CurrentPointer;
        CurrentPointer := CurrentPointer^.NextField
      END
END;

{ ------------------------- Процедура BuildList ------------------------ }
PROCEDURE BuildList(VAR FirstPointer :ListPointer;
                    DataItem :DataRecord);
{ Эта процедура создает связанный список или добавляет в него узлы.}
{ Примечание: FirstPointer передается с использованием ключевого
слова VAR, поскольку он будет обновлен этой процедурой. }
VAR
  ToolPointer :ListPointer;
BEGIN
  NEW(ToolPointer);
  ToolPointer^.DataField := DataItem;
  ToolPointer^.NextField := FirstPointer;
  FirstPointer := ToolPointer
END;

{ ------------------------- Процедура ReadList ------------------------- }
PROCEDURE ReadList(FirstPointer :ListPointer);
{ Эта процедура читает и выводит на экран содержимое списка. }
VAR
  CurrentPointer :ListPointer;
BEGIN
```

```
  CurrentPointer := FirstPointer;
  WHILE CurrentPointer <> NIL DO
    BEGIN
      WITH CurrentPointer^.DataField DO
        BEGIN
          WRITE(ID :7);
          WRITE(Name :22);
          WRITE(Position :22);
          WRITE(SSN :13);
          WRITELN(' $' ,Rate :0:2)
        END;
      CurrentPointer := CurrentPointer^.NextField
    END;
  WRITELN
END;

{ ------------------------ Процедура DelRecord -------------------------- }
PROCEDURE DelRecord(VAR FirstPointer :ListPointer);
{ Эта процедура удаляет узел из списка. Если удаляется первый узел,
FirstPointer перемещается на следующий узел; в противном случае
в поле указателя предыдущего узла заносится адрес следующего узла.
В обоих случаях CurrentPointer освобождается. }
VAR
  CurrentPointer, PreviousPointer :ListPointer;
  Found :BOOLEAN;
  SSNumber: SSNstring;
BEGIN
  Found := FALSE;
  WRITELN(Separator);
  WRITE('Введите номер социального страхования удаляемого сотрудника:');
  READLN(SSNumber);
  SearchList(FirstPointer, CurrentPointer, PreviousPointer, SSNumber, Found);
  IF NOT Found THEN
    WRITELN('SSN: ', SSNumber, ' Не найден')
  ELSE
    BEGIN
      IF PreviousPointer = NIL THEN
{ Должен быть удален первый узел. }
        FirstPointer := FirstPointer^.NextField
      ELSE
{ Удаляемый узел имеет предшественника. }
        PreviousPointer^.NextField := CurrentPointer^.NextField;
      DISPOSE(CurrentPointer);
      WRITELN('Запись удалена из списка.')
    END;
  WRITE('Для продолжения нажмите любую клавишу...');
  READLN
END;

{ -------------------------- Процедура GetData ------------------------- }
PROCEDURE GetData(VAR FirstPointer :ListPointer);
```

```
{ Эта процедура принимает с клавиатуры данные о сотруднике и передает
процедуре BuildList информацию записи для добавления ее в связанный список. }
VAR
  CurrentPointer, DummyPointer :ListPointer;
  Item    :DataRecord;
  SSNumber: SSNstring;
  Found   :BOOLEAN;
BEGIN
  Found := FALSE;
  WRITE('Пожалуйста, введите номер социального страхования сотрудника: ');
  READLN(SSNumber);
  SearchList(FirstPointer, CurrentPointer, DummyPointer, SSNumber, Found);
  IF NOT Found THEN
    BEGIN
      WRITELN('Пожалуйста, введите информацию о сотруднике:');
      WITH Item DO
        BEGIN
          SSN := SSNumber;
          WRITE('ID: '); READLN(ID);
          WRITE('Имя: '); READLN(Name);
          WRITE('Должность: '); READLN(Position);
          WRITE('Тариф: '); READLN(Rate);
          WRITELN(Separator)
        END;
      BuildList(FirstPointer, Item);
      WRITELN('Сотрудник добавлен в список.')
    END
  ELSE
    WRITELN('The SSN: ', SSNumber, ' уже в списке.');
  WRITE('Для продолжения нажмите любую клавишу...');
  READLN
END;

{ -------------------------- Процедура DisplayItAll ---------------------- }
PROCEDURE DisplayItAll(FirstPointer :ListPointer);
{ Эта процедура выводит на экран заголовки полей в соответствующем формате
и вызывает процедуру ReadList для вывода на экран содержимого списка. }
BEGIN
  WRITELN(Separator);
  WRITELN('Содержимое списка: ');
  WRITELN('ID' :7, 'Имя' :22, 'Должность' :22, 'SSN' :13, 'Тариф' :7);
  WRITELN;
  ReadList(FirstPointer);
  WRITE('Для продолжения нажмите любую клавишу...');
  READLN
END;

{ -------------------------- Процедура DisplayRec ------------------------ }
PROCEDURE DisplayRec(FirstPointer :ListPointer);
{ Эта процедура выводит на экран информацию об определенном
сотруднике. Она вызывает процедуру SearchList для поиска
в списке по номеру социального страхования сотрудника. }
```

```
VAR
  CurrentPointer, DummyPointer :ListPointer;
  SSNumber :SSNstring;
  Found    :BOOLEAN;
{ Примечание: DummyPointer используется для вызова процедуры SearchList
(принимающей в качестве параметров три указателя),
но в данной процедуре этот указатель не требуется. }
BEGIN
  Found := FALSE;
  WRITELN(Separator);
  WRITE('Введите номер социального страхования сотрудника:');
READLN(SSNumber);
  SearchList(FirstPointer, CurrentPointer, DummyPointer, SSNumber, Found);
  IF NOT Found THEN
    WRITELN('SSN: ', SSNumber, ' Не найден')
  ELSE
    WITH CurrentPointer^.DataField DO
      BEGIN
        WRITELN('ID: ',ID);
        WRITELN('Имя: ',Name);
        WRITELN('Должность: ', Position);
        WRITELN('Номер социального страхования: ', SSN);
        WRITELN('Почасовой тариф: $', Rate :2:2)
      END;
  WRITE('Для продолжения нажмите любую клавишу...');
  READLN
END;

{ ------------------------- Процедура UpdateRec ------------------------ }
PROCEDURE UpdateRec(FirstPointer :ListPointer);
{ Эта процедура обновляет информацию записи определенного сотрудника.
Она вызывает процедуру SearchList для поиска в списке по номеру социального
страхования сотрудника. Принимается новая информация, вводимая
пользователем,
в противном случае выдается сообщение "Не найден". }
VAR
  CurrentPointer, DummyPointer :ListPointer;
  SSNumber :SSNstring;
  Found    :BOOLEAN;
{ Примечание: DummyPointer используется для вызова процедуры SearchList
(которая принимает в качестве параметров три указателя),
но в данной процедуре этот указатель не требуется. }
BEGIN
  Found := FALSE;
  WRITELN(Separator);
  WRITE('Введите номер социального страхования сотрудника:');
READLN(SSNumber);
  SearchList(FirstPointer, CurrentPointer, DummyPointer, SSNumber, Found);
  IF NOT Found THEN
    WRITELN('SSN: ', SSNumber, ' Не найден')
  ELSE
```

```
    WITH CurrentPointer^.DataField DO
      BEGIN
        WRITELN('Пожалуйста, введите новую информацию о',
' сотруднике (SSN: ', SSNumber,'):');
        WRITE('ID: '); READLN(ID);
        WRITE('Имя: '); READLN(Name);
        WRITE('Должность: '); READLN(Position);
        WRITE('Почасовой тариф: '); READLN(Rate);
        WRITELN('Запись обновлена.')
      END;
  WRITE('Для продолжения нажмите любую клавишу...');
  READLN
END;

{ -------------------------- Процедура SaveList -------------------------- }
PROCEDURE SaveList(FirstPointer :ListPointer;
                   VAR MyListFile: EmpFile);
{Эта процедура сохраняет поля данных связанного списка в файле типа RECORD. }
VAR
  CurrentPointer :ListPointer;
BEGIN
  ASSIGN(MyListFile, FileName);
  REWRITE(MyListFile);
  CurrentPointer := FirstPointer;
  WHILE CurrentPointer <> NIL DO
    BEGIN
      WRITE(MyListFile, CurrentPointer^.DataField);
      CurrentPointer := CurrentPointer^.NextField
    END;
  CLOSE(MyListFile);
  WRITELN('Список сохранен в файле.');
  WRITE('Для продолжения нажмите любую клавишу...');
  READLN
END;

{ -------------------------- Процедура ReadFile -------------------------- }
PROCEDURE ReadFile(VAR FirstPointer :ListPointer;
                   VAR MyListFile: EmpFile);
{ Эта процедура читает данные из файла emplist.bin и добавляет данные в
связанный список. }
VAR
  Item :DataRecord;
BEGIN
  ASSIGN(MyListFile, FileName);
  RESET(MyListFile);
  WHILE NOT EOF (MyListFile) DO
    BEGIN
      READ(MyListFile, Item);
      BuildList(FirstPointer, Item);
    END;
  CLOSE(MyListFile);
  WRITELN('Список сотрудников уже в памяти.');
```

```
  WRITE('Для продолжения нажмите любую клавишу...');
  READLN
END;

{ ---------------------------- Процедура Menu ---------------------------- }
PROCEDURE Menu;
VAR
  Option :INTEGER;
BEGIN
  WRITELN(Header);
  WRITELN('1. Добавить записи в список.');
  WRITELN('2. Вывести на экран весь список.');
  WRITELN('3. Вывести на экран запись о сотруднике.');
  WRITELN('4. Добавить записи из файла.');
  WRITELN('5. Сохранить список в файле.');
  WRITELN('6. Удалить запись.');
  WRITELN('7. Обновить запись.');
  WRITELN('8. Выход.');
  WRITELN(Separator);
  WRITE('Сделайте выбор и нажмите цифру: ');
  READLN(Option);
  CASE Option OF
    1 : GetData(FirstPointer);
    2 : DisplayItAll(FirstPointer);
    3 : DisplayRec(FirstPointer);
    4 : ReadFile(FirstPointer, MyListFile);
    5 : SaveList(FirstPointer, MyListFile);
    6 : DelRecord(FirstPointer);
    7 : UpdateRec(FirstPointer);
    8 : Exit
  END;
  Menu
END;

{ ------------------------------ Главная программа ----------------------- }
BEGIN
{ Инициализация пустого списка }
  FirstPointer := NIL;
  menu
END.
```

В следующем тесте была выбрана опция 1 для добавления нового сотрудника (SSN: 222-22-2222). Однако программа отказалась его добавить, поскольку процедура SearchList обнаружила этот SSN в списке.

Пример выполнения:

```
----------- Главное меню -------------
1. Добавить записи в список.
2. Вывести на экран весь список.
3. Вывести на экран запись о сотруднике.
4. Добавить записи из файла.
5. Сохранить список в файле.
```

```
6. Удалить запись.
7. Обновить запись.
8. Выход.
--------------------------------------
Сделайте выбор и нажмите цифру: 4
Список сотрудников уже в памяти.
Для продолжения нажмите любую клавишу...
----------- Главное меню -------------
1. Добавить записи в список.
2. Вывести на экран весь список.
3. Вывести на экран запись о сотруднике.
4. Добавить записи из файла.
5. Сохранить список в файле.
6. Удалить запись.
7. Обновить запись.
8. Выход.
--------------------------------------
Сделайте выбор и нажмите цифру: 1
Пожалуйста, введите номер социального страхования сотрудника: 222-22-2222
The SSN: 222-22-2222 уже в списке.
Для продолжения нажмите любую клавишу...
```

Заключение

В этой главе мы рассмотрели следующие особенности указателей:

1. Указатель может ссылаться на любой тип данных.
2. Указатель на определенный тип данных связан с этим типом.
3. Нельзя читать или выводить на экран значение указателя. Можно только читать или выводить на экран значение, на которое ссылается этот указатель.
4. К указателям можно применять только операции присваивания и сравнения (= или <>). Указателю можно присвоить только константу NIL или значение другого указателя, связанного с тем же самым типом данных.
5. Тип указателя определяется в общих чертах так:

   ```
   TYPE
   тип-указателя = ^ определение-типа;
   ```

   где «определение-типа» – это стандартный или определенный пользователем тип данных.
6. Для выделения памяти указателю применяется процедура NEW, а для освобождения выделенной памяти – процедура DISPOSE. Эти две процедуры принимают указатель в качестве параметра.

   ```
   NEW(PtrПеременная);
   DISPOSE(PtrПеременная);
   ```

7. Вы также научились использовать связанные списки в качестве улучшенных структур данных, которые динамически расширяются и сжимаются в процессе выполнения программы. Ниже приведены наиболее важные особенности связанных списков:

- В связанном списке данные хранятся в узлах. Каждый узел содержит поле данных и поле указателя. Поле указателя ссылается на следующий узел.
- Узлы в связанном списке могут хранить данные любого типа, однако чаще всего они используются для хранения записей.
- Для объявления связанного списка используется следующая основная форма:

```
TYPE
    тип-данных = определение-типа;
    ListPointer = ^ListRecord;
    ListRecord = RECORD
                    DataField : Data-Type;
                    NextField :ListPointer
             END;
```

- Связанные списки строятся и обрабатываются при помощи указателей.
- Можно добавлять узлы в связанный список и удалять их из связанного списка. Добавлять узлы можно только в начало списка.

Упражнения

1. Истинно или ложно:
   a. Указатели нельзя напечатать с помощью операторов WRITE/WRITELN.
   b. Указатели можно прочитать с помощью операторов READ/READLN.
   c. Перед использованием указателя ему нужно выделить память.
   d. Память, выделенная указателю, может быть освобождена с помощью процедуры DISPOSE.
   e. Значение указателя (данные, на которые ссылается указатель) можно присвоить статической переменной.
   f. После освобождения указатель становится неопределенным.
   g. Можно объявить указатель для любого типа данных.
   h. Указатель можно присвоить другому указателю того же типа.
   i. Нельзя сравнивать два указателя с помощью логических операторов.
   j. Можно складывать и вычитать указатели.

2. Даны следующие объявления:

```
TYPE
  Employee = RECORD
               ID  :INTEGER;
               Wage: REAL;
             END;
  Empptr = ^Employee;
  Person = RECORD
             Name :STRING[25];
             SSN  :STRING[11];
           END;
  Personptr = ^Person;
VAR
  Ptr1, Ptr2 : Empptr;
  Ptr3       : Personptr;
```

Определите, какие из этих операций допустимы, а какие нет:

a. Ptr1 := Ptr2;

b. WRITELN(Ptr1=Ptr2);

c. Ptr1 := Ptr3;

d. WRITELN(Ptr1=Ptr3);

3. Напишите объявление типа для связанного списка для описи предметов, который хранит следующие поля:
   - Номер предмета: может состоять из букв и цифр.
   - Название предмета: может состоять из букв и цифр.
   - Количество: целое число.
   - Цена по каталогу: вещественное число.

   Также объявите переменные, необходимые для обработки списка и сохранения его в файле.

Ответы

1. b, I и j ложно.
2. a и b допустимы. c и d недопустимы (не совпадают типы).
3. Объявления записи и списка:

