Виталий Потопахин

Turbo Pascal

ОСВОЙ НА ПРИМЕРАХ

Санкт-Петербург
«БХВ-Петербург»
2005

УДК 681.3.068+800.92Turbo Pascal
ББК 32.973.26-018.1
П64

Потопахин В. В.

П64 Turbo Pascal. Освой на примерах. — СПб.: БХВ-Петербург, 2005. — 240 с.: ил.

ISBN 5-94157-656-0

В первой части книги приведено неформальное описание языка Pascal, сопровождающееся большим количеством полностью законченных примеров, работающих в среде Turbo Pascal 7.0 компании Borland. Во второй части рассмотрено решение различных типовых задач программирования, нацеленных на формирование у обучаемого особой программистской логики и дающих возможность изучить и отработать на практике все существенные особенности языка Pascal. Подробно и последовательно освещены вопросы работы со статической и динамической графикой, организации диалогов, обработки массивов и строк, работа с файлами, указателями, списками и др.

Для начинающих программистов

УДК 681.3.068+80.92Turbo Pascal
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Екатерина Капалыгина*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Наталия Першакова*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 31.05.05.
Формат 60×90 1/16. Печать офсетная. Усл. печ. л. 15.
Тираж 5000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

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

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

ISBN 5-94157-656-0

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

Отпечатано с готовых диапозитивов
в ОАО "Техническая книга"
190005, Санкт-Петербург, Измайловский пр., 29.

Отпечатано с диапозитивов в ГПП "Печатный двор"
Министерства Российской Федерации по делам печати,
телерадиовещания и средств массовых коммуникаций.
197110, Санкт-Петербург, Чкаловский пр.,15.

Оглавление

Введение

Книгу, которую вы собираетесь изучать, можно рассматривать как самоучитель программирования. Она полностью построена на примерах прикладного характера и почти все программы приведены в завершенном виде. Вы можете набрать пример из книги в редакторе компилятора Turbo Pascal компании Borland, и программа будет работать именно так, как обещано. Теоретический материал также есть в книге, но он привязан к прикладным потребностям. Из всего сказанного можно сделать вывод: если вы хотите за разумный срок приобрести уверенные навыки решения задач с использованием языка программирования Паскаль, то эта книга принесет вам заметную пользу, хотя материал изложен нестрого и неформально.

Но, тем не менее, эта книга не слишком проста, как может показаться на первый взгляд. В ней есть теоретические разделы и отдельные задачи, над которыми придется подумать и довольно искушенному человеку. Смею утверждать, что учащийся, сумевший проработать весь материал, станет хорошо эрудированным в деле программирования. Для первых же занятий никаких программистских знаний не требуется, начинать работать можно с чистого листа.

В первой главе книги дано описание языка Паскаль. Здесь вы не найдете большого количества технических подробностей, и в этом плане такое описание, конечно, уступает техническим руководствам, но технических руководств очень много, и создавать еще одно нет необходимости. Но все, что действительно нужно для записи логики алгоритма, вы найдете именно здесь. Есть даже описания таких редко используемых структур, как вариантные записи и множества. Совсем без технических языковых деталей мы все же не обошлись, но и эти детали выбирались из необходимости описания логики алгоритмов.

Следующие восемь глав книги посвящены восьми типам прикладных задач. В каждой главе примерно по 30 задач, часть из

которых решена, а часть нет. Решенные задачи вы используете в качестве примера, а нерешенные предлагаются вам для практики. Кроме того, решенные задачи снабжаются пояснениями, комментирующими то, что сделано. Пояснения помогут лучше понять текст программы. Нерешенные задачи снабжены подсказками, их цель — помощь в решении. Эти главы и есть собственно самоучитель, теоретически можно работать только с ними, но если на протяжении своего самообучения вы будете пользоваться первой главой, то процесс пойдет и быстрее, и эффективнее.

Для учителей. Данная книга есть ядро апробированного в течении 10 лет авторского курса программирования. Поэтому ее второе предназначение — организация кружковой работы со школьниками в сфере программирования.

Глава 1

Язык Паскаль

Изучать программирование по описанию языка нельзя, даже если это очень хорошее описание, потому что программирование — это искусство решения задач, а никак не искусство владения языком, но язык, без сомнения, нужен, т. к. он является базой. Поэтому лучшая стратегия использования данной главы заключается в следующем.

1. Внимательно просмотрите всю главу, а ее первые разделы, посвященные самым основным конструкциям, проработайте более внимательно.
2. После этого переходите ко второй главе — решению задач. В процессе борьбы с задачами периодически, по мере необходимости, возвращайтесь к первой главе и уже берите то, что нужно для решения конкретной проблемы.

Основные конструкции и операторы

Любой человек владеет хотя бы одним языком (родным), и поэтому любой человек знает, что такое язык, но объяснить, что это такое, сможет уже не каждый. Естественный человеческий язык — очень сложная вещь, у него необыкновенно много функций и возможностей. Причина тому — сложность человеческого интеллекта.

Любой из нас согласится, что компьютер при всей его сложности все же неизмеримо проще человеческого мышления, и, ви-

димо, машинный язык также должен быть неизмеримо проще. Это действительно так:

- ❒ язык программирования описывает только действия;
- ❒ не любые действия, а только простейшие;
- ❒ язык программирования опирается на очень простые понятия. Пожалуй, мы не слишком сильно ошибемся, если скажем, что этих понятий только два: число и символ.

Языков программирования очень много, но при всем их многообразии, в каждом присутствует обязательный набор языковых конструкций. Например, любой язык должен обеспечивать возможность выбора из нескольких действий одного, удовлетворяющего определенным условиям.

Пример. **Если** рабочий день закончился, **то** можно идти домой, **иначе** необходимо продолжать работать.

Любой язык должен предоставить возможность повторять команды без многократного их написания.

Пример.

Для каждого числа от 1 до 100 делаем:

- ❒ находим сумму его делителей;
- ❒ печатаем найденную сумму.

Можно продолжать примеры команд, которые должны быть в любом языке. Например, необходимо уметь вычислять арифметические выражения, выводить значения на экран и т. д. Можно сказать, что в языке программирования должны присутствовать любые конструкции и команды, которые могут встретиться при записи алгоритмов. Следовательно, можно предположить, что команды языка программирования — это форма записи команд алгоритма, поэтому далее (по крайнем мере первое время) мы будем изучать язык программирования по следующей схеме.

1. Записываем алгоритм на русском языке.
2. Выделяем в полученной записи команды.

3. Записываем выделенные команды командами языка программирования.

Пример 1. Требуется ввести два числа, найти их среднее арифметическое и вывести полученное значение на экран монитора.

Требуемый алгоритм выглядит следующим образом.

1. Ввести число A.
2. Ввести число B.
3. Вычислить C = (A + B) / 2.
4. Вывести значение C.

Ясно, что для записи программы по этому алгоритму нужны команды ввода значений переменных величин, вывода полученных значений и вычисления значений. В языке Паскаль команда ввода записывается словом **`read`**, команда вывода словом **`write`**. Команда вычисления записывается с помощью знака `:=` (но называется эта команда не вычислением, а присваиванием). Теперь мы наш фрагмент программы можем записать так:

```
read(A);
read(B);
C:=(A+B)/2;
write(C);
```

Примечание

Буквы `A`, `B`, `C` обозначают переменные величины, т. е. величины, чье значение можно изменять. Записывать имена переменных большими буквами совершенно необязательно. Каждая команда заканчивается точкой с запятой. Команды в программе на языке Паскаль не нумеруются и их можно располагать совершенно произвольным образом, компьютер же будет их читать слева направо и сверху вниз.

В дальнейшем попробуем придерживаться общепринятой терминологии в обозначении команд. А они обозначаются тремя разными словами: оператор, процедура, функция. Пока не будем давать точного определения этих слов, просто привыкнем к ним. Позже это различие прояснится.

Оператор присваивания

Оператор присваивания в языке Паскаль имеет следующую форму:

```
Переменная : = Арифметическое или логическое выражение
```

Под арифметическим выражением понимается любое допустимое в языке выражение, содержащее числа и арифметические операции *(см. далее)*. Есть, однако, несколько особенностей, игнорирование которых может привести к серьезным проблемам.

- Тип выражения, вычисляемого справа от знака оператора, должен совпадать с типом переменной, указанной слева. Целое число должно равняться целому. Действительное число может равняться любому.
- Необходимо следить за числовым интервалом, в который попадает вычисляемое значение. Это следует из того, что каждый тип имеет вполне определенный допустимый интервал значений. Если ваше выражение даст значение за пределами этого интервала, то в ходе работы программы возникнет ошибка, которую компилятор не сможет обнаружить. Например, целые числа определены в интервале от −32 768 до 32 767. Более подробно обо всех эти проблемах можно почитать в разделе, посвященном типам данных.

Оператор цикла

Пример 2. Найти сумму *N* последовательных чисел. То есть найти сумму следующего ряда: 1 + 2 + 3 + ... + *N*.

Конечно, неинтересно создавать алгоритм, который всегда считал бы сумму одних и тех же чисел. Значительно интереснее было бы получить алгоритм, для которого мы сможем вводить произвольное *N*. Но тогда необходимо уметь выполнять операцию сложения много раз. Оператор, предоставляющий такую возможность, называется оператором цикла.

Суть оператора заключается в том, что какая-то переменная изменяется от начального значения до конечного значения с шагом единица, и на каждом шаге выполняются какие-то операции, называемые телом цикла.

Наш алгоритм тогда будет выглядеть следующим образом:

```
Ввести N
S=0
Для всех i от 1 до N делать
  Начало цикла
   S=S+i
  Конец цикла
Вывести S
```

Фрагмент Паскаль-программы будет выглядеть так:

```
read(N);
s:=0;
for i:=1 to n do
   s:=s+i;
write(s);
```

Примечание

Никогда, пожалуйста, не забывайте, что параметр цикла изменяется с *шагом 1*. После служебного слова do можно записать только один оператор, и только он будет исполняться в цикле. Как сделать так, чтобы циклически исполнялась группа операторов, мы рассмотрим позже, когда будем говорить о сложном операторе.

Параметр цикла обязательно является *целым* числом, начальное и конечное значения параметра цикла также целые числа. Особенность заключается в том, что целое число в языке программирования совсем не то же самое, что целое число в математике. Математическое целое может быть бесконечно большим, но целое машинное число не может быть слишком большим, т. к. для представления каждого числа нужна область памяти, а память компьютера не безгранична. В результате этого на величину целого числа накладываются довольно жесткие ограничения, о которых мы поговорим немного позже.

Не забывайте также, что в данной форме цикла параметр изменяется *от меньшего к большему*.

Постарайтесь свои программы проектировать таким образом, чтобы переменные, использующиеся в описании заголовка цикла, не изменялись в теле цикла. Иное не будет ошибкой, но может привести к неоправданному усложнению логики программы.

Кроме того, в изменении параметров нет никакой необходимости, т. к. существуют еще две формы организации цикла, при которых с любыми переменными можно делать все, что вам заблагорассудится (они рассматриваются далее). Приведем простейший пример плохой организации цикла.

```
for i:=1 to 10 do
   i:=i*i;
```

Возможно, цель этого фрагмента программы вычислить квадраты первых десяти чисел, но реально этого не получится, т. к. параметр цикла будет изменяться не с шагом единица, а значительно быстрее.

Цикл по параметру **for to do** не единственный в языке Паскаль. Во-первых, возможна запись цикла с параметром, в которой параметр изменяется от большего к меньшему, уменьшаясь на каждом шаге на единицу. Наша программа с таким оператором цикла будет выглядеть так:

```
read(N);
s:=0;
for i:=n downto 1 do
   s:=s+i;
write(s);
```

Программа суммирует те же самые числа, что и предыдущая, но в обратном порядке. Кроме того, Паскаль имеет еще два типа цикла по условию. Это циклические конструкции, в которых группа операторов или один оператор выполняются либо до тех пор, пока истинно некоторое условие, либо до тех пор, пока некоторое условие не станет истинным. Приведем примеры действия таких циклов на той же задаче сложения чисел.

```
read(N);
s:=0;i:=1;
while i<=n do
  begin
   s:=s+i;i:=i+1;
  end;
write(s);
```

```
read(N);
s:=0;i:=1;
repeat
   s:=s+i;i:=i+1;
until i>n;
write(s);
```

Цикл *while do*

После слова **while** записывается условие, а после слова **do** записывается выполняемый оператор, который, конечно, может быть сложным. Что такое сложный оператор, видно на примере цикла в форме **while**. А именно, сложным оператором мы будем называть группу операторов, записанных между ключевыми словами **begin end**.

Данный цикл исполняет оператор, записанный после слова **do**, до тех пор, пока истинно условие. Здесь сначала проверяется условие, а затем выполняется оператор, поэтому такая конструкция называется *циклом с предусловием*.

Цикл *repeat ... until*

После слова **until** записывается условие. Операторы, записанные между словом **repeat** и словом **until**, исполняются до тех пор, пока условие не станет истинным. В данном типе цикла сначала выполняются действия, а затем проверяется условие, поэтому такая конструкция называется *циклом с постусловием*.

Отличия циклов по условию от цикла с параметром

В циклах по условию нет понятия параметра, и поэтому все переменные, используемые внутри цикла, могут меняться произвольным образом.

Параметр "Цикла по параметру" изменяется только на 1. Прибавить (или отнять) 1 к числу проще, чем произвольное число, поэтому операция +1 обрабатывается компилятором быстрее, чем произвольное сложение, и, следовательно, цикл по параметру работает быстрее, чем цикл по условию.

Условный оператор (оператор выбора между несколькими действиями)

Пример 3. Ввести множество чисел и определить, сколько среди них положительных.

Идея решения: заведем переменную, которая будет играть роль счетчика положительных чисел, и каждый раз, когда вводимое число окажется положительным, будем значение этого счетчика увеличивать на единицу. А для того, чтобы иметь возможность так делать, нам нужна конструкция, которая позволяла бы выполнять определенную команду только в том случае, если некоторое условие окажется истинным.

Алгоритм будет выглядеть так:

```
Введем N
s=0
Для всех i от 1 до N делать
  Начало
   Ввести Число
   Если Число > 0 То s=s+1
  Конец
Вывести s
```

При записи программы по данному алгоритму возникнет серьезная проблема. Нам необходимо циклически выполнять две команды: ввод числа и проверка на положительность. Но мы уже знаем, что после слова **do** можно записывать только одну команду. Выход заключается во введении сложного оператора. Об этой языковой конструкции уже говорилось, но повторимся.

Сложный оператор — это группа операторов, записанных между словами **begin** (начало) и **end** (конец). Такой составной оператор воспринимается компьютером как один оператор, и его можно поместить, например, в цикл. С учетом этого наша программа будет записана следующим образом:

```
read(n);
s:=0;
for i:=1 to n do
  begin
   read(a);
   if a>0 then s:=s+1;
  end;
write(s);
```

Английское слово "if" переводится как "если", а оператор:

```
if a>0 then s:=s+1;
```

называется *условным оператором*. Его устройство таково: после слова **if** записывается условие, а после слова **then** записывается один оператор (можно сложный), который исполняется, если условие оказывается истинным. Условный оператор имеет еще одну форму:

if *условие* then *оператор* else *оператор*;

В этой форме, если условие истинно, то исполняется оператор, записанный после слова **then**, а если условие ложно, выполняется оператор, записанный после слова **else**.

Примечание

Те, кто только начал практиковаться в программировании на языке Паскаль, часто допускают грубую ошибку в условном операторе. Рассмотрим пример: `if t=5 then t:=6; else t:=8;`. В записанном операторе после `t:=6` стоит точка с запятой. Ставят ее из тех соображений, что любой оператор в языке Паскаль завершается точкой с запятой. Но завершается так логически законченная конструкция. В данном случае на `t:=6` конструкция условного оператора еще не завершена, и точка с запятой здесь не к месту. Ставить ее нужно только после оператора, следующего за словом **else**. Следовательно, правильная конструкция такова: `if t=5 then t:=6 else t:=8;`.

Структура программы

К сожалению, написанные нами фрагменты программ нельзя просто так взять и исполнить. Чтобы программа была исполняемой, к ней необходимо сделать некоторые дополнения. Далее мы опишем структуру программы на языке Паскаль, а в качестве примера будем использовать уже известный нам фрагмент с подсчетом количества положительных чисел.

У программы должно быть имя. *Имя программы* — это набор латинских букв и цифр. Имя может быть почти произвольным, но оно не должно совпадать с ключевыми словами, т. е. такие слова, как "read", "write", "for", не могут быть именем программы. Имя программы также не может начинаться с цифры, хотя циф-

ры могут в нем присутствовать. Имя программы не может совпадать с именами используемых в программе переменных. Оно пишется после ключевого слова **program**. После имени программы обязательно ставится точка с запятой, например:

```
Program proba;
```

Далеко не все команды, которые мы можем записать в программе, непосредственно обрабатываются компилятором. Беда в том, что возможных команд слишком много, чтобы обязывать компилятор знать их все. Поэтому в современных языках программирования используется такое понятие, как *библиотека готовых команд*. И, если мы используем что-то неизвестное компилятору, мы должны указать имя библиотеки, где записан код этой команды. Приведем пример, как это сделать.

Предположим, что у нас есть желание использовать процедуру `clrscr`, она очищает экран. Если мы просто вставим эту команду в текст программы, то компилятор сообщит нам, что такой идентификатор ему не известен. Поэтому необходимо обратиться к справочнику языка, найти в нем описание `clrscr`, а в описании найти слово **unit**. И то, что будет записано после этого слова, есть имя библиотеки. Конкретно в нашем случае имя библиотеки будет **crt**. А указать это имя в программе можно так:

```
Uses crt;
```

Теперь наша программа будет выглядеть следующим образом:

```
Program proba;
  Uses crt;
```

Далее в программе следует описание типов переменных:

```
Var
  список переменных : описание типа;
  список переменных : описание типа;
  ...
  список переменных : описание типа;
```

В этой книге есть целая глава, посвященная описанию различных типов данных. Сейчас нам достаточно одного типа. Договоримся, что вводимые нами числа будут обязательно целыми. Тип

целых чисел в языке Паскаль называется **integer**. Теперь наша программа будет выглядеть так:

```
Program proba;
  Uses crt;
  Var
    n,s,i,a:integer;
```

Далее записывается текст программы, заключенный между словами **begin** (начало) и **end** (конец).

Обратите внимание: сложный оператор также оформляется с помощью этих ключевых слов, из чего следует, что программу можно рассматривать как сложный оператор, но программа имеет то небольшое отличие, что после слова **end** в сложном операторе ставится *точка с запятой*, а после слова **end**, заканчивающего программу, ставится *точка*.

Окончательный вариант программы будет выглядеть так:

```
Program proba;
  Uses crt;
  Var
    n,s,i,a:integer;
begin
  read(n);
  s:=0;
  for i:=1 to n do
    begin
      read(a);
      if a>0 then s:=s+1;
    end;
  write(s);
end.
```

Примечание

Блок описания переменных необязательно содержит описания только одного типа. После слова **var** можно создать сколько угодно описаний.

В нашем примере структура программы описана не полностью. Здесь отсутствует блок описания типов, определенных пользователем, блок описания процедур и функций, но этому дальше будут посвящены отдельные главы.

Конструкция *case*

Рассмотрим следующую задачу: пусть две величины L и U могут принимать только три значения, причем эти величины взаимосвязаны, т. е. величина U принимает свои значения в зависимости от того, какое значение принимает величина L. Предположим, что взаимосвязь устанавливается следующим образом: при $L = 1$ $U = 4$; при $L = 4$ $U = -5$; при $L = 0$ $U = 11$. Предположим далее, что величина L вводится с клавиатуры, а величина U вычисляется в зависимости от введенного L. Запишем каким образом это можно реализовать на языке Паскаль:

```
Program example;
  Var
    U,L:integer;
begin
  read(L);
  case L of
    1: U:=4;
    4: U:=-5;
    0: U:=11;
  end; { Этот end завершает выполнение case }
end.
```

Рассмотрим еще одну интересную ситуацию. Предположим, что взаимосвязь L с U немного усложнилась. Старые значения связаны так же, как и раньше, но теперь L может принимать любые значения, и, если L принимает значение, отличное от уже описанных, то $U = 20$. Опишем эту ситуацию с помощью конструкции **case** (или оператора выбора).

```
Program example;
  Var
    U,L:integer;
```

```
begin
  read(L);
  case L of
    1: U:=4;
    4: U:=-5;
    0: U:=11;
    else U:=20;
  end; { Этот end завершает выполнение case }
end.
```

При исполнении этой конструкции, если *L* примет значение, отличное от 1, 4, 0, то *U* примет значение 20. Возможна и такая структура оператора, при которой *U* будет принимать одно и то же значение при различных *L*. Приведем пример:

```
Program example;
  Var
    U,L:integer;
begin
  read(L);
  case L of
    1,5,8: U:=4;
    4,7: U:=-5;
    0,3: U:=11;
    else U:=20;
  end; { Этот end завершает выполнение CASE }
end.
```

Заключение

Язык Паскаль устроен таким образом, что для понимания его основных конструкций нужно только лишь немного знать английский язык. Смысл основных языковых конструкций совпадает со смыслом перевода английских слов, обозначающих эти конструкции.

Убедитесь сами:

if	then	else	for	to	do
если	то	иначе	для	до	делать

Очень важной конструкцией языка является сложный оператор, помогающий эффективно разбивать программу на логические блоки. Программа, с одной стороны, состоит из сложных операторов, а с другой стороны сама является сложным оператором.

И последнее, обратите внимание на то, каким образом записываются операторы. Каждый оператор, если он связан с вышестоящим, смещается немного вправо. Например, вот так:

```
for i:=1 to 10 do
  s:=s+1;
```

Это необязательно. Всю программу можно записать в одну строку. Но такая форма записи помогает лучше понять структуру программы. А именно: какие операторы связаны в группы, именуемые сложными операторами, а какие нет. При написании больших программ это становится очень существенным.

Операции языка Паскаль

Сейчас наша цель понять, как в языке Паскаль вычислить арифметическое и логическое выражение, какие существуют арифметические и логические операции, каков порядок их выполнения и какие существуют библиотечные математические функции.

Арифметические операции

Для начала можно, конечно, дать определение арифметического выражения, но будем надеяться, что все понимают, что это такое. В табл. 1.1 приводится список арифметических операций.

Таблица 1.1. *Арифметические операции*

Знак	Операция
-	Вычитание
+	Сложение
*	Умножение
/	Деление
Div	Целочисленное деление
Mod	Остаток от деления

Высший приоритет в арифметическом выражении имеют следующие операции: `div`, `mod`, `*`, `/`. Низший приоритет имеют операции `+`, `-`. Если две операции одинакового приоритета стоят рядом, то первой выполняется та, которая стоит первой от левого конца выражения. И, конечно, выражение в скобках, так же, как и в математике, имеет более высокий приоритет, чем выражение за скобкой. Приведем несколько примеров правильных арифметических выражений:

```
x*col-7*(g-u+5)
s/t/y-y+8*(u/7-5-g)*(u-8.78)
5.89+6*(y+7*u*(t+6))
```

`5 mod g` (В этом выражении ищется остаток от деления 5 на *g*.)

`g div 2` (В этом выражении вычисляется результат от деления *g* на 2.)

Примечание

Дробная часть числа отделяется от целой не запятой, как это принято в математике, а точкой.

Некоторые полезные арифметические функции:

- **`sin`** — вычисление синуса. Аргумент задается в радианах;
- **`cos`** — вычисление косинуса. Аргумент задается в радианах;
- **`exp`** — вычисление экспоненты;
- **`sqr`** — вычисление квадрата выражения;

- `sqrt` — вычисление квадратного корня выражения;
- `abs` — вычисление модуля выражения;
- `arctan` — вычисление арктангенса выражения;
- `frac` — вычисление дробной части выражения;
- `int` — вычисление целой части выражения;
- `round` — преобразование к целому типу;
- `random` — вычисление случайного числа в указанном интервале.

Мы здесь не будем разбирать, как записывать арифметическое выражение с использованием арифметических функций, т. к. это можно посмотреть в документации, прилагаемой к любому компилятору.

Логические операции

В табл. 1.2 представлен список логических операций.

Таблица 1.2. *Логические операции*

Операция	Пояснение
`and`	Логическое умножение
`or`	Логическое сложение
`not`	Логическое отрицание
`xor`	Логическое деление

Приведем определения логических операций.

- *Отрицание.* Если логическая величина *C* является отрицанием логического выражения *A*, то *C* истинно, если *A* ложно, и ложно, если *A* истинно.
- *Логическое умножение.* Если *A* и *B* истинны, то *C* также истинно. Если же хотя бы одно из них ложно, то *C* также ложно.
- *Логическое сложение.* Если *A* и *B* ложны, то *C* также ложно. Если же хотя бы одно из логических выражений *A* и *B* истинно, то *C* также истинно.