```
TYPE
  InventoryItem = RECORD
                    ItemNo       :STRING[10];
                    ItemName     :STRING[20];
                    Quantity     :INTEGER;
                    InvoicePrice :REAL
                  END;
{Объявление списка: }
  ListPointer = ^ListRecord;
```

```
  ListRecord = RECORD
                 DataField :InventoryItem;
                 NextField :ListPointer
               END;
{Объявление файла записей: }
  InventoryFile = FILE OF InventoryItem;
VAR
  FirstPointer :ListPointer;
  MyFile       :InventoryFile;
  MyRecord     :InventoryItem;
```

Следующий шаг

Вы приобрели достаточный инструментарий для практического программирования на Паскале и создания хороших программных приложений. Однако некоторые темы, не затронутые в этой книге, также могут быть вам интересны:

- Упорядоченные связанные списки
- Двоичные поисковые деревья
- Файлы прямого/произвольного доступа
- Графика

Последние две не входят в стандартный Паскаль, поэтому придется обратиться к документации по конкретному компилятору.

Хотя в этой книге не упоминалось программирование под Windows, совершенствование в стандартном Паскале является необходимым шагом на пути к Windows-программированию.

A

Таблица ASCII-кодов

Печатаемые символы

Десятичный	Восьмеричный	Шестнадцатеричный	Символ
32	40	20	пробел
33	41	21	!
34	42	22	"
35	43	23	#
36	44	24	$
37	45	25	%
38	46	26	&
39	47	27	'
40	50	28	(
41	51	29	)
42	52	2a	*
43	53	2b	+
44	54	2c	,
45	55	2d	-
46	56	2e	.
47	57	2f	/
48	60	30	0
49	61	31	1
50	62	32	2
51	63	33	3
52	64	34	4
53	65	35	5
54	66	36	6

Десятичный	Восьмеричный	Шестнадцатеричный	Символ
55	67	37	7
56	70	38	8
57	71	39	9
58	72	3a	:
59	73	3b	;
60	74	3c	<
61	75	3d	=
62	76	3e	>
63	77	3f	?
64	100	40	@
65	101	41	A
66	102	42	B
67	103	43	C
68	104	44	D
69	105	45	E
70	106	46	F
71	107	47	G
72	110	48	H
73	111	49	I
74	112	4a	J
75	113	4b	K
76	114	4c	L
77	115	4d	M
78	116	4e	N
79	117	4f	O
80	120	50	P
81	121	51	Q
82	122	52	R
83	123	53	S
84	124	54	T
85	125	55	U
86	126	56	V
87	127	57	W
88	130	58	X
89	131	59	Y
90	132	5a	Z

Десятичный	Восьмеричный	Шестнадцатеричный	Символ
91	133	5b	[
92	134	5c	\
93	135	5d	]
94	136	5e	^
95	137	5f	_
96	140	60	`
97	141	61	a
98	142	62	b
99	143	63	c
100	144	64	d
101	145	65	e
102	146	66	f
103	147	67	g
104	150	68	h
105	151	69	i
106	152	6a	j
107	153	6b	k
108	154	6c	l
109	155	6d	m
110	156	6e	n
111	157	6f	o
112	160	70	p
113	161	71	q
114	162	72	r
115	163	73	s
116	164	74	t
117	165	75	u
118	166	76	v
119	167	77	w
120	170	78	x
121	171	79	y
122	172	7a	z
123	173	7b	{
124	174	7c	|
125	175	7d	}
126	176	7e	~

Управляющие символы

Десятичный	Восьмеричный	Шестнадцатеричный	Ключ	Мнемонический
0	0	0	^@	NUL
1	1	1	^A	SOH
2	2	2	^B	STX
3	3	3	^C	ETX
4	4	4	^D	EOT
5	5	5	^E	ENQ
6	6	6	^F	ACK
7	7	7	^G	BEL
8	10	8	^H	BS
9	11	9	^I	HT
10	12	a	^J	LF
11	13	b	^K	VT
12	14	c	^L	FF
13	15	d	^M	CR
14	16	e	^N	SO
15	17	f	^O	SI
16	20	10	^P	DLE
17	21	11	^Q	DC1
18	22	12	^R	DC2
19	23	13	^S	DC3
20	24	14	^T	DC4
21	25	15	^U	NAK
22	26	16	^V	SYN
23	27	17	^W	ETB
24	30	18	^X	CAN
25	31	19	^Y	EM
26	32	1a	^Z	SUB
27	33	1b	ESC	ESC
28	34	1c		FS
29	35	1d		GS
30	36	1e		RS
31	37	1f		US
127	177	7f	DEL	DEL

B

Зарезервированные слова и стандартные идентификаторы

Зарезервированные слова

AND	FORWARD	PROCEDURE
ARRAY	FUNCTION	PROGRAM
BEGIN	GOTO	RECORD
CASE	IF	REPEAT
CONST	IN	SET
DIV	LABEL	THEN
DO	MOD	TO
DOWNTO	NIL	TYPE
ELSE	NOT	UNTIL
END	OF	VAR
FILE	OR	WHILE
FOR	PACKED	WITH

Дополнительные слова, зарезервированные в Турбо Паскале:

ABSOLUTE	INTERFACE	STRING
EXTERNAL	INTERRUPT	UNIT
IMPLEMENTATION	SHL	USES
INLINE	SHR	XOR

Стандартные идентификаторы

Константы		
FALSE	MAXINT	TRUE
Типы		
BOOLEAN	INTEGER	TEXT
CHAR	REAL	
Файлы		
INPUT	OUTPUT	
Функции		
ABS	EXP	SIN
ARCTAN	LN	SQR
CHR	ODD	SQRT
COS	ORD	SUCC
EOF	PRED	TRUNC
EOLN	ROUND	
Процедуры		
DISPOSE	PUT	UNPACK
GET	READ	WRITE
NEW	READLN	WRITELN
PACK	RESET	
PAGE	REWRITE	

Дополнительные идентификаторы, предопределенные в Турбо Паскале:

Константы		
MAXLONGINT	PI	
Типы		
BYTE	EXTENDED	SINGLE
COMP	LONGINT	WORD
DOUBLE	SHORTINT	
Функции (рассмотрены в этой книге)		
CONCAT	LENGTH	POS
COPY	PI	RANDOM
Процедуры (рассмотрены в этой книге)		
APPEND	CLOSE	EXIT
ASSIGN	DELETE	INSERT

Стандартные процедуры GET, PUT, PACK, UNPACK и PAGE в Турбо Паскале не определены.

C

Ответы к заданиям

Глава 1

Задание 1–1

```
{ ------------------------------ ЗАДАНИЕ 1-1 ----------------------------- }
PROGRAM DisplayText(OUTPUT);
BEGIN
  WRITELN('wordware Publishing, Inc.');
  WRITELN('-------------------------');
  WRITELN('2320 Los Rios Boulevard');
  WRITELN('Plano, Texas 75074')
END.
```

Задание 1–2

```
{ ------------------------------- ЗАДАНИЕ 1-2 -------------------------- }
PROGRAM Expressions(OUTPUT);
BEGIN
  WRITELN;
  WRITELN('A. 144/12 = ',144 / 12:0:2);
  WRITELN('B. 144 DIV 12 = ', 144 DIV 12);
  WRITELN('C. 17 MOD 5 = ', 17 MOD 5);
  WRITELN('D. 3 MOD 5 = ', 3 MOD 5);
  WRITELN('E. 3e+02 + 3 = ',3E+02+3:0:2);
  WRITELN('F. 345E-01 -1 = ',345E-01-1:0:2);
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Задание 1–3

```
{ ------------------------------ ЗАДАНИЕ 1-3 ---------------------------- }
PROGRAM Expressions1(OUTPUT);
BEGIN
  WRITELN;
```

```
  WRITELN('A. 15 - 15 DIV 15 = ',15-15 DIV 15);
  WRITELN('B. 22 + 10 / 2 = ', 22+10/2:0:2);
  WRITELN('B. (22 + 10) / 2 = ', (22+10)/2:0:2);
  WRITELN('C. 50 * 10 - 4 MOD 3 * 5 + 80 = ',50*10-4 MOD 3*5+80);
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Задание 1–4

```
{ ------------------------------ ЗАДАНИЕ 1-4 --------------------------- }
PROGRAM Expressions2(OUTPUT);
{ Определение переменных }
VAR
  a, b :INTEGER;
{ Программный блок }
BEGIN
  a := 2;
  b := 9;
  WRITELN;
  WRITELN('a=',a);
  WRITELN('b=',b);
  WRITELN('a+b DIV 2 = ',a+b DIV 2);
  WRITELN('(a+b) DIV 2 = ',(a+b) DIV 2)
END.
```

Задание 1–5

```
{ ------------------------------ ЗАДАНИЕ 1-5 --------------------------- }
PROGRAM Payroll(INPUT,OUTPUT);
{ Определение переменных }
VAR
  HoursWorked, PayRate, Wages :REAL;
{ Программный блок }
BEGIN
  WRITE('Пожалуйста, введите отработанные часы: ');
  READLN(HoursWorked);
  WRITE('Пожалуйста, введите тариф: ');
  READLN(PayRate);
  Wages := HoursWorked * PayRate;
  WRITELN;
  WRITELN('Зарплата = $', Wages:0:2)
END.
```

Глава 2

Задание 2–1

```
{ ------------------------------ ЗАДАНИЕ 2-1 --------------------------- }
{ Бакалейная лавка }
PROGRAM Grocery(INPUT,OUTPUT);
VAR
```

```
  Change, AmountPaid, TotalPrice :REAL;
  IntChange, Dollars, Quarters, Dimes, Nickels, Cents :INTEGER;
BEGIN
  WRITE('Введите общую стоимость в долларах:');
  READLN(TotalPrice);
  WRITE('Введите уплаченную сумму в долларах:');
  READLN(AmountPaid);
  { Доллары }
    Change := AmountPaid - TotalPrice;
    Dollars := TRUNC(Change);
    Change := (Change - Dollars)*100;
    IntChange := ROUND(Change);
  { 25-центовые монеты }
    Quarters := IntChange DIV 25;
    IntChange := IntChange MOD 25;
  { 10-центовые монеты }
    Dimes := IntChange DIV 10;
    IntChange := IntChange MOD 10;
  { 5-центовые монеты }
    Nickels := IntChange DIV 5;
    IntChange := IntChange MOD 5;
  { Центы }
    Cents := IntChange;
  WRITELN('Сдача:');
  WRITELN(Dollars, ' долларов');
  WRITELN(Quarters, ' 25-центовые монеты');
  WRITELN(Dimes,' 10-центовые монеты');
  WRITELN(Nickels, ' 5-центовые монеты');
  WRITELN(Cents, ' 1-центовые монеты');
  READLN
END.
```

Задание 2–2

```
{-------------------------------- ЗАДАНИЕ 2-2 -------------------------- }
{ Решение квадратного уравнения }
PROGRAM Quadratic(INPUT,OUTPUT);
VAR
  A, B, C, D, X1, X2 :REAL;
BEGIN
  WRITE('Введите значения A,B и C для квадратного уравнения:');
  READLN(a,b,c);
  { Детерминант}
  D:=SQR(B)-4.0*A*C;
  { Roots }
  X1:=(-B+SQRT(D))/(2*A);
  X2:=(-B-SQRT(D))/(2*A);
  WRITELN('X1=',X1:2:2,' X2=',X2:2:2);
  WRITELN('Для продолжения нажмите ENTER...');
  READLN
END.
```

```
{ Выполнение примера:
  A=2, B=4, C=1
  X1=0.29
  X2=1.70
  A=1, B=2, C=1
  X1=-1
  X2=-1 }
```

Задание 2-3

```
{ ------------------------------ ЗАДАНИЕ 2-3 --------------------------- }
{ Логические выражения }
PROGRAM CompoundBoolean(OUTPUT);
VAR
  A, X, Y, Z            :INTEGER;
  One, Two, Three, Four :BOOLEAN;
BEGIN
  { Запустите программу для различных значений A, X, Y, Z и посмотрите
    на  результаты }
  WRITE('Введите значения A, X, Y, Z:');
  READLN(A,X,Y,Z);
  One := A < 55.5;
  Two := (X=Y) OR (X>=Z);
  Three := (X=40) OR (Y=80);
  Four := (X=40)<>(Y=80);
  { Four := (X=40) XOR (Y=80); } { Версия Турбо Паскаля }
  WRITELN('Выражение #1= ', One);
  WRITELN('Выражение #2= ', Two);
  WRITELN('Выражение #3= ', Three);
  WRITELN('Выражение #4= ', Four)
END.
```

Глава 3

Задание 3-1

```
{ ------------------------------ ЗАДАНИЕ 3-1 --------------------------- }
PROGRAM CharsTester(INPUT,OUTPUT);
VAR
  InputChar :CHAR;
BEGIN
  WRITE('Пожалуйста, введите алфавитный символ:');
  READLN(InputChar);
  IF (ORD(InputChar) > 64) AND (ORD(InputChar) < 91) THEN
    WRITELN('Это символ верхнего регистра.');
  IF (ORD(InputChar) > 96) AND (ORD(InputChar) < 123) THEN
    WRITELN('Это символ нижнего регистра.');
  IF (ORD(InputChar) > 47) AND (ORD(InputChar) < 58) THEN
    WRITELN('Это число.');
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Задание 3-2

```
{------------------------------- ЗАДАНИЕ 3-2 -------------------------- }
{ Полное решение квадратного уравнения }
PROGRAM Quadratic2(INPUT,OUTPUT);
VAR
  A, B, C, D, X1, X2 :REAL;
BEGIN
  WRITE('Введите значения A,B,C, разделенные пробелами:');
  READLN(a,b,c);
  { Детерминант}
  D:=SQR(B)-4.0*A*C;
  IF D < 0 THEN
    BEGIN
      WRITELN('Корни мнимые.');
      WRITELN('X1=',-B/(2*A):0:2,'+j',SQRT(ABS(D))/(2*A):0:2);
      WRITELN('X2=',-B/(2*A):0:2,'-j',SQRT(ABS(D))/(2*A):0:2)
    END
  ELSE
    BEGIN
      WRITELN('Корни вещественные.');
      WRITELN('X1=',(-B+SQRT(D))/(2*A):0:2);
      WRITELN('X2=',(-B-SQRT(D))/(2*A):0:2)
    END;
  WRITELN('Для продолжения нажмите ENTER');
  READLN
END.

{ Выполнение примера:
  Вещественные корни:
  A=2, B=4, C=1
  X1=0.29
  X2=1.70
  Равные вещественные корни:
  A=1, B=2, C=1
  X1=-1
  X2=-1
  Мнимые корни:
  A=1, B=1,C=1
  X1=-0.5+j0.87
  X2=-0.5-j0.87 }
```