- *Логическое деление* (иногда эту операцию еще называют *исключающим или*). Это логическая операция, устанавливающая соответствие между логическими выражениями *A* и *B* и логической величиной *C* следующим образом: *C* ложно, если *A* и *B* либо одновременно истинны, либо одновременно ложны.

Упомянутые ранее определения логических операций можно также описать в виде *таблиц истинности* (табл. 1.3).

Таблица 1.3. *Таблица истинности для всех логических операций*

A	B	not A	A and B	A or B	A xor B
true	true	false	true	true	false
true	false	false	false	true	true
false	true	true	false	true	true
false	false	true	false	false	false

А сейчас несколько примеров логических выражений:

```
(a and b)or(b and not c)
(a xor b) and ( a or c) xor (not h)
```

Логическое выражение — это такое выражение, которое может принимать только два значения: истина (`true`) и ложь (`false`). В языке Паскаль логическими выражениями могут быть:

- специальные логические переменные (тип **`boolean`** — *см. разд. "Типы данных"*);
- выражения, содержащие сравнения (например, `s<>h`);
- сложные выражения, содержащие выражения двух описанных ранее типов, соединяемых знаками логических операций и скобками.

А сейчас еще несколько примеров сложных логических выражений:

```
(a<>b)and(f  or  (h<5))
f  xor  (5=6*i)
(s<>6) or (g>8)
```

Заключение

Логические выражения представляют собой мощный математический аппарат, позволяющий проверять несколько элементарных условий в одном операторе. В их отсутствие для использования условного оператора пришлось бы осуществлять проверку сложного условия как проверку нескольких элементарных условий, для каждого из которых потребовался бы собственный оператор условия. Приведем пример:

```
if (y<7) and (h>5) then h:=y;
```

Этот же оператор, без применения логических операций, можно записать так:

```
if (y<7) then if (h>5) then h:=y;
```

Видно, что вторая запись длиннее и сложнее. Точно так же более просто организуются (с применением логических операций) и циклы по условию.

Еще одна очень серьезная выгода связана с наличием в языке Паскаль специального типа данных (логического типа). Этот тип данных позволяет создавать и вычислять логические выражения в операторе присваивания аналогично тому, как это делается с арифметическими выражениями. Например, выражение

```
if t=6 then y:=true else y:=false;
```

вполне можно заменить следующей более простой конструкцией:

```
y:=t=6;
```

Типы данных

К настоящему моменту мы знаем, что все переменные, используемые в программе, должны быть описаны. Рассмотрим подробнее, какие типы встречаются в языке Паскаль, и как их описывать.

Что такое описание типов переменных?

Описание типа — это информация для компилятора о том, что из себя представляет переменная, сколько ячеек памяти для нее отводить и какие операции с ней можно выполнять.

Для чего нужно описывать переменные?

- Назначение переменной часто бывает связано с ее типом. Например, параметр цикла — это обязательно целая величина. Если же вдруг мы внутри цикла по параметру присвоим параметру дробное значение, то это будет ошибкой, и вот такие ошибки компилятор может обнаружить, имея описания типов переменных. Поиск подобных ошибок является важнейшей функцией компилятора.
- Память даже самого мощного компьютера ограничена. Кроме того, статическая память нашего компилятора языка Паскаль — это всего лишь один сегмент памяти в 64 Кбайт. Все остальные мегабайты вашего компьютера для вас недоступны (пока вы не изучили язык получше). Следовательно, важно точно определить, сколько памяти нужно под ту или иную переменную, чтобы не столкнуться с нехваткой памяти в процессе работы программы.

Какие типы переменных существуют в языке Паскаль?

- **`integer`** — целый тип. Данный тип представляет собой целые числа в интервале от −32 768 до 32 767. Отсюда следует, что вам необходимо очень внимательно следить за границами изменения всех целых величин. Особенно следует помнить о параметрах цикла **`for`**, т. к. все они целого типа. И, если вы опишите цикл, параметр которого будет изменяться, например, до 32 788, то ваша программа не будет работать хорошо. Для хранения значения используется 2 байта.
- **`longint`** — это тоже целый тип, но в отличие от типа **`integer`** позволяет задавать числа в значительно большем интервале. Интервал изменения для этого типа от −2 147 483 648 до 2 147 483 647. Данный тип позволяет использовать большие целые числа, но он требует памяти в два раза больше, чем тип **`integer`**, т. е. 4 байта.
- **`real`** — представляет вещественные числа. Используя этот тип данных, можно записать число длиной 11—12 знаков. Для хранения значения отводится 6 байт.
- **`byte`** — является разновидностью целого типа. Интервал изменения от 0 до 255. Для хранения значения используется 1 байт.

- **word** — также является разновидностью целого. Интервал изменения от 0 до 65 535. Для хранения значения используется 2 байта.
- **char** — символьный тип. Величина этого типа есть ни что иное, как просто любой символ. Для хранения значения используется 1 байт.
- **string** — строковый тип данных. Переменная данного типа представляет собой строку символов. Примеры описания:

```
f:string;
g:string[20];
```

  В первом примере объявлена строка символов максимально возможной длины, т. е. 255 символов. Во втором примере объявлена строка символов с максимальной длиной 20 символов. Для хранения каждого символа строки требуется 1 байт.
- **boolean** — позволяет представлять логические переменные, которые могут иметь только два значения: истина (единица) или ложь (ноль). Такой узкий интервал представления данных означает, что для данного типа требуется очень мало памяти. Тип **boolean** — это очень экономный тип данных. Присвоить значение логической переменной можно так: `h:=false;` (`h` присваивается значение ложь) или `h:=true;` (`h` присваивается значение истина). Логическим переменным можно также присваивать значение логических выражений, например: `h:=g<10;`. Над логическими переменными можно выполнять логические операции. (Что такое логические операции и как ими пользоваться, смотрите в описании условного оператора **if**). Для хранения значения используется 1 байт.

На этом мы заканчиваем описание элементарных структур данных, но в языке Паскаль есть еще сложные структуры, такие как: массивы, записи, файлы данных. Им будут посвящены отдельные главы, а сейчас мы рассмотрим эти структуры кратко.

Массивы

Массив — это упорядоченное множество данных одинаковой природы.

Например:

```
F: array[1..10] of integer;
```
(Десять целых чисел.)

```
H: array[1..100] of string;
```
(Сто строковых величин.)

В качестве примера приведем программу, в которой вычисляются квадраты первых ста целых чисел.

```
Program example;
  Var
    i:integer;
    d: array[1..100] of integer;
begin
  for i:=1 to 100 do
   begin
     d[i]:=i*i;
     write(d[i]);
   end;
end.
```

Массивы могут быть многомерными, например, двумерными:

```
f: array[1..10,1..100] of integer;
```

Здесь объявлен массив в 10 раз по 100 целых чисел.

Записи

Запись — это сложное данное, содержащее в себе простые данные различной природы. Например:

```
d: record
    i:integer;
    t:string;
    y:array[1..10] of integer;
   end;
```

Переменная типа "запись" используется так же, как и обычная переменная, разница только в способе доступа к ее значению. Для доступа к значению компонента записи необходимо указать как имя записи, так и имя компонента:

```
d.i:=67;
```

Имя записи отделяется от имени компонента точкой.

Файловые переменные

Файловые переменные используются для организации работы с файлами. Такая переменная связывается с именем существующего файла, и все операции производятся с файловыми переменными, как с файлами. Далее приведен пример программы, работающей с файлом целых чисел.

Задача. Ввести *N* чисел, записать их в файл, затем прочитать из файла и вывести на экран.

```
Program example;
  Uses crt;
  Var
    i,a,n:integer;
    f:file of integer; {Обьявляется файловая переменная,
                        далее с ней будет связан файл целых
                        чисел}
begin
  read(n);
  assign(f,'file'); {С файловой переменной связывается
                     файл целых чисел по имени file}
  rewrite(f); {Создается файл, связанный с файловой
               переменной f,
               если такой файл уже есть,
               то он уничтожается и создается заново}
  for i:=1 to n do
    begin
      read(a); {Запрашивается с клавиатуры очередное число}
      write(f,a); {Введенное выше число записывается в файл}
    end;
  close(f); {Файл закрывается}
  reset(f); {Файл открывается. Данной процедурой можно
             открыть
             только уже существующий файл.
             После выполнения данной процедуры
             над файлом можно выполнять любые операции}
```

```
  for i:=1 to n do
    begin
      read(f,a); {Очередное число читается из файла}
      writeln(a); {Прочитанное выше число выводится на экран
                   монитора}
    end;
end.
```

Массивы, записи и файлы — это сложные структуры данных, им дальше посвящены специальные главы. А сейчас рассмотрим еще одну интересную возможность. Любой язык профессионального программирования разрешает программисту создавать собственные типы данных. Есть такая возможность и в языке Паскаль.

Типы данных, определенные пользователем

Что такое типы, определенные пользователем?

Предположим, что вы разрабатываете программу, в которой много данных одинакового типа, но этот тип не является базовым (т. е. он не **integer**, не **real** и т. д.). Например, это данные, имеющие структуру массива определенной длины. Тогда вы можете сэкономить на описании переменных, сделав описание следующим образом:

```
Type
  shablon=array[1..100] of real;
Var
  t,y,u: shablon;
```

Свойства, определенные для `shablon`, переходят на переменные, указанные в описании **var**, а `shablon` называется новым типом данных. Это означает, что с помощью `shablon` мы можем определять тип переменных, но `shablon` не является переменной и его нельзя использовать как переменную. Конечно, в таком простом примере не обязательно вводить новую структуру.

Мы могли бы решить эту же проблему следующим описанием:

```
Var
  t,y,u: array[1..100] of real;
```

Но достаточно часто встречаются ситуации, в которых собственные типы существенно экономят место в описаниях. Именно такая ситуация представлена в следующем примере. Здесь описываются переменная и массив типа "запись" (`record`) с одинаковой структурой.

```
Program example;
  Var
    a: record
        i:integer;
        f:string;
        g:real;
       end;
    b:array[1..100] of record
    i:integer;
    f:string;
    g:real;
end;
```

Это же самое можно сделать несколько короче:

```
Program example;
  Type
    d=record
       i:integer;
       f:string;
       g:real;
      end;
  Var
    a: d;
    b: array[1..100] of d;
```

Здесь переменная `a` объявлена типом `d`, это означает, что все описания, сделанные для `d`, автоматически переходят на `a`. Так же происходит и с объявлением массива `b`. Чем длиннее описа-

ние переменной, тем выгоднее применить собственные типы данных.

Совершенно очевидной полезность собственных типов данных становится при написании больших программ, состоящих из большого количества программных блоков, в каждом из которых есть описания переменных и сложных структур переменных. Использование собственных типов позволяет не описывать одну и ту же структуру в каждом новом блоке. Теперь мы можем сделать это описание только один раз, создав соответствующий собственный тип.

Константы

Константа — это нечто постоянное, неизменяемое. И конечно, это совершенно не обязательно числовая величина.

Объявление константы

Объявляется константа в блоке объявлений следующим образом:

```
const
  a=79;
  d=4.5;
  s='hh';
```

Константа обязательно имеет значение. Нельзя включать в объявление константы переменные величины, т. к. их значение в блоке объявлений еще не определено. Объявление константы вполне может содержать выражения, в которых также будут присутствовать уже объявленные ранее константы.

```
const
  a='gfgfgf';
  s=a+'dsd';
  w=5.6;
  q=w + 7;
```

Использование константы

Очевидно полезна константа в том случае, когда некая величина используется во многих частях программы, причем эта величина

может оказаться длинной в своей записи, что серьезно затруднит ее написание в программе. Конечно, можно использовать понятие переменной (т. е. обозначить величину буквой), однако в этом случае мы должны сами позаботиться о ее неизменяемости. А ведь часто программистские ошибки возникают именно от того, что разные по смыслу величины оказываются обозначенными одним именем. Понятие константы в этом отношении дает дополнительную защиту. Любая попытка изменить значение константы будет поймана уже на этапе компиляции.

Пример использования:

```
Program example;
  Uses crt;
  Const
    g=9.8;
  Var
    max,s,t:real;
begin
  read(max);
  t:=0;
  repeat
    s:=g*t*t/2;
    writeln('s=',s,' t=',t);
    t:=t+0.1;
  until t>=max;
end.
```

Заключение

Одна из сложнейших проблем, возникающих в общении программиста и компьютера, — это разные типы данных, которыми пользуются люди и компьютеры. Компьютер в сущности понимает только один тип данных — последовательность нулей и единиц, занимающую определенное количество ячеек памяти (бит, байт, несколько байт, машинное слово). Человек пользуется очень сложными структурами данных: число, слово, предложение и т. д. Поэтому одной из важнейших задач языка про-

граммирования является поиск таких структур данных, которые было бы несложно преобразовать в машинное представление. В то же время эти структуры данных не должны слишком сильно отличаться от данных, используемых человеком. Нетрудно убедиться, что типы данных языка Паскаль неплохо решают эту задачу.

Массивы — это упорядоченные множества данных, т. е. такие, в которых данные пронумерованы. Числа типа **integer** и **real** — это целые и действительные числа, от которых машинные типы отличаются только допустимым интервалом представления.

Вторая важнейшая функция описания данных — поиск ошибок на этапе компиляции. В правильно работающей программе значение может быть передано только в переменную соответствующего типа. Например, число 9.89 может быть передано только в переменную типа **real**. Если же где-то в программе это правило нарушается, то можно с уверенностью говорить об ошибке, и такие ошибки можно обнаружить на этапе компиляции, используя описания типов переменных. Из этого следует очень важный принцип языка Паскаль: *любая переменная и структура данных должна быть описана до своего первого использования.*

Строковые величины

Строковый тип данных — это тип данных, похожий на массив символьного типа. Если имеется данное, определенное как:

```
f: array[1..100] of char;
```

то это же данное возможно определить так:

```
f: string[100];
```

Эти два описания имеют один и тот же смысл. Строку всегда можно представить как массив символов, однако не всегда массив символов можно представить как строку. Дело в том, что строка в языке Паскаль может иметь только весьма ограниченную длину, не более чем 255 символов. Поэтому если нам нужно работать со множеством символов, количество элементов которого больше чем 255, то придется использовать конструкцию массива.

Если же длина строки в 255 символов вполне достаточна, то лучше воспользоваться строковым типом, т. к. этот тип предоставляет в наше распоряжение ряд очень удобных операций. Перечислим основные:

- **copy** — копирует из строки подстроку;
- **delete** — уничтожает в строке подстроку;
- **insert** — вставляет строку в строку;
- **length** — вычисляет длину строки;
- **pos** — вычисляет номер символа, начиная с которого подстрока входит в строку.

Здесь мы не будем разбирать подробно, как работают данные функции. Это можно посмотреть в информационной системе компилятора. Кроме того, здесь не упоминается еще целый ряд возможностей, предоставляемых компилятором языка Паскаль. Подробное описание всех имеющихся функций и процедур не входит в задачи этой книги.

А сейчас приведем несколько примеров, на которых можно посмотреть, как работать с данными этого нового типа.

Пример 1. Вычеркнуть заданный символ из строки везде, где он встречается.

Искомое решение состоит из двух циклически выполняемых операций:

- поиск вхождения символа в строке;
- если вхождение найдено, то символ уничтожается.

```
Program example;
  Uses crt;
  Var
    s:string;
    c:char;
    n:integer;
begin
  clrscr;
  readln(s);readln(c);
```

```
  repeat
    n:=pos(c,s);
    if n>0 then delete(s,n,1);
  until n=0;
  write(s);
end.
```

Пример 2. Проверить, содержит ли строка строку. И если содержит, то сколько раз. Функцией **pos** пользоваться запрещается.

Решение: функция **pos** занимается поиском вхождения подстроки в строку, и если разрешить ей пользоваться, то решение будет слишком простым. Не сложно оно и сейчас. Все, что требуется, это проверить, входит ли подстрока в строку с первого элемента, затем со второго элемента и т. д.

Алгоритм:

1. Вводим строку и подстроку.
2. Вычисляем длины строки и подстроки.
3. Копируем из строки подстроку с длиной, равной длине введенной подстроки. Подстрока копируется, начиная с текущей позиции строки.
4. Сравниваем полученную подстроку с введенной подстрокой. Если имеет место равенство, то счетчик вхождений увеличиваем на 1.
5. Переходим вправо на следующую позицию строки. Если с этой позиции строки можно получить подстроку нужной длины, то возвращаемся на пункт 3. Если невозможно, то печатаем значение счетчика вхождений и прекращаем работу.

```
Program example;
  Uses crt;
  Var
    i,n,h,k:integer;
    t,a,b:string[100];
begin
  clrscr;
  write('введи строку ');readln(a);
```

```
  write('введи подстроку');readln(b);
  n:=0;
  i:=length(a);k:=length(b);
  for h:=1 to i-k+1 do
    begin
      t:=copy(a,h,k);
      if t=b then n:=n+1;
    end;
  write(n);
end.
```

Что отличает строку от символьного массива?

- Нулевой элемент строки содержит количество элементов строки. Хранится оно в виде символа, чей код в таблице ASCII равен требуемому числу. Получить количество элементов строки можно следующим образом:

```
Program example;
  Var
    d:string;
begin
  readln(d);
  write(ord(d[0]));
end.
```

- Строку нельзя заполнить как символьный массив, а затем работать с ней как со строкой. Например, следующая программа должна была бы выдать в результате число 10 (длина строки), но напечатается 0. Это оттого, что не сформирован нулевой элемент.

```
Program example;
  Var
    d:string;
    i:integer;
```

```
begin
  for i:=1 to 10 do d[i]:='w';
  write(length(d));
end.
```

Заключение

Строковый тип предназначен для удобства в работе с символьными данными. Используя этот тип данных, мы получаем в свое распоряжение готовые и очень полезные процедуры и функции, для реализации которых, используя конструкцию массива, пришлось бы потратить некоторое время и усилия. Однако это удобство ограничивает нас в длине используемых строк. Это, конечно, не хорошо, но ограничение в 255 символов, наверное, все же не очень страшное. В реальных задачах этого, как правило, вполне достаточно.

Массивы

В математике, да и в других областях знания, часто применяется такая структура данных, как множество чего-либо, каких-либо объектов. Например, множество изменений температуры в течение дня или множество домов на улице и т. д. У всех этих множеств есть два общих признака. Во-первых, элементы одного множества имеют одну и ту же природу, т. е. любой дом на улице — это, прежде всего, дом. Во-вторых, элементы любого такого множества пронумерованы для того, чтобы один элемент было легко отличить от другого. Например, об измерении температур можно сказать, что это конкретное измерение было первым, а это вторым, и т. д.

Можно сказать, что элементы множества имеют одинаковое имя, упорядочены по какому-либо признаку и отличаются друг от друга порядковым номером. Описанный нами тип данных настолько распространен, что было совершенно естественно предусмотреть подобную структуру и в языке программирования. И такая структура действительно есть.

Кроме простых типов данных (**integer**, **boolean**, **real** и т. д) в языке Паскаль предусмотрена такая структура данных, как массив. *Массив* — это упорядоченное множество элементов какой-либо природы. Приведем несколько примеров описания массивов:

```
a: array[1..10] of integer;
d,c: array[1..100] of boolean;
f: array[1..10,1..100] of real;
```

В первом примере описан массив из десяти целых чисел с именем `a`. Во втором примере описаны два массива `d` и `c`, состоящих из 100 логических значений. И в третьем примере описан двумерный массив `f` вещественных чисел с границами изменения индексов от 1 до 10 и от 1 до 100.

Обращение к элементу массива производится по значению индекса, которое указывается в квадратных скобках:

```
a[1]=2;
b[1,3]=x+5;
```

Чтобы лучше понять, как работать с массивами, рассмотрим несколько примеров.

Пример 1. Дан массив чисел. Найти наибольший элемент массива.

Решение: пусть наибольший элемент содержится в переменной `max`. В процессе поиска наибольшего мы будем просматривать весь массив, начиная с первого элемента. В тот момент, когда мы возьмем из массива первый элемент, значение `max` будет неопределенным. Поэтому на этот момент максимальный элемент естественно будет равен первому. Затем организуем циклическую операцию сравнения каждого последующего элемента массива с уже найденным наибольшим, и если окажется, что очередной элемент больше наибольшего, то наибольшему присвоим значение этого очередного элемента.

```
Program example;
  Uses crt;
  Var
    a:array[1..1000] of real;
    i,n:integer;
    max:real;
```

```
begin
  clrscr;
  write('Сколько чисел - ');
  readln(n);
  for i:=1 to n do
    begin
      write(i,') ');
      readln(a[i]);
    end;
  max:=a[1];
  for i:=2 to n do
    if a[i]>max then max:=a[i];
  write('Наибольшее =',max);
end.
```

Пример 2. Даны два числа в виде массивов цифр. Выполнить операцию сложения.

Решение: сложение столбиком — это сложение по разрядам. При операции сложения цифр разряда с учетом цифры, которую мы держим в уме, возможны только четыре случая:

- сумма цифр разрядов и цифры в уме равна нулю. Тогда в суммарный разряд записывается ноль, и на ум пошел ноль;
- сумма цифр разрядов и цифры в уме равна единице. Тогда в суммарный разряд записывается единица, и на ум пошел ноль;
- сумма цифр разрядов и цифры в уме равна двум. Тогда в суммарный разряд записывается ноль, и на ум пошла единица;
- сумма цифр разрядов и цифры в уме равна трем. Тогда в суммарный разряд записывается единица, и на ум пошла единица.

Операция поразрядного сложения выполнятся с каждым разрядом, т. е. внутри цикла. Количество шагов цикла равно количеству разрядов самого длинного из двух суммируемых чисел. Кроме того, возможна еще ситуация, когда все разряды просуммированы, и на "уме" осталась единица. Это означает, что в чис-

ле, обозначающем сумму, надо добавить еще один старший разряд, содержащий единицу.

```
Program example;
  Uses crt;
  Var
    a,b,c:array[1..10] of integer;
    i,j,n1,n2,n,q:integer;
begin
  clrscr;
  write('Введите длину первого числа - ');readln(n1);
  write('Введите длину второго числа - ');readln(n2);
  { Цифры вводятся, начиная с последней }
  writeln('Вводим первое число');
  for i:=1 to 10 do
    begin
      a[i]:=0;b[i]:=0;c[i]:=0;
    end;
  for i:=1 to n1 do
    begin
      write('Цифра N',i,'  ');readln(a[i]);
    end;
  writeln('Вводим второе число');
  for i:=1 to n2 do
    begin
      write('Цифра N',i,'  ');readln(b[i]);
    end;
  j:=0;
  if n2>n1 then n:=n2 else n:=n1;
  for i:=1 to n do
    begin
      q:=a[i]+b[i]+j;
      if q=0 then begin c[i]:=0;j:=0;end;
      if q=1 then begin c[i]:=1;j:=0;end;
      if q=2 then begin c[i]:=0;j:=1;end;
```

```
      if q=3 then begin c[i]:=1;j:=1;end;
    end;
  if j=1 then c[n2+1]:=1;
  for i:=10 downto 1 do write(c[i]);
end.
```

Пример 3. Дан двумерный массив размером $N \times N$. Выяснить, является ли этот массив магическим квадратом.

Решение: магический квадрат — это двумерный массив размерности $N \times N$, в котором все суммы элементов по столбцам и по строкам одинаковы. Поэтому для проверки можно поступить следующим образом:

1. Вычислим сумму первой строки и обозначим эту сумму, через *S*. Ясно, что все последующие суммы должны быть равны *S*;
2. Будем вычислять суммы столбцов и строк и сравнивать их с *S*. Если хотя бы одна сумма окажется не равна *S*, то этот квадрат не является магическим, в противном случае он действительно магический. В программе введем величину *K*, которая изначально пусть равна 1, и если нашлась сумма, не равная S, то пусть $K = 0$. Если по окончании работы программы $K = 1$, то квадрат магический, если же $K = 0$, то квадрат магическим не является.

Примечание

В данном алгоритме есть один существенный недостаток. Возможно, вы уже на первом шаге сможете обнаружить, что данный квадрат не является магическим, но алгоритм, тем не менее, будет продолжать свою работу. Конечно, было бы значительно эффективнее, если бы программа прекращала свою работу сразу, как только обнаружит сумму, отличную от вычисленной первой.

```
Program example;
  Uses crt;
  Var
    a:array[1..10,1..10] of integer;
    i,j,n,s,w,k,q:integer;
```

```
begin
  clrscr;
  write('Ввести размерность квадрата =');readln(n);
  for i:=1 to n do
    for j:=1 to n do
      begin
        write('A(',j,',',i,')=');readln(a[j,i]);
      end;
    s:=0;
    for i:=1 to n do
      s:=s+a[1,i];
    k:=1;
    for i:=1 to n do
      begin
        w:=0;q:=0;
        for j:=1 to n do
          begin
            w:=w+a[j,i];
            q:=q+a[i,j];
          end;
        if (s<>w)or(q<>s) then k:=0;
      end;
    if k=0 then write('Квадрат не является магическим')
    else write('Квадрат магический');
end.
```

Работа с массивом как единым объектом

Язык Паскаль предоставляет некоторые возможности для работы с массивом как единым целым. Например, допустима такая программа:

```
Program example;
  Uses crt;
  Var
    a, b: array[1..100] of integer;
    n,i:integer;
```

```
begin
  read(n);
  for i:=1 to n do readln(a[i]);
  b:=a;
  for i:=1 to n do writeln(b[i]);
end.
```

В программе вводится массив *A*, затем значения его элементов присваиваются соответствующим элементам массива *B*, каковые далее распечатываются. Далее приводится пример работы с двумерным массивом, в котором Паскаль также корректно выполняет поставленные перед ним задачи присваивания.

```
Program example;  Uses crt;  Var    c,a,b:array[1..10,1..10]
of integer;    i,j,n:integer;begin
  {Вводится двумерный массив A}  readln(n);  for i:=1 to n do
for j:=1 to n do      read(a[i,j]);
  {Завершен ввод массива}
  {Массив A присваивается массиву C, как единое целое}
  c:=a;
  {Присваивание завершено}
  {Распечатка массива C}  for i:=1 to n do    begin      for
j:=1 to n do write('  ',c[i,j]);      writeln;    end;
  {Завершена распечатка массива C}
  {Массив A присваивается массиву B построчно}
  for i:=1 to n do
    b[i]:=a[i];
  {Завершено построчное присваивание}
  {Распечатка массива B}  for i:=1 to n do    begin      for
j:=1 to n do write('  ',b[i,j]);      writeln;    end;
{Завершена распечатка массива B}
end.
```

Не все операции можно выполнять с целым массивом. Например, нельзя указывать имя массива в качестве аргумента некоторых процедур:

```
write(a);
```

ошибка, если `a` — массив.

Нельзя также присваивать имени массива число. Например:

```
w:=5;
```

будет ошибкой, если `w` — имя массива.

Необходимо помнить!

- Индекс элемента массива обязательно должен быть числом целого типа.
- Организовать в виде массива можно любой тип данных, кроме файлового.
- Количество индексов массива может быть практически любым, но длина массива ограничена памятью, доступной компилятору, а чем длиннее массив, тем больше памяти он требует для своего размещения.
- Для обращения к элементу многомерного массива необходимо указывать все индексы. К примеру, если массив был объявлен так:

  ```
  f:array[1..10,1..10] of integer;
  ```

  то следующий оператор `f[5]:=1;` будет неправильным.
- Если вы в качестве индекса элемента массива укажете константу, значение которой выходит за границы определения массива, то компилятор обнаружит такую ошибку, но компилятор бессилен обнаружить эту ошибку, если пределы определения массива нарушает переменная величина. Пусть, например, был определен массив `r:array[1..10] of integer`. В операторе `r[11]:=3;` ошибка будет обнаружена, а в операторах `i:=11;` и `r[i]:=3;` с точки зрения компилятора ошибки нет.
- Массив можно обрабатывать как единую структуру, т. е. можно некоторые операции языка применять не к отдельному элементу массива, а ко всему массиву сразу по его имени.

Заключение

Массив позволяет не перегружать описание переменных большим количеством имен. Даже в случае небольшого количества

переменных выгода очевидна. Далее приводятся два описания одинакового количества переменных:

```
a1, a2, a3, a4, a5, a6, a7, a8, a9, a10 : integer;
a : array[1..10] of integer;
```

Второе описание очевидно короче. При большем количестве переменных первый способ описания данных становится совершенно невозможным (представьте, что вам нужно 500 переменных).

Массив позволяет организовать циклическую обработку данных.

Например, без массива:

```
a1:=sin(1);
a2:=sin(2);
a3:=sin(3);
a4:=sin(4);
a5:=sin(5);
a6:=sin(6);
```

С использованием массива:

```
For i:=1 to 6 do a[i]:=sin(i);
```

Конечно, массивы создают и сложности. Когда мы обращаемся к обычной переменной, мы обращаемся к ней по имени и не может быть никаких сомнений, что наше обращение произошло по верному адресу. Например, можно не сомневаться, что после выполнения оператора `a1:=9;` переменная `a1` будет равняться `9`. Если же мы обращаемся к переменной по индексу, то мы рискуем ошибиться, т. к. значение индекса может изменяться довольно хитроумным способом. Даже, более того, при использовании массивов есть серьезный риск обратиться к вообще несуществующему элементу. Но, тем не менее, выгоды, создаваемые структурами массивов, настолько значительны, что с лихвой компенсируют все создаваемые неудобства.

Множества

Рассмотренные нами ранее массивы можно определить как упорядоченное множество. В языке Паскаль есть еще один тип

данных, который представляет собой обычное множество, т. е. неупорядоченное. Данное понятие чем-то похоже на понятие массива. Массив — это набор элементов. Множество — также набор элементов, но, в отличие от массива, множество — неупорядоченный набор. То есть если мы не можем ни для одного элемента указать его порядковый номер, то это множество, а если можем, то это массив. Множество — очень своеобразный тип данных. Так как его элементы никак не упорядочены, то мы не можем обратиться к его элементам и вынуждены работать с ним, как с единым целым. Однако есть ситуации, когда этого достаточно. Например, мы вводим строку символов и желаем узнать, состоит ли она из одних цифр или там есть и другие символы. С использованием структуры множества задача решается легко и просто, и ее решение мы приведем далее. А сейчас обсудим, как множество определить, задать и какие операции над ним можно выполнять.

Тип "множество" определяется через базовые простые типы. Но т. к. множество может содержать не более чем 255 элементов, то типы **integer**, **real**, **string** не подходят для создания множества. Базовые типы для множеств — это типы **char**, **boolean**, **byte**. Также множество можно определить через типы-перечисления. С помощью базовых типов множество можно задать так:

```
a: set of byte;
b: set of boolean;
c: set of char;
```

Элементами этих множеств будут все значения, допустимые для данных типов.

С помощью типа-перечисления и типа-диапазона множество можно определить так:

```
Type
  y=(p1,p2,p3,p4,p5,p6,p7,p8);
  d=p4..p7;
Var
  a: set of y;
  s: set of d;
```

В этом примере мы создаем два собственных типа. Идентификатор `y` — перечисляемый тип, идентификатор `d` — тип-диапазон. Соответственно идентификаторы `a` и `s` — множества типа-перечисления и типа-диапазона.

Еще одно важное понятие, относящееся к множествам, это операция определения содержимого множества. Такая операция в языке Паскаль называется *конструктор*.

Пример:

```
a:=['w','r'];
```

В этом операторе создано множество с именем `a`, состоящее из двух литер `'w'` и `'r'`.

А теперь перечислим операции над множествами — табл. 1.4.

Таблица 1.4. *Операции над множествами*

Знак	Наименование операции	Получаемый результат
+	Объединение множеств	Множество, содержащее все элементы объединяемых множеств
-	Вычитание множеств	Множество, содержащее все элементы уменьшаемого множества, за исключением тех элементов, которые содержатся в обоих множествах
*	Пересечение множеств	Множество, содержащее только те элементы, которые одновременно содержатся во всех множествах, чье пересечение находится
=	Проверка эквивалентности двух множеств	Логическая величина, которая принимает значение "истина", если множества эквиваленты, и "ложь", если множества неэквивалентны
<>	Проверка неэквивалентности двух множеств	Логическая величина, которая принимает значение "ложь", если множества эквиваленты, и "истина", если множества неэквивалентны

Таблица 1.4 *(окончание)*

Знак	Наименование операции	Получаемый результат
>=	Проверка, является ли правое множество подмножеством левого	Логическая величина, которая принимает значение "истина", если правое множество есть подмножество левого, и "ложь", если это не так
in	Проверка, входит ли элемент, указанный слева, в множество, указанное справа	Логическая величина, которая принимает значение "истина", если элемент содержится во множестве, и "ложь", если это не так
<=	Проверка, является ли левое множество подмножеством правого	Логическая величина, которая принимает значение "истина", если левое множество есть подмножество правого, и "ложь", если это не так

Далее приведен пример, в котором используются все операции над множествами, и в ремарках указан получаемый результат.

```
Program example;
  Uses crt;
  Var
    c,a,b:set of char;
    q:boolean;
begin
  clrscr;
  a:=['a','s','d','f','g'];
  b:=['s','f','x'];
  c:=a+b;
  { Множество c='a','s','d','f','g','s','f','x'}
  c:=a*b;
  { Множество c='s','f'}
  c:=a-b;
  { Множество c='a','d','g'}
  q:=a=b;
  { Логическая переменная принимает значение ложь }
```

```
  q:=a<>b;
  { Логическая переменная принимает значение ложь }
  q:=b<=a;
  { Логическая переменная принимает значение ложь }
  q:=b>=a;
  { Логическая переменная принимает значение ложь }
  q:='x' in b;
  { Логическая переменная принимает значение истина }
end.
```

Примечание

Конструктор можно использовать не только для первоначального задания множества. Конструктор может участвовать и в операциях над множествами. Можно записать следующие операторы: `f:=f+['h'];f:=f+['g']-['f','g'];`. Пустое множество задается так: `f:=[];`. Вполне возможно и вычисление сложных выражений, содержащих различные операции над множествами и скобки. Например: `f:=['g','f','d']-(g*h+['w']);`

А сейчас рассмотрим содержательный *пример*. Итак, пусть введена строка символов, и нам требуется выяснить, состоит ли она из одних цифр, или в ней имеются и другие символы. Идея решения такова. Среди операций, разрешенных со множествами, есть операция **in**, которая проверяет, входит ли элемент в множество или нет. Поэтому мы создадим множество цифр, а затем будем брать из введенной строки все элементы по очереди и проверять, входит ли этот очередной элемент в созданное нами множество. Если входит, то он — цифра, а если не входит, он — иной символ.

```
Program example;
  Uses crt;
  Var
    s:string;
    a: set of char;
    n:integer;
    q:boolean;
```

```
begin
  clrscr;
  a:=['1','2','3','4','5','6','7','8','9','0'];
  readln(s);
  q:=length(s)>0;
  n:=1;
  while (q)and(n<=length(s)) do
    begin
      q:=s[n] in a;
      n:=n+1;
    end;
  if q then write('Строка состоит из одних только цифр')
  else write('Имеются посторонние символы');
end.
```

Примечание

Цикл **while** завершается в двух случаях: или, во-первых, мы просмотрели все элементы строки, или, во-вторых, мы смогли обнаружить элемент, не являющийся цифрой. Если мы найдем элемент, не являющийся цифрой, то этот элемент не будет элементом множества, и результатом оператора `q:=s[n] in a;` будет "ложь", а тогда выполнение цикла **while** прекратится. Если же мы ничего не введем в операторе **read**, то переменная `q` сразу будет иметь значение "ложь", и цикл **while** ни разу не будет исполнен. В нашей программе такая ситуация будет эквивалентна строке, содержащей нецифровые символы.

Заключение

Тип "множество" не позволяет работать с большим количеством данных, и это, возможно, недостаток, но в качестве компенсации Паскаль предлагает очень мощный набор операций. Реализация каждой из операций другими средствами языка потребует написания небольших программных модулей. Кроме того, действительно существуют структуры данных, в которых не существенен порядок элементов и этих элементов не слишком много.

Примеры таких множеств: алфавит, названия дней недели, названия месяцев. А задач, для которых не существенен порядок элементов, достаточно много. Например, структура множества будет полезна, если вам потребуется определить, имеется ли фамилия человека в списке сотрудников предприятия и т. д.

Множества — это хороший пример того, как язык высокого уровня может обрабатывать сложные структуры данных и реализовывать сложные математические операции.

Записи

Предположим, что программа предназначена для работы с группой объектов, каждый из которых характеризуется набором величин, и этот набор может оказаться весьма значительным по объему. Конечно, любому разработчику программы хотелось бы не забыть, какая величина к какому объекту относится. Покажем на примере, как можно поступить в этом случае. Предположим, наш объект — это книга, и она описывается такими величинами, как объем в страницах, фамилия автора и цена. Тогда мы можем создать следующее описание:

```
Var
  book_cost,book_volum:integer;
  book_family:string;
```

Это совершенно нормальное описание, и оно позволяет нам запомнить, что данные переменные относятся к описанию объекта "книга" (book). Но у данного описания есть один существенный недостаток. С точки зрения компилятора эти переменные никак не связаны друг с другом. Это означает, например, следующее: если вы пожелаете передать в другой программный модуль полную информацию о книге, то в списке передаваемых параметров должно быть указано три параметра. Для нашего примера это совершенно не существенно, но если объект будет описываться несколькими десятками величин, то передача параметров в иной программный модуль окажется очень утомительным занятием. Еще одна очень существенная трудность возникнет при работе с файлами данных. В объявлении файла необходимо точно указать, какого типа величина будет храниться в файле. Исходя из

этого, совершенно не ясно, что делать, если величины, описывающие объект, окажутся разного типа. По всей видимости, придется формировать несколько разных файлов, что создаст огромные трудности, о которых даже и не хочется говорить.

Решением проблемы будет создание структуры данных, которая заключала бы в себе данные самой разной природы. Таковой структурой и являются записи языка Паскаль. Для пояснения этого термина возьмем тот же пример и опишем запись `book`.

```
Var
  book: record
         cost,volum: integer;
         family: string;
        end;
```

Слово **end** завершает описание записи. Теперь наши три переменные связаны между собой общим названием объекта. Это означает, что мы можем передать данные на объект куда угодно, используя только имя объекта, а если нам потребуется работать с файлами данных, то мы просто создадим файл, содержащий данные типа "запись". А сейчас, чтобы понять, как работать с этой сложной структурой, рассмотрим пример.

Пример. Сохранение в файле анкетных данных сотрудников.

Для начала договоримся, что нас интересуют следующие данные: стаж работы (в годах), месячный заработок, фамилия, имя, отчество. Возможности программы минимальны: ввод данных и занесение их в файл. Алгоритм работы простейший: программа при запуске запрашивает, сколько данных вводить; затем, используя конструкцию цикла, запрашивает их и заносит в файл.

Опишем структуру записи:

```
Program example;
  Uses crt;
  Type
    shablon=record
              d: integer;
              z: integer;
```

```
             f: string;
             e: string;
             t: string;
            end;
  Var
    i,n: integer;
    anketa: shablon;
    h: file of shablon;
begin
  clrscr;
  assign(h,'anketa.dat');reset(h);
  write(' Сколько данных вводить'); readln(n);
   for i:=1 to n do
     begin
       write(' Введите фамилию ');readln(anketa.f);
       write(' Введите имя '); readln(anketa.e);
       write(' Введите отчество '); readln(anketa.t);
       write(' Введите стаж работы '); readln(anketa.d);
       write(' Введите величину месячного заработка ');
readln(anketa.z);
       write(h,anketa);
     end;
   close(h);
end.
```

Структура записи еще интересна тем, что ее компонентом может быть массив. Конечно, это ясно уже из предыдущего примера, т. к. тип **`string`** — это не что иное, как массив типа **`char`**. Но чтобы лучше уяснить эту возможность, рассмотрим еще одну задачу. Возьмем тот же пример с анкетой, но добавим в анкету еще один пункт: предположим, есть необходимость запоминать все суммы, выплаченные человеку. Для учета нового пункта необходимо завести массив типа **`real`**. Покажем, как это можно сделать, но для краткости не будем записывать данные в файл.

```
Program example;  Uses crt;  Var    j,i,n,m: integer;
anketa:array[1..10] of  record
           d: integer;
```

```
           z: integer;
           f: string[20];
           e: string[20];
           t: string[20];
           g: array[1..100] of real;
         end;
begin
  clrscr;
  write(' Сколько данных вводить'); readln(n);
  for i:=1 to n do
    begin
      with anketa[i] do
        begin
          write(' Введите фамилию ');readln(f);
          write(' Введите имя '); readln(e);
          write(' Введите отчество '); readln(t);
          write(' Введите стаж работы '); readln(d);
          write(' Введите величину месячного заработка ');
readln(z);
          write(' Сколько выплат было произведено');
readln(m);
          for j:=1 to m do
            read(g[j]);
        end;
    end;
end.
```

Обратите внимание на оператор **with**. Этот чрезвычайно полезный оператор позволяет не упоминать имя записи, а записывать только имена компонентов записи. Действует он, от слова **begin** до соответствующего слова **end**. Использование **with** накладывает определенные ограничения на имена переменных. Например, в приведенной ранее программе вполне можно объявить переменную с именем `t`, несмотря на то, что есть компонент записи с таким именем. Однако использовать эту переменную в пределах действия **with** уже нельзя, т. к. этот оператор не сможет от-

личить компонент записи с именем `t` от переменной с именем `t`, что вызовет сообщение об ошибке от компилятора.

Вариантные записи

Рассмотрим следующую задачу: пусть требуется создать базу данных из анкет сотрудников предприятия. Причем известно, что предприятие состоит из нескольких подразделений. И для каждого подразделения кроме общих данных (например, возраст, стаж работы), которые нужно знать о сотруднике в любом случае, требуются и некоторые специальные данные. Например, наличие допуска к определенной работе для расчетчика заработной платы или уборщицы роли не играет, но для станочника, работающего на станках высокой сложности, это важно. Могут быть, конечно, и другие примеры разной потребности в информации в различных подразделениях предприятия.

Из этого затруднения можно попытаться выйти, разработав отдельные базы данных для каждого подразделения, но это создаст сложности в управлении предприятием. А именно:

- для каждой базы данных придется писать отдельные программы обработки, что приведет к лишним затратам;
- появятся трудности при получении общей статистической информации о всем предприятии.

Можно, конечно, привести примеры и других проблем. Ясно, что нужна такая структура записей, в которой учитывались бы все возможные варианты и, в то же время, в каждой записи присутствовали только те поля, которые нужны для данной записи и только для нее. Паскаль для решения описанной проблемы предлагает такую структуру данных, как *вариантная запись*. Ее общая конструкция выглядит следующим образом:

```
Program example;  Type
    figure=(square,triangle,circle);
    param=record
          x,y:real;
          case fig: figure of
            square:(side:real);
            triangle:(side1,side2:real;angle:integer);
```

```
            circle:(radius:real);
          end;
  Var
    t:param;
begin
  t.fig:=square;
  t.side:=3;
  t.fig:=circle;
  t.radius:=4.2;
end.
```

В данном примере компонент записи с именем `fig` определяет варианты полей. Причем, как видно из примера, можно определить вариант, состоящий из нескольких полей. В процессе работы программы определение варианта записи происходит согласно текущему значению переключателя **`case`**, в нашем случае переключателем является компонент `fig`.

Заключение

Запись — крайне полезная конструкция, позволяющая обходиться со сложными данными как с единым целым. Это важно для таких операций, как занесение данных в файл, передача данных в другой программный блок. Благодаря этому запись является очень удобным инструментом для создания баз данных.

Графика языка Паскаль

Графика языка Паскаль — это перечень графических процедур и функций. Приведем список этих процедур и функций и поясним, как ими пользоваться.

- **`line`** — процедура рисования отрезка. Аргументы процедуры: координаты точек концов отрезка. Отрезок рисуется от начальной точки до последней. Пример: `line(x1,y1,x2,y2);`. Отрезок рисуется цветом, установленным последней процедурой **`setcolor`**.

- **circle** — процедура рисования окружности. Аргументы процедуры: координаты центра и радиус. Пример: `circle(x,y,r);`. Окружность рисуется цветом, установленным последней процедурой **setcolor**.
- **putpixel** — процедура установки точки. Аргументы процедуры: координаты точки и цвет точки. Пример: `putpixel(x,y,15);`.
- **initgraph** — процедура инициализации графического экрана. Данная процедура должна выполняться перед первым вызовом графической процедуры или функции. Пример: `initgraph(driver,mode,'');`. Внутри апострофов должно стоять имя драйвера и быть указан путь к нему, но это не обязательно. Если имя пропущено, то будет загружен драйвер egavga.bgi из текущего каталога.
- **getpixel** — функция, определяющая цвет заданной точки. Аргументы процедуры: координаты точки. Пример: `c:=getpixel(x,y);`.
- **setfillstyle** — процедура, определяющая способ закраски контура. Аргументы процедуры: шаблон закраски и цвет закраски. Пример: `setfillstyle(1,8);`.
- **floodfill** — процедура, осуществляющая закраску. Аргументы процедуры: координаты точки, в которую льется краска, и номер цвета контура, до которого льется краска. Краска начинает литься в указанную точку и растекаться во всех направлениях до тех пор, пока не встретит контур указанного цвета. Если контур не встретится, то краска зальет весь экран. Пример: `floodfill(x,y,6);`.
- **setcolor** — процедура установки цвета для рисующих процедур. Аргумент процедуры: номер устанавливаемого цвета. Пример: `setcolor(9);`.
- **setbkcolor** — процедура установки цвета фона. Аргумент процедуры: номер устанавливаемого цвета. Пример: `setbkcolor(9);`.
- **outtextxy** — процедура вывода строки символов в графическом режиме экрана. Аргументы процедуры: координаты точки, с которой начинается вывод строки, и выводимая

строка. Выводимая строка может быть как типа **string**, так и типа **char**. Пример: `outtextxy(12,45,'Пример выводимой строки');`.

- **closegraph** — процедура закрытия графического экрана.
- **cleardevice** — процедура очистки экрана.

Примечание

Все аргументы всех графических процедур и функций могут быть только целого типа.

Пример 1. Простейшая графическая задача.

Условие: нарисуем пирамиду (снеговика) из закрашенных окружностей, в которой каждая из окружностей рисуется и закрашивается своим (отличным от соседей) цветом.

```
Program example;
  Uses crt,graph; {библиотека graph содержит графические процедуры}
  Var
    i,driver,mode:integer;
begin
  driver:=detect;
  initgraph(driver,mode,'');
  for i:=1 to 8 do
    begin
      setfillstyle(1,i);
      setcolor(i);
      circle(300,180-i*23,150-i*18);
      floodfill(300,180-i*23,i);
    end;
  closegraph;
end.
```

Пример 2. Построение динамических объектов — изображение летящего отрезка. Идея решения следующая: изобразим некото-

рый отрезок с помощью процедуры **line**. Затем будем циклически выполнять два действия:

- ❒ пририсовывать одну точку цвета отрезка к правому концу отрезка;
- ❒ закрашивать крайнюю левую точку отрезка точкой цвета фона.

Последовательное и многократное выполнение указанных действий создаст эффект движения отрезка. Движение будет происходить слишком быстро, поэтому в программу полезно включить процедуру временной задержки **delay**. Вот как будет выглядеть программа:

```
Program example;
  Uses crt,graph;
  Var
    driver,mode,i:integer;
begin
  driver:=detect;
  initgraph(driver,mode,'');
  setbkcolor(0);setcolor(6);
  line(5,150,25,150);
  for i:=5 to 600 do
    begin
      putpixel(i,150,0);
      putpixel(i+21,150,6);
      delay(10);
    end;
end.
```

Заключение

Описанный набор графических процедур далеко не полон. Даже в версиях, работающих под DOS, их значительно больше. Еще больше возможностей по созданию графических объектов предлагают графические библиотеки компиляторов, создающие программы под Windows. Но необходимо понять, что логическая сложность графических программ определяется совершенно не

тем, что они графические. Какой-либо особой логики здесь нет, освоение дополнительных возможностей сводится к внимательному чтению документации библиотек процедур и функций, прилагаемых к компилятору.

Процедуры и функции

Большинство современных программных средств представляют собой огромные тексты в тысячи, десятки тысяч и даже сотни тысяч команд. Разобраться в такой программе исключительно сложно, но и написать такую программу не менее трудно. Чем больше объем информации, тем труднее держать его в голове. Эта сложная проблема решается очень просто. Если вы не можете обработать задачу целиком, разбейте ее на части, решите по частям и потом сложите полученные небольшие программки вместе.

Пример 1. Вычисление суммы факториалов.

Эту задачу можно разбить на две подзадачи. Первая — вычисление факториала, вторая — суммирование факториалов, как будто их значения уже известны. Составим для начала алгоритм из тех соображений, что факториалы уже вычислены:

```
сумма=0
цикл по i от 1 до N
  начало
    сумма=сумма+факториал(i)
  конец
конец алгоритма
```

Мы написали команду `факториал(i)` с таким расчетом, что исполнитель данного алгоритма, дойдя до команды, вычислит факториал от числа `i` и использует его для дальнейших расчетов. Вообще говоря, в своей вычислительной практике мы часто так поступаем. Например, в выражении `5*sin(x)` предполагается, что метод вычисления синуса известен. Если все же это не так, и исполнитель не знает, как вычислить факториал, то нужно

описать метод вычисления, причем не обязательно в главном алгоритме.