Задание 3-3

```
{ ------------------------------- ЗАДАНИЕ 3-3 --------------------------- }
PROGRAM WeatherTester(INPUT,OUTPUT);
VAR
  Temperature :INTEGER;
  Hot, Cool, Cold, Freezing :BOOLEAN;
BEGIN
  WRITE('Пожалуйста введите температуру:');
  READLN(Temperature);
  Hot := (Temperature >= 75) AND (Temperature < 140);
```

```
  Cool := (Temperature >= 50) AND (Temperature < 75);
  Cold := (Temperature >= 35) AND (Temperature < 50);
  Freezing := (Temperature < 35) AND (Temperature > -80);
  WRITELN;
{ Начало конструкции IF }
{ --------------------------- }
  IF Hot THEN
    WRITELN('На улице жара!')
  ELSE IF Cool THEN
    WRITELN('Да, погода прохладная.')
  ELSE IF Cold THEN
    WRITELN('Ох, холодно. ')
  ELSE IF Freezing THEN
    WRITELN('Ой, мороз.')
  ELSE
    WRITELN('Ну и ну, я никогда не слышал о такой температуре!');
{ Конец конструкции IF }
{ ---------------------- }
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Задание 3-4

```
{ ------------------------------ ЗАДАНИЕ 3-4 --------------------------- }
PROGRAM DateConverter(INPUT,OUTPUT);
VAR
  Day, Month, Year :INTEGER;
BEGIN
  WRITE('Пожалуйста, введите дату в формате mm dd yy:');
  READLN(Month, Day, Year);
  CASE Month OF
    1: WRITE('January ');
    2: WRITE('February ');
    3: WRITE('March ');
    4: WRITE('April ');
    5: WRITE('May ');
    6: WRITE('June ');
    7: WRITE('July ');
    8: WRITE('August ');
    9: WRITE('September ');
    10: WRITE('October ');
    11: WRITE('November ');
    12: WRITE('December ')
  END;
  IF (Day=1) OR (Day=21) OR (Day=31) THEN
    WRITE(Day,'st')
  ELSE IF (Day=2) OR (Day=22) THEN
    WRITE(Day,'nd')
  ELSE IF (Day=3) OR (Day=23) THEN
    WRITE(Day,'rd')
  ELSE
```

```
    WRITE(Day,'th');
  WRITELN(', 19', Year);
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Глава 4

Задание 4–1

```
{ ------------------------------ ЗАДАНИЕ 4-1 --------------------------- }
PROGRAM LeapYears(OUTPUT);
VAR
  Year :INTEGER;
BEGIN
  FOR Year := 1990 TO 2000 DO
    BEGIN
      IF (Year MOD 4 = 0) AND
          (Year MOD 100<>0) OR (Year MOD 400 = 0) THEN
        WRITELN('Год ', Year, ' високосный.')
      ELSE
        WRITELN('Год ', Year, ' не високосный.')
    END;
  WRITELN('Для продолжения нажмите ENTER...');
  READLN
END.
```

Задание 4–2

```
{ ------------------------------ ЗАДАНИЕ 4-2 --------------------------- }
PROGRAM FactorialProg3(INPUT,OUTPUT);
VAR
  Factorial :REAL;
  Kounter, Number :INTEGER;
BEGIN
  WRITE('Дайте мне число (или 0 для выхода): ');
  READLN(Number);
  IF Number = 0 THEN
     EXIT;
  Factorial := 1;
  FOR kounter := Number DOWNTO 1 DO
      Factorial := Factorial * Kounter;
  WRITELN('Факториал ', Number,' равен ', Factorial:0:0);
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Задание 4-3

```
{ ------------------------------ ЗАДАНИЕ 4-3 -------------------------- }
PROGRAM FlagLoop(OUTPUT);
VAR
  Row, Column :INTEGER;
BEGIN
  FOR Row := 1 TO 5 DO
    BEGIN
      FOR Column := 1 to 10 DO
        WRITE('* ');
      WRITELN
    END;
  WRITELN;
  WRITELN('Для продолжения нажмите ENTER...');
  READLN
END.
```

Задание 4-4

```
{ ------------------------------ ЗАДАНИЕ 4-4 -------------------------- }
PROGRAM Multiplication(INPUT,OUTPUT);
VAR
  Result, Kounter, Number :INTEGER;
BEGIN
  WRITE('Дайте мне число: ');
  READLN(Number);
  Kounter := 1;
  WHILE Kounter <= 9 DO
    BEGIN
      Result := Kounter * Number;
      WRITELN(Kounter,' * ',Number, ' = ', Result);
      Kounter := Kounter +1
    END;
  WRITELN;
  WRITELN('Для продолжения нажмите ENTER...');
  READLN
END.
```

Задание 4-5

```
{ ------------------------------ ЗАДАНИЕ 4-5 -------------------------- }
PROGRAM FactorialProg3(INPUT,OUTPUT);
VAR
  Factorial :REAL;
  Kounter, Number :INTEGER;
BEGIN
  WRITE('Дайте мне число (или 0 для выхода): ');
  READLN(Number);
  WHILE Number<>0 DO
    BEGIN
      Factorial := 1;
      FOR kounter := Number DOWNTO 1 DO
```

```
      Factorial := Factorial * Kounter;
        WRITELN('Факториал ', Number,' равен ', Factorial:0:0);
      WRITE('Дайте мне число (или 0 для выхода): ');
      READLN(Number)
    END
END.
```

Глава 5

Задание 5–1

```
{ ------------------------------- ЗАДАНИЕ 5-1 -------------------------- }
PROGRAM Subrange2(INPUT,OUTPUT);
VAR
  UpperCase :'A'..'Z';
  LowerCase :'a'..'z';
  Digit :'0'..'9';
BEGIN
  WRITE('Пожалуйста, введите букву из нижнего регистра: ');
  READLN(LowerCase);
  WRITE('Пожалуйста, введите букву из верхнего регистра: ');
  READLN(UpperCase);
  WRITE('Пожалуйста, введите цифру: ');
  READLN(Digit);
  WRITELN('-----------------------------------------------');
  WRITELN('Ваш ввод:');
  WRITELN('Буква нижнего регистра : ', LowerCase);
  WRITELN('Буква верхнего регистра : ', UpperCase);
  WRITELN('Цифра : ', Digit);
  WRITELN('Для продолжения нажмите ENTER...');
  READLN
END.
```

Задание 5–2

```
{ ------------------------------- ЗАДАНИЕ 5-2 -------------------------- }
PROGRAM SubrangesAndEnum(INPUT,OUTPUT);
{
          ************************************************
  *** Удалите маркеры комментария для просмотра сообщений об ошибках ***
          ************************************************
}
TYPE
  Football = (Saints, Cowboys);
{ Games = (Football, Baseball, Basketball) }
                          { Дублирующийся идентификатор: Football }
  Week = (Mon, Tue, Wed, Thu, Fri, Sat, Sun);
  Weekend = Sat..Sun;
  Compiler = (C, Pascal, Fortran, Ada, Basic);
VAR
  WholeWeek :Week;
```

```
{ WorkingDay :(Mon, Tue, Wed, Fri);}
                          { Дублирующиеся идентификаторы: Mon,..}
  Weekday :Mon..Fri;
{ SW :(Compiler, OperatingSystem, ApplicationProgram); }
                          { Дублирующийся идентификатор: Compiler}
  DpTools :(Hardware, Software, PeopleWare);
  DpTool :(HW, SW, PW);
{ C :(TurboC, QuickC); }
                          { Дублирующийся идентификатор: C}
  Margin : -10..+10;
BEGIN
END.
```

Задание 5-3

```
{ ------------------------------- ЗАДАНИЕ 5-3 -------------------------- }
PROGRAM Scores4(INPUT,OUTPUT);
CONST
  NumberOfStudents = 5;
  Tab = '         '; { 9 пробелов }
  Dash = '-';
  NumberOfDashes = 23;
VAR
  Score :ARRAY[1..NumberOfStudents] OF REAL;
  Average, SumOfScores :REAL;
  Index                :INTEGER;
BEGIN
{ Чтение массива результатов }
{ ------------------------- }
  FOR Index := 1 TO NumberOfStudents DO
  BEGIN
    WRITE('Введите результат ученика #', Index,': ');
    READLN(Score[Index])
  END;
{ Вычисление среднего результата }
{ ----------------------------- }
  SumOfScores := 0;
  FOR Index := 1 TO NumberOfStudents DO
    SumOfScores := SumOfScores + Score[Index];
  Average := SumOfScores / NumberOfStudents;
{ Вывод результатов на экран }
{ ------------------------- }
  WRITELN;
  WRITE(Tab, 'Ученик #');
  WRITE(Tab, 'Результат');
  WRITELN;
  WRITE(Tab);
  FOR Index := 1 TO NumberOfDashes DO
    WRITE(Dash);
  WRITELN;
  FOR Index := 1 TO NumberOfStudents DO
    WRITELN(Tab,Index:3,tab,Score[Index]:10:2);
```

```
  WRITE(Tab);
  FOR Index := 1 TO NumberOfDashes DO
    WRITE(Dash);
  WRITELN;
  WRITELN(Tab,'Средний результат = ', Average:0:2);
  WRITELN;
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Задание 5-4

```
{ ------------------------------- ЗАДАНИЕ 5-4 -------------------------- }
PROGRAM Scores5(INPUT,OUTPUT);
CONST
  NumberOfStudents = 5;
  Tab = ' ';
  Dash = '-';
  NumberOfDashes = 23;
VAR
  Score :ARRAY[1..NumberOfStudents] OF REAL;
  Average, SumOfScores, BestScore :REAL;
  Index, BestOfClass :INTEGER;
BEGIN
{ Чтение массива результатов }
{ ------------------------- }
  FOR Index := 1 TO NumberOfStudents DO
    BEGIN
      WRITE('Введите результат ученика  #', Index,': ');
      READLN(Score[Index])
    END;
{ Вычисление среднего результата }
{ ---------------------------- }
  SumOfScores := 0;
  FOR Index := 1 TO NumberOfStudents DO
    SumOfScores := SumOfScores + Score[Index];
  Average := SumOfScores / NumberOfStudents;
{ Получение лучшего результата }
{ -------------------------- }
  BestScore := Score[1]; { начальное значение }
  BestOfClass := 1; { начальное значение }
  FOR Index := 2 TO NumberOfStudents DO
    BEGIN
      IF Score[Index] > BestScore THEN
        BEGIN
          BestScore := Score[Index];
          BestOfClass := Index;
        END
    END;
{ Вывод результатов на экран }
{ ------------------------- }
  WRITELN;
  WRITE(Tab, 'Ученик #', Tab, 'Результат');
```

```
  WRITELN;
  WRITE(Tab);
  FOR Index := 1 TO NumberOfDashes DO
    WRITE(Dash);
  WRITELN;
  FOR Index := 1 TO NumberOfStudents DO
    WRITELN(Tab, Index:3, Tab, Score[Index]:10:2);
  WRITE(Tab);
  FOR Index := 1 TO NumberOfDashes DO
    WRITE(Dash);
  WRITELN;
  WRITELN(Tab,'Средний результат = ', Average:0:2);
  WRITELN(Tab,'Лучший результат = ', BestScore:0:2);
  WRITELN(Tab,'Лучший в классе ученик #',BestOfClass);
  WRITELN;
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Задание 5–5

```
{ ------------------------------ ЗАДАНИЕ 5-5 --------------------------- }
PROGRAM Scores6(INPUT,OUTPUT);
{Использование двумерного массива }
CONST
  NumberOfClasses = 3;  { Измените это число для большего числа классов }
  NumberOfStudents = 4; { Измените это число для большего числа учеников }
  Tab = '     ';        { 5 пробелов }
  Dash = '-';
  NumberOfDashes = 23;
TYPE
  ScoreArray = ARRAY[1..NumberOfStudents, 1..NumberOfClasses] OF REAL;
  AverageArray = ARRAY[1..NumberOfStudents] OF REAL;
  NameArray = ARRAY[1..NumberOfStudents] OF STRING;
VAR
  Score                                :ScoreArray;
  Average                              :AverageArray;
  Name                                 :NameArray;
  SumOfScores, AveragePot              :REAL;
  StudentCount, ScoreCount, DashCount :INTEGER;
  I, J                                 :INTEGER;
  NamePot                              :STRING;
BEGIN
{ Чтение массива результатов }
{ ------------------------- }
  FOR StudentCount := 1 TO NumberOfStudents DO
    BEGIN
      WRITELN;
      WRITE('Имя ученика #', StudentCount ,': ');
      READLN(Name[StudentCount]);
      WRITELN('Результат ', Name[StudentCount], ': ');
      FOR ScoreCount := 1 TO NumberOfClasses DO
        BEGIN
```

```
          WRITE('Введите результат в классе #', ScoreCount,': ');
          READLN(Score[StudentCount, ScoreCount])
        END;
    END;
{ Вычисление среднего для каждого ученика }
{ ---------------------------------------- }
  FOR StudentCount := 1 TO NumberOfStudents DO
    BEGIN
      SumOfScores := 0; { Initialize for each student }
      FOR ScoreCount := 1 TO NumberOfClasses DO
        SumOfScores := SumOfScores + Score[StudentCount, ScoreCount];
      Average[StudentCount] := SumOfScores/NumberOfClasses
    END;
{ Сортировка средних значений в порядке убывания }
{ ------------------------------------------------ }
  FOR I := 1 TO NumberOfStudents-1 DO
    BEGIN
      FOR J := I+1 TO NumberOfStudents DO
        IF Average[J] > Average[I] THEN
          BEGIN
          { Перестановка средних значений }
            AveragePot := Average[I];
            Average[I] := Average[J];
            Average[J] := AveragePot;
          { Перестановка соответствующих номеров учеников }
            NamePot := Name[I];
            Name[I] := Name[J];
            Name[J] := NamePot
          END                  { Конец IF и внутреннего цикла }
    END;                       { Конец внешнего цикла }
{ Вывод результатов на экран }
{ -------------------------- }
  WRITELN;
  WRITELN(Tab, 'Имя ученика', Tab, 'Среднее значение');
  WRITE(Tab);
  FOR DashCount := 1 TO NumberOfDashes DO
    WRITE(Dash);
  WRITELN;
  FOR StudentCount := 1 TO NumberOfStudents DO
    BEGIN
      WRITE(Tab, Name[StudentCount]);
      FOR I := 1 TO 15 - LENGTH(Name[StudentCount]) DO
        WRITE(' ');
      WRITELN(Average[StudentCount]:8:2)
    END;
  WRITE(Tab);
  FOR DashCount := 1 TO NumberOfDashes DO
    WRITE(Dash);
  WRITELN;
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Глава 6

Задание 6-3

```
{ ------------------------------ ЗАДАНИЕ 6-3 --------------------------- }
PROGRAM AlphaCounter(INPUT,OUTPUT);
VAR
  Ch      :CHAR;
  Counter :INTEGER;
BEGIN
  Counter := 0;
  WHILE NOT EOLN DO
    BEGIN
      READ(Ch);
      IF (Ch >= 'a') AND (Ch <= 'z')
        OR (Ch >= 'A') AND (Ch <= 'Z') THEN
          Counter := Counter + 1
    END;
  WRITELN('Количество букв= ', Counter);
  READLN; { Перемещение указателя за метку конца строки }
  WRITELN('Для продолжения нажмите ENTER...');
  READLN
END.
```