Сейчас запишем алгоритм функции для расчета факториала:

```
факториал=1
цикл по j от 1 до i делать
  начало
    факториал=факториал*j
  конец
конец алгоритма
```

Итак, исполнитель, отрабатывая главный алгоритм, дойдет до команды `факториал(i)`, после чего он найдет описание алгоритма расчета факториала, вычислит значение факториала и вернется в главный алгоритм. Мы, таким образом, смогли разбить алгоритм на две независимые части, одна из которых называется главным алгоритмом, а вторая — функцией.

Программа будет выглядеть так:

```
Program example;
  Var
    n,i,s:integer;
    { Начало описания функции }
    function factorial(m:integer):integer;
    var
     j,k:integer;
    begin
      k:=1;
      for j:=1 to m do k:=k*j;
      factorial:=k;
    end;
    { Конец описания функции }
begin
  readln(n);s:=0;
  for i:=1 to n do s:=s+factorial(i);
    writeln(s);
end.
```

Как это работает? Работа программы начинается не в теле функции, а после слова **begin**, которым начинается главная программа. В тот момент, когда выполнение программы подходит к вызову функции `factorial`, управление передается на список операторов, из которых состоит тело функции. Значение параметра `i` передается параметру `m`, описанному в заголовке функции, который затем используется в тексте функции (параметр `i` называется фактическим, а параметр `m` формальным). Оператор `factorial:=k;` определяет, какое значение будет являться результатом вычисления функции.

А сейчас необходимо сделать важнейшее пояснение относительно применяемых в этой программе понятий формального параметра и локальной переменной.

Имена параметров, перечисляемые в заголовке функции (в данном случае мы имеем только один параметр `m`), называются *формальными параметрами*. Формальность их заключается в том, что они как бы не имеют собственного значения. Значения им передаются в момент вызова функции. Чтобы понять, как это делается, приведем пример. Предположим, что в описании функции `Summa` дан такой список формальных параметров: `f`, `g`, `h`, `j`. И пусть в вызове этой функции дан следующий список *фактических параметров*: `3`, `y`, `0`, `u+1`. Тогда передача значений фактических параметров формальным параметром произойдет так: `f=3`; `g=y`; `h=0`; `j=u+1`.

Локальная переменная — это такая переменная, значение которой доступно только в данной конкретной процедуре (что такое процедура, будет сказано чуть позже) или функции. Локальными переменными являются любые переменные, описанные внутри процедур и функций. В нашем примере в функции `factorial` локальными переменными являются переменные `m`, `j`, `k`. Полезность локальных переменных заключается в том, что, именно благодаря им нам удается изолировать вычислительный процесс, проходящий внутри процедуры или функции, от вычислительных процессов, проходящих в других частях программы. Причем, даже если локальные переменные будут иметь те же самые имена, что и переменные, используемые в главной программе или в других программных блоках, все равно это будут разные, совершенно не связанные между собой переменные.

В отличие от локальных переменных, переменные, описанные в главной программе, называются *глобальными*, то есть доступными в любом программном блоке, за исключением лишь тех случаев, когда в программном блоке используется локальная переменная с тем же именем. А сейчас на нашем же примере покажем, каких ошибок помогает избежать применение локальных переменных. Изменим программу вычисления суммы факториалов следующим образом:

```
Program example;
  Var
    n,i,s:integer;
    function factorial(m:integer):integer;
    var
      j:integer;
    begin
      s:=1;
      for j:=1 to m do s:=s*j;
      factorial:=s;
    end;
begin
  readln(n);s:=0;
    for i:=1 to n do s:=s+factorial(i);
  writeln(s);
end.
```

В этом варианте программы для вычисления факториала используется глобальная переменная `s`, которая в главной программе играет роль суммы факториалов. Это означает, что при каждом вызове функции `factorial` переменной `s` будет присваиваться значение факториала и, следовательно, эта переменная никак не будет выполнять возложенной на нее роли. Эта проблема становится очень серьезной при написании больших программ, когда иногда бывает очень сложно проследить за всеми изменениями переменных.

А теперь, что такое процедура?

Процедура отличается от функции тем, что она, выполняя нужные действия, не возвращает никакого значения. В принципе на

этом различия заканчиваются, но для лучшего понимания приведем пару примеров.

Пример 2. Предположим, что в нашей программе есть необходимость выводить на экран различные сообщения, помещенные в рамку.

Напишем программу, выводящую одно такое сообщение:

```
Program example;
  Uses crt;
  Var
    x,y,i:integer;
begin
  clrscr;
  x:=10;y:=10;
  for i:=1 to 20 do
    begin
      gotoxy(x-3+i,y);write('*');
      gotoxy(x-3+i,y+2);write('*');
    end;
  gotoxy(x-2,y+1);write('*');
  gotoxy(x+17,y+1);write('*');
  gotoxy(x-1,y+1);write('first');
end.
```

Фрагмент, выделенный жирным шрифтом, рисует рамку. Заметим сразу, что этой части программы достаточно безразлично, чему равны переменные `x` и `y`. Это означает, что выделенный фрагмент программы можно записать в виде отдельной процедуры.

Немного усложним задачу. Пусть теперь требуется выводить несколько сообщений в разных местах программы. Сейчас процедуры помогут сделать программу существенно короче. Мы не будем записывать нужный фрагмент каждый раз, когда потребуется рамка. Мы оформим его в виде процедуры, которая вызывается каждый раз, когда в том возникает необходимость.

```
Program example;
  Uses crt;
    procedure ramka(x,y:integer;s:string);
    var
      i:integer;
    begin
      for i:=1 to 20 do
        begin
          gotoxy(x-3+i,y);write('*');
          gotoxy(x-3+i,y+2);write('*');
        end;
      gotoxy(x-2,y+1);write('*');
      gotoxy(x+17,y+1);write('*');
      gotoxy(x-1,y+1);write(s);
    end;
begin
  clrscr;
  ramka(10,10,'first');
  ramka(15,15,'second');
  ramka(20,20,'third');
end.
```

Пример 3. Предположим, что нам поручено разработать программу учета кадров некой организации. Организация состоит из большого количества отделов, в них работают сотрудники, на которых заведены личные карточки. Карточка в нашей терминологии — это сложное данное типа "запись". Конечно, форма личной карточки общая, независимо от отдела, в котором работает сотрудник. Но у отдела кадров есть необходимость сортировать дела по отделам. То есть для каждого отдела заводится отдельный массив типа "запись". В программе такая организация данных выглядела бы следующим образом:

```
Type
  delo=record
       поле1: тип1;
```

```
       поле2: тип2;
       .....
      end;
Var
otdel1, otdel2: array[1..1000] of delo;
otdel2: array[1..1000] of delo;
.............................
```

Далее, требуется организовать обработку этих данных. Способ обработки одинаковый для всех личных дел, независимо от отдела, в котором работает сотрудник, и написать программу такой обработки совершенно несложно, но есть небольшая проблема. Массивы записей, хранящие данные на разные отделы, имеют различные имена, и, чтобы выполнить нашу программу для каждого нового отдела, нам придется ее изменять, меняя имена массивов, что, конечно, неудобно. Механизм работы процедуры позволяет этого избежать. Будем просто передавать в процедуру имя личного дела в качестве фактического параметра, и процедура будет его обрабатывать, не заботясь о том, о каком конкретно отделе идет речь. А теперь покажем на уровне программы, как это можно сделать. Но, чтобы демонстрационная программа не оказалась слишком громоздкой, договоримся о некоторых упрощениях.

- Пусть отдела будет только два: `otdel1` и `otdel2`.
- Пусть на сотрудников вводятся только три типа данных: фамилия, количество проработанных в организации лет и ежегодный заработок.
- Требуемая обработка будет заключаться в вычислении общего заработка сотрудника за весь период работы и выводе результата на экран.

```
Program example;
  Uses crt;
  Type
    delo=record
         fio:string;
         age:integer;
         salary:real;
        end;
```

```
  Var
    otdel1:array[1..10] of delo;
    otdel2:array[1..10] of delo;
    i:integer;
    procedure worker(pupil:delo);
    var
      s:real;
    begin
      s:=pupil.age*pupil.salary;
       writeln(pupil.fio,' ',s);
    end;
begin
  clrscr;
  { Ввод данных }
  for i:=1 to 10 do
    begin
      readln(otdel1[i].fio);
      readln(otdel1[i].age);
      readln(otdel1[i].salary);
      readln(otdel2[i].fio);
      readln(otdel2[i].age);
      readln(otdel2[i].salary);
    end;
  { Обработка данных }
  for i:=1 to 10 do
    begin
      worker(otdel1[i]);
      worker(otdel2[i]);
    end;
end.
```

Процедуры без параметров

Процедуры (и функции) совершенно не обязательно должны получать на входе какие-либо значения. Можно создавать про-

цедуры (и функции) без параметров. Поясним это на следующем примере. Предположим, нам необходимо ввести некоторое количество анкетных данных. Пусть анкета состоит из двух пунктов: фамилия и возраст. Для организации таких данных создадим массив записей с двумя компонентами (строка для фамилии и тип "целый" для возраста). Ввод данных — это циклический процесс, но хотелось бы, чтобы все фамилии вводились в какой-то определенной строке и все возрасты также вводились в определенной строке. Для чего после ввода данных на человека эти две строчки, в которых происходил ввод, необходимо очищать, но воспользоваться процедурой **clrscr** нельзя, т. к. она очищает весь экран, а у нас вверху экрана будет находиться сообщение о том, что сейчас происходит ввод данных, поэтому очищать надо только две конкретные строки.

Сложного в этом нет ничего, просто будем на этих двух строках печатать много пробелов, и старые записи при этом сотрутся. Можно даже эти пробелы печатать прямо в цикле ввода информации, но мы (для иллюстрации применения процедур) оформим такое стирание в виде процедуры без параметров. Вот как это будет выглядеть:

```
  Program example;
  Uses crt;
  Var
    anketa:array[1..10] of record
             fio:string;
             age:integer;
           end;
    i:integer;
    procedure cls;
    begin
      gotoxy(5,5);write('
');
      gotoxy(5,7);write('
');
    end;
begin
  clrscr;
```

```
  gotoxy(30,2);write('Ввод анкетных данных');
  for i:=1 to 2 do
    begin
      gotoxy(5,5);write('Введите фамилию -
');readln(anketa[i].fio);
      gotoxy(5,7);write('Введите возраст -
');readln(anketa[i].age);
      cls;{ Вызов очищающей процедуры }
    end;
end.
```

Примечание

Функция также может быть описана без списка входных параметров. Только следует заметить, что в этом мало смысла.

О передаваемых параметрах

И передавать, и получать можно переменные любых базовых типов, в том числе и данные строкового типа. Непосредственно массивы передавать нельзя, и массив не может быть результатом функции. Однако существует механизм, позволяющий передать массив. Суть его в следующем: если определить массив как собственный тип данных *(см. разд. "Типы данных")*, то для компилятора этот тип будет новым объектом, который он не имеет права воспринимать как обыкновенный массив (т. е. множество объектов одинаковой природы). И вместо множества элементов компилятор теперь видит один объект, хотя и сложный, и, следовательно, теперь нет никаких проблем с его передачей как фактического параметра. Покажем на примере, как это можно осуществить.

```
Program example;
  Uses crt;
  Type
     c=array[1..100] of integer;
  Var
    a:c;
    i:integer;
```

```
    procedure print(g:c);
    var
      i:integer;
    begin
      clrscr;
      for i:=1 to 50 do
        write(' ',g[i]);
    end;
begin
  for i:=1 to 50 do
    a[i]:=i;
  print(a);
end.
```

В этом примере в главной программе формируется массив, затем он передается как фактический параметр в процедуру `print` и в ней распечатывается.

Механизм передачи параметра следует рассмотреть особо. Оказывается, он устроен таким образом, что, используя его, можно организовать возврат данных из процедуры. То есть выполнить операцию, для которой процедуры, вообще говоря, не предназначены. Рассмотрим следующий пример:

```
Program example;
  Uses crt;
  Var
    u:integer;
    procedure goga(var g:integer);
    begin
      g:=20;
    end;
begin
  clrscr;
  goga(u);
  write(u);
end.
```

Эта программа интересна тем, что она распечатает значение переменной u, равное 20. Однако эта переменная в программе нигде не определяется. Можно предположить, что процедура использует область памяти, в которой хранится переменная u для обработки переменной, объявленной в качестве формального параметра. Это и приводит к эффекту возврата значения.

Вызов процедуры/функции

На первый взгляд все, что нужно для вызова, это написать имя процедуры или функции и затем указать в скобках список фактических параметров. Однако это не так. Возможна ситуация, в которой один модуль (будем так для краткости называть и процедуры, и функции, и главную программу) не виден из другого. Начнем с примеров.

```
Program example;
  procedure proba1;
  begin { Начало процедуры proba1}
  end; { Конец процедуры proba1}
  procedure proba2;
  begin { Начало процедуры proba2}
  end; { Конец процедуры proba2}
begin { Начало главной программы}
end. { Конец главной программы }
```

В этом примере в главной программе можно вызывать обе процедуры, и в процедуре proba2 можно вызывать процедуру proba1. Процедуру proba2 внутри процедуры proba1 вызывать нельзя вследствие общего запрета языка Паскаль использовать имена до их описания.

```
Program example;
  procedure proba1;
  procedure proba2;
  begin { Начало процедуры proba2}
  end; { Конец процедуры proba2}
  begin { Начало процедуры proba1}
  end; { Конец процедуры proba1}
```

```
begin { Начало главной программы}
end. { Конец главной программы }
```

В этом примере в главной программе доступна только процедура `proba1`. Процедура `proba2` доступна только внутри процедуры `proba1`.

Из приведенных ранее примеров можно вывести общее правило:

- модуль можно использовать только внутри того модуля, в котором он описан;
- модуль может использовать только те модули, которые описаны ранее него.

В табл. 1.5 приведены вызовы нескольких процедур.

Таблица 1.5. *Примеры процедур*

Правильно	**Неправильно**

`Program example;`
`  procedure proba1;`
`  procedure proba2;`
`  begin {Начало процедуры proba2}`
`  end; {Конец процедуры proba2}`
`  begin {Начало процедуры proba1}`
`  proba2;`
`  end; {Конец процедуры proba2}`
`begin {Начало главной программы}`
`  proba1;`
`end. {Конец главной программы}`
	`Program example;`
	`  procedure proba1;`
	`  procedure proba2;`
	`  begin {Начало процедуры proba2}`
	`  end; {Конец процедуры proba2}`
	`  begin {Начало процедуры proba1}`
	`  proba2;`
	`  end; {Конец процедуры proba2}`
	`begin {Начало главной программы}`
	`  proba2; {Неправильный вызов процедуры}`
	`end. {Конец главной программы}`

Требование описывать имена перед их использованием является одним из важнейших принципов языка Паскаль. Однако иногда

встречаются ситуации, когда данный принцип сильно затрудняет работу. Не часто, но возможны программы, в которых желательно использовать модуль, чье описание будет сделано позже. В компиляторе Турбо Паскаль эта проблема решается введением ключевого слова **forward**. Теперь мы можем использовать опережающий вызов процедур. Делается это так:

```
Program example;
  forward proba2;
  procedure proba1;
  begin { Начало процедуры proba1}
    proba2;
  end; { Конец процедуры proba1}
  procedure proba2;
  begin { Начало процедуры proba2}
  end; { Конец процедуры proba2}
begin { Начало главной программы}
end. { Конец главной программы }
```

Ключевое слово **forward** сообщает компилятору, что описание процедуры `proba2` встретится позже.

Заключение

В применении процедур и функций есть две большие полезности. Во-первых, их можно вызывать из любого места программы или другой процедуры/функции и тем самым серьезно сократить текст программы. Во-вторых, упрощается разработка программы. Большую задачу можно разбить на более мелкие задачи, решить их отдельно, оформить в виде процедур или функций, потом скомпоновать их вместе.

Важнейшим понятием этого раздела является локальная переменная, сущность которой заключается в том, что ее значение доступно только в том программном блоке, в котором она описана. По большому счету именно локальные переменные дают возможность процедурам и функциям выполнять успешно свою роль, т. к. недоступность локальных переменных для других блоков программы делает процессы, проходящие в процедурах/

функциях, изолированными. А это в свою очередь дает возможность процедурам/функциям быть строительными кирпичиками для больших программ.

Рекурсия

Предположим, нам нужно вычислить некоторую величину, пусть она называется `Value`. Предположим также, что разумно проводить ее вычисление в отдельной функции. А теперь представьте себе, что `Value` — это рекуррентная функция и выглядит она например так:

```
Value(n)=Value(n-1)+1
```

Как видите, имя функции стоит как в правой части выражения, так и в левой части. Это означает, что в процессе вычисления величины возникает потребность вычисления подобной же величины. С точки зрения программирования можно сказать, что внутри функции, вычисляющей величину `Value`, появляется вызов этой же функции. Такой вызов и называется *рекурсией*. Чтобы понять, как это работает, рассмотрим несложный пример расчета факториала.

Согласно математическому определению факториал числа равен произведению всех чисел от единицы до данного числа. В виде формулы это можно записать так: `n!=1*2*3...*(n-1)*n`. Теперь заметим, что если в этой формуле, в правой части, отбросить последний сомножитель, то оставшееся `1*2*3...*(n-1)` будет не что иное, как факториал числа `n-1`. Таким образом, мы приходим к новому определению факториала: `n!=(n-1)!*n`. Это определение существенно короче. А его математический смысл заключается в сведении вычисления факториала к вычислению его от меньшего аргумента.

Программа, вычисляющая значение факториала, будет вызывать функцию, вычисляющую значение факториала, а эта функция на каждом шаге своей работы должна определять значение факториала, как факториал от меньшего числа, умноженного на константу. И происходить это будет до тех пор, пока аргумент функции не станет равным единице, факториал от которой равен единице. После чего начнется обратный процесс возврата

вычисленных значений. Запишем эту программу и попробуем ее проанализировать.

```
Program example;
  Var
    n:integer;
    function factorial(n:integer):integer;
    begin
      if n>1 then factorial=n*factorial(n-1)
      else factorial=1;
    end;
begin
  read(n);
  write('Факториал=',factorial(n);
end.
```

А теперь проведем анализ. Ответим на вопрос: "Что происходит при вызове функции `factorial`?" Пусть требуется вычислить `factorial(4)`. Чтобы это сделать, необходимо найти `factorial(3)`, что достигается рекурсивным обращением функции `factorial` к себе самой. Вычисление `factorial(4)` временно приостанавливается до тех пор, пока не будет получен результат вызова `factorial(3)`. Как только это произойдет, вычисление `factorial(4)` сможет, наконец, завершиться: величина `factorial(3)` будет просто помножена на `4`.

Но для вычисления `factorial(3)` нужно располагать значением `factorial(2)`: будучи помноженным на `3`, оно даст `factorial(3)`. Для определения `factorial(2)` требуется получить `factorial(1)` и умножить его на `2`. Факториал от единицы по определению равен `1`. С этим значением мы продолжим вычисление `factorial(2)`, временно находившееся в состоянии ожидания ответа от `factorial(1)`. Теперь наша программа может вычислить `factorial(2)`, как `2*factorial(1)=2`. Далее полученная двойка возвращается к вычислению `factorial(3)` и выполняет его умножением `3*2=6`. Затем значение `6` позволяет завершить приостановленное вычисление `factorial(4)`. Получаем `factorial(4)=6*4=24`.

Пример 1. Вычисление рекуррентной функции.

Напишем программу вычисления функции, заданной следующими условиями:

- `f(1)=1`;
- `f(2N)=f(N)`;
- `f(2N+1)=f(N)+f(N+1)`.

Решение: из определения видно, что вычисление функции заключается в сведении функции от аргумента к вычислению функции от меньшего аргумента, и процесс такого сведения продолжается до тех пор, пока все получаемые функции не будут иметь своим аргументом единицу. Значение же функции от единицы единственно известное.

Таким образом, работа алгоритма будет состоять из некоторого количества шагов, на каждом из которых будут выполняться два действия:

1. Определение четности или нечетности аргумента (от этого зависит выбор формулы преобразования).
2. Выполнение преобразования. Фактически такое преобразование будет сводиться к определению нового аргумента функции.

```
Program example;
  Uses crt;
  Var
    n,a:integer;
    c:char;
    function f(n:integer):integer;
    begin
      if n=1 then f:=1
      else begin
        if frac(n/2)=0 then begin
          n:=round(int(n/2));
          f:=f(n);
        end
```

```
        else begin
          n:=round(int(n/2));
          f:=f(n)+f(n+1);
        end;
      end;
    end;
begin
clrscr;
write('Введите число - ');readln(n);
a:=f(n);
write('Результат =',a);
end.
```

В рекурсивной процедуре/функции могут быть определены локальные переменные. То есть переменные, описанные в заголовке процедуры/функции или описанные после ключевого слова **var**. Может возникнуть вопрос, что с ними происходит при следующем самовызове процедуры. Чтобы на него ответить, вернемся к уже рассмотренной программе расчета факториала. Пусть мы вычисляем факториал `4`. Вызов (пока аргумент не равен `1`) осуществляется так: `factorial:=n*factorial(n-1)`. На первом шаге эта формула будет выглядеть следующим образом: `factorial:=4*factorial(3)`. Далее функция будет вычислять факториал от `3`, но рано или поздно вернется к своему первому вызову для того, чтобы рассчитать факториал от `4` при уже известном факториале от `3`. Ясно, что она (рекурсивная функция) должна запомнить, чему было равно на этом шаге `n`.

Все локальные переменные сохраняют свои значения для каждого вызова рекурсивной процедуры/функции. Однако появляется проблема памяти. Для хранения всех локальных переменных используется специальная структура памяти называемая *стеком*, и конечно его объем не безграничен. Поэтому, если в вашей программе окажется слишком много рекурсивных вызовов или список локальных данных слишком велик, то вы рискуете исчерпать стек до того, как будут получены требуемые результаты.

Заключение

Из всех приведенных ранее примеров можно сделать следующий важнейший вывод: применение рекурсии выгодно тогда, когда необходимо запустить процесс, выполнение которого состоит из совершенно идентичных шагов, и результаты выполнения каждого шага рекурсии становятся исходными данными последующего шага рекурсии.

Однако если процесс линейный, то рекурсия становится, пожалуй, слишком сложным методом, для линейного процесса подойдет и обычная конструкция цикла. Рекурсия, безусловно, выгодна, когда процесс разветвляющийся, т. к. в разветвляющемся вычислительном процессе главная сложность заключается в запоминании, откуда был совершен переход на текущее ответвление. Механизм рекурсии берет решение этой проблемы на себя. Примером такого разветвляющегося процесса было вычисление рекуррентной функции.

Файлы данных

Файл данных — это пространство на магнитном носителе, зарезервированное для хранения информации и имеющее определенное имя. *Файл* — это также последовательность чисел (кодов), некоторые из которых понимаются как управляющие коды (например, существует код признака конца файла, для текстовых файлов существует признак конца строки). Что представляет из себя эта последовательность чисел, можно решить только в программе. В самом файле нет никаких признаков, позволяющих определить характер информации.

Работать с файлом на физическом уровне крайне сложно. Поэтому для файла, как и для обычной переменной, Паскаль определяет тип. Например, можно записать:

`a: file of integer;` (Файл целых чисел.)

`f: file of string;` (Файл строк.)

Далее все операции производятся не с физическим файлом, а с определенной файловой переменной, и если в файловую пере-

менную заносится величина соответствующего типа (т. е. такого же типа, как и тип файла), то эта величина на самом деле записывается в файл. Однако, конечно, файловая переменная — не совсем обычная переменная, поэтому прежде чем начать с ней работу, нужно выполнить некоторые операции.

Алгоритм работы с файлом:

1. Файл специальной процедурой связывается с файловой переменной.
2. Открывается либо создается файловая переменная (физически будет открыт файл, но программист в этом процессе не участвует).
3. Выполняются необходимые операции чтения и записи данных.
4. Файловая переменная закрывается.

Пример. Напишем программу, в которой будут выполняться следующие действия.

1. Открывается файл.
2. Записывается 100 последовательных целых чисел.
3. Файл закрывается.
4. Файл открывается.
5. Считываются числа, содержащиеся в файле, и распечатываются на экран.

```
Program example;
  Var
    i,u: integer;
  { Определяется переменная, которую затем можно привязать к
    файлу, содержащему целые числа }
    f: file of integer;
begin
{ Файловая переменная привязывается к файлу,
  имя которого указано в апострофах }
  assign(f,'file');
```

```
{ Так как такой файл еще не существует, то он создается и
открывается}
  rewrite(f);
{ Числа записываются в файл }
  for i:=1 to 100 do write(f,i);
{ Файл закрывается}
  close(f);
{ Так как файл уже существует, то он просто открывается }
  reset(f);
{ Числа читаются из файла и выводятся на экран дисплея}
  for i:=1 to 100 do
    begin
      read(f,u);
      write(' ',u);
    end;
end.
```

Файловая переменная не может быть совершенно обычной переменной. Поэтому для файловых переменных предусмотрены специальные процедуры и функции. Краткий список процедур и функций, работающих с файлами в Borland Pascal, приведен далее.

- **assign** — процедура, связывающая файл с файловой переменной.
- **reset** — открывает существующий файл и устанавливает указатель позиции файла на нулевой элемент.
- **rewrite** — создает файл.
- **truncate** — обрезает файл, начиная с текущей позиции.
- **seek** — устанавливает указатель файла в указанную позицию.
- **eof** — функция, возвращающая истину, если был достигнут конец файла, и ложь в противном случае.
- **filesize** — вычисляет размер файла в количестве записей того типа, который указан в объявлении файла.

Примечание

Для файла существует такое понятие, как указатель на текущую позицию. Это величина целого типа (для Borland Pascal — это величина типа **longint**), в которой хранится номер текущей позиции файла. При каждой операции чтения/записи указатель смещается на следующую запись. Под записью понимается длина типа, указанного в объявлении файла. Нумерация записей в файле начинается с нуля.

Приведенная далее программа показывает пример использования специальных файловых средств.

```
Program example;
  Uses crt;
  Var
    i,k:integer;
    f:file of integer;
begin
  clrscr;
  {Открываем файл целых чисел и записываем в него
   100 последовательных чисел}
  assign(f,'proba.dat');rewrite(f);
  for i:=1 to 100 do  write(f,i);
  reset(f);
  { Необходимо убедиться, что числа были записаны правильно.
    Для этого читаем файл до тех пор, пока не будет достигнут
    признак конца файла, на каждом шаге читаем очередное
    значение и распечатываем его. }
  while not eof(f) do
    begin
      read(f,k);write(k,'  ');
    end;
  {Обрезаем 10 последних элементов файла.}
  seek(f,filesize(f)-10);  truncate(f);  reset(f);
  writeln;  writeln;
```

```
  {Повторяем процедуру чтения и распечатки значений элементов
   файла}
  while not eof(f) do
    begin
      read(f,k);write(k,'  ');
    end;
end.
```

В дополнение приведем ряд примеров, показывающих важные особенности.

```
Program example;  Uses crt;  Var    a:record
      s:string;
      i:integer;
     end;
     f: file of record
                s:string;
                i:integer;
               end;
begin
  assign(f,'file.dat');rewrite(f);
  a.s:='fsfsfsf';
  a.i:=8;
  write(f,a);
end.
```

Структуры данных в приведенном примере определены вполне корректно, но в операторе `write(f,a);` компилятор выдаст сообщение об ошибке. А именно: компилятор сообщит, что имеет место несоответствие типов. Кажется, что типы переменных `a` и `f` одинаковы. Однако это не так с точки зрения компилятора. Мы описали две разные структуры, и компилятор справедливо полагает, что они могут быть различны, и не берет на себя заботу по проверке их одинаковости.

Указанная проблема решается следующим образом:

```
Program example;
  Uses crt;
```

```
  Type
    r=record
      s:string;
      i:integer;
     end;
  Var
    a:r;
    f: file of r;
begin
  assign(f,'file.dat');rewrite(f);
  a.s:='fsfsfsf';
  a.i:=8;
  write(f,a);
end.
```

Эта программа реализует ту же задачу, что и предыдущая, однако здесь для компилятора не возникает никаких проблем.

Файл может быть компонентом сложной структуры. Например, вполне допустим массив файлов:

```
f:array[1..10] file of integer;
```

Файл вполне может оказаться компонентом записи:

```
Program example;
  Var
    a:record
      s:string;
      f: file of integer;
     end;
begin
  assign(a.f,'file.dat');rewrite(a.f);
end.
```

Еще один интересный пример:

```
Program example;
  Uses crt;
  Type
    d=array[1..10] of integer;
```

```
  Var
    f:file of d;
    i:integer;
    a:d;
begin
  clrscr;
  for i:=1 to 5 do a[i]:=i;
  assign(f,'file.dat');rewrite(f);
  write(f,a,a);
  reset(f);
  while not eof(f) do
    begin
      read(f,a);
      for i:=1 to 10 do write(a[i],'  ');
      writeln;
    end;
end.
```

В этом примере открывается файл массивов. То есть каждая запись файла — это массив длиной в 10 целых чисел. Причем мы можем не определять значения всех десяти элементов, в файле все равно будет записано их десять, как дано в определении массива.

Один и тот же физический файл можно открыть как файл одного типа, а затем его же как файл другого типа:

```
Program example;  Uses crt;
  Var
    s:string;
     i:integer;
    f:file of string;
    d:file of integer;
begin
  clrscr;
  assign(f,'file.dat');rewrite(f);
  s:='gdgdgjagdjasg';
```

```
  for i:=1 to 10 do write(f,s);
  close(f);
  assign(d,'file.dat');reset(d);
  while not eof(d) do
    begin
      read(d,i);write(i,' ');
    end;
end.
```

В этом примере файл с именем file.dat открывается как строковый и заполняется некоторым содержимым, затем закрывается и опять открывается, но уже как файл чисел. Такие операции для языка Паскаль вполне законны, это следствие того, что на физическом уровне тип файла никак не фиксируется.

Заключение

Файл — это структура позволяющая хранить большие наборы информации. То, что файловые типы представляют собой обычные языковые типы данных, добавляет им удобства в обработке. Единственный недостаток хранения данных в файлах — это относительно низкая скорость доступа.

Указатели

Чтобы понять, что такое указатели, разберем различие между статической и динамической памятью.

Итак, предположим, дано такое определение: `var i:integer;`. Мы знаем, что в приведенной строке переменная `i` определяется как целая переменная, или, говоря более точным языком, переменная целого типа. Это означает следующее: в момент определения переменной в памяти компьютера выделяется два байта памяти, в которых будет храниться ее значение. Причем "затолкать" туда переменную большей длины (например, типа **`real`**) мы не сможем; кроме того, если потребность в переменной `i` отпадет, мы все равно не сможем использовать память,

отведенную под нее. В этих двух особенностях и заключается статичность *статической памяти.*

Динамическая память обладает прямо противоположными свойствами. Во-первых, память под переменную отводится не в момент определения, а тогда, когда в переменной возникает реальная необходимость, и во-вторых, если потребность в переменной отпадает, память, предназначенную под ее хранение, можно присоединить к свободной памяти и использовать для других данных.

Для организации такой модели памяти вводится специальная переменная, именуемая *указателем*, которая хранит адрес области памяти, предназначенной для хранения структуры данных.

В нужный момент специальной процедурой такая область будет создана, а указателю будет автоматически присвоен адрес начала области. Причем, что особенно полезно, программисту совершенно не требуется знать этот адрес, он вычисляется и присваивается указателю автоматически.

Далее указатель ведет себя так, как будто бы он и есть структура данных, хранящаяся в области, на которую он указывает. Однако здесь возникает небольшая проблема. Указатель сам по себе тоже данное, и для его хранения также требуется память. Поэтому, если мы будем создавать указатель для хранения простой переменной или константы, то память будет тратиться и на указатель, и на саму переменную. Выгодным указатель становится тогда, когда он представляет собой крупную структуру данных, например, запись с большим числом компонентов, настолько большую, что память, затраченная на хранение самого указателя, несравнима с памятью, затраченной на хранение этой записи. Кроме того, указатель позволяет создавать совершенно новую структуру данных — связные списки, но об этом будет рассказано позже, а сейчас несколько простых примеров.

Пример 1. Организуем запись с несколькими компонентами, один из которых массив, и заполним.

```
Program example;
  Type
    s=record
```

```
      f:string;
      a:array[1..10] of real;
      g:integer;
     end;
     r=^s; {r — Указатель на запись типа s}
  Var
    i:integer;
    u:r;
begin
  new(u); { Эта процедура создает новую запись }
  { Ввод компонентов записи }
  readln(u^.g);
  for i:=1 to u^.g do
    readln(u^.a[i]);
  readln(u^.f);
end.
```

Процедура **new** выполняет сразу два действия:

- ❒ отводит память под структуру данных;
- ❒ возвращает указателю адрес начала отведенной области.

Пример 2. Требуется ввести на несколько человек следующие данные: фамилия и возраст. Затем определить самого молодого из списка.

Оформим записи с помощью указателей.

```
Program example;
  Type
    s=record
      f:string;
      g:integer;
     end;
     r=^s; {r - Указатель на запись типа s}
  Var
    n,i,h:integer;
    uk:array[1..100] of r;
```

```
begin
  { Ввод данных }
  readln(n);
  for i:=1 to n do
    begin
      new(uk[i]);
      readln(uk[i]^.f);readln(uk[i]^.g);
    end;
  { Поиск самого молодого }
  h:=1;
  for i:=2 to n do
    if uk[i]^.g<uk[h]^.g then h:=i;
  { Распечатка результата }
  write(uk[h]^.f,' ',uk[h]^.g );
end.
```

Три полезные процедуры

Если у вас больше нет потребности в динамической переменной, вы можете установить указатель в никуда, присвоив ему значение **nil**, но это плохая идея. Дело в том, что, установив так указатель, вы не уничтожаете динамическую переменную, а просто теряете к ней доступ. Сама же память так и остается распределенной, а следовательно, поступив так, вы просто теряете участок памяти для дальнейшего использования. Процедура **dispose** решает эту проблему. Если вы создали динамическую переменную, вызвав процедуру с указателем в качестве аргумента, то, вызвав процедуру **dispose** с тем же аргументом, вы освободите память, занятую динамической переменной. Однако такой способ не решает всех проблем. Если в процессе работы программы динамические переменные непрерывно создаются и уничтожаются, то память вашего компьютера будет представлять собой смесь свободных и занятых участков. И может случиться так, что в один прекрасный момент программа не сможет найти свободного участка для достаточно большой структуры данных, несмотря на то, что маленьких свободных участков будет много.

К сожалению, процедура **dispose** не умеет перераспределять память и объединять свободные участки в один большой массив.

Процедуры **mark** и **release** решают проблему, с которой не справляется процедура **dispose**. Делают они свою работу так: вызов процедуры **mark** вычисляет адрес начала области свободной памяти и передает его указателю, являющемуся аргументом процедуры (например, `mark(rec)`). Выполнять процедуру **mark** следует перед первым вызовом процедуры **new**. Все последующие вызовы процедуры **new** будут выделять память под динамические переменные в указанной области. Процедура **release** освобождает всю память, занятую динамическими переменными, после чего ее опять можно использовать, но недостаток этого метода в том, что нельзя освободить память, занятую конкретной переменной, как было сказано, уничтожаются сразу все динамические переменные.

Связные списки

Использовать понятие указателя можно еще более сильным способом. Указатель позволяет разместить в памяти массив данных, не используя массив как структуру. Такого интересного результата можно добиться организацией структуры данных, именуемой связным списком. *Связным списком* называется последовательность динамических записей, одним из компонентов которых является указатель на следующую запись. Таким образом, каждая запись хранит: во-первых, полезную информацию и, во-вторых, ссылку на последующую запись. Прочитав очередную запись из такого списка, мы получаем и данные, и возможность прочитать следующую запись из списка, но, к сожалению, этот способ имеет недостаток. Для того чтобы прочитать, например, пятую запись, необходимо прочитать первую, вторую, третью и четвертую. Далее приведен текст программы, в которой создается и заполняется связный список, а затем осуществляется сквозной проход через весь список.

```
Program example;
  Type
    ukaz=^zapis;
    zapis=record
```

```
          number:integer;
          nextukaz:ukaz;
        end;
  Var
    spisok,temp:ukaz;
    i:integer;
begin
  { Создание связного списка }
  new(spisok);temp:=spisok;
  for i:=1 to 5 do
    begin
      spisok^.number:=i;
      new(spisok^.nextukaz);
      spisok:=spisok^.nextukaz;
    end;
  { Сквозной проход связного списка }
  spisok:=temp;
  for i:=1 to 5 do
    begin
      writeln(spisok^.number);
      spisok:=spisok^.nextukaz;
    end;
end.
```

Еще одна интересная проблема

Дело в том, что функция по определению не может вернуть более чем одно значение, значение типа **record** также не возвращается функцией. Однако, применяя совместно понятия записи и указателей, эту проблему решить можно. Идея следующая:

1. Сформируем внутри функции запись.
2. Найдем адрес этой записи и вернем его в качестве вычисленного значения функции.
3. Возвращаемое значение присвоим указателю, после чего значения записи становятся доступны за пределами функции.

```
Program example;
  Uses crt;
  Type
    s=record
      d,d1:integer;
      f:string;
     end;
     d=^s;
  Var
    q:d;
    w:s;
    function proba(a:s):d;
    begin
      a.d:=45;proba:=addr(a); {Здесь вычисляется адрес
записи}
    end;
begin
  clrscr;
  w.f:='fgfgfgfgf';
  q:=proba(w);
  writeln(q^.f,' ',q^.d);
end.
```

Указатель на тип

Во всех предыдущих примерах мы использовали указатели, связанные с переменными, имеющими определенные имена. Это не обязательно. Можно создать указатель, связанный просто с типом данных, без указания имени структуры данных, на которую он будет указывать. Далее приведен пример, как это сделать:

```
Program example;
  Uses crt;
  Type
    pointer=^integer;
```

```
  Var
    a,b:pointer;
    i:integer;
begin
  clrscr;
  new(a);b:=a;
  for i:=1 to 100 do
    begin
      a^:=i; inc(a);
    end;
  for i:=1 to 100 do
    begin
      write(b^,' ');
      inc(b);
    end;
end.
```

В этом примере указатель настраивается на тип **integer**, и в программе фактически создается массив большого размера без объявления массива как типа данных. Однако таким приемом следует пользоваться осторожно, результат приведенного примера сильно зависит от состояния памяти.

Заключение

Динамическая память — это, образно говоря, способ распределения памяти по ситуации. С использованием динамической памяти у нас отпадает необходимость подсчитывать, сколько тех или иных переменных может понадобиться. Это, конечно, очень удобно, но необходимо помнить о некоторых существенных минусах.

- Статическая память хороша тем, что многие ошибки с распределением памяти мы можем обнаружить на этапе компиляции. Так как динамические переменные создаются уже в процессе работы, то и обнаружение ошибок также приходится на этап работы программы, что осложняет ее отладку.

- Перераспределение памяти само по себе может оказаться довольно сложной алгоритмической проблемой. Накладываясь на собственные сложности решаемой задачи, проблема перераспределения может слишком сильно усложнить решение.
- Паскаль не предоставляет средств для собирания вместе свободных областей памяти и может так оказаться, что при наличии большого количества небольших свободных кусков некуда будет разместить большую динамическую структуру.

Однако понятие указателя в сочетании с другими возможностями языка находит себе неожиданное и очень полезное применение. Например, расширяются возможности такой конструкции, как функция, для которой, как мы видели, можно организовать возврат нескольких величин, что невозможно сделать обычными средствами языка. Возможно также организовать такую сложную структуру, как связный список.

Глава 2

Организация диалога с программой

В данной главе мы рассмотрим средства взаимодействия пользователя и программы. Прежде всего изучим набор команд ввода-вывода. Этот набор невелик, и правила записи не сложны, но организовать диалог так, чтобы он был понятен пользователю и позволял ему избегать ошибок на этапе ввода данных, достаточно непросто. Кстати, правильное понимание результатов работы программы — тоже задача не слишком тривиальная. Вот этим всем мы сейчас и займемся. Тема, между прочим, ключевая, без умения организовать ввод-вывод мало что можно сделать.

Задача 1. Организовать ввод двух целых чисел и вывод их суммы так, чтобы и ввод, и вывод сопровождался поясняющими фразами.

```
Program example;
  Uses crt;
  Var
    a,b,sum:integer;
begin
  write('Введите первое число');readln(a);
  write('Введите второе число');readln(b);
  sum:=a+b;
  write('Сумма =',sum);
end.
```

Пояснения

Оператор **write** выводит на экран сообщение, но не переводит курсор на следующую строчку. В результате ввод числа будет осуществляться в той же строке, в которой находится поясняющая фраза.

Задача 2. Ввести три числа и затем вывести их на экран монитора следующими способами.

- В одной строке, так чтобы они были отделены друг от друга пробелами.
- В столбец, так чтобы они были отделены друг от друга пустыми строками.
- Разбросать числа по экрану в беспорядке.

```
Program example;
  Uses crt;
  Var
    a,s,d:integer;
begin
  read(a,s,d);clrscr;
  write(a,' ',s,' ',d);delay(10000);clrscr;
  writeln(a);writeln;
  writeln(s);writeln;
  writeln(d);delay(10000);clrscr;
  gotoxy(10,10);write(a);
  gotoxy(12,24);write(s);
  gotoxy(34,5);write(d);
  delay(10000);clrscr;
end.
```

Пояснения

Процедура **delay** производит задержку выполнения программы на количество миллисекунд, указанное в скобках. Правда, на современных компьютерах и под современные операционные системы это, видимо, уже не так.

Пустой оператор **writeln** ничего не распечатывает, а просто переводит курсор на новую строку. Таким образом, последовательность операторов `writeln(a);writeln;` осуществляет два последовательных перевода курсора на новую строку. В результате одна строка оказывается пустой.

Процедура **gotoxy** устанавливает курсор в том месте экрана, координаты которого указаны в скобках. Необходимо помнить, что эта процедура не работает в графическом режиме.

Задача 3. Вводится три числа *A*, *X*, *Y*. Вывести значение *A* на экране монитора в координатах *X*, *Y*.

Подсказки

Необходимо воспользоваться процедурой **gotoxy**, а в качестве ее аргументов взять величины *X*, *Y*.

Задача 4. Как организовать ввод чисел, чтобы запрос на очередное число осуществлялся в центре экрана монитора? Пусть вводится 10 чисел.

```
Program example;
  Uses crt;
   Var
     i,k:integer;
begin
  for i:=1 to 10 do
    begin
      clrscr; gotoxy(40,12);read(k);
    end;
end.
```

Пояснения

После ключевого слова **do** необходимо выполнить три команды, а Паскаль разрешает только одну. Поэтому эти три команды объединены в сложный оператор ключевыми словами **begin end**.

Процедура **clrscr** — это необходимая очистка экрана, т. к. в противном случае ввод нового числа будет производиться поверх предыдущего, чем затруднит восприятие вводимого нового числа.

Задача 5. Вывести на экран рамку, построенную из знаков "*".

Подсказки

Необходимо построить четыре линии из символов "*". Это можно выполнить с помощью четырех циклических процессов, каждый из которых будет выводить одну линию. А можно все линии создать и в одном цикле.

Устанавливать курсор в нужную позицию можно процедурой **gotoxy**.

В качестве примера напишем программу, устанавливающую одну вертикальную линию.

```
Program example;
  Uses crt;
   Var
     i: integer;
begin
  clrscr;
  for i:=1 to 10 do
    begin
      gotoxy(40,5+i);write('*');
    end;
end.
```

Пояснения

В данном примере вертикальная линия устанавливается в позицию по $X = 40$. По вертикальной оси линия начинается с шестой позиции и печатается 10 символов.

Задача 6. Организовать вывод ста последовательных чисел (1, 2, 3, ..., 100) так, чтобы все они выводились в центре экрана через определенный временной промежуток.

Подсказки

Задача совершенно аналогична задаче 4. Разница заключается лишь в том, что в задаче 4 организуется ввод чисел, а здесь речь идет о выводе.

Временную задержку на определенный интервал можно организовать с помощью процедуры **delay**.

Задача 7. Организовать движение курсора по экрану (влево, вправо) так, чтобы курсор двигался под управлением соответствующих клавиш.

```
Program example;
  Uses crt;
  Var
   x,d:integer;
    c:char;
procedure move;
begin
      c:=readkey;
      d:=ord(c);
       if d=0 then d:=ord(readkey);
       case d of
       77: if  x<79 then x:=x+1;
       75: if x>1 then x:=x-1;
       end;
       gotoxy(x,10);
    end;
begin
  d:=0;x:=1;gotoxy(x,10);
  while d<>13 do
    begin
      move;
    end;
end.
```

Пояснения

С помощью функции **readkey** выясняется, была ли нажата какая-либо клавиша. Функция **ord** определяет код нажатой клавиши.

Клавиши управления курсором имеют двойной код. Первый код — это ноль. Поэтому если функция **ord** выдала значение ноль, то операцию опроса клавиатуры следует повторить для получения второго кода.

Конструкция **case** передает управление на альтернативу, на которую указывает значение `d`. А т. к. `d` имеет значение кода клави-

ши управления курсором, то, следовательно, если была нажата клавиша влево, значение x уменьшается на 1, и, наоборот, если была нажата клавиша вправо, значение x увеличивается на 1.

Процедура **gotoxy** помещает курсор в новую позицию.

Переменная x может изменяться только в определенных пределах (в программе от 1 до 80). Поэтому, прежде чем изменять x, программа в операторах **if** проверяет, не выходит ли переменная x за допустимые пределы.

В этой программе все операции обработки движения выполняются в процедуре, что не обязательно, рассматривайте это как пример простейшей организации процедуры.

Задача 8. Организовать ввод строки символов так, чтобы каждый вводимый символ отображался на экране, и при этом не нажималась клавиша ввода. Ввод завершается по нажатию клавиши <Esc>.

```
Program example;
  Uses crt;
  Var
    b:char;
    i,k:integer;
begin
  i:=0;
  repeat
    b:=readkey;
    k:=ord(b);
    if k<>27  then
      begin
        i:=i+1;
       gotoxy(i,10);
       write(b);
     end;
  until k=27;
end.
```

Пояснения

Ввод осуществляется с помощью функции **readkey**. Эта функция срабатывает сразу после нажатия клавиши, без нажатия на клавишу <Enter> (в отличие от **read**).

Функция **ord** определяет код введенного символа, и если он не равен 27 (код клавиши <Esc>), то значение введенного символа тут же выводится на экран монитора в очередной *i*-ой позиции.

Все операции по вводу очередного символа выполняются внутри цикла по условию, который завершается, когда переменная `k` получает значение 27 (код клавиши <Esc>).

Задача 9. Пусть в программе выполняется циклическая операция по вычислению какой-либо величины и выводу ее на экран. Как сделать так, чтобы ее выполнение прерывалось нажатием клавиши <Esc>?

Подсказки

Цикл можно организовать с помощью конструкции цикла по условию. Причем условием его прекращения будет нажатие на клавишу <Esc>.

Внутри конструкции цикла необходимо организовать опрос клавиатуры, например, так, как это сделано в предыдущей задаче.

Задача 10. Пусть в программе организовано какое-либо меню (произвольное). От выбора пункта меню зависит, какое действие будет выполняться, пункты меню пронумерованы. Как сделать так, чтобы выбор пункта осуществлялся простым нажатием клавиши с цифрой?

Подсказки

Пункты меню можно распечатать любыми процедурами вывода. В качестве таких пунктов можно выбрать определенные действия, например, распечатку каких-либо сообщений.

После вывода меню на экран опрашивается клавиатура и в зависимости от того, какая клавиша была нажата, производится выбор.

Выбор можно производить с помощью конструкции **case**. Покажем на примере, как это делать. Предположим, что с клавиатуры может быть либо получен символ "1", либо символ "2". Если нажата <1>, то вывести сообщение: "Нажата клавиша 1"; если же нажата

клавиша <2>, то вывести сообщение: "Нажата клавиша 2". Программный фрагмент будет выглядеть так:

```
case k of
  '1' : write('Нажата клавиша 1');
  '2' : write('Нажата клавиша 2');
end;
```

Переменная k должна быть обязательно символьного типа.

Задача 11. Пусть в программе выполняется некоторый бесконечный цикл. Как сделать так, чтобы при нажатии любой клавиши начиналось выполнение произвольной операции (например, печать символа на экране монитора) и действие прекращалось, когда клавиша отжимается?

```
Program example;
 Uses crt;
 Var
  a,h:char;
begin
  clrscr;
  h:='g';
  repeat
    if keypressed then
       while keypressed do
         begin
          write('d');
          h:=readkey;
          end;
  until h='j';
end.
```

Пояснения

Цикл **repeat until** выполняется бесконечно, т. к. переменная a в программе никогда не изменяется и, следовательно, всегда равна символу j. Прерваться же цикл может только тогда, когда a станет неравно j.

Переменная **keypressed** — это логическая переменная, которая истинна, если нажата какая-либо клавиша, и ложна в противоположном случае.

Следовательно, если будет нажата клавиша, то программа начнет печатать символ 'd'. В этом же сложном операторе записана функция **readkey**, считывающая символ или, иначе говоря, забирающая символ из буфера клавиатуры. Поэтому если мы перестанем нажимать на клавишу, то за один шаг оператора **while do** функция **readkey** очистит буфер клавиатуры, и переменная **keypressed** примет значение "ложь"; после чего выполнение цикла **while do** прекратится, и программа перестанет печатать символ 'd'.