Задание 6-4

```
{ ------------------------------ ЗАДАНИЕ 6-4 --------------------------- }
PROGRAM ScramblingStrings(INPUT,OUTPUT);
CONST
  NumberOfElements = 3;
TYPE
  ScrambleArray = Array[1..NumberOfElements] OF STRING[10];
VAR
  A          :ScrambleArray;
  I1, I2, I3 :INTEGER;
BEGIN
  WRITE('Введите слово: ');
  READLN(A[1]);
  WRITE('Введите слово: ');
  READLN(A[2]);
  WRITE('Введите слово: ');
  READLN(A[3]);
  FOR I1 := 1 TO 3 DO
    FOR I2 := 1 TO 3 DO
      IF I2<>I1 THEN
        BEGIN
          I3 := 6- (I1 + I2);
          WRITELN(A[I1],' ',A[I2],' ',A[I3]);
        END;
  WRITELN('Для продолжения нажмите ENTER...');
  READLN
END.
```

Задание 6-5

```
{ ------------------------------ ЗАДАНИЕ 6-5 --------------------------- }
PROGRAM StringFunctions2(INPUT,OUTPUT);
VAR
  Name                :STRING[30];
  First, Middle, Last :STRING[10];
BEGIN
  WRITE('Пожалуйста, введите ваше имя: ');
  READLN(First);
{Перевод первой буквы в верхний регистр, если она в нижнем регистре }
  IF ORD(First[1]) > 90 THEN
    First[1] := CHR(ORD(First[1]) - 32);
  First := CONCAT(First, ' ');
  WRITE('Пожалуйста, введите ваше отчество: ');
  READLN(Middle);
{ Перевод первой буквы в верхний регистр, если она в нижнем регистре }
  IF ORD(Middle[1]) > 90 THEN
    Middle[1] := CHR(ORD(Middle[1]) - 32);
{ Получение инициала из отчества }
  Middle := COPY(Middle, 1, 1);
  Middle := CONCAT(Middle, '. ');
  WRITE('Пожалуйста, введите вашу фамилию: ');
  READLN(Last);
{ Перевод первой буквы в верхний регистр, если она в нижнем регистре }
  IF ORD(Last[1]) > 90 THEN
  Last[1] := CHR(ORD(Last[1]) - 32);
  Name := CONCAT(First, Middle, Last);
  WRITELN;
  WRITELN('Ваше полное имя: ',Name);
  WRITELN('Для продолжения нажмите ENTER...');
  READLN
END.
```

Глава 7

Задание 7-1

```
{ ------------------------------ ЗАДАНИЕ 7-1 --------------------------- }
PROGRAM Header(OUTPUT);
VAR
  Len, Tab, Kounter :INTEGER;
  TestSentence      :STRING;
  LineChar          :CHAR;
{ ---------------------- Начало процедуры ---------------------- }
PROCEDURE DrawLine(LineLength, TabLength :INTEGER; LineCh: CHAR);
VAR
  Counter :INTEGER;
BEGIN
  FOR Counter := 1 TO TabLength DO
  WRITE(' ');
  FOR Counter := 1 TO LineLength DO
```

```
  WRITE(LineCh);
  WRITELN
END;
{ ---------------------------- Конец процедуры ------------------------- }
{ --------------------------- Главная программа ------------------------ }
BEGIN
  WRITE('Пожалуйста, введите предложение: ');
  READLN(TestSentence);
  Len := LENGTH(TestSentence);
  Tab := (80 - Len) DIV 2;
  WRITE('Пожалуйста, введите символ строки: ');
  READLN(LineChar);
  WRITELN;
  Drawline(Len, Tab, LineChar);
  FOR Kounter := 1 TO Tab DO
    WRITE(' ');
  WRITELN(TestSentence);
  Drawline(Len, Tab, LineChar);
  WRITELN('Для продолжения нажмите ENTER...');
  READLN
END.
```

Задание 7–2

```
{ ------------------------------ ЗАДАНИЕ 7-2 --------------------------- }
PROGRAM FunctionMax(INPUT, OUTPUT);
CONST
  ArraySize = 6;
TYPE
  Range = 1..ArraySize;
  NumbersArray = ARRAY[Range] OF INTEGER;
VAR
  Numbers :NumbersArray;
{ -------------------------- Процедура чтения ------------------------- }
PROCEDURE ReadNumbers(L: INTEGER; VAR R :NumbersArray);
VAR
  I :INTEGER;
BEGIN
  WRITELN('Дайте мне массив чисел из шести элементов.');
  FOR I := 1 TO L DO
    BEGIN
      WRITE('Введите элемент #', I,': ');
      READLN(R[I])
    END
END;
{ ---------------------- Функция MaxNumber ------------------------- }
FUNCTION MaxNumber(S :INTEGER; N: NumbersArray) :INTEGER;
VAR
  K, Maximum :INTEGER;
BEGIN
  Maximum := N[1];
  FOR K := 1 TO S DO
    IF N[K] > Maximum THEN
```

```
      Maximum := N[K];
  MaxNumber := Maximum { Присвоение функции максимального значения }
END;
{ ------------------------------ Конец функции --------------------------- }
{ --------------------------- Главная программа ------------------------- }
BEGIN
  ReadNumbers(ArraySize, Numbers);
  WRITELN('Максимальное число: ', MaxNumber(ArraySize, Numbers));
  WRITELN;
  WRITELN('Для продолжения нажмите ENTER...');
  READLN
END.
```

Задание 7–3

```
{ ------------------------------- ЗАДАНИЕ 7-3 --------------------------- }
PROGRAM ProcedureRecursion(INPUT, OUTPUT);
VAR
  A    :INTEGER;
  Fact :REAL;
{ ------------- Определение процедуры -------------- }
PROCEDURE Factorial(X :INTEGER; VAR Fac :REAL);
BEGIN
  IF X > 1 THEN
    BEGIN
      Factorial(X-1, Fac);
      Fac := Fac * X
    END
  ELSE
    Fac := 1;
END;
{ ------------------ Конец процедуры ------------------ }
{ ----------------- Главная программа ----------------- }
BEGIN
  WRITE('Введите число: ');
  READLN(A);
  Factorial(A, Fact);
  WRITELN('Факториал ', A,' = ', Fact:0:0);
  WRITELN('Для продолжения нажмите ENTER...');
  READLN
END.
```

Глава 8

Задание 8–1

```
{ ------------------------------- ЗАДАНИЕ 8-1 --------------------------- }
Выражение                                   Значение
1. ['A','B','C','D'] + ['E','F']            ['A','B','C','D','E','F']
2. ['A','B','C','D'] + ['B','C','E','F']    ['A','B','C','D','E','F']
3. [1,3,7] + []                             [1,3,7]
```

```
4. ['A','D','F'] * ['O','F']                  ['F']
5. [1,2,3,4] * [5,6,7]                        []
6. [1,2,3,4] - [5,6,7]                        [1,2,3,4]
7. [5,6,7] - []                               [5,6,7]
8. [Able, Baker, Charlie] - [Able, Charlie]   [Baker]
```

Задание 8-2

```
{ ------------------------------ ЗАДАНИЕ 8-2 --------------------------- }
PROGRAM TestExpressions(OUTPUT);
CONST
  CR = CHR(13);
  LF = CHR(10);
  T = ' ';
  A1 = '[1,0] = [1,0]';
  B1 = '[1,0]<>[1,4]';
  C1 = '[1,2,3] >= [1,2]';
  D1 = '[1,2,3] >= [1,2,3]';
  E1 = '[] <= [1,2,3]';
  F1 = '[1,2,3] <= [1,2,3]';
VAR
  A, B, C, D, E, F :BOOLEAN;
BEGIN
  A := [1,0] = [1,0];
  B := [1,0]<>[1,4];
  C := [1,2,3] >= [1,2];
  D := [1,2,3] >= [1,2,3];
  E := [] <= [1,2,3];
  F := [1,2,3] <= [1,2,3];
  WRITELN(A1:30, T, A, CR, LF, B1:30, T, B, CR, LF,
          C1:30, T, C, CR, LF, D1:30, T, D, CR, LF,
          E1:30, T, E, CR, LF, F1:30, T, F);
  WRITELN;
  WRITELN('Для продолжения нажмите ENTER...');
  READLN
END.
```

Задание 8-3

```
{------------------------------ ЗАДАНИЕ 8-3 ----------------------------}
PROGRAM NestedRecord2(OUTPUT);
TYPE
  AddressRecord = RECORD
                    Street :STRING[18];
                    City   :STRING[15];
                    State  :STRING[2];
                    Zip    :String[5];
                  END;
  EmployeeRecord = RECORD
                     Name          :STRING[25];
                     AddressRec    :AddressRecord;
                     Phone         :STRING[12];
                     Rate          :REAL;
```

```
                    MaritalStatus :CHAR;
                END;
VAR
  EmployeeRec :EmployeeRecord;
BEGIN
{ Assign values to the fields }
  WITH EmployeeRec DO
    WITH AddressRec DO
      BEGIN
        Name := 'Sally A. Abolrous';
        Street := '5 Belle Chasse Dr.';
        City := 'LaPlace';
        State := 'LA';
        Zip := '70068';
        Phone := '504-285-3434';
        Rate := 22.5;
        MaritalStatus := 'S';
{ Вывод на экран информации записи }
          WRITELN('Имя сотрудника:     ', Name);
          WRITELN('Address:            ', Street);
          WRITELN('                    ', City);
          WRITE('                    ', State);
          WRITELN('                    ', Zip);
          WRITELN('Телефон #:          ', Phone);
          WRITELN('Почасовой тариф:    $', Rate:0:2);
          WRITELN('Семейное положение: ', MaritalStatus)
      END;
  WRITELN('Для продолжения нажмите ENTER...');
  READLN
END.
```

Глава 9

Задание 9-1

```
{ ------------------------------- ЗАДАНИЕ 9-1 -------------------------- }
PROGRAM EmployeeInfoFile(INPUT,OUTPUT,TimeFile);
{ Эта программа создает месячный график работы }

TYPE
  EmployeeRecord = RECORD
                     ID          :INTEGER;
                     Name        :STRING[25];
                     HoursWorked :INTEGER;
                   END;
VAR
  TimeFile    :TEXT;
  EmployeeRec :EmployeeRecord;
{ ---------- Процедура FileInfo ------------ }
PROCEDURE FileInFo(VAR F:TEXT; Employee :EmployeeRecord);
{ Процедура сохраняет в файле за один раз одну запись }
```

```
BEGIN
  WITH Employee DO
    BEGIN
      WRITELN(F, ID);
      WRITELN(F, Name);
      WRITELN(F, HoursWorked)
    END
END;
{ ------------- Процедура GetData -------------- }
{ Процедура принимает данные с клавиатуры и передает их в процедуру "FileInfo" }
PROCEDURE GetData(VAR F:TEXT; VAR Employee :EmployeeRecord);
VAR
  Counter :INTEGER;
BEGIN
  Counter := 0;
  WITH Employee DO
    BEGIN
      WRITE('ID сотрудника (или 0 для выхода): '); READLN(ID);
      WHILE ID<>0 DO
        BEGIN
          Counter := Counter + 1;
          WRITE('Имя сотрудника: '); READLN(Name);
          WRITE('Рабочие часы  : '); READLN(HoursWorked);
          FILEINFO(F, Employee);
          WRITE('ID сотрудника (или 0 для выхода): '); READLN(ID);
        END
    END;
  WRITELN(Counter,' Записи сотрудника были сохранены в файле.')
END;
{ --------------- Главная программа ----------------- }
BEGIN
  ASSIGN(TimeFile, 'TIMSHEET.TXT');
    REWRITE(TimeFile);
    GetData(TimeFile, EmployeeRec);
  CLOSE(TimeFile)
END.
```

Задание 9–2

```
{ ------------------------------- ЗАДАНИЕ 9-2 -------------------------- }
PROGRAM CreateEmpFile(INPUT,OUTPUT,F);
TYPE
  AddressRecord = RECORD
                    Street :STRING[18];
                    City   :STRING[15];
                    State  :STRING[2];
                    Zip    :String[5];
                  END;
  EmployeeRecord = RECORD
                    ID            :INTEGER;
                    Name          :STRING[25];
                    AddressRec    :AddressRecord;
```

```
                    Phone         :STRING[12];
                    Rate          :REAL;
                    MaritalStatus :CHAR;
                  END;
VAR
  F           :TEXT; { Файловая переменная }
  EmployeeRec :EmployeeRecord;
{ ----------- Процедура WriteRecord ---------- }
PROCEDURE WriteRecord;
BEGIN
{ Сохранение одной записи в файле }
  WITH EmployeeRec DO
    WITH AddressRec DO
      BEGIN
        WRITELN(F, ID);
        WRITELN(F, Name);
        WRITELN(F, Street);
        WRITELN(F, City);
        WRITELN(F, State);
        WRITELN(F, Zip);
        WRITELN(F, Phone);
        WRITELN(F, Rate:0:2);
        WRITELN(F, MaritalStatus)
      END
END;
{ ----------- Процедура GetData ---------- }
PROCEDURE getdata;
VAR
  Counter :INTEGER;
BEGIN
  Counter := 0;
  WITH EmployeeRec DO
    WITH AddressRec DO
      BEGIN
        WRITE('Пожалуйста, введите ID сотрудника (или 0 для выхода): ');
         READLN(ID);
         WHILE ID<>0 DO
           BEGIN
            Counter := counter + 1;
            WRITE('Имя сотрудника: '); READLN(Name);
            WRITE('Адрес: Улица: '); READLN(Street);
            WRITE(' Город: '); READLN(City);
            WRITE(' Штат: '); READLN(State);
            WRITE(' Почтовый индекс: '); READLN(Zip);
            WRITE('Номер телефона: '); READLN(Phone);
            WRITE('Почасовой тариф: '); READLN(Rate);
            WRITE('Семейное положение (S/M): '); READLN(MaritalStatus);
            WriteRecord;
            WRITE('Пожалуйста, введите ID сотрудника (или 0 для выхода): ');
            READLN(ID);
          END
      END;
```

```
  WRITELN(Counter, ' Записи сотрудника были сохранены в файле.')
END;
{ ---------------- Главная программа -------------- }
{ Главная программа }
BEGIN
  ASSIGN(F, 'EMPFILE.TXT');
  REWRITE(F);
  GetData;
  CLOSE(F)
END.
```

Задание 9–3

```
{ --------------------------------- ЗАДАНИЕ 9-3 ------------------------- }
PROGRAM PayRoll2(INPUT,OUTPUT,MasterFile,PayFile);
TYPE
  AddressRecord = RECORD
                    Street :STRING[18];
                    City   :STRING[15];
                    State  :STRING[2];
                    Zip    :String[5];
                  END;
  EmployeeRecord = RECORD
                     ID            :INTEGER;
                     Name          :STRING[20];
                     AddressRec    :AddressRecord;
                     Phone         :STRING[12];
                     Rate          :REAL;
                     MaritalStatus :CHAR;
                   END;
  PayRecord = RECORD
                ID    :INTEGER;
                Name  :STRING[20];
                Wages :REAL;
              END;
VAR
  MasterFile, PayFile :TEXT;
  EmployeeRec         :EmployeeRecord;
  PayRec              :PayRecord;
  HoursWorked, Wages  :REAL;

{ --------------- Процедура Getinfo ------------------ }
{ Эта процедура читает файл сотрудника EMPFILE.TXT и выводит на экран его ID,
имя и почасовой тариф. Затем она принимает с клавиатуры рабочие часы за этот
период }
PROCEDURE Getinfo(VAR F:TEXT);
BEGIN
  WITH EmployeeRec DO
    WITH AddressRec DO
      BEGIN
        READLN(F,ID);   WRITELN('ID: ',ID);
        READLN(F,Name); WRITELN('Имя: ',Name);
```

```
        READLN(F,Street);
        READLN(F,City);
        READLN(F,State);
        READLN(F,Zip);
        READLN(F,Phone);
        READLN(F,Rate); WRITELN('Почасовой тариф: $', Rate:0:2);
        READLN(F,MaritalStatus);
      END;
END;

{ --------------- Процедура CalcWages ---------------- }
{ Эта процедура вычисляет зарплату и округляет центы результата.
Результат передается главной программе }
PROCEDURE CalcWages(HoursWorked:REAL; VAR Wages:REAL);
BEGIN
  WITH EmployeeRec DO
    WITH AddressRec DO
      Wages := Hoursworked * Rate;
  Wages := ROUND(100 * Wages) / 100;
END;

{ -------------- Процедура FilePayRoll --------------- }
{ Эта процедура сохраняет одну запись в выходном файле PAYFILE.TXT }
PROCEDURE FilePayRoll(VAR P :TEXT; Wages :REAL);
BEGIN
  WITH EmployeeRec DO
    WITH AddressRec DO
      BEGIN
        PayRec.ID := ID;
        PayRec.Name := Name;
        Payrec.Wages := Wages
      END;
  WITH PayRec DO
  WRITELN(P, ID:3, Name:20, Wages:10:2)
END;
{ -------------- Процедура ReadPayRoll --------------- }
{ Эта процедура читает файл  PAYFILE.TXT
и в конце работы программы выводит на экран записи }
PROCEDURE ReadPayRoll(VAR P :TEXT);
BEGIN
  WITH PayRec DO
    BEGIN
      READLN(P, ID, Name, Wages);
      WRITELN(ID:3, Name:20, ' $',Wages:0:2)
    END;
END;
{ --------------- Главная программа ------------------ }
BEGIN
  ASSIGN(MasterFile, 'EMPFILE.TXT');
  ASSIGN(Payfile, 'PAYFILE.TXT');
  REWRITE(PayFile);
  RESET(MasterFile);
```

```
  WHILE NOT EOF(MasterFile) DO
    BEGIN
      Getinfo(MasterFile);
      WRITE('Пожалуйста, введите отработанные часы за этот период: ');
      READLN(HoursWorked);
      CalcWages(HoursWorked, Wages);
      FilePayRoll(PayFile, Wages)
    END;
  CLOSE(MasterFile);
  CLOSE(PayFile);
  RESET(PayFile);
  WRITELN('--------- Итоговая платежная ведемость --------- ');
  WRITELN('ID ------------- Имя ------------ Оклад');
  WHILE NOT EOF(PayFile) DO
    ReadPayroll(PayFile);
  CLOSE(PayFile);
  WRITELN('------------------------------------------------ ');
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Глава 10

Задание 10-1

```
{ ------------------------------ ЗАДАНИЕ 10-1 ---------------------------- }
PROGRAM CreateEmployeeDataBase2(INPUT, OUTPUT, PayrollFile, NewFile);
{ Программа создает TEXT-файл PR.TXT, в котором хранятся записи сотрудника. }
CONST
  FileName = 'PR.TXT';
  Header = '---------- Создание файла платежной ведомости ----------';
  Separator = '---------------------------------------';
TYPE
  EmployeeRecord = RECORD
                     ID             :STRING[5];
                     Name, Position :STRING[20];
                     SSN            :STRING[11];
                     CASE Category  :CHAR OF
                       '1' :(MonthlySalary :REAL);
                       '2' :(HourlyRate    :REAL);
                       '3' :(Commission,
                             BasicSalary   :REAL;
                             Area          :STRING[20])
                END;
VAR
  PF :TEXT;
  EmployeeRec     :EmployeeRecord;
  Title           :ARRAY [1..9] OF STRING[20];
BEGIN
{ Присваивание заголовков }
  Title[1] := 'ID: ';
```

```
  Title[2] := 'Имя: ';
  Title[3] := 'Должность: ';
  Title[4] := 'Номер социального страхования: ';
  Title[5] := 'Оклад: ';
  Title[6] := 'Тариф: ';
  Title[7] := 'Комиссионные: ';
  Title[8] := 'Основной оклад: ';
  Title[9] := 'Регион: ';
  ASSIGN(PF, FileName);
  REWRITE(PF);
  WRITELN(Header);
  WITH EmployeeRec DO
    BEGIN
      WRITE('Пожалуйста, введите ID сотрудника: '); READLN(ID);
      WRITELN(PF,ID);
      WRITE('Имя: '); READLN(Name);
      WRITELN(PF,Name);
      WRITE('Должность: '); READLN(Position);
      WRITELN(PF,Position);
      WRITE('Номер социального страхования (xxx-xx-xxxx): '); READLN(SSN);
      WRITELN(PF,SSN);
      WRITE('Категория оплаты: '); READLN(Category);
        CASE Category OF
         '1' : BEGIN
                 WRITE('Месячный оклад: ');
                 READLN(MonthlySalary);
                 WRITELN(PF,MonthlySalary)
               END;
         '2' : BEGIN
                 WRITE('Почасовой тариф: ');
                 READLN(HourlyRate);
                 WRITELN(PF,HourlyRate)
               END;
         '3' : BEGIN
                 WRITE('Комиссионный тариф: ');
                 READLN(Commission);
                 WRITELN(PF,Commission);
                 WRITE('Основной оклад: ');
                 READLN(BasicSalary);
                 WRITELN(PF,BasicSalary);
                 WRITE('Регион: ');
                 READLN(Area);
                 WRITELN(PF,Area)
               END
        END
    END;
  CLOSE(PF);
  WRITELN(Separator);
  WRITELN('Файл платежной ведомости был создан. Нажмите клавишу');
  READLN
END.
```

Задание 10-2

```
{ ---------------------------- ЗАДАНИЕ 10-2 ---------------------------- }
PROCEDURE DelRec(VAR NewFile, PayrollFile :TEXT; Employee :EmployeeRecord);
VAR
  SSNumber :STRING[11];
  Found    :INTEGER;
BEGIN
  Found := 0;
  ASSIGN(PayrollFile, FileName);
  RESET(PayrollFile);
  ASSIGN(NewFile, TempFile);
  REWRITE(NewFile);
  WRITE('Пожалуйста, введите номер социального страхования удаляемого
  сотрудника: ');
  READLN(SSNumber);
  WHILE NOT EOF(PayrollFile) DO
    BEGIN
      WITH Employee DO
        BEGIN
          READLN(PayrollFile, ID);
          READLN(PayrollFile, Name);
          READLN(PayrollFile, Position);
          READLN(PayrollFile, SSN);
          READLN(PayrollFile, Category);
          CASE Category OF
           '1' : READLN(PayrollFile, MonthlySalary);
           '2' : READLN(PayrollFile, HourlyRate);
           '3' : BEGIN
                   READLN(PayrollFile, Commission);
                   READLN(PayrollFile, BasicSalary);
                   READLN(PayrollFile, Area)
                 END
          END; { Конец структуры CASE }
          IF SSNumber<>SSN THEN
            BEGIN
              WRITELN(NewFile,ID);
              WRITELN(NewFile,Name);
              WRITELN(NewFile,Position);
              WRITELN(NewFile,SSN);
              WRITELN(NewFile,Category);
              CASE Category OF
               '1' : WRITELN(NewFile,MonthlySalary:0:2);
               '2' : WRITELN(NewFile,HourlyRate:0:2);
               '3' : BEGIN
                       WRITELN(NewFile,Commission:0:2);
                       WRITELN(NewFile,BasicSalary:0:2);
                       WRITELN(NewFile,Area)
                     END
              END; { Конец структуры CASE }
            END
           ELSE
```

```
            Found := 1;
        END { Конец блока WITH }
      END;
  CLOSE(NewFile);
  CLOSE(PayrollFile);
{ Копирование NewFile обратно в файл платежной ведомости }
  ASSIGN(PayrollFile, FileName);
  REWRITE(PayrollFile);
  ASSIGN(NewFile, TempFile);
  RESET(NewFile);
  WHILE NOT EOF(NewFile) DO
    BEGIN
      READLN(NewFile,OneLine);
      WRITELN(PayrollFile,OneLine)
    END;
  CLOSE(NewFile);
  ERASE(NewFile); { Удаление временного файла }
  CLOSE(PayrollFile);
{ Сообщения пользователя }
  IF Found =1 THEN
    WRITELN('Сотрудник', SSNumber, ' удален из файла.')
  ELSE
    BEGIN
      WRITELN('SSN ', SSNumber, ' не найден.');
      WRITELN('Проверьте номер и попытайтесь снова.');
      WRITELN
    END
END;
```

Задание 10-3

```
{ ----------------------------- ЗАДАНИЕ 10-3 ----------------------------- }
PROGRAM EmployeeDataBase2(INPUT, OUTPUT, PayrollFile, NewFile);
CONST
  FileName = 'Payroll.TXT';
  TempFile = 'TEMP.TXT';
  Header = '------------- Главное меню --------------';
  Header1 = '------ База данных сотрудников ----------';
  Header2 = '--------- Запись сотрудника ------------';
  Separator = '----------------------------------------';
TYPE
  EmployeeRecord = RECORD
                     ID             :STRING[5];
                     Name, Position :STRING[20];
                     SSN            :STRING[11];
                     CASE Category  :CHAR OF
                       '1' :(MonthlySalary :REAL);
                       '2' :(HourlyRate    :REAL);
                       '3' :(Commission,
                             BasicSalary   :REAL;
                             Area          :STRING[20])
                     END;
```

```
VAR
  NewFile, PayrollFile :TEXT;
  EmployeeRec     :EmployeeRecord;
  Title           :ARRAY [1..9] OF STRING[20];
  OneLine         :STRING[80];

{ ------------------ Процедура ReadRec -------------------- }
PROCEDURE ReadRec(VAR PayrollFile :TEXT;
  Employee :EmployeeRecord);
VAR
  SSNumber :STRING[11];
  Found    :INTEGER;
BEGIN
  Found := 0; {Reset the flag}
  ASSIGN(PayrollFile, FileName);
  RESET(PayrollFile);
  WRITELN;
  WRITE('Пожалуйста, введите номер социального страхования сотрудника: ');
  READLN(SSNumber);
  WHILE NOT EOF(PayrollFile) DO
    BEGIN
      WITH Employee DO
        BEGIN
          READLN(PayrollFile, ID);
          READLN(PayrollFile, Name);
          READLN(PayrollFile, Position);
          READLN(PayrollFile, SSN);
          READLN(PayrollFile, Category);
          CASE Category OF
           '1' : READLN(PayrollFile, MonthlySalary);
           '2' : READLN(PayrollFile, HourlyRate);
           '3' : BEGIN
                   READLN(PayrollFile, Commission);
                   READLN(PayrollFile, BasicSalary);
                   READLN(PayrollFile, Area)
                 END
          END; { Конец структуры CASE }
          IF SSNumber = SSN THEN
            BEGIN
              WRITELN(Header2);
              WRITELN(Title[1],ID);
              WRITELN(Title[2],Name);
              WRITELN(Title[3],Position);
              WRITELN(Title[4], SSN);
              CASE Category OF
               '1' : WRITELN(Title[5], MonthlySalary:0:2);
               '2' : WRITELN(Title[6], HourlyRate:0:2);
               '3' : BEGIN
                       WRITELN(Title[7], Commission:0:2);
                       WRITELN(Title[8], BasicSalary:0:2);
                       WRITELN(Title[9], Area)
                     END
```

```
                END; { Конец структуры CASE }
                Found := 1
              END
          END { Конец блока WITH }
      END;
    CLOSE(PayrollFile);
    IF Found<>1 THEN
      BEGIN
        WRITELN('SSN не найден в файле.');
        WRITELN('Пожалуйста, попытайтесь снова.');
        WRITELN
      END
END;

{ ------------------ Процедура DelRec ---------------------- }
PROCEDURE DelRec(VAR NewFile, PayrollFile :TEXT;
  Employee :EmployeeRecord);
VAR
  SSNumber :STRING[11];
BEGIN
  ASSIGN(PayrollFile, FileName);
  RESET(PayrollFile);
  ASSIGN(NewFile, TempFile);
  REWRITE(NewFile);
  WRITE('Пожалуйста, введите номер социального страхования удаляемого
  сотрудника: ');
  READLN(SSNumber);

  WHILE NOT EOF(PayrollFile) DO
    BEGIN
      WITH Employee DO
        BEGIN
          READLN(PayrollFile, ID);
          READLN(PayrollFile, Name);
          READLN(PayrollFile, Position);
          READLN(PayrollFile, SSN);
          READLN(PayrollFile, Category);
          CASE Category OF
           '1' : READLN(PayrollFile, MonthlySalary);
           '2' : READLN(PayrollFile, HourlyRate);
           '3' : BEGIN
                   READLN(PayrollFile, Commission);
                   READLN(PayrollFile, BasicSalary);
                   READLN(PayrollFile, Area)
                 END
          END; { Конец структуры CASE }
          IF SSNumber<>SSN THEN
            BEGIN
              WRITELN(NewFile,ID);
              WRITELN(NewFile,Name);
              WRITELN(NewFile,Position);
              WRITELN(NewFile,SSN);
```

```
                WRITELN(NewFile,Category);
                CASE Category OF
                  '1' : WRITELN(NewFile,MonthlySalary:0:2);
                  '2' : WRITELN(NewFile,HourlyRate:0:2);
                  '3' : BEGIN
                          WRITELN(NewFile,Commission:0:2);
                          WRITELN(NewFile,BasicSalary:0:2);
                          WRITELN(NewFile,Area)
                        END
                END; { Конец структуры CASE }
              END
          END { Конец блока WITH }
      END;
    CLOSE(NewFile);
    CLOSE(PayrollFile);
{ Копирование NewFile обратно в файл платежной ведомости}
    ASSIGN(PayrollFile, FileName);
    REWRITE(PayrollFile);
    ASSIGN(NewFile, TempFile);
    RESET(NewFile);
    WHILE NOT EOF(NewFile) DO
    BEGIN
      READLN(NewFile,OneLine);
      WRITELN(PayrollFile,OneLine)
    END;
    CLOSE(NewFile);
    ERASE(NewFile); { Удаление временного файла }
    CLOSE(PayrollFile);
    WRITELN('Сотрудник ', SSNumber, ' удален из файла.')
END;