По нажатию на клавишу `j` работа программы прекращается.

Задача 12. Пусть внутри некоего бесконечного цикла вычисляется какая-либо величина. Как сделать так, чтобы ее значение распечатывалось по нажатию (и только в этом случае) на клавишу <Esc>? Причем одиночное нажатие должно вызывать печать только одного значения.

Подсказки

После каждой расчетной операции необходимо проверять, нажата или нет какая-либо клавиша, и если да, то распечатывать полученное значение и очищать буфер клавиатуры.

Задача 13. Пусть в программе организовано два циклически выполняемых действия. Например, печать символа в строку и печать символа с переводом на новую строку. Как организовать переход управления программой с одного действия на другое по нажатию на любую клавишу?

Подсказки

Если действий выполняются циклически, то, следовательно, они организованы посредством конструкции цикла. Так как мы не знаем, как долго выполнять действия, используем цикл по условию.

Так как действий два, то циклов также будет два. Кроме того, оба цикла должны находиться внутри некоторого внешнего цикла по условию.

Таким образом, структура программы будет примерно такая:

```
Пока верно условие1, делать
  Начало
    Повторять
      Действие1
    Пока не станет истинным  условие3
    Очистка буфера клавиатуры
    Повторять
      Действие2
    Пока не станет истинным  условие5
    Очистка буфера клавиатуры
  Конец внешнего цикла
```

Задача 14. Пусть в программе выполняются два циклических действия. Как сделать так, чтобы по завершению первой циклической операции программа зависала и ждала нажатия любой клавиши, после чего выполняла вторую циклическую операцию? Циклически выполняемые действия организовать в виде процедуры с одним параметром. Параметр этой процедуры может быть символом, предназначенным для вывода на экран.

Подсказки

Определить, нажата клавиша или нет, можно с помощью переменной **keypressed**. Она принимает значение "истина", если нажата любая клавиша, и "ложь", если ничего не нажато.

Так как переменная **keypressed** логическая, то ее имя можно использовать в качестве условия; в предыдущих примерах это уже демонстрировалось.

Решение поставленной задачи заключается в организации пустого цикла, в условии окончания которого должна проверяться истинность или ложность переменной **keypressed**.

Тело процедуры будет состоять из оператора цикла, внутри которого имеется один оператор вывода.

Задача 15. Как организовать ожидание из условия предыдущей задачи с той разницей, что программа ожидает нажатия не любой клавиши, а конкретной?

Подсказки

Задача похожа на предыдущую, но использованный ранее прием не годится. Переменная **`keypressed`** не выясняет, что именно было нажато, а просто констатирует факт, нажата клавиша или нет.

Для данной задачи хорошо подходит функция **`readkey`**, т. к. она устанавливает код, по которому можно определить нажатую клавишу.

Фрагмент программы, обладающий требуемым свойством, будет представлять собой цикл по условию, внутри которого производится опрос клавиатуры, а выход из цикла происходит тогда, когда полученный код соответствует коду ожидаемой клавиши.

Задача 16. Организовать ввод символов так, чтобы вводились только разрешенные символы, например, только цифры. То есть если вводимый символ цифра, он отображается на экране, иначе нет. Для определения допустимости символа разработать специальную функцию, которая должна возвращать истину при допустимом символе и ложь в противоположном случае.

Подсказки

Попытка ввода очередного символа осуществляется до тех пор, пока не будет введен разрешенный. Следовательно, ввод осуществляется по следующему простому алгоритму: пока не введен допустимый символ — вводить символ. Это можно реализовать с помощью любой формы цикла по условию.

Цикл ввода одного символа повторяется до тех пор, пока не будет введено определенное количество символов; для такого процесса хорошо подходит цикл по параметру. Следовательно, программа представляет собой два вложенных цикла. Внешний цикл по параметру обеспечивает ввод множества символов, и внутренний по условию будет выполнять ввод символов до тех пор, пока мы не введем допустимый.

Непосредственно для ввода символа можно использовать какую-либо функцию ввода символа.

Задача 17. Как организовать управляемое движение символа? Управление движением осуществлять клавишами управления курсора.

Подсказки

В задаче 7 реализовано движение курсора по горизонтали. Для решения данной задачи достаточно в задачу 7 добавить (в конструкцию `case`) еще два пункта: движение вниз и движение вверх.

Для определения кода клавиш <↑> и <↓> можно воспользоваться программой, приведенной далее.

```
Program example;
  Uses crt;
  Var
    k: integer;
begin
  clrscr;
   k:=0;
   while k<>27 do
      begin
         k:=ord(readkey);
          writeln(k);
       end;
end.
```

Здесь 27 — это код клавиши <Esc>, поэтому выполнение программы прекращается по нажатию клавиши <Esc>. Обратите внимание, что клавиши управления курсором имеют двойной код.

Задача 18. Вывести на экран таблицу умножения.

Подсказки

Для вывода таблицы необходимо два цикла (можно по параметру). Внешний цикл пробегает все строки таблицы, а внутренний выводит все элементы текущей строки. В качестве элементов таблицы можно взять произведение параметров внешнего и внутреннего циклов.

После вывода очередной строки необходимо выполнить процедуру, переводящую курсор на новую строку. В Паскале такой процедурой может быть **writeln**.

Задача 19. Организовать ввод множества целых чисел таким образом, чтобы:

- после ввода каждого числа курсор переводился на новую строку;
- перед вводом каждого нового элемента на той строке, на которой будет вводиться элемент, печаталось значение его индекса с круглой скобкой. Например, ввод пятого элемента будет выглядеть следующим образом: `5) _` (после номера одна позиция — это пробел).

Подсказки

Оператор вывода, решающий нашу задачу, будет содержать три компонента: номер элемента, символьную строку со скобкой и сам распечатываемый элемент.

Задача 20. Организовать вывод множества целых чисел таким образом, чтобы:

- после вывода каждого числа курсор переводился на новую строку;
- перед выводом значения элемента выводилось его символьное обозначение. Например, вывод пятого элемента будет выглядеть так: `a(5)=12`.

Подсказки

Задача представляет собой немного усложненный вариант предыдущей задачи.

Задача 21. Вывести таблицу квадратов натуральных чисел с рамкой, построенной из символов псевдографики.

Подсказки

В этой таблице необходимы две колонки. Колонка с последовательными натуральными числами и вторая колонка с квадратами

этих чисел. Символы псевдографики — это следующие символы: ¬L -+ +T + + - ¦. К ним можно обращаться с помощью функций, обеспечивающих доступ к символам по их коду в таблице ASCII.

Задача 22. Создать меню с подсветкой пунктов при движении по ним курсора.

```
Program example;
 Uses crt;
 Var
  x,c:integer;
procedure move(color:integer);
begin
  textbackground(color);
  gotoxy((x-1)*8+1,1);
  case x of
   1:write('File');
   2:write('Edit');
   3:write('Search');
   4:write('Run');
  end;
end;
begin
 clrscr;
 write('File    Edit    Search  Run');
 x:=1;gotoxy(x,1);
 textbackground(2);
 write('File');
 repeat
  c:=ord(readkey);
  if c=0 then c:=ord(readkey);
  move(0);
  case c of
   77: if x<4 then x:=x+1;
   75: if x>1 then x:=x-1;
  end;
```

```
  move(2);
 until ord(c)=13;
end.
```

Пояснения

При переходе к новому пункту есть необходимость перепечатать текущий пункт с черным фоном и только затем перепечатать новый пункт с подсветкой. Здесь идеально подходит процедура, в которую просто передается цвет подсветки. Нулевой цвет — это тоже цвет.

Задача 23. Организовать двухуровневое меню. То есть меню, состоящее из нескольких пунктов, при обращении к которым появляется меню второго уровня. Подобное меню — достаточно обычная конструкция. Таким является (и даже немного более сложным) меню среды компилятора, которым мы пользуемся.

Подсказки

Перемещение по меню первого уровня должно производиться посредством клавиш управления курсором. Положение меню второго уровня, конечно, должно быть привязано к выбранному пункту меню первого уровня.

Задача 24. Построить модель змеи, перемещающейся с помощью клавиш управления курсора. Змея движется, если клавиша нажата. Длина змеи фиксирована. Змея состоит из букв "о".

```
Program example;
  Uses crt;
  Var
    s,xx,yy,i,d:integer;
    x,y:array[1..10] of integer;
    c:char;
begin
  d:=1;s:=0;
  textbackground(0);
  textcolor(15);
  clrscr;
```

```
for i:=1 to 10 do
 begin
  x[i]:=i+10;y[i]:=10;
  gotoxy(i+10,10);write('o');
 end;
gotoxy(x[1],y[1]);
while d<>13 do
 begin
  c:=readkey;
  d:=ord(c);
  if d<>0 then
   begin
    if s=0 then
     begin
      gotoxy(x[10],y[10]);
      write(' ');
     end
  else
   begin
    gotoxy(xx,yy);
    write(' ');
   end;
  s:=0;
  xx:=x[10];yy:=y[10];
  for i:=10 downto 2 do
   begin
    x[i]:=x[i-1];
    y[i]:=y[i-1];
   end;
  if (d=77) and (x[1]<79) then x[1]:=x[1]+1;
  if (d=75) and (x[1]>1) then x[1]:=x[1]-1;
  if (d=72) and (y[1]>1) then y[1]:=y[1]-1;
  if (d=80) and (y[1]<24) then y[1]:=y[1]+1;
  gotoxy(x[1],y[1]);
```

```
    write('o');
  end;
 end;
end.
```

Пояснения

Основой для данной задачи является задача, в которой было необходимо организовать управляемое движение курсора. Разница заключается в том, что перемещать нужно не один объект, а целую группу.

Принципиально по-разному перемещается голова змеи. Голова движется в направлении, которое определяется клавишами управления курсором. Тело движется проще. Каждый "нолик" встает на место следующего.

Задача 25. Перемещение змеи. От предыдущей задачи движение отличается тем, что нажатие клавиши определяет направление движения, а само движение продолжается и после того, как клавиша отпущена.

Подсказки

От предыдущей задачи отличие в способе опроса клавиатуры. В задаче 24 на каждом шаге большого цикла программа ожидает нажатия клавиши. Здесь же ожидание не нужно. На каждом шаге большого цикла необходимо только проверять, нажато что-нибудь или нет. И, только если нажато, проверять что именно.

Движение же змеи должно осуществляться независимо от сигналов с клавиатуры.

Задача 26. Перемещение змеи. От предыдущей задачи имеется следующее отличие: на экране выкладывается добыча, координаты которой определяются случайным образом. Цель играющего добраться до угощения и съесть его, после чего длина змеи увеличивается на одну букву "o".

Подсказки

Для выкладки обеда необходимо только вычислить его координаты, а это два случайных числа.

Далее на каждом шаге змеи необходимо проверять, равны ли координаты ее головы координатам обеда и, если да, увеличить длину змеи и добавить в ее тело еще одну букву "о". Ясно, что в этой программе должна появиться еще одна переменная для хранения змеиной длины.

Задача 27. Дана большая таблица квадратов целых чисел. Таблица состоит из двух колонок, самих чисел и их квадратов. Таблица выводится постранично, переход между страницами осуществляется клавишами управления курсором: вверх и вниз.

Задача 28. Дана большая таблица квадратов целых чисел. Таблица состоит из трех пар колонок, самих чисел и их квадратов. Таблица выводится постранично, переход между страницами осуществляется клавишами управления курсором: вверх и вниз.

```
Program example;
 Uses crt;
 Var
  m,c:integer;
procedure table(m:integer);
var
 i:integer;
 n:longint;
begin
 clrscr;
 for i:=m*20+1 to m*20+20 do
   begin
    n:=i;
    writeln(i,'   ',n*n);
   end;
end;
begin
 m:=0;
 table(m);
 repeat
```

```
  c:=ord(readkey);
  case c of
   72: if m>0 then
       begin
         m:=m-1;
         table(m);
       end;
   80: begin
         m:=m+1;
         table(m);
       end;
  end;
 until c=27;
end.
```

Пояснения

Необходимо пояснить смысл выделенной команды. Если обойтись без нее и выводить просто произведение `i*i`, то мы быстро выйдем за пределы целого типа. Если же написать команду `n:=i*i`, то справа все равно будет выражение целого типа, которое не преобразуется в тип "длинное целое". Хотя, конечно, можно было просто все величины объявить как длинное целое.

Задача 29. Дана большая таблица квадратов целых чисел. Таблица состоит из двух колонок, самих чисел и их квадратов. Перемещение по таблице не постраничное, а плавное, когда курсор доходит до верхней границы, таблица смещается на одну позицию вниз, и наоборот, когда курсор доходит до нижней границы, таблица смещается на одну позицию вверх. Пользователь управляет курсором с помощью клавиш управления курсором.

Подсказки

В предыдущей задаче просто перепечатывается вся страница. Здесь, как кажется, этого делать не надо. Однако эта задача также решается перепечатыванием всей страницы, разница в величине смещения. Здесь нам смещение нужно не на величину страницы, а только на одну позицию. Видимо, главная программа в обоих задачах выглядит одинаково.

Задача 30. Дана большая таблица квадратов целых чисел. Таблица состоит из большого количества пар колонок. Переход между парами колонок осуществляется посредством клавиш управления курсором. При достижении левой или правой границы таблицы таблица смещается в соответствующую сторону на одну пару колонок.

Подсказки

Эта задача отличается уже существенно. Таблицу так же, как и раньше, необходимо перепечатывать, но сама таблица меняет вид, и интервал распечатываемых значений изменяется скачком на размер одной пары колонок.

Распечатку таблицы можно организовать в виде двух вложенных циклов. Внешний может пересчитывать пары колонок, а внутренний — элементы очередной пары.

Глава 3

Статическая графика

Пожалуй, самая легкая тема. Во-первых, мы будем в каждой задаче получать легко наблюдаемый результат, что сильно упрощает понимание, а во-вторых, графика в программировании чаще всего играет вспомогательные роли, поэтому потребность в сложных графических алгоритмах не очень велика. Особенно, если говорить о графике статической.

Задача 1. Нарисовать 20 вертикальных отрезков в ряд, используя конструкцию цикла.

```
Program example;
  Uses graph;
   Var
      i,driver,mode:integer;
begin
  driver:=detect;initgraph(driver,mode,'');
   for i:=1 to 20 do
     line(20*i,100,20*i,200);
end.
```

Пояснения

Обе координаты *X* имеют одинаковое значение, изменяющееся вместе с изменением параметра цикла. За счет этого на каждом шаге рисуется отрезок на 20 точек правее предыдущего отрезка.

Задача 2. Нарисовать ряд из окружностей (10 окружностей) так, чтобы соседние окружности касались одной точкой, и каждая окружность закрашивалась в другой цвет.

Подсказки

Задача очень похожа на предыдущую, только вместо отрезков рисуются окружности.

Для касания окружностей в одной точке необходимо, чтобы их центры находились на одной линии и отстояли друг от друга на величину диаметра.

В качестве подсказки напишем программу, рисующую три окружности радиуса 20 с центрами, отстоящими друг от друга по прямой на 100 точек, и закрашенных белым цветом.

```
Program example;
  Uses graph;
   Var
      i,driver,mode:integer;
begin
  driver:=detect;initgraph(driver,mode,'');
   setfillstyle(1,15);
   setcolor(15);
   for i:=1 to 3 do
     begin
       circle(100*i,200,20);
        floodfill(100*i,200,15);
      end;
end.
```

Задача 3. Нарисовать квадрат.

```
Program example;
  Uses graph;
   Var
     driver,mode:integer;
begin
  driver:=detect;
```

```
  initgraph(driver,mode,'');
  line(100,100,200,100);line(200,100,200,200);
  line(200,200,100,200);line(100,200,100,100);
end.
```

Пояснения

В программе рисуется четыре линии одинаковой длины (100 точек). И каждая линия начинается в точке, в которой заканчивается предыдущая.

Задача 4. Нарисовать квадрат и окружность, касающуюся всех сторон квадрата.

```
Program example;
  Uses graph;
   Var
     driver,mode:integer;
begin
  driver:=detect;
  initgraph(driver,mode,'');
  line(100,100,200,100);line(200,100,200,200);
  line(200,200,100,200);line(100,200,100,100);
  circle(150,150,50);
end.
```

Пояснения

Для того чтобы окружность касалась сторон квадрата, необходимо, чтобы ее центр совпадал с центром квадрата, и радиус был равен половине длины стороны.

Задача 5. Нарисовать пирамиду (наверное, точнее было бы назвать это снеговиком) из десяти окружностей, каждую закрасить своим цветом. Получившиеся круги могут пересекаться.

Подсказки

Для организации процесса рисования окружностей хорошо подойдет цикл по параметру.

Если самая большая окружность находится внизу и мы рисуем от большей к меньшей, то значение координаты *Y* должно уменьшаться с ростом параметра цикла. Так же должен уменьшаться и радиус.

Следите за тем, чтобы радиус уменьшался не слишком быстро. Если величина, на которую уменьшается радиус, будет слишком велика, и радиус станет нулевым или отрицательным, то программа может сработать неправильно.

После того как очередная окружность нарисована, необходимо ее закрасить. Для этого внутри цикла потребуется сложный оператор, содержащий два действия: рисование очередной окружности и ее закраску.

Задача 6. Нарисовать пять квадратов, расположенных друг относительно друга по диагонали так, чтобы правая нижняя вершина предыдущего квадрата совпадала с левой верхней вершиной последующего.

Подсказки

Процесс рисования квадратов — это, очевидно, циклический процесс, в ходе которого каждый квадрат рисуется правее на величину стороны предыдущего квадрата и на величину стороны ниже.

Для того чтобы точку нарисовать правее предыдущей, надо к ее координате *X* прибавить величину смещения. Аналогично для рисования ниже надо также прибавить величину смещения, но уже к координате *Y*.

Задача 7. Нарисовать горизонтальный отрезок, пользуясь только процедурой установки точки.

```
Program example;
  Uses graph;
  Var
    driver,mode,i:integer;
begin
  driver:= detect;
  initgraph(driver,mode,'');
  for i:=1 to 200 do
    putpixel(10+i,200,15);
end.
```

Пояснения

Координата *Y* является константой, следовательно, все рисуемые точки будут находиться на одной горизонтали.

Координата *X* от точки к точке изменяется на единицу, следовательно, все точки будут рисоваться вплотную друг к другу, что и создаст эффект сплошного отрезка.

Задача 8. Вывести текстовое сообщение в произвольном месте экрана.

```
Program example;
  Uses graph;
  Var
    driver,mode:integer;
begin
  driver:= detect;
  initgraph(driver,mode,'');
  outtextxy(100,200,'Выводим сообщение в графическом
режиме');
end.
```

Пояснения

Несмотря на то, что сообщение представляет собой текст, координаты используются графические.

Задача 9. Вывести сообщение в произвольном месте и заставить его переливаться разными цветами.

```
Program example;
  Uses crt,graph;
  Var
    driver,mode,i:integer;
begin
  driver:= detect;
  initgraph(driver,mode,'');
  for i:=1 to 15 do
    begin
      setcolor(i);
```

```
      outtextxy(100,200,'Выводим сообщение в графическом
                режиме');
      delay(10000);
    end;
end.
```

Пояснения

Смысл программы заключается в том, что одно и то же сообщение выводится в одном и том же месте экрана, но на каждом шаге цикла меняется цвет, поэтому сообщение каждый раз печатается новым цветом.

Процедура **delay** организует временную задержку, для того чтобы все разноцветные сообщения не промелькнули слишком быстро.

Задача 10. Нарисовать систему координат и около делений проставить числа.

Подсказки

Программа состоит из трех частей. Во-первых, надо нарисовать оси, представляющие собой две линии. Во-вторых, необходимо проставить деления на осях через определенный интервал. В-третьих, около делений проставить числа.

В качестве главной подсказки напишем программу, рисующую отрезок, проставляющую на нем деления и числа возле них.

```
Program example;  Uses crt,graph;  Var
i,driver,mode:integer;   s:string;begin
driver:=detect;  initgraph(driver,mode,'');
line(100,200,400,200);  for i:=1 to 11 do
    begin
     line(100+30*(i-1),198,100+30*(i-1),202);
     str(i,s);
     outtextxy(98+30*(i-1),190,s);
    end;
end.
```

Процедура **line** внутри цикла рисует небольшие вертикальные отрезки.

Задача 11. Нарисовать лестницу из отрезков.

Подсказки

Лестница состоит из ступенек. Ступенька — это две линии, одна горизонтальная и одна вертикальная. Каждая ступенька отстоит от предыдущей на длину линии вправо и на длину линии вниз.

Обозначим длину линии через *L*. Тогда, чтобы получить новую ступеньку, достаточно сместить все координаты в процедурах `line` на величину *L* вправо и на величину *L* вниз.

Если же двух подсказок окажется недостаточно, то можно еще посмотреть шестую задачу, она аналогична данной.

Задача 12. Нарисовать незакрашенную шахматную доску 8×8.

Подсказки

Шахматная доска состоит из девяти равноотстоящих вертикальных линий и девяти равноотстоящих горизонтальных линий. Как рисуются равноотстоящие вертикальные линии, можно посмотреть в задаче 1. Равноотстоящие горизонтальные линии рисуются аналогично. Подогнать их так, чтобы получилась доска, а не набор линий, можно и экспериментальным путем.

Задача 13. Нарисовать шахматную доску, если из графических процедур разрешается пользоваться только процедурой установки точки. Для решения задачи создать процедуру, рисующую закрашенный квадрат в заданных координатах.

Подсказки

Заметим, что шахматная доска состоит из горизонтальных полос квадратиков. Каждая полоса состоит из закрашенных квадратов, квадрат из горизонтальных отрезков, горизонтальный отрезок из точек. Отсюда следует простая вещь — достаточно нарисовать один отрезок. Затем его многократное повторение даст квадрат, многократное повторение квадрата даст полосу, многократное повторение полосы даст шахматную доску.

Цвета квадратов в процессе рисования должны меняться от квадрата к квадрату. Чтобы сделать так, необходимо ввести дополнительную переменную, которая будет менять свое значение после каждого нарисованного квадрата.

В помощь напишем программу, выполняющую два уровня работы: рисование отрезка из точек и рисование квадрата из отрезков.

```
Program example;
 Uses graph;
 Var
  dr,md,x,y:integer;
begin
 dr:=detect;initgraph(dr,md,'');
 for y:=1 to 40 do
  for x:=1 to 40 do
   putpixel(x+50,y+50,15);
end.
```

Теперь осталось эти два цикла повторить 8 раз для получения полосы квадратов, затем уже три цикла повторить еще 8 раз для получения доски и затем подумать о перемене цвета от квадрата к квадрату и о том, как часть программы оформить в виде процедуры.

Задача 14. Нарисовать N концентрически сходящихся разноцветных окружностей.

Подсказки

Задача аналогична задаче 5. Различие между ними в том, что в данной задаче координаты центра не смещаются.

Задача 15. Дан произвольный прямоугольник, заданный координатами начальной и конечной точек одной из диагоналей. Требуется закрасить этот прямоугольник произвольным цветом, используя только процедуру рисования линии `line`.

Подсказки

То, что прямоугольник задан координатами вершин диагонали, означает, что нам известны четыре величины: $x1$, $y1$ — координаты левого верхнего угла и $x2$, $y2$ — координаты правого нижнего угла. Если точка расположена внутри прямоугольника, то ее координаты удовлетворяют следующим неравенствам: $x1 < X < x2$ и $y1 < Y < y2$.

Можно сказать, что прямоугольник состоит из отрезков с координатой *X* начальной точки, равной *x*1, и координатой *X* конечной точки, равной *x*2. Точки самого верхнего отрезка имеют координату *Y*, равную *y*1; и точки самого нижнего отрезка имеют координату *Y*, равную *y*2.

Таким образом, чтобы закрасить прямоугольник, достаточно провести все горизонтальные отрезки с координатами: *x*1, *x*2 и *y*1 < < *Y* < *y*2.

Задача 16. Предположим, что в квадрате с диагональю (0, 0)—(200, 200) нарисована закрашенная окружность, требуется скопировать эту окружность в произвольно расположенный квадрат того же размера при условии, что пользоваться процедурами рисования и закрашивания окружности нельзя. Оба квадрата расположены так, что одна пара сторон параллельна одной стороне экрана, а вторая пара — второй стороне экрана.

Подсказки

Если мы сможем узнать цвет каждой точки внутренней области квадрата, то мы сможем установить во втором квадрате точку с соответствующими координатами такого же цвета.

Узнать цвет точки с заданными координатами можно с помощью процедуры `getpixel(x,y)`. `x`, `y` — это координаты точки, цвет которой определяется.