{ ------------------ Процедура AddRec --------------------- }
PROCEDURE AddRec(VAR NewFile, PayrollFile :TEXT;
    Employee: EmployeeRecord);
BEGIN
    ASSIGN(PayrollFile, FileName);
    RESET(PayrollFile);
    ASSIGN(NewFile, TempFile);
    REWRITE(NewFile);
{ Проверка конца текстового файла }
    WHILE NOT EOF(PayrollFile) DO
      BEGIN
{ Копирование каждой записи из PayrollFile в NewFile }
        READLN(PayrollFile,OneLine);
        WRITELN(NewFile,OneLine)
      END;
{ Прием новой записи с клавиатуры }
    WITH Employee DO
      BEGIN
        WRITE('Пожалуйста, введите ID сотрудника: ');
        READLN(ID);
        WRITE('Имя: ');                                    READLN(Name);
```

```
      WRITE('Должность: ');                                    READLN(Position);
      WRITE('Номер социального страхования (xxx-xx-xxxx): '); READLN(SSN);
      WRITE('Категория оплаты: ');                             READLN(Category);
      CASE Category OF
       '1' : BEGIN
               WRITE('Месячный оклад: ');
               READLN(MonthlySalary);
             END;
       '2' : BEGIN
               WRITE('Тариф: ');
               READLN(HourlyRate);
             END;
       '3' : BEGIN
               WRITE('Комиссионные: ');
               READLN(Commission);
               WRITE('Основной оклад: ');
               READLN(BasicSalary);
               WRITE('Регион: ');
               READLN(Area)
             END
      END;
{ Сохранение информации в NewFile }
      WRITELN(NewFile, ID);
      WRITELN(NewFile, Name);
      WRITELN(NewFile, Position);
      WRITELN(NewFile, SSN);
      WRITELN(NewFile, Category);
      CASE Category OF
       '1' : WRITELN(NewFile, MonthlySalary:0:2);
       '2' : WRITELN(NewFile, HourlyRate:0:2);
       '3' : BEGIN
               WRITELN(NewFile, Commission:0:2);
               WRITELN(NewFile, BasicSalary:0:2);
               WRITELN(NewFile, Area)
             END
      END
    END;
  CLOSE(NewFile);
  CLOSE(PayrollFile);
{ Копирование NewFile обратно в файл платежной ведомости }
  ASSIGN(PayrollFile, FileName);
  REWRITE(PayrollFile);
  ASSIGN(NewFile, TempFile);
  RESET(NewFile);
  WHILE NOT EOF(NewFile) DO
    BEGIN
      READLN(NewFile,OneLine);
      WRITELN(PayrollFile,OneLine)
    END;
  CLOSE(NewFile);
  ERASE(NewFile); { Удаление временного файла }
  CLOSE(PayrollFile)
END;
```

```
{ ----------------- Процедура UpdateRec ------------------- }
PROCEDURE UpdateRec(VAR NewFile, PayrollFile :TEXT;
  Employee :EmployeeRecord);

VAR
  SSNumber :STRING[11];
  Found    :INTEGER;
BEGIN
  Found := 0;
  ASSIGN(PayrollFile, FileName);
  RESET(PayrollFile);
  ASSIGN(NewFile, TempFile);
  REWRITE(NewFile);
  WRITE('Пожалуйста, введите номер социального страхования удаляемого
  сотрудника: ');
  READLN(SSNumber);
  WHILE NOT EOF(PayrollFile) DO
    BEGIN
      WITH Employee DO
        BEGIN
          READLN(PayrollFile, ID);
          READLN(PayrollFile, Name);
          READLN(PayrollFile, Position);
          READLN(PayrollFile, SSN);
          READLN(PayrollFile, Category);
          CASE Category OF
           '1' : READLN(PayrollFile, MonthlySalary);
           '2' : READLN(PayrollFile, HourlyRate);
           '3' : BEGIN
                   READLN(PayrollFile, Commission);
                   READLN(PayrollFile, BasicSalary);
                   READLN(PayrollFile, Area)
                 END
          END; { Конец структуры CASE }
          IF SSNumber<>SSN THEN
            BEGIN
              WRITELN(NewFile,ID);
              WRITELN(NewFile,Name);
              WRITELN(NewFile,Position);
              WRITELN(NewFile,SSN);
              WRITELN(NewFile,Category);
                CASE Category OF
                 '1' : WRITELN(NewFile,MonthlySalary:0:2);
                 '2' : WRITELN(NewFile,HourlyRate:0:2);
                 '3' : BEGIN
                         WRITELN(NewFile,Commission:0:2);
                         WRITELN(NewFile,BasicSalary:0:2);
                         WRITELN(NewFile,Area)
                       END
                END; { Конец структуры CASE }
            END
          ELSE
```

```
                BEGIN
                  Found := 1;
                  WRITELN('Пожалуйста, введите обновленную информацию:');
                  WRITE('ID: '); READLN(ID);
                  WRITELN(NewFile,ID);
                  WRITE('Имя: '); READLN(Name);
                  WRITELN(NewFile,Name);
                  WRITE('Должность: '); READLN(Position);
                  WRITELN(NewFile,Position);
                  WRITELN(NewFile,SSN);
                  WRITE('Категория: '); READLN(Category);
                  WRITELN(NewFile,Category);
                    CASE Category OF
                     '1' : BEGIN
                             WRITE('Оклад: ');
                             READLN(MonthlySalary);
                             WRITELN(NewFile,MonthlySalary:0:2)
                           END;
                     '2' : BEGIN
                             WRITE('Почасовой тариф: ');
                             READLN(HourlyRate);
                             WRITELN(NewFile,HourlyRate:0:2)
                           END;
                     '3' : BEGIN
                             WRITE('Комиссионные: ');
                             READLN(Commission);
                             WRITELN(NewFile,Commission:0:2);
                             WRITE('Основной оклад: ');
                             READLN(BAsicSalary);
                             WRITELN(NewFile,BasicSalary:0:2);
                             WRITE('Регион: ');
                             READLN(Area);
                             WRITELN(NewFile,Area)
                           END
                    END; { Конец структуры CASE }
                END
            END { Конец блока WITH }
      END;
    CLOSE(NewFile);
    CLOSE(PayrollFile);
  { Копирование NewFile обратно в файл платежной ведомости }
    ASSIGN(PayrollFile, FileName);
    REWRITE(PayrollFile);
    ASSIGN(NewFile, TempFile);
    RESET(NewFile);
    WHILE NOT EOF(NewFile) DO
      BEGIN
        READLN(NewFile,OneLine);
        WRITELN(PayrollFile,OneLine)
      END;
    CLOSE(NewFile);
    ERASE(NewFile); { Удаление временного файла }
```

```
  CLOSE(PayrollFile);
{ Сообщения пользователя }
  IF Found =1 THEN
    WRITELN('Сотрудник ', SSNumber, ' обновлен.')
  ELSE
    BEGIN
      WRITELN('SSN ', SSNumber, ' не найден.');
      WRITELN('Проверьте номер и попытайтесь снова.');
      WRITELN
    END
END;
{ -------------------- Процедура Menu --------------------- }
PROCEDURE Menu;
VAR
  Option :INTEGER;
BEGIN
  WRITELN(Header);
  WRITELN;
  WRITELN('1. Показать запись сотрудника.');
  WRITELN('2. Добавить нового сотрудника.');
  WRITELN('3. Удалить сотрудника.');
  WRITELN('4. Обновить запись сотрудника.');
  WRITELN('5. Выход.');
  WRITELN(Separator);
  WRITE('Сделайте выбор и нажмите цифру: ');
  READLN(Option);
  CASE Option OF
     1 : ReadRec(PayrollFile, EmployeeRec);
     2 : AddRec(NewFile, PayrollFile, EmployeeRec);
     3 : DelRec(NewFile, PayrollFile, EmployeeRec);
     4 : UpdateRec(NewFile, PayrollFile, EmployeeRec);
     5 : Exit
  END;
  Menu
END;
{ -------------------- Главная программа ---------------------- }
BEGIN
{ Присваивание заголовков }
  Title[1] := 'ID: ';
  Title[2] := 'Имя: ';
  Title[3] := 'Должность: ';
  Title[4] := 'SSN: ';
  Title[5] := 'Оклад: ';
  Title[6] := 'Тариф: ';
  Title[7] := 'Комиссионные: ';
  Title[8] := 'Основной оклад: ';
  Title[9] := 'Регион: ';
  Menu
END.
```

Глава 11

Задание 11–1

```
{ ------------------------------ ЗАДАНИЕ 11-1 --------------------------- }
PROGRAM CreateEmployeeDataBase3(INPUT, OUTPUT, PayrollFile, NewFile);
{ Эта программа создает текстовый файл платежной ведомости PR.TXT }
CONST
  FileName = 'PR.TXT';
  Header = '---------- Создание файла платежной ведомости ----------';
  Separator = '------------------------------------------------';
TYPE
  EmployeeRecord = RECORD
                     ID             :STRING[5];
                     Name, Position :STRING[20];
                     SSN            :STRING[11];
                     CASE Category  :CHAR OF
                       '1' :(MonthlySalary :REAL);
                       '2' :(HourlyRate    :REAL);
                       '3' :(Commission,
                             BasicSalary   :REAL;
                             Area          :STRING[20])
                     END;
  EmployeePointer = ^EmployeeRecord;
VAR
  PF         :TEXT;
  RecPointer :EmployeePointer;
  Title      :ARRAY [1..9] OF STRING[20];

BEGIN
{ Присваивание заголовков }
  Title[1] := 'ID: ';
  Title[2] := 'Имя: ';
  Title[3] := 'Должность: ';
  Title[4] := 'SSN: ';
  Title[5] := 'Оклад: ';
  Title[6] := 'Тариф: ';
  Title[7] := 'Комиссионные: ';
  Title[8] := 'Основной оклад: ';
  Title[9] := 'Регион: ';

  ASSIGN(PF, FileName);
  REWRITE(PF);
  WRITELN(Header);
  WITH RecPointer^ DO
  BEGIN
    WRITE('Пожалуйста, введите ID сотрудника: ');          READLN(ID);
    WRITELN(PF,ID);
    WRITE('Имя: ');                                        READLN(Name);
    WRITELN(PF,Name);
    WRITE('Должность: ');                                  READLN(Position);
    WRITELN(PF,Position);
```

```
    WRITE('Номер социального страхования (xxx-xx-xxxx): '); READLN(SSN);
    WRITELN(PF,SSN);
    WRITE('Категория оплаты: ');                              READLN(Category);
      CASE Category OF
       '1' : BEGIN
               WRITE('Месячный оклад: ');
               READLN(MonthlySalary);
               WRITELN(PF,MonthlySalary)
             END;
       '2' : BEGIN
               WRITE('Почасовой тариф: ');
               READLN(HourlyRate);
               WRITELN(PF,HourlyRate)
             END;
       '3' : BEGIN
               WRITE('Комиссионный тариф: ');
               READLN(Commission);
               WRITELN(PF,Commission);
               WRITE('Основной оклад: ');
               READLN(BasicSalary);
               WRITELN(PF,BasicSalary);
               WRITE('Регион: ');
               READLN(Area);
               WRITELN(PF,Area)
             END
      END
    END;
  CLOSE(PF);
  WRITELN(Separator);
  WRITELN('Файл платежной ведомости был создан. Нажмите клавишу');
  READLN
END.
```

Задание 11–2

```
{ ---------------------------- ЗАДАНИЕ 11-2 ---------------------------- }
PROGRAM EmployeeDataBase2(INPUT, OUTPUT, PayrollFile, NewFile);
{ Эта программа использует файл базы данных сотрудников payroll.txt
для обработки записей с помощью указателей. }

CONST
  FileName = 'payroll.txt';
  TempFile = 'temp.txt';
  Header = '------------- Главное меню --------------';
  Header1 = '------ База данных сотрудников ----------';
  Header2 = '--------- Запись сотрудника -------------';
  Separator = '-----------------------------------------';

TYPE
  EmployeeRecord = RECORD
                     ID             :STRING[5];
                     Name, Position :STRING[20];
                     SSN            :STRING[11];
```

```
                    CASE Category  :CHAR OF
                     '1' :(MonthlySalary :REAL);
                     '2' :(HourlyRate    :REAL);
                     '3' :(Commission,
                           BasicSalary   :REAL;
                           Area          :STRING[20])
                    END;
  SSNstring = STRING[11];
  EmployeePointer = ^EmployeeRecord;
VAR
  NewFile, PayrollFile :TEXT;
  EmployeeRec          :EmployeePointer;
  Title                :ARRAY [1..9] OF STRING[20];
  OneLine              :STRING[80];

{ ----------------- Процедура SearchRec ------------------- }
PROCEDURE SearchRec(VAR PayrollFile :TEXT;
                    Employee :EmployeePointer;
                    SSNumber :SSNstring;
                    VAR Found :INTEGER);
BEGIN
  Found := 0;
  ASSIGN(PayrollFile, FileName);
  RESET(PayrollFile);
  WHILE NOT EOF(PayrollFile) DO
    BEGIN
      WITH Employee^ DO
        BEGIN
          READLN(PayrollFile, ID);
          READLN(PayrollFile, Name);
          READLN(PayrollFile, Position);
          READLN(PayrollFile, SSN);
          READLN(PayrollFile, Category);
          CASE Category OF
           '1' : READLN(PayrollFile, MonthlySalary);
           '2' : READLN(PayrollFile, HourlyRate);
           '3' : BEGIN
                   READLN(PayrollFile, Commission);
                   READLN(PayrollFile, BasicSalary);
                   READLN(PayrollFile, Area)
                 END
          END; { Конец структуры CASE }
          IF SSNumber = SSN THEN
            Found := 1;
        END { Конец блока WITH }
    END;
  CLOSE(PayrollFile);
END;

{ ------------------ Процедура ReadRec -------------------- }
PROCEDURE ReadRec(VAR PayrollFile :TEXT;
  Employee :EmployeePointer);
```

```
VAR
  SSNumber :STRING[11];
  Found    :INTEGER;
BEGIN
  WRITELN;
  WRITE('Пожалуйста, введите номер социального страхования сотрудника: ');
  READLN(SSNumber);
  SearchRec(PayrollFile, Employee, SSNumber, Found);
  IF Found =1 THEN
    BEGIN
      ASSIGN(PayrollFile, FileName);
      RESET(PayrollFile);
      WHILE NOT EOF(PayrollFile) DO
        BEGIN
          WITH Employee^ DO
            BEGIN
              READLN(PayrollFile, ID);
              READLN(PayrollFile, Name);
              READLN(PayrollFile, Position);
              READLN(PayrollFile, SSN);
              READLN(PayrollFile, Category);
              CASE Category OF
               '1' : READLN(PayrollFile, MonthlySalary);
               '2' : READLN(PayrollFile, HourlyRate);
               '3' : BEGIN
                       READLN(PayrollFile, Commission);
                       READLN(PayrollFile, BasicSalary);
                       READLN(PayrollFile, Area)
                     END
              END; { Конец структуры CASE }
              IF SSNumber = SSN THEN
                BEGIN
                  WRITELN(Header2);
                  WRITELN(Title[1],ID);
                  WRITELN(Title[2],Name);
                  WRITELN(Title[3],Position);
                  WRITELN(Title[4], SSN);
                  CASE Category OF
                   '1' : WRITELN(Title[5], MonthlySalary:0:2);
                   '2' : WRITELN(Title[6], HourlyRate:0:2);
                   '3' : BEGIN
                           WRITELN(Title[7], Commission:0:2);
                           WRITELN(Title[8], BasicSalary:0:2);
                           WRITELN(Title[9], Area)
                         END
                  END; { Конец структуры CASE }
                END
            END { Конец блока WITH }
        END;
      CLOSE(PayrollFile)
    END
      ELSE {Если не найден }
```