Пробежать все точки внутренней области исходного квадрата можно с помощью двух вложенных циклов. Один цикл по координате *X*, и второй по координате *Y*.

Процесс копирования легко представить себе наглядно. Представьте себе, что по обоим квадратам одновременно и с одинаковой скоростью ползут две точки, и тот цвет, который видит перед собой первая точка, автоматически рисует вторая точка.

Скажем еще, как вычислить координаты соответствующей точки второго квадрата. Так как квадраты одинакового размера, то существует параллельный перенос, переводящий исходный квадрат в его копию. Очевидно, что этот же параллельный перенос переводит любую точку исходного квадрата в соответствующую ей точку второго квадрата.

Задача 17. На экране монитора нарисована закрашенная окружность. Координаты центра и цвет внутренней области вводятся с

клавиатуры. Нужно выяснить из какого количества точек состоит внутренняя область. Определить их количество необходимо по возможности оптимальным способом. То есть просто проверить цвета всех точек экрана нельзя, т. к. это займет слишком много времени.

Подсказки

И все же от проверки точек экрана уйти не получится. Проверка осуществляется функцией `getpixel`. Если результатом функции будет величина, равная номеру искомого цвета (цвета внутренней области), то, следовательно, мы нашли точку внутренней области; и к величине, содержащей количество точек, необходимо прибавить единицу.

Минимальной областью, содержащей окружность, будет квадрат, в который вписана окружность. Центр этого квадрата будет совпадать с центром окружности, а его сторона равна двум радиусам окружности.

Задача 18. Создать таблицу цветов в виде колонки квадратов, каждый из которых закрашен в свой цвет. Все квадраты пронумеровать. Для рисования квадратов разрешается из графических процедур использовать только процедуру `putpixel`. Для рисования закрашенных квадратов написать отдельную процедуру, рисующую квадрат в указанных координатах. Можно использовать процедуру из задачи 13.

Подсказки

Задача создания вертикальной колонки из квадратов аналогична задаче рисования лесенки квадратов из задачи 6.

В условии требуется проставить номера цветов в каждом квадрате. Как это сделать, можно посмотреть в задаче 10.

Как нарисовать один квадрат, пользуясь только процедурой установки точки, можно посмотреть в задаче о шахматной доске.

Задача 19. Построить пирамиду из закрашенных в разные цвета прямоугольников. Для рисования прямоугольников разрешается из графических процедур использовать только процедуру `putpixel`. Для рисования закрашенных прямоугольников напи-

сать отдельную процедуру, рисующую прямоугольник в указанных координатах.

Подсказки

Ранее был рассмотрен аналогичный пример — пирамида из кругов. Однако требуемую пирамиду построить сложнее, т. к. прямоугольник уменьшать в размерах сложнее.

Процедуру, рисующую квадрат, мы уже делали дважды. Сейчас надо ее немного модифицировать, для того чтобы обеспечить учет изменяющихся размеров прямоугольника.

Договоримся, что уменьшаться будет только ширина прямоугольников, их высота остается прежней. Тогда смещение прямоугольника вверх можно выполнять так, как и окружности, перемещением всех его вершин вверх на величину смещения.

Кроме смещения вершин вверх их также необходимо смещать к центру. Величина смещения вполне может быть константой. Левые вершины смещаются вправо, а правые смещаются влево.

Таким образом, программа представляет собой конструкцию цикла, внутри которого выполняются следующие операции:

1. Рисуется прямоугольник в текущих координатах.
2. Координаты *Y* всех вершин уменьшаются на величину высоты.
3. Координаты *X* левых вершин смещаются вправо.
4. Координаты *X* правых вершин смещаются влево.
5. Возвращаемся в пункт 1.

Задача 20. Нарисовать 100 одноцветных точек внутри квадрата с вершинами (100, 100); (300, 100); (300, 300); (100, 300) и координатами, определяемыми случайным образом.

Подсказки

Случайно выбранные координаты — это два случайных числа. Случайные числа генерируются функцией случайных чисел **random**(N). Эта функция даст целое случайное число в интервале от 0 до N–1.

Нам нужно получить два случайных числа в пределах от 100 до 300. Функция `random(200)` даст случайное число от 0 до 199. Тогда `random(200)+100` даст случайное число в интервале от 100 до 299. Таких случайных чисел требуется два, для координаты *X* и для координаты *Y*.

Программа будет представлять собой конструкцию цикла, в котором выполняются следующие действия:

1. Получение двух случайных чисел.
2. Рисование точки с полученными координатами.

Задача 21. Даны четыре вершины квадрата. Построить систему из пяти квадратов, сходящихся к общему центру. Использовать конструкцию цикла.

Подсказки

Задача очень похожа на задачу 18. Разница заключается в том, что здесь требуется смещать все вершины к центру.

Договоримся, что вершины смещаются на некоторую фиксированную величину *L*. Тогда:

1. У верхних вершин координаты *Y* уменьшаются на *L*.
2. У нижних вершин координаты *Y* увеличиваются на *L*.
3. У левых вершин координаты *X* увеличиваются на *L*.
4. У правых вершин координаты *X* уменьшаются на *L*.

Задача 22. Нарисовать вот такую картинку:

с использованием одной конструкции цикла и одной процедуры рисования линии.

```
Program example;
 Uses graph;
 Var
  i,dr,md:integer;
begin
 dr:=detect;initgraph(dr,md,'');
 for i:=1 to 4 do
   line(50*i,300-(5-i)*50,50*i,300);
end.
```

Пояснения

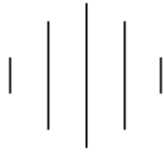
Отрезки отстоят друг от друга на одинаковое расстояние, нижние концы всех отрезков имеют одну и ту же координату *Y*, а верхний конец смещается вниз на каждом шаге.

Задача 23. Эта задача аналогична задаче 21. Необходимо нарисовать картинку из отрезков, используя одну конструкцию цикла и одну процедуру рисовании линии:

Подсказки

Здесь так же, как и в предыдущей задаче, должен быть построен циклический процесс, однако его характер меняется в середине. Для того чтобы это учесть, нужен специальный прием. Например, можно устроить проверку значения параметра цикла, и если он достиг определенного значения, то увеличение отрезка должно смениться уменьшением.

Задача 24. Еще одна задача из этой же серии. Нарисовать следующую картинку, используя только одну конструкцию цикла и одну процедуру рисования линии:



Подсказки

Данная задача от предыдущей отличается только тем, что здесь нет фиксированного конца отрезка, разница впрочем не принципиальная, и, следовательно, внешне новая программа почти не будет отличаться от предыдущей.

Задача 25. Нарисовать радугу с использованием процедуры рисования эллипса.

```
Program example;
 Uses graph;
 Var
  i,dr,md:integer;
begin
 dr:=detect;initgraph(dr,md,'');
 for i:=1 to 8 do
  begin
   ellipse(320,200,0,180,150-i*10,110-i*10);
   if i<8 then
    begin
     line(320-150+i*10,200,330-150+i*10,200);
     line(320+150-i*10,200,310+150-i*10,200);
    end;
   if i>1 then
    begin
     setfillstyle(1,i);
     floodfill(320,200-116+i*10,15);
    end
  end;
end.
```

Задача 26. Нарисовать волну с использованием процедуры рисования эллипса.

Подсказки

Процедура рисования эллипса позволяет рисовать различные дуги. Можно нарисовать дугу от 0 до 180 градусов, и можно нарисовать дугу от 180 до 360 градусов. Если эти две дуги сместить относительно друг друга (по горизонтали) на величину диаметра, то и получится нечто похожее на волну. Останется только повторить эту процедуру многократно.

Задача 27. На экране монитора случайным образом нарисовано несколько белых точек. Найти их все и перекрасить в зеленый цвет.

Подсказки

Функция `getpixel` позволяет определить цвет точки, а для просмотра всех точек экрана необходимо два вложенных цикла, один из которых пробегает все строки экрана, а второй — все точки на строке.

Задача 28. На экране монитора нарисован прямоугольник со сторонами, параллельными краям монитора. Его местоположение определяется случайным образом. Закрасить его внутреннюю область. Этот прямоугольник единственный графический объект, существующий на экране.

Подсказки

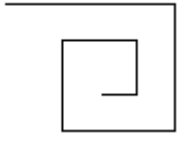
Если решена предыдущая задача, то два вложенных цикла обеспечивающих просмотр всех точек экрана, можно взять из нее. Далее обратите внимание, что, проходя очередную горизонтальную линию, мы можем (за исключением двух горизонтальных сторон) или ни разу не встретить точку цвета сторон прямоугольника, или только два раза.

Задача 29. На экране монитора нарисован прямоугольник. Его местоположение определяется случайным образом. Определить, является ли этот прямоугольник квадратом.

Подсказки

Данная задача серьезно пересекается с предыдущей. Все что нужно для ее решения, это подсчитать количество точек на одной горизонтальной и одной вертикальной стороне.

Задача 30. Нарисовать прямоугольную спираль. Примерно так, как это изображено на рисунке:



Подсказки

Можно написать один цикл, внутри которого рисуется отрезок, и на каждом шагу ориентацию отрезка менять на 90 градусов, а длину немного уменьшать.

Глава 4

Динамическая графика

Эта тема продолжает предыдущую, но ее сложность существенно выше, что вполне понятно, т. к. динамические картинки сложнее организовать. Здесь существенной может стать проблема понимания того, что видишь. Попробуйте в первой задаче убрать процедуру `delay`, в результате вы увидите только одну точку. Это не означает, что движения точки не было. Это означает, что точка двигалась очень быстро, и человеческий глаз за этим движением просто не успел.

Задача 1. Организовать движение точки по экрану.

```
Program example;
  Uses graph;
  Var
    driver,mode,i:integer;
begin
  driver:= detect;
  initgraph(driver,mode,'');
  for i:=1 to 600 do
    begin
      putpixel(i,200,0);
      putpixel(i+1,200,15);
      delay(200);
    end;
end.
```

Пояснения

Процесс перемещения точки по экрану состоит из двух действий: рисования точки цвета экрана и рисования точки белого цвета (белый цвет определяется по умолчанию). Точка цвета фона рисуется в старой позиции точки и тем самым закрашивает старую точку. Точка белого цвета рисуется на 1 пиксел правее предыдущей позиции, таким образом точка смещается за один шаг цикла на один пиксел.

Задача 2. Движение по экрану горизонтального отрезка.

```
Program example;
  Uses crt,graph;
  Var
    driver,mode,i:integer;
begin
  driver:= detect;
  initgraph(driver,mode,'');
  setcolor(15);line(10,100,60,100);
  for i:=1 to 600 do
   begin
     putpixel(9+i,100,0);
     putpixel(60+i,100,15);
     delay(100);
   end;
end.
```

Пояснения

На каждом шаге цикла самая левая точка закрашивается цветом фона и к правому концу пририсовывается одна точка белого цвета. Таким образом, за один шаг цикла отрезок смещается на одну точку вправо.

Задача 3. Организовать движение по экрану буквы в графическом режиме.

Подсказки

Задача аналогична предыдущей. Программа будет представлять собой цикл, внутри которого буква перемещается за счет того, что

за каждый шаг цикла она закрашивается в своей старой позиции и заново рисуется в новой.

Процедура установки символа — **`outtextxy(x,y,'строка символов')`**.

Задача 4. Изобразить движущуюся по окружности стрелку (стрелка от часов).

Подсказки

Стрелка — это отрезок. Чтобы сместить отрезок, надо его закрашивать в старой позиции и заново рисовать в новой. Отличие от предыдущей задачи в том, что отрезок смещается одним концом вдоль некоторой окружности радиуса R.

Определить положение конца стрелки, двигающего вдоль окружности, можно по формулам: $x = R\cos(a)$; $y = R\sin(a)$, где a — угол между осью OX и отрезком-стрелкой.

Приведенные формулы описывают окружность с центром в начале координат. Начало же координат находится в верхнем левом углу экрана. Чтобы переместить центр окружности по оси OX, надо прибавить число к координате X, аналогично можно поступить и с координатой Y. Для произвольного центра приведенные ранее формулы имеют следующий вид: $x = x0 + R\cos(a)$; $y = y0 + R\sin(a)$.

Задача 5. Некое тело брошено вверх. Известна его начальная скорость. Создать графическую модель движения такого тела. Максимальную высоту, которую видно на экране, принять постоянной. Для наглядности изобразить вертикальную ось координат с делениями. Для прорисовки тела (небольшой кружок) написать процедуру.

Подсказки

Физическая суть задачи в следующем: телу в момент времени t_0 придается некоторая начальная скорость V_0, благодаря которой тело начинает подниматься вверх. При этом зависимость высоты от времени дается следующим законом: $h = V_0t - (gt^2)/2$ где g — ускорение свободного падения, равное примерно 9,8 м/с^2. Подниматься тело будет до тех пор, пока скорость не уменьшится до нуля. Момент времени, когда это произойдет, можно определить из закона изменения скорости: $V = V_0 - gt$. Зная время подъема, можно определить достигнутую высоту. Обозначим ее через H_{max}.

Далее тело начнет падать, и закон изменения высоты будет таким: $h = H_{max} – (gt^2)/2$. Начальная скорость в этой формуле отсутствует, т. к. она равна нулю.

Таким образом, программа может состоять из описания двух циклических процессов: первый процесс моделирует движение тела вверх, и второй моделирует движение тела вниз. В обоих циклах главным параметром будет время, изменяющееся на определенную величину. Цикл движения вверх завершается, когда скорость становится равной нулю. Цикл движения вниз завершается, когда высота становится равной нулю.

Внутри циклов выполняются три действия:

1. Вычисляется очередное значение высоты.
2. Стирается старое изображение тела.
3. Рисуется новое изображение тела.

Текст программы, приведенный далее, частично решает поставленную задачу.

```
Program example;
 Uses crt,graph;
 Var
  dr,md:integer;
  h,v0,v,t:real;
procedure ris(n:integer);
begin
  setcolor(n);
  circle(300,round(400-h),5);
end;
begin
 read(v0);
 dr:=detect;initgraph(dr,md,'');
 h:=0;
 repeat
  h:=v0*t-4.9*t*t;
  v:=v0-9.8*t;
  ris(2);
  delay(10);
```

```
  ris(0);
  t:=t+0.001;
 until v<=0;
end.
```

Задача 6. Все условия предыдущей задачи сохраняются, но тело брошено не вверх, а под углом к горизонту. Кроме вертикальной оси еще нужно изобразить горизонтальную.

Подсказки

Движение под углом к горизонту разлагается на два независимых движения. Одно движение из предыдущей задачи, законы которого уже описаны. Другое — вдоль оси *OX*, с постоянной скоростью.

Для описания обоих движений нужно знать их начальные скорости. Если мы обозначим известную начальную скорость через *V*, то $V_x = V\cos(a)$ и $V_y = V\sin(a)$, где *a* — угол между осью *OX* и вектором начальной скорости.

Программа будет иметь точно такую же структуру, как и предыдущая. Отличие заключается в том, что в предыдущей задаче положение тела описывалось только одной координатой, а в данной задаче положение описывается двумя координатами, каждая из которых вычисляется независимо, и вычисляются они для каждого значения времени.

Задача 7. Тело падает с некоторой высоты. Построить графическую модель движения, для наглядности добавить к картине вертикальную ось с делениями.

Подсказки

Задача является второй половиной задачи 5, когда тело уже поднялось на максимальную высоту.

Задача 8. Тело опять пущено с некоторой скоростью под углом к горизонту, но на этот раз необходимо учесть сопротивление воздуха. Известно, что сила сопротивления прямо пропорциональна скорости движения тела.

Подсказки

Эта задача отличается от задачи движения под углом к горизонту под действием силы тяжести законом движения. Сейчас суммар-

ная сила равна разности силы тяготения и силы сопротивления воздуха.

Не забудьте учесть тот факт, что сила сопротивления воздуха непрерывно изменяется, т. к. непрерывно изменяется скорость.

Задача 9. Тело, покоящееся в пустом пространстве, начинает участвовать в двух взаимоперпендикулярных гармонических колебаниях. Построить модель такого движения.

Подсказки

Гармоническое колебание описывается следующим законом: $x = A\cos(wt)$, если тело колеблется вдоль оси *OX*, и $y = A\sin(wt)$, если тело колеблется вдоль оси *OY*.

В задаче сказано, что тело участвует в двух колебаниях, это означает, что его координата *X* изменяется по закону косинуса, а координата *Y* по закону синуса.

Программа представляет собой цикл по времени от t_0 до t_1.

Так как время желательно изменять на величину, меньшую единицы, то для организации циклического процесса хорошо подойдет цикл по условию.

Задача 10. Создать эффект горизонтального движения закрашенного прямоугольника по экрану монитора.

```
Program exapmle;
 Uses crt,graph;
 Var
  dr,md,x:integer;
begin dr:=detect;initgraph(dr,md,'');
 setcolor(4);
 rectangle(10,100,110,200);
 setfillstyle(1,4);
 floodfill(15,105,4);
 for x:=111 to 640 do
  begin
   setcolor(0);
   line(x-101,100,x-101,200);
```

```
    setcolor(4);
    line(x,100,x,200);
    delay(100);
  end;
end.
```

Пояснения

В задаче 2 показано, как двигать горизонтальный отрезок. Прямоугольник — это не что иное, как множество горизонтальных отрезков, прилегающих вплотную друг к другу.

Если мы будем сдвигать каждый из отрезков, из которых составлен прямоугольник, то возникнет эффект движения самого прямоугольника.

Таким образом, чтобы сдвинуть прямоугольник на одну точку вправо, мы должны пририсовать одну точку ко всем отрезкам, составляющим прямоугольник. Проще говоря, мы должны пририсовать к правому краю прямоугольника новый отрезок и после этого закрасить цветом фона один отрезок с левого края прямоугольника.

Циклическое повторение операции пририсовывания отрезка справа и затирания отрезка слева создаст эффект движения прямоугольника.

Задача 11. Создать эффект плавного сжатия окружности по оси *OY*.

Подсказки

Как видно из предыдущих задач, любое движение — это циклический процесс, состоящий из двух действий: рисования нового изображения и стирания старого. Следовательно, и в данной задаче необходимо также определить, как строить новое изображение и как стирать старое.

В компиляторе языка Паскаль фирмы Borland есть специальная процедура, рисующая эллипсы. Это процедура **ellipse**. Окружность является частным случаем эллипса, в котором диаметр по оси *OX* равен диаметру по *OY*. Таким образом, пользуясь данной процедурой, можно нарисовать изначальную окружность.

Далее можно организовать циклический процесс, состоящий из двух действий: закраски ранее нарисованного эллипса (самым первым ранее нарисованным эллипсом является исходная окружность) и рисования нового эллипса, у которого диаметр по оси *OY* будет несколько меньше.

Задача 12. На экране монитора нарисован закрашенный квадрат. Необходимо заставить его менять свой цвет, иначе говоря, перекрашивать в другой цвет.

Подсказки

Если на экране монитора нарисован квадрат, то нетрудно закрасить его. Для этого необходимо установить цвет и способ закраски в процедуре `setfillstyle` и затем в процедуре `floodfill` указать точку внутри контура и цвет контура. Как пользоваться процедурами закраски, можно посмотреть в задаче 2 предыдущей главы.

Для того чтобы заставить квадрат перекрашиваться, необходимо процедуру закраски повторять внутри цикла, в котором роль цвета закраски играл бы параметр цикла.

Задача 13. На экране монитора нарисована лента, состоящая из разноцветных квадратов. Как нарисовать такую ленту? Как сделать так, чтобы квадраты начали двигаться вдоль ленты?

Подсказки

В первой задаче предыдущей главы показано, как нарисовать ряд из вертикальных отрезков. Квадрат отличается от отрезка только тем, что состоит из множества линий, а не из одной. Следовательно, для изображения ленты мы создаем цикл, внутри которого рисуется квадрат, и его вершины на каждом шаге цикла смещаются на величину стороны квадрата вдоль оси *OX*.

Так как на каждом шаге цикла рисуется новый квадрат, то мы в качестве цвета для закраски можем взять параметр цикла.

Над движением квадратов вдоль ленты можно думать только после того, как все квадраты ленты будут нарисованы.

В задаче 9 подробно разобрано, как заставить двигаться прямоугольник. Эта задача от девятой отличается только тем, что здесь мы имеем ряд из квадратов. Смещение всех квадратов вдоль ленты состоит из одиночных смещений каждого квадрата.

Лента имеет конец, поэтому последний квадрат будет продолжаться не за пределами ленты, а в ее начале. Конец ленты как бы приклеен к ее началу. Обеспечить переход последнего квадрата в начало ленты можно так: правая нижняя точка каждого квадрата имеет координату *X*, которая изменяется на каждом шаге цикла на единицу, и когда точка с координатой *X* доходит до конца ленты, то она попадает в начало ленты. Следовательно, нам сейчас необходимо решить следующую задачу: как изменять некоторое чис-

ло с шагом 1 от 1 до *N* так, чтобы по достижении *N* число превращалось в 1 и процесс повторялся. Для достижения поставленной задачи можно применить оператор нахождения остатка **mod**. Выражение (k mod *N*) будет равно k при k<*N*, и k при превышении *N* будет обращаться в ноль. Тогда фрагмент программы, решающий нашу частную задачу, будет выглядеть так:

```
k:=1;
Repeat
  k:=k+1;
  k:=k mod N;
  if k=0 then k:=1;
Until любое, никогда не выполняемое условие.
```

Задача 14. Изобразить движущуюся змейку.

Подсказки

Задача аналогична задаче о движении горизонтального отрезка. В данной задаче тоже организуется движение отрезка, но его точки располагаются не на прямой, а на некоторой синусоиде. Синусоида — это линия, точки которой расположены по закону синуса $y = A\sin(wx)$.

Программа должна в начале своей работы нарисовать кусок синусоиды, который будет представлять собой змейку. Затем организуется циклический процесс, в котором на каждом шаге рисуется одна новая точка графика синуса, и одна точка на левом конце графика зарисовывается цветом фона.

Для данной задачи можно взять программу перемещения отрезка. В ней перевычисляются координаты *X*; для нашей задачи необходимо добавить операцию перевычисления координаты *Y*.

Необходимо помнить, что, применяя формулу, указанную ранее, мы получим синусоиду, расположенную возле начала координат, а оно находится в левом верхнем углу. Поэтому, чтобы наша змейка была хорошо видна, необходимо ко всем координатам *Y* прибавить некоторую величину, т. е. осуществить параллельный перенос картинки вдоль оси *OY*.

Задача 15. Создать эффект движения отрезка по диагонали.

Подсказки

Поставленная задача аналогична задаче о перемещении горизонтального отрезка. Разница лишь в том, что в данной задаче координата *Y* изменяется одновременно с координатой *X*. Это не-

большое изменение, и его можно выполнить непосредственно в тексте задачи о горизонтальном отрезке.

Задача 16. Построить модель хаотичного движения частицы со следующими ограничениями:

- за один шаг частица смещается на одно и то же расстояние;
- смещаться она может только в четырех направлениях: вверх, вниз, вправо, влево;
- направление смещения выбирается случайным образом.

Подсказки

Программа представляет собой циклический процесс, внутри которого выполняются следующие операции:

1. Вычисление направления смещения.
2. Закрашивание старого изображения цветом фона.
3. Рисование нового изображения.

Направление смещения мы можем обозначать целым числом (например: вверх — 1, вправо — 2, вниз — 3, влево — 4). Случайное число можно получить функцией **`random`**. Выбирать смещение можно, пользуясь конструкцией **`case`**. Алгоритмически это будет выглядеть так:

1. Если *смещение* = 1, то *Y* = *Y* – *величина смещения*.
2. Если *смещение* = 2, то *X* = *X* + *величина смещения*.
3. Если *смещение* = 3, то *Y* = *Y* + *величина смещения*.
4. Если *смещение* = 4, то *X* = *X* – *величина смещения*.

Чтобы тело не выпрыгнуло за пределы экрана монитора, в каждой из приведенных ранее альтернатив необходимо поставить проверку на допустимость смещения. Например: если *смещение* = 1, то, если *верхний предел* не достигнут, *Y* = *Y* – *величина смещения*.

Полезно в начале программы выполнить процедуру **`randomize`**. Ее выполнение обеспечит случайность работы программы при каждом новом запуске.

Задача 17. Построить модель движения тела по синусоиде.

Подсказки

Решение данной задачи можно получить комбинацией решений двух задач. Первая — это как смещать тело, прием, используе-

мый почти во всех задачах данной главы, и вторая полезная задача — это перемещение змейки.

Задача 18. Построить модель отражения мячика от стенки. Стенка расположена вертикально, ее положение фиксировано. Координаты мячика вводятся с клавиатуры. Мячик движется к стенке по прямой, скорость его движения постоянна. Решить задачу с условием, что компьютеру не известны координаты стенки, но известен ее цвет.

Подсказки

В качестве мячика можно взять окружность малого радиуса. Как организовать движение в этой главе, рассматривалось уже много раз. После отражения мячик начинает двигаться в противоположном направлении.

В этой задаче есть проблема, не встречающаяся нам ранее. В процессе движения мячика требуется поймать момент, в который он коснется стенки, цвет которой известен. Идея здесь следующая: в момент соприкосновения расстояние между центром шарика и стенкой будет равно величине радиуса. Следовательно, точка, расположенная по горизонтали на расстоянии радиуса от центра шарика, будет цвета стенки. А отсюда следует, что для определения, коснулся шарик стенки или нет, достаточно проверять цвет точки, отстоящей от центра по горизонтали на расстоянии радиуса. Проверять цвет этой точки необходимо до рисования шарика в новой позиции, потому что в момент касания шариком стенки его край закрасит точки стенки, и мы не сможем ее обнаружить.

Задача 19. Построить модель кругового движения отрезка. Отрезок движется по окружности, а наблюдатель видит его сбоку так, что его длина сначала уменьшается до нуля, а затем увеличивается до некоторого максимума, и процесс повторяется. Фактически наблюдатель видит отрезок переменной длины. Скорость движения не учитывать.

Подсказки

Отрезок растет вверх до максимума, затем уменьшается до нуля, затем растет вниз до максимума, затем опять уменьшается, и процесс колебания повторяется. Ясно, что это циклический процесс, на каждом шаге которого можно вычислять новое значение вершины отрезка, затем закрашивать цветом фона старый отрезок и рисовать новый.

Есть, однако, более красивая идея. Заметим, что реально движется только вершина отрезка, и эта вершина колеблется возле фиксированной точки (назовем ее центром колебания), и колебание происходит параллельно оси *OY*. Возможны четыре ситуации:

1. Если вершина движется вверх от центра колебания, то отрезок рисуется.
2. Если вершина движется вниз к центру колебания, то отрезок затирается.
3. Если вершина движется вниз от центра колебания, то отрезок рисуется.
4. Если вершина движется вверх к центру колебания, то отрезок затирается.

Таким образом, программа представляет собой бесконечный цикл, внутри которого постоянно рисуется точка, но в зависимости от описанных ранее ситуаций ее цвет меняется с цвета отрезка на цвет фона и наоборот.

Координату *Y* можно изменять по следующему закону: `Y:=Y+L`. Где `L` — направление движения. Если отрезок движется вверх, то `L:=1`. Если же отрезок движется вниз, то `L:=-1`. Величина `L` изменяется в крайней нижней и в крайней верхней точке. Если мы обозначим эти точки через `Ymax` и `Ymin`, то внутри нашего бесконечного цикла можно записать следующие операторы:

```
if (y=Ymax) then L:=-1;
if (y=Ymin) then L:=1;
```

Как вычислить `Ymax` и `Ymin`? Обозначим через `Y0` координату *Y* центра колебания и через `h` — длину колеблющегося отрезка. Эти величины можно вводить процедурами ввода перед началом работы программы. Тогда:

```
Ymax := Y0 + h
Ymin := Y0 - h
```

Задача 20. Построить модель пульсирующего круга. На экране монитора нарисован закрашенный круг, который периодически плавно меняет свои размеры, сначала увеличиваясь до некоторого предела, а затем уменьшаясь.

Подсказки

Задача совершенно аналогична задаче 18. Внутри некоторого бесконечного цикла рисуется окружность, радиус которой сначала

увеличивается от нулевого до максимального, и окружность при этом рисуется, например, белым цветом. Затем радиус изменяется от максимального до нулевого, и окружность при этом рисуется цветом фона.

Задача 21. Построить пульсирующую модель, аналогичную предыдущей задаче, но вместо круга взять закрашенный квадрат.

Подсказки

Задача полностью аналогична предыдущей. Разница заключается только в том, что здесь речь идет не об окружности, а о квадрате.

Задача 22. Создать модель стоячей волны в одномерном дискретном пространстве. Каждая точка волны должна изображаться отрезком, проведенным от горизонтальной оси до точки фронта волны.

Подсказки

Программа будет представлять собой цикл пересчета координат точек волны. После пересчета можно процедурой очистки экрана стереть старое изображение волны и нарисовать новое, но это не очень хорошо. Дело в том, что процедура очистки работает достаточно медленно и в сочетании с потребностями счетных операторов создаст серьезную задержку, которая будет выражаться в мигании изображения.

Для того чтобы решить проблему мелькания, надо каждый отрезок не стирать, а дорисовывать. Пусть, например, некая точка после пересчета получила координату *Y* на 1 большую, чем старая координата *Y*. Это означает, что вместо того, чтобы полностью стирать старый отрезок и рисовать новый, достаточно к старому дорисовать одну точку. А если новый на точку ниже, то достаточно на старом отрезке затереть одну верхнюю точку.

Каждый отрезок представляет собой гармоническое колебание, в ходе которого по синусоидальному закону изменяется его амплитуда. Таким образом, вся волна — это комбинация двух гармонических колебаний.

Задача 23. Тараканьи бега. На экране нарисовано несколько беговых дорожек. Роль тараканов исполняют небольшие кружки. Скорость бега каждого таракана случайна и может меняться в процессе движения.

```
Program example;
 Uses crt,graph;
 Var
  n,i,dr,md:integer;
  x:array[1..4] of integer;
function finish:boolean;
var
 i:integer;
begin
 finish:=false;
 for i:=1 to 4 do
  if x[i]=450 then finish:=true;
end;
begin
 randomize;
 dr:=detect;initgraph(dr,md,'');
 for i:=1 to 5 do
  line(50,40*i,450,40*i);
 for i:=1 to 4 do
  begin
   circle(50,60+40*(i-1),5);
   x[i]:=50;
  end;
  repeat
   n:=random(4)+1;
   setcolor(0);
   {Выбранный таракан стирается на старом месте}
   circle(x[n],60+40*(n-1),5);
   x[n]:=x[n]+1;
   setcolor(15);
   {И рисуется на новом}
   circle(x[n],60+40*(n-1),5);
   delay(600);
  until finish;
```

```
  closegraph;
  write('champion -',n);
end.
```

Задача 24. Более сложный вариант тараканьих бегов. Тараканы существенно меньше беговых дорожек и могут двигаться во всех возможных направлениях, но направление вперед имеет бóльшую вероятность выбора его тараканом. Так же как и в предыдущей задаче, использовать функцию для определения финиша.

Подсказки

От предыдущей задачи отличие только в том, что случайно выбирается не только таракан, но и направление его движения. Для лучшего визуального эффекта хорошо бы выбор осуществлять из 8 направлений.

Для того чтобы обеспечить неравноправность направлений, можно воспользоваться следующим приемом: пусть каждому направлению соответствует не одно направление, а интервал, тогда более вероятным будет то направление, у которого интервал больше. Регулируя величину интервала, мы можем изменять вероятность в нужную нам сторону.

Задача 25. Модель солнечной системы. В центре монитора находится неподвижное Солнце, вокруг него по круговой орбите движется Земля, а вокруг Земли также по круговой орбите движется Луна.

Подсказки

Задача организации вращения Земли решается легко, достаточно вспомнить задачу о вращении отрезка.

Задача о вращении Луны вокруг Земли сводится к задаче о вращении Земли вокруг Солнца с той разницей, что координаты Земли величина переменная. Если же вращение Луны организовать внутри процедуры, в которую координаты Земли будут передаваться, то переменность даже не будет заметна с точки зрения этой процедуры.

Задача 26. В пустом пространстве начинают движение два тела с известной массой и начальным вектором скорости. Между тела-

ми действует сила взаимного тяготения. Построить траектории их движения.

```
Program example;
 Uses crt,graph;
 type
  s=record
     x,y,vx,vy,m,ax,ay:real;
    end;
 Var
  dr,md:integer;
  telo1,telo2:s;
  t,q,l:real;
begin
  clrscr;
  with telo1 do readln(x,y,vx,vy,m);
  with telo2 do readln(x,y,vx,vy,m);
  dr:=detect;initgraph(dr,md,'');
  setfillstyle(1,2);setcolor(2);
  t:=0;
  repeat
   l:=sqrt(sqr(telo1.x-telo2.x)+sqr(telo1.y-telo2.y));
   telo1.ax:=telo2.m/(l*l*l)*(telo2.x-telo1.x);
   telo1.ay:=telo2.m/(l*l*l)*(telo2.y-telo1.y);
   telo2.ax:=telo1.m/(l*l*l)*(telo1.x-telo2.x);
   telo2.ay:=telo1.m/(l*l*l)*(telo1.y-telo2.y);
   with telo1 do
    begin
     x:=x+vx*t+ax*t*t/2;
     y:=y+vy*t+ay*t*t/2;
     circle(round(x),round(y),2);floodfill(round(x),round(y),2);
    end;
   with telo2 do
    begin
     x:=x+vx*t+ax*t*t/2;
```

```
     y:=y+vy*t+ay*t*t/2;
     circle(round(x),round(y),2);floodfill(round(x),round(y),2);
    end;
    t:=t+0.001;
  until t>1000;
end.
```

Пояснения

Тела воздействуют друг на друга непрерывно, непрерывно меняется их взаимное положение, следовательно, непрерывно меняется и сила взаимодействия. Однако мы не умеем считать непрерывные величины. Поэтому в данной программе сделано важное допущение о способе взаимодействия тел. Полагается, что тела действуют друг на друга мгновенно и воздействие осуществляется в некоторых временных точках, между которыми воздействия нет. При достаточно малых интервалах между точками воздействия картина движения будет хорошо соответствовать реальности.

Кроме того, полагается, что сначала первое тело воздействует на второе, а затем второе на первое. Оба этих положения приводят к погрешности в расчетах, но уменьшая временной интервал, в течение которого, как полагается, тела не взаимодействуют, эту погрешность можно уменьшить до сколь угодно малой величины. Таким образом, борьба с погрешностью — это вопрос быстродействия компьютера, на котором выполняются расчеты.

Возможно, покажется, что этот оператор не имеет смысла: `telo2.ay:=telo1.m/(L*L*L)*(telo1.y-telo2.y);`. Здесь в знаменателе стоит величина, численно равная величине `L` и отличающаяся от нее только, быть может, знаком, но сокращать нельзя именно из-за этого знака, т. к. он учитывает векторный характер силы и соответственно ускорения, а следовательно, и скоростей.

Задача 27. Модель вращающегося куба.

Подсказки

Нетрудно представить себе движение точки в пространстве по окружности. А для того чтобы пространственное движение проецировать на плоскость, достаточно немного знать тригонометрию.

Осталось вспомнить, что модель куба состоит из восьми точек, одни из которых соединены отрезками. Для вращения куба необходимо все его вершины смещать одновременно.

Некоторые ребра куба видимы, некоторые нет, и в процессе движения это их состояние меняется. Учитывать видимость ребер — достаточно трудная задача, можете этого не делать.

Задача 28. Модель движения бильярдного шара на бильярдном столе. Стол представляет собой прямоугольник, для бильярдного шара указывается начальный вектор скорости.

```
Program example;
 Uses crt,graph;
 Var
  dr,md,vx,vy,x,y:integer;
  x1,y1,Lx,Ly:real;
begin
 dr:=detect;initgraph(dr,md,'');
 setcolor(15);
 rectangle(50,50,450,350);
 x1:=90;y1:=90;vx:=7;vy:=12;
 x:=round(x1);y:=round(y1);
 Lx:=vx/(sqrt(vx*vx+vy*vy));
 Ly:=vy/(sqrt(vx*vx+vy*vy));
 circle(x,y,5);
 repeat
  setcolor(0);
  circle(x,y,5);
  if (getpixel(x-6,y)=15) or (getpixel(x+6,y)=15) then Lx:=-Lx;
  if (getpixel(x,y-6)=15) or (getpixel(x,y+6)=15) then Ly:=-Ly;
  x1:=x1+Lx;y1:=y1+Ly;
  x:=round(x1);y:=round(y1);
  setcolor(15);
  circle(x,y,5);
  delay(500);
 until keypressed;
end.
```

Пояснения

Шар отражается от стенки по законам оптики, т. е. угол падения равен углу отражения. Однако углы считать не надо. Наша задача сильно упрощается тем, что бильярдный стол прямоугольный. Если шар падает на горизонтальную стенку, то удар никак не влияет на горизонтальную составляющую скорости, а вертикальная составляющая просто меняет направление на противоположное.

В программе для работы с координатами заведено две переменные: одна целого типа, а вторая действительного. Это связано с необходимостью накапливать небольшие изменения координат, а функции округления небольшие изменения отбрасывают. Например, если округлить выражение 3 + 0.02, то получится 3. Этого, конечно, могло и не быть, если бы в процедурах графики можно было пользоваться действительными величинами.

Задача 29. Небольшое усложнение предыдущей задачи. Бильярдных шаров на столе теперь несколько.

Подсказки

Теперь мы должны иметь дело не с парой координат, а с парой массивов координат или массивом пар.

А внутри условного цикла, обеспечивающего движения, должен быть цикл по параметру, пробегающий все шары.

Задача 30. На экране монитора построена фигура, край которой представляет собой замкнутую ломаную линию без пересечений. Любая сторона фигуры паралелльна стороне экрана монитора. Некий путешественник (на экране он изображается окружностью маленького диаметра) должен, начав свой путь из произвольной точки экрана, дойти до фигуры и совершить ее полный обход.

Подсказки

Двигаться в произвольном направлении мы уже умеем, это, например, приходилось делать в задаче о бильярдных шарах.

Определить направление движения можно, если известны координаты вершин фигуры. Если мы предположим, что путешественник видит фигуру, то разумно допустить, что он знает координаты вершин фигуры. Тогда координаты точки, в направлении которой

необходимо двигаться, можно определить, как среднее арифметическое координат вершин фигуры. Назовем эту точку ТОЧКОЙ.

Столкновение с фигурой можно определить, проверяя цвета точек экрана, по которым движется путешественник (а он тоже точка).

Действительно сложной задачей является обход фигуры, особенно если учесть, что ее форма достаточно произвольна. Представьте себе, что в ТОЧКЕ лежат работающие часы. Зная направление движения часовой стрелки, каждую сторону нашей фигуры можно представить в виде вектора (т. е. направленного отрезка). Тогда, столкнувшись с отрезком (вектором), путешественник должен искать точку цвета контура в направлении данного вектора до тех пор, пока точку в данном направлении найти удается.

Глава 5

Вычислительные задачи

Математика — это отрасль знания, в которой методы программирования применяются наиболее часто. Вычислительные алгоритмы очень сложны и требуют как хорошего знания математики, так и виртуозного владения техникой программирования. В данной главе нет по-настоящему сложных задач, но даже в этих простых задачах все не так уж просто, поэтому будьте внимательны.

Задача 1. Дано два числа. Напечатать число, большее их среднего арифметического.

```
Program example;
Var
     a,s,d:real;
begin
  read(a,s);d:=(a+s)/2;
  if a>d then write(a);
  if s>d then write(s);
end.
```

Пояснения

Все числа, используемые в программе, действительные. Величина `d` играет роль среднего арифметического.

Задача 2. Квадратное уравнение задано своими коэффициентами. Вычислить его корни.

```
Program example;
  Var
    x1,x2,a,b,c,d:real;
```

```
begin
 write('Введите коэффициенты уравнения');read(a,b,c);
 d:=b*b-4*a*c;
 if d<0 then write('Уравнение не имеет корней');
 if d=0 then begin
              x1:=-b/2/a;
              write('x=',x1);
             end
 if d>0 then begin
              x1:=(-b+sqrt(d))/2/a;x2:=(-b-sqrt(d))/2/a;
              write('x1=',x1,'  x2=',x2);
             end;
end.
```

Пояснения

Программа вычисляет дискриминант и в зависимости от его значения принимает решение о том, сколько у уравнения корней, и какие они принимают значения.

Задача 3. Дано два числа. Выяснить, делится ли большее на меньшее нацело или нет. Какое из них больше, а какое меньше — не известно.

```
Program example;
Var
    a,b:integer;
begin
read(a,b);
  if (a>=b) then
    if a mod b=0 then write('Делится') else write ('Не
    делится')
  else
    if b mod a=0 then write('Делится') else write (' Не
    делится' );
end.
```

Пояснения

Так как неизвестно какое число больше, то программа рассматривает две ситуации: первую, когда `b` больше `a`, и вторую, когда `a` больше `b`.

Операция **mod** находит остаток от деления. Числа делятся в том случае, когда остаток равен нулю.

Задача 4. Решить предыдущую задачу при условии, что пользоваться операцией деления нацело и процедурой нахождения остатка нельзя.

Подсказки

Если число *A* делится нацело на число *B*, то это означает, что существует такое число *K*, что *A* = *B* * *K*. Это выражение в свою очередь означает, что если будем складывать *B* само с собой, то вычисляемая сумма рано или поздно станет равна *A*. Если же число *A* не делится на *B*, то вычисляемая сумма станет больше, чем *B*.

Алгоритм проверки может быть следующим:

```
Сумма = 0
Начало цикла
  Сумма = Сумма + B
   Если Сумма = A, то число A делится на B. Конец
   работы
   Если Сумма > A, то число A не делится на B. Конец
   работы
   Вернуться на начало цикла.
 Конец цикла.
```

Реализовать описанный алгоритм можно с помощью цикла по условию.

Алгоритм работает в том случае, когда сравнивается большее число с меньшим, но нам не известно, какое число больше, а какое меньше. Поэтому полезно договориться, что переменная *A* будет хранить большее значение, а переменная *B* меньшее. Перед запуском алгоритма проверки необходимо проверять больше *A* чем *B* или нет. Если же условие *A* > *B* не выполняется, то их можно переставить местами. Осуществить перестановку можно так: `C:=A; A:=B; B:=C;`.

Обратите внимание, что в предложенном алгоритме две точки, в которых завершается работа алгоритма. Придется вам подумать об исправлении алгоритма. Желательно, чтобы точка завершения была одна.

Задача 5. Вычислить сумму и среднее арифметическое множества чисел.

```
Program example;
  Var
    i,n:integer;
    a,s,f:real;
begin
  read(n);s:=0;
  for i:=1 to n do
    begin
      readln(a);s:=s+a;
    end;
  f:=s/n;
  writeln('Сумма =',s,'  Среднее арифметическое =',f);
end.
```

Пояснения

Величины `i` и `n` целого типа, т. к. они являются параметрами цикла. Величина `n` — это количество вводимых чисел; `a` — это переменная, в которую заносятся вводимые числа, и `s` — это переменная, содержащая сумму.

Внутри цикла выполняются два действия: вводится очередное число; значение очередного числа прибавляется к сумме.

По окончании работы цикла в переменной `s` — сумма чисел. Мы делим ее на величину `n` и получаем среднее арифметическое.

Задача 6. Вычислить сумму N последовательных целых чисел.

Подсказки

Задача отличается от предыдущей тем, что нет необходимости вводить числа. Если мы организуем цикл по параметру, и его параметр будет пробегать все числа от 1 до N, то это и будут нуж-

ные нам числа. Следовательно, к сумме необходимо на каждом шаге цикла прибавлять величину параметра цикла.

Задача 7. Вычислить *N*-й член ряда Фиббоначи. Рядом Фиббоначи называется ряд чисел, в котором два первых члена равны 1, а все последующие равны сумме двух предыдущих. То есть это следующий ряд: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

Для вычисления нового члена ряда написать функцию, в которую в качестве аргументов передаются два уже известных члена, а ее результат — новый член ряда.

Подсказки

Числа ряда Фиббоначи обладают интересным свойством, мешающим вычислить *N*-й член ряда. Для того чтобы посчитать A_n, необходимо знать два числа: A_{N-1} и A_{N-2} и т. д. Таким образом, чтобы вычислить *N*-е число, необходимо вычислить все предыдущие значения ряда Фиббоначи.

Программа будет представлять собой циклический процесс, в ходе которого вычисляются все числа от третьего до N-го. Первое и второе числа известны: $A_1 = 1$ и $A_2 = 1$. Каждое последующее равно сумме двух предыдущих.

Задача 8. Вычислить факториал от числа *N*.

Подсказки

Факториал от числа *N* вычисляется как произведение последовательных чисел от 1 до *N*. Таким образом, задача вычисления факториала очень сильно похожа на задачу вычисления суммы. Есть правда одна тонкость. Факториал растет намного быстрее суммы, и его значение быстро превысит допустимый интервал целых чисел, поэтому сумму целых можно объявить как целое, а для факториала желательно использовать тип действительный или длинное целое.

Задача 9. Вычислить сумму числового ряда $1 + 1/2 + 1/3 + 1/4 + + 1/N$.

Подсказки

Это опять задача вычисления суммы, в которой вместо числа берется обратное ему. Присутствие знака деления говорит о том, что необходимо использовать тип действительных чисел.

Задача 10. Вычислить сумму числового ряда $1 - 1/2 + 1/3 - 1/4$ до N-го члена.

Подсказки

На этот раз задача вычисления суммы осложнена тем, что члены ряда берутся с переменным знаком: то суммируются, то вычитаются. Для учета данного обстоятельства можно воспользоваться следующим приемом: заведем дополнительную переменную, которая при входе в цикл вычисления суммы равна 1, а затем на каждом шаге цикла меняет свой знак на противоположный. Сделать это можно следующим оператором условия: `if a=1 then a:=-1 else a:=1;` или еще проще: `a:=-a;`.

Задача 11. Арифметическая прогрессия задана своим начальным членом и разностью. Вычислить сумму N членов при условии, что пользоваться формулой суммы запрещается.

Подсказки

Опять мы имеем дело с задачей вычисления суммы, в которой каждое суммируемое число предварительно вычисляется по определенной формуле. В нашем случае это формула члена арифметической прогрессии.

Задача 12. Вычислить число $(((1/2)/3)/4))/N$.

Подсказки

То же самое выражение можно представить в следующем виде: $1/2/3/4\ldots/N$. Из данного представления видно, что мы имеем дело с операцией последовательного деления единицы на натуральные числа, начиная с двойки.

Задача 13. Даны два числа: A и B. Вычислить их произведение при условии, что пользоваться операцией умножения нельзя. В главной программе оформить только ввод и вывод. Для расчета произведения написать отдельную функцию.

Подсказки

Произведение числа A на число B — это суммирование числа A самого с собой B раз. Следовательно, данная задача — это задача нахождения суммы.

Задача 14. Дано натуральное число *N*. Выяснить, является оно простым или нет. Для выяснения факта, есть ли у числа делитель или нет, написать собственную функцию.

Подсказки

Если число *N* простое, то оно не имеет делителей. Все возможные делители находятся в интервале от 2 до числа *N* – 1. Следовательно, необходимо организовать цикл, в котором должна быть осуществлена проверка на делимость, и если ни одного делителя обнаружено не будет, то число простое.

Для того чтобы сделать вывод о простоте числа *N*, необходимо каким-либо образом запомнить факт обнаружения делителя. Сделать это можно, например, так: пусть некоторое число-индикатор перед циклом проверки равно единице. А если внутри цикла проверки будет обнаружен делитель, то индикатор приравнивается к нулю. Тогда по окончании цикла проверки достаточно посмотреть, чему равно значение индикатора. Если оно равно единице, то, следовательно, не было найдено ни одного делителя и, следовательно, число *N* простое. Кстати такой индикатор принято называть флагом.

Здесь интервал, в котором находятся все возможные делители, указан очень неточно. На самом деле он значительно меньше.

Задача 15. Дано натуральное число. Выяснить, является ли оно совершенным. Для поиска суммы делителей написать собственную функцию.

Подсказки

Совершенное число — это число, равное сумме своих делителей, включая единицу. Примеры: 6 = 1 + 2 + 3; 28 = 1 + 2 + 14 + 4 + 7.

Из определения ясно, что мы должны найти все делители, суммировать их и в конце проверить, равна или нет полученная сумма самому числу.

Решение задачи можно получить из предыдущей небольшим усовершенствованием. В предыдущей задаче мы каждое число проверяли, делитель оно или нет. Сейчас же, если число делитель, то его нужно добавлять к сумме, которая перед циклом проверки равна нулю.

Задача 16. Дано четырехзначное натуральное число. Разложить его на цифры.

Подсказки

Если мы разделим четырехзначное число на 1000 и округлим его до целого, то получим первую цифру исходного числа. Если после этого полученную цифру умножить на 1000 и вычесть из исходного числа, то мы получим трехзначное число, и операцию выделения цифры можно повторить.

Необходимо, однако, организовать описанную ранее операцию в виде циклического процесса. Это нетрудно сделать, если заметить, что на первом шаге мы делим и умножаем на 1000, на втором шаге на 100 и т. д. То есть число, помогающее нам выделить очередную цифру на каждом шаге цикла, уменьшается в 10 раз.

Задача 17. Дано три цифры. Составить из них натуральное число.

Подсказки

Задача обратная предыдущей