```
        BEGIN
          WRITELN('SSN не найден в файле.');
          WRITELN('Пожалуйста, попытайтесь снова.');
          WRITELN
        END
END;
{ ------------------ Процедура DelRec ---------------------- }
PROCEDURE DelRec(VAR NewFile, PayrollFile :TEXT;
  Employee :EmployeePointer);

VAR
  SSNumber :STRING[11];
  Found    :INTEGER;

BEGIN
  WRITE('Пожалуйста, введите номер социального страхования удаляемого
  сотрудника: ');
  READLN(SSNumber);
  SearchRec(PayrollFile, Employee, SSNumber, Found);
  IF Found =1 THEN
    BEGIN
      ASSIGN(NewFile, TempFile);
      REWRITE(NewFile);
      ASSIGN(PayrollFile, FileName);
      RESET(PayrollFile);
      WHILE NOT EOF(PayrollFile) DO
        BEGIN
          WITH Employee^ DO
            BEGIN
              READLN(PayrollFile, ID);
              READLN(PayrollFile, Name);
              READLN(PayrollFile, Position);
              READLN(PayrollFile, SSN);
              READLN(PayrollFile, Category);
              CASE Category OF
               '1' : READLN(PayrollFile, MonthlySalary);
               '2' : READLN(PayrollFile, HourlyRate);
               '3' : BEGIN
                       READLN(PayrollFile, Commission);
                       READLN(PayrollFile, BasicSalary);
                       READLN(PayrollFile, Area)
                     END
              END; { Конец структуры CASE }
              IF SSNumber<>SSN THEN
                BEGIN
                  WRITELN(NewFile,ID);
                  WRITELN(NewFile,Name);
                  WRITELN(NewFile,Position);
                  WRITELN(NewFile,SSN);
                  WRITELN(NewFile,Category);
                  CASE Category OF
                   '1' : WRITELN(NewFile,MonthlySalary:0:2);
                   '2' : WRITELN(NewFile,HourlyRate:0:2);
```

```
                    '3' : BEGIN
                            WRITELN(NewFile,Commission:0:2);
                            WRITELN(NewFile,BasicSalary:0:2);
                            WRITELN(NewFile,Area)
                          END
                  END; { Конец структуры CASE }
                END;
            END { Конец блока WITH }
        END; {End of DO }
      CLOSE(NewFile);
      CLOSE(PayrollFile);
{ Копирование NewFile обратно в файл платежной ведомости }
      ASSIGN(PayrollFile, FileName);
      REWRITE(PayrollFile);
      ASSIGN(NewFile, TempFile);
      RESET(NewFile);
      WHILE NOT EOF(NewFile) DO
        BEGIN
        READLN(NewFile,OneLine);
        WRITELN(PayrollFile,OneLine)
        END;
      CLOSE(NewFile);
      ERASE(NewFile); { Удаление временного файла }
      CLOSE(PayrollFile);
{ Сообщения пользователя }
      WRITELN('Сотрудник ', SSNumber, ' удален из файла.')
    END { Конец блока "Если найден.." }
  ELSE {Если не найден }
    BEGIN
      WRITELN('SSN ', SSNumber, ' не найден.');
      WRITELN('Проверьте номер и попытайтесь снова.');
      WRITELN
    END
END;

{ ------------------ Процедура AddRec ---------------------- }
PROCEDURE AddRec(VAR NewFile, PayrollFile :TEXT;
  Employee: EmployeePointer);

BEGIN
  ASSIGN(PayrollFile, FileName);
  RESET(PayrollFile);
  ASSIGN(NewFile, TempFile);
  REWRITE(NewFile);
  WHILE NOT EOF(PayrollFile) DO
    BEGIN
{ Копирование каждой записи из PayrollFile в NewFile }
      READLN(PayrollFile,OneLine);
      WRITELN(NewFile,OneLine)
    END;
{ Прием новой записи с клавиатуры }
  WITH Employee^ DO
    BEGIN
```

```
      WRITE('Пожалуйста, введите ID сотрудника: ');
      READLN(ID);
      WRITE('Имя: ');                                          READLN(Name);
      WRITE('Должность: ');                                    READLN(Position);
      WRITE('Номер социального страхования (xxx-xx-xxxx): ');  READLN(SSN);
      WRITE('Категория оплаты: ');                             READLN(Category);
      CASE Category OF
       '1' : BEGIN
               WRITE('Месячный оклад: ');
               READLN(MonthlySalary)
             END;
       '2' : BEGIN
               WRITE('Тариф: ');
               READLN(HourlyRate)
             END;
       '3' : BEGIN
               WRITE('Комиссионные: ');
               READLN(Commission);
               WRITE('Основной оклад: ');
               READLN(BasicSalary);
               WRITE('Регион: ');
               READLN(Area)
             END
    END;
{ Сохранение информации в NewFile }
    WRITELN(NewFile, ID);
    WRITELN(NewFile, Name);
    WRITELN(NewFile, Position);
    WRITELN(NewFile, SSN);
    WRITELN(NewFile, Category);
    CASE Category OF
     '1' : WRITELN(NewFile, MonthlySalary:0:2);
     '2' : WRITELN(NewFile, HourlyRate:0:2);
     '3' : BEGIN
             WRITELN(NewFile, Commission:0:2);
             WRITELN(NewFile, BasicSalary:0:2);
             WRITELN(NewFile, Area)
           END
    END
  END;
  CLOSE(NewFile);
  CLOSE(PayrollFile);
{ Копирование NewFile обратно в файл платежной ведомости }
  ASSIGN(PayrollFile, FileName);
  REWRITE(PayrollFile);
  ASSIGN(NewFile, TempFile);
  RESET(NewFile);
  WHILE NOT EOF(NewFile) DO
    BEGIN
      READLN(NewFile,OneLine);
      WRITELN(PayrollFile,OneLine)
    END;
```

```
  CLOSE(NewFile);
  ERASE(NewFile); { Удаление временного файла }
  CLOSE(PayrollFile)
END;

{ ---------------- Процедура UpdateRec -------------------- }
PROCEDURE UpdateRec(VAR NewFile, PayrollFile :TEXT;
  Employee :EmployeePointer);
VAR
  SSNumber :STRING[11];
  Found    :INTEGER;
BEGIN
  WRITE('Пожалуйста, введите номер социального страхования удаляемого
  сотрудника: ');
  READLN(SSNumber);
  SearchRec(PayrollFile, Employee, SSNumber, Found);
  IF Found = 1 THEN
    BEGIN
      ASSIGN(PayrollFile, FileName);
      RESET(PayrollFile);
      ASSIGN(NewFile, TempFile);
      REWRITE(NewFile);
      WHILE NOT EOF(PayrollFile) DO
        BEGIN
          WITH Employee^ DO
            BEGIN
              READLN(PayrollFile, ID);
              READLN(PayrollFile, Name);
              READLN(PayrollFile, Position);
              READLN(PayrollFile, SSN);
              READLN(PayrollFile, Category);
              CASE Category OF
               '1' : READLN(PayrollFile, MonthlySalary);
               '2' : READLN(PayrollFile, HourlyRate);
               '3' : BEGIN
                       READLN(PayrollFile, Commission);
                       READLN(PayrollFile, BasicSalary);
                       READLN(PayrollFile, Area)
                     END
              END; { Конец структуры CASE }
              IF SSNumber<>SSN THEN
                BEGIN
                  WRITELN(NewFile,ID);
                  WRITELN(NewFile,Name);
                  WRITELN(NewFile,Position);
                  WRITELN(NewFile,SSN);
                  WRITELN(NewFile,Category);
                  CASE Category OF
                   '1' : WRITELN(NewFile,MonthlySalary:0:2);
                   '2' : WRITELN(NewFile,HourlyRate:0:2);
                   '3' : BEGIN
                           WRITELN(NewFile,Commission:0:2);
```

```
                        WRITELN(NewFile,BasicSalary:0:2);
                        WRITELN(NewFile,Area)
                      END
                END { Конец структуры CASE }
              END { Конец блока }
            ELSE
              BEGIN
                WRITELN('Пожалуйста, введите обновленную информацию:');
                WRITE('ID: '); READLN(ID);
                WRITELN(NewFile,ID);
                WRITE('Имя: '); READLN(Name);
                WRITELN(NewFile,Name);
                WRITE('Должность: '); READLN(Position);
                WRITELN(NewFile,Position);
                WRITELN(NewFile,SSN);
                WRITE('Категория: '); READLN(Category);
                WRITELN(NewFile,Category);
                CASE Category OF
                 '1' : BEGIN
                         WRITE('Оклад: ');
                         READLN(MonthlySalary);
                         WRITELN(NewFile,MonthlySalary:0:2)
                       END;
                 '2' : BEGIN
                         WRITE('Почасовой тариф: ');
                         READLN(HourlyRate);
                         WRITELN(NewFile,HourlyRate:0:2)
                       END;
                 '3' : BEGIN
                         WRITE('Комиссионные: ');
                         READLN(Commission);
                         WRITELN(NewFile,Commission:0:2);
                         WRITE('Основной оклад: ');
                         READLN(BAsicSalary);
                         WRITELN(NewFile,BasicSalary:0:2);
                         WRITE('Регион: ');
                         READLN(Area);
                         WRITELN(NewFile,Area)
                       END
                END { Конец структуры CASE }
              END { Конец блока ELSE }
          END { Конец блока WITH }
      END; { Конец DO }
    CLOSE(NewFile);
    CLOSE(PayrollFile);
{ Копирование NewFile обратно в файл платежной ведомости }
    ASSIGN(PayrollFile, FileName);
    REWRITE(PayrollFile);
    ASSIGN(NewFile, TempFile);
    RESET(NewFile);
    WHILE NOT EOF(NewFile) DO
      BEGIN
```

```
          READLN(NewFile,OneLine);
          WRITELN(PayrollFile,OneLine)
        END;
      CLOSE(NewFile);
      ERASE(NewFile); { Удаление временного файла }
      CLOSE(PayrollFile);
{ Сообщения пользователя }
      WRITELN('Сотрудник ', SSNumber, ' обновлен.')
    END { Конец блока }
  ELSE
    BEGIN
      WRITELN('SSN ', SSNumber, ' не найден.');
      WRITELN('Проверьте номер и попытайтесь снова.');
      WRITELN
    END
END;

{ -------------------- Процедура Menu --------------------- }
PROCEDURE Menu;
VAR
  Option :INTEGER;
BEGIN
  WRITELN(Header);
  WRITELN;
  WRITELN('1. Показать запись сотрудника.');
  WRITELN('2. Добавить нового сотрудника.');
  WRITELN('3. Удалить сотрудника.');
  WRITELN('4. Обновить запись сотрудника.');
  WRITELN('5. Выход.');
  WRITELN(Separator);
  WRITE('Сделайте выбор и нажмите цифру: ');
  READLN(Option);
  CASE Option OF
     1 : ReadRec(PayrollFile, EmployeeRec);
     2 : AddRec(NewFile, PayrollFile, EmployeeRec);
     3 : DelRec(NewFile, PayrollFile, EmployeeRec);
     4 : UpdateRec(NewFile, PayrollFile, EmployeeRec);
     5 : Exit
  END;
  Menu
END;

{ --------------------- Главная программа ---------------------- }

BEGIN
{ Присваивание заголовков }
  Title[1] := 'ID: ';
  Title[2] := 'Имя: ';
  Title[3] := 'Должность: ';
  Title[4] := 'SSN: ';
  Title[5] := 'Оклад: ';
  Title[6] := 'Тариф: ';
  Title[7] := 'Комиссионные: ';
```

```
  Title[8] := 'Основной оклад: ';
  Title[9] := 'Регион: ';
  Menu
END.
```

Задание 11–3

```
{ ----------------------------- ЗАДАНИЕ 11-3 ----------------------------- }
PROGRAM LinkedListDemo(INPUT, OUTPUT, NamesFile);

{ Программа инициализирует связанный список, который хранит строки. Она добав-
ляет данные в список, выводит на экран его содержимое и сохраняет его в файле
namelist.bin. Она также читает файл и добавляет его содержимое в список. }

CONST
  FileName = 'namelist.bin';
  Header = '------------- Главное меню --------------';
  Separator = '-----------------------------------------';

TYPE
  DataString = STRING[30];
  ListPointer = ^ListRecord;
  ListRecord = RECORD
                 DataField :DataString;
                 NextField :ListPointer
               END;
  NamesFile = FILE OF DataString;

VAR
  FirstPointer :ListPointer;
  MyListFile   :NamesFile;
{ ---------------- Процедура BuildList -------------------- }
PROCEDURE BuildList(VAR FirstPointer :ListPointer;
  DataItem :DataString);
{Примечание: FirstPointer передается с помощью ключевого слова VAR,
так как он будет обновлен этой процедурой. }
VAR
  ToolPointer :ListPointer;
BEGIN
  NEW(ToolPointer);
  ToolPointer^.DataField := DataItem;
  ToolPointer^.NextField := FirstPointer;
  FirstPointer := ToolPointer
END;

{ ----------------- Процедура ReadList -------------------- }
PROCEDURE ReadList(FirstPointer :ListPointer);

VAR
  CurrentPointer :ListPointer;
BEGIN
  CurrentPointer := FirstPointer;
  WHILE CurrentPointer<>NIL DO
    BEGIN
      WRITELN(CurrentPointer^.DataField);
```

```
      CurrentPointer := CurrentPointer^.NextField
    END;
  WRITELN
END;
{ ----------------- Процедура GetData ---------------------- }
PROCEDURE GetData(VAR FirstPointer :ListPointer);

VAR
  Name :DataString;

BEGIN
  WRITELN('Введите имена, добавляемые в список,', ' затем нажмите ENTER.');
{ Чтение первого элемента данных }
  READLN(Name);
{ Проверка конца данных }
  WHILE LENGTH(Name)<>0 DO
    BEGIN
      BuildList(FirstPointer, Name);
      READLN(Name)
    END
END;

{ ----------------- Процедура DisplayInfo ----------------- }
PROCEDURE DisplayInfo(FirstPointer :ListPointer);

BEGIN
  WRITELN(Separator);
  WRITELN('Содержимое списка: ');
  ReadList(FirstPointer);
  WRITE('Для продолжения нажмите клавишу');
  READLN
END;

{ ---------------- Процедура SaveList ---------------------- }
PROCEDURE SaveList(FirstPointer :ListPointer;
  VAR MyListFile: NamesFile);

VAR
  CurrentPointer :ListPointer;

BEGIN
  ASSIGN(MyListFile, FileName);
  REWRITE(MyListFile);
  CurrentPointer := FirstPointer;
  WHILE CurrentPointer<>NIL DO
    BEGIN
      WRITE(MyListFile, CurrentPointer^.DataField);
      CurrentPointer := CurrentPointer^.NextField
    END;
  CLOSE(MyListFile)
END;

{ ----------------- Процедура ReadFile --------------------- }
PROCEDURE ReadFile(VAR FirstPointer :ListPointer;
  VAR MyListFile: NamesFile);
```

```
VAR
  Name :DataString;

BEGIN
  ASSIGN(MyListFile, FileName);
  RESET(MyListFile);
  WHILE NOT EOF (MyListFile) DO
    BEGIN
      READ(MyListFile, Name);
      BuildList(FirstPointer, Name);
    END;
  CLOSE(MyListFile)
END;

{ ------------------ Процедура Menu ----------------------- }
PROCEDURE Menu;
VAR
  Option :INTEGER;

BEGIN
  WRITELN(Header);
  WRITELN('1. Добавить данные с клавиатуры.');
  WRITELN('2. Показать список.');
  WRITELN('3. Добавить данные из файла.');
  WRITELN('4. Сохранить список в файле.');
  WRITELN('5. Выход.');
  WRITELN(Separator);
  WRITE('Сделайте выбор и нажмите цифру: ');
  READLN(Option);
  CASE Option OF
     1 : GetData(FirstPointer);
     2 : DisplayInfo(FirstPointer);
     3 : ReadFile(FirstPointer, MyListFile);
     4 : SaveList(FirstPointer, MyListFile);
     5 : Exit
  END;
  Menu
END;
{ --------------------- Главная программа ---------------------- }
BEGIN
{ Инициализация пустого списка }
  FirstPointer := NIL;
  menu
END.
```

Задание 11–4

```
{ ------------------------ Процедура UpdateRec -------------------------- }
PROCEDURE UpdateRec(FirstPointer :ListPointer);
{ Эта процедура обновляет информацию записи определенного сотрудника.
  Она вызывает процедуру SearchList для поиска в списке по номеру социального
  страхования сотрудника. Новая информация принимается от пользователя,
  в противном случае выдается сообщение "не найден".}
```

```
VAR
  CurrentPointer :ListPointer;
  SSNumber       :SSNstring;
  Found          :BOOLEAN;
BEGIN
  Found := FALSE;
  WRITELN(Separator);
  WRITE('Введите номер социального страхования сотрудника:'); READLN(SSNumber);
  SearchList(FirstPointer, CurrentPointer,
             SSNumber, Found);
  IF NOT Found THEN
    WRITELN('SSN: ', SSNumber, ' Не найден')
  ELSE
    WITH CurrentPointer^.DataField DO
      BEGIN
        WRITELN('Пожалуйста, введите новую информацию о',
                ' сотруднике (SSN: ', SSNumber,'):');
        WRITE('ID: '); READLN(ID);
        WRITE('Имя: '); READLN(Name);
        WRITE('Должность: '); READLN(Position);
        WRITE('Почасовой тариф: '); READLN(Rate);
        WRITELN('Запись обновлена.')
      END;
  WRITE('Для продолжения нажмите клавишу');
  READLN
END;
```

О дискете

Прилагаемая дискета содержит исходные тексты и скомпилированные версии приведенных в книге примеров и заданий, а также другие файлы, необходимые для выполнения этих упражнений. После установки корневой каталог с материалами будет иметь имя **Learn Pascal** и включать в себя 12 каталогов, по одному на каждую из 11 глав и один под именем TextFiles. Примеры имеют названия вида 1-01.pas. Решения заданий можно найти в файлах вида Drl1-1.pas.

Для установки файлов на ваш жесткий диск необходимо запустить программу Setup.exe, находящуюся на дискете, выбрать подходящий каталог для установки и нажать кнопку Unzip.

Для получения более подробной информации смотрите файл Readme, находящийся на дискете.

Алфавитный указатель

A

ABS, смешанная функция, 32
AND, логический оператор, 44
APPEND, процедура, 187
ARCTAN, тригонометрическая функция, 32
ASCII, код, 36
  порядковый номер символа, 37
    пример вывода на экран, 37, 38
ASSIGN, процедура, 167

B

BEGIN, ключевое слово, 11
BEGIN-END, блок, 52
BOOLEAN, тип данных, 29, 42
BYTE, целочисленный тип, 30, 38

C

CASE, конструкция множественного выбора, 61
CASE-ELSE, 66
CHAR, тип данных, 29
CHR, символьная функция, 37
CLOSE, процедура, 168
COMP, 31
CONCAT, функция, 126
CONST, секция раздела объявлений, 22
COPY, функция, 126
COS, тригонометрическая функция, 32

D

DELETE, процедура, 126
DISPOSE, процедура, 228
DIV, оператор целочисленного деления, 16, 18
DOUBLE, вещественный тип, 31
DOWNTO, ключевое слово, 76

E

ELSE-IF, 55
END, ключевое слово, 11
EOF, функция, 121, 169
EOLN, функция, 120, 169
ERASE, процедура, 187
EXP, смешанная функция, 32
EXTENDED, вещественный тип, 31

F

FILE, тип, 165
FOR, цикл, 72
FORWARD, ключевое слово, 141
FRAC, дополнительная функция, 35

G

GOTO, оператор безусловного перехода, 64

I

IF-ELSE, вложенность, 60
IF-THEN, конструкция, 50
IF-THEN-ELSE, конструкция, 53
IN, оператор, 151
INPUT, оператор, 111
INPUT, файл, 110
INSERT, процедура, 127
INT, дополнительная функция, 35
INTEGER, тип данных, 29

L

LENGTH, функция определения длины строки, 41, 125
LN, смешанная функция, 32
LONGINT, целочисленный тип, 30

M

MAXINT, константа, 22, 30
MOD, 16

N

NEW, процедура, 228
NOT, логический оператор, 44

O

OR, логический оператор, 44
ORD, символьная функция, 37
OUTPUT, оператор, 110
OUTPUT, файл, 110

P

PACKED ARRAY OF CHAR, упакованный массив символов, 40
POS, функция, 126
PRED, символьная функция, 37
PROGRAM, ключевое слово, 12

R

RANDOM, дополнительная функция, 35
READ, оператор, 24, 113, 114, 117
READ, процедура, 169
READLN, оператор, 24, 111, 117
  ввод символов, 116
READLN, процедура, 169
REAL, вещественный тип, 29, 31
REPEAT, цикл, 82
RESET, процедура, 167
REWRITE, процедура, 175
ROUND, арифметическая функция, 23, 32

S

SHORTINT, целочисленный тип, 30
SIN, тригонометрическая функция, 32
SINGLE, вещественный тип, 31
SQR, смешанная функция, 32
SQRT, смешанная функция, 32
STRING, строковый тип, 40, 123
SUCC, символьная функция, 37

T

TEXT, тип файла, 166
  вывод на экран, 172
  создание, 175
  чтение, 166, 173
TRUNC, арифметическая функция, 23, 32
TYPE, оператор, 91
TYPE, секция, объявление массивов в, 100

V

VAR, секция раздела объявлений, 18, 22

W

WHILE, цикл, 79
WITH, оператор, 156
WORD, целочисленный тип, 30
WRITE, оператор, 13
WRITE, процедура, 176
WRITELN, оператор, 13
WRITELN, процедура, 176

X

XOR, логический оператор (Турбо Паскаль), 45

А

арифметические выражения, вычисление, 16
арифметический оператор, 14, 17
  приоритет, 16
арифметический цикл, 71

Б

базовый тип, 147
безусловный переход, 64
бесконечный цикл, 65
бинарный оператор, 17
блоки BEGIN-END, 52

В

вариантные записи, 194
  объявление, 196
  список полей, 196
ввод символов, 114, 116
вещественное деление, 15
вещественные типы данных, 87
вещественные типы Турбо Паскаля, 31
вещественные числа, 15

вложенные записи, 158
вложенные операторы, 60
вложенные условия, 57
вложенный цикл, 78
вывод текста, 13
вывод, форматирование, 25

Г

глобальные переменные, 136

Д

двоичный файл, 166
двумерный массив, 94, 103
деление по модулю, оператор DIV, 16
динамическое выделение памяти, 227
длина строки, 41

З

записи, 153
  вариантные, 194
  вложенные, 158
  обновление, 212
  объявление, 154
  удаление, 203
  указатели на, 232
зарезервированные слова, 12

И

идентификатор, 12
  стандартные, 19
именованные константы, 21
исключающее ИЛИ, 45

К

комментарии, 11, 14, 16, 18, 35, 36, 46, 53, 60, 64, 78, 79, 82
конец строки, 110, 112
конец файла, 121
константы, 21
  именованные, 21
  литеральные, 21

Л

литеральные константы, 21
логическое выражение, 42
локальные переменные, 136

М

массивы, 94
  двумерный, 94, 103
  инициализация, 106
  объявление в секции TYPE, 100
  одномерный, 94
машинный файл, 166
метка, объявление, 64
множества, 146
  объявление, 147
  ограничения на, 148
  присваивание элементов, 148
множественные операторы
  использование, 149
множество символов, 36

Н

нетекстовые файлы, 182

О

область видимости переменных, 140
объявление переменных, 18
одномерный массив, 94
операторы
  +, сложение, 14, 18, 149
  –, вычитание, 14, 18, 149
  *, умножение, 14, 18, 149
  /, деление вещественное, 14, 16, 18
  <, меньше, 43
  <=, меньше или равно, 43
  =, равно, 43
  <>, не равно, 43
  >, больше, 43
  >=, больше или равно, 43
  :=, присваивание, 20
  AND, 44
  DIV, деление целочисленное, 16, 18
  IN, 151
  INPUT, 111
  MOD, деление по модулю, 16
  NOT, 44
  OR, 44
  READ, 113, 114, 117
  READLN, 111, 117
  TYPE, 91
  WITH, 156
  XOR, 45

операторы
вложенные, 60
исключающее ИЛИ (XOR), 45
логические (булевы), 44
отношений, 42, 150
объединения, 149
пересечения, 149
старшинство, 16, 45, 151
типы
арифметический, 14, 16
бинарный, 17
множественные, 149
отношений, 42
присваивания, 20
унарный, 17
Турбо Паскаля, 45
операции с указателями, 229
присваивание, 230
сравнение, 230

П

параметры, 133
параметры-значения, 135
параметры-переменные, 135
передача по значению, 134
передача по ссылке, 134
передача указателей в качестве, 234
фактические, 133
файловые, 12, 167
формальные, 133
Паскаля соглашения, 13
переменные
глобальные, 136
локальные, 136
область видимости, 140
объявление, 18
полевые, 155
статические, 227
файлового буфера, 189
файловые, 167
пересечения оператор, 149
перечисления, 87
ограничения на применение, 91
перечислимый тип, 88
повторяющиеся циклы, 65
поддиапазон, 89
ограничения на применение, 91
подпрограмма, 130
поле, 153
данных, 235
доступ, 155
метки, 195
связи, 235
указателя, 235
полевые переменные, 155
пользовательский ввод, чтение, 24
порядковые типы данных, 87
определяемые пользователем, 87
последовательного доступа файлы, 165
построение циклов, 71
преобразование типов, 23
примеры
анализатор текста, 151, 170
база данных сотрудника, 245, 254
ввод символов, 115
вывод TEXT-файла на экран, 172
записи сотрудников, 155, 177
количество дней в месяце, 62, 66
кредитная карта на Паскале, 50
перестановка букв, 118
результаты и отметки, 57, 96, 104
связанные списки, 238
система расчета зарплаты, 179, 183, 196, 206, 216
сортировка, 136
сортировка массива, 101
сортировка строк, 123
среднее значение, 75, 79
степени двойки, 74
счетчик сдачи (бакалейная лавка), 34
счетчик символов, 120
счетчик частоты повторений, 121
тестер символов, 56, 65
торговый автомат, 61
факториал, 77, 82, 142
функция возведения в степень, 33
чтение TEXT-файла, 173
принятие решений, 49
приоритет, 16
программа
качество, 95
структура, 11
простые типы данных, 29
процедуры, 130
APPEND, 187
ASSIGN, 167
CLOSE, 168
ERASE, 187
READ, 169
READLN, 169
RESET, 167

процедуры
REWRITE, 175
WRITE, 176
WRITELN, 176
возврат значений из, 134
определение, 131
передача значений в, 132
прямого доступа файлы, 165
пустое множество, 148

Р

рекурсия, 142
решения, принятие, 49

С

связанный список
объявление, 234
поиск в, 244
построение, 235
сохранение в файле, 241
удаление узлов из, 251
чтение, 236
из файла, 242
символьные функции, 37
символьный тип, CHAR, 36
синтаксис, 12
скалярные типы данных, 29
соглашения Паскаля, 13
списки, 22
стандартные идентификаторы, 19, 270
стандартные типы данных, 29
старшинство операторов, 45
статическая переменная, 227
строки, 39, 122
объявление, 40
определение длины при помощи функции LENGTH, 41
чтение, 173
строковые процедуры, 125
строковые функции, 125
структуры данных, 227

Т

текст, вывод, 13
типы
BOOLEAN, 42
CHAR, 36
определенные пользователем, 92
переименование, 91
типы данных
вещественный, 87
порядковый, 87
стандартный, 29
числовой, 30
Турбо Паскаль, 12, 23, 38, 65, 93, 167, 175, 187
возможности, 66
операторы, 45
функции, 35
числовые типы в, 30

У

узлы, 234
удаление из связанного списка, 251
указатели, 227
на записи, 232
объявление, 228
ограничения на использование, 232
операции с, 229
передача в качестве параметра, 234
поле указателя, 235
унарный оператор, 17
управляющие структуры, 49
упреждающее объявление, 141
условия вложенные, 57

Ф

фактический параметр, 133
файловая переменная, 167
файловые параметры, 12, 167
файлы
INPUT, 110
OUTPUT, 110
двоичный, 166
закрытие, 168
машинный, 166
нетекстовые, 182
открытие, 167
последовательного доступа, 165
присоединение к файлу, 187
прямого доступа, 165
сохранение связанного списка в файле, 241
тип TEXT, 166
чтение связанных списков из файлов, 242
формальный параметр, 133
форматирование вывода, 25

функции, 138
EOF, 169
EOLN, 169
арифметические, 31
возведение в степень, 33
преобразования, 31
символьные, 37
смешанные, 31
тригонометрические, 31
Турбо Паскаль, 35

Ц

целые типы в Турбо Паскале, 30
циклы
FOR, 72
REPEAT, 82
WHILE, 79
арифметический, 71
бесконечный, 65
вложенный, 78
повторяющийся, 65
построение, 71

Ч

числовые типы данных, 30
в Турбо Паскале, 30

По договору между издательством «Символ-Плюс» и Интернет-магазином «Books.Ru – Книги России» единственный легальный способ получения данного файла с книгой ISBN 5-93286-057-X, название «Программирование на Pascal, 3-е издание» – покупка в Интернет-магазине «Books.Ru – Книги России». Если Вы получили данный файл каким-либо другим образом, Вы нарушили международное законодательство и законодательство Российской Федерации об охране авторского права. Вам необходимо удалить данный файл, а также сообщить издательству «Символ-Плюс» (piracy@symbol.ru), где именно Вы получили данный файл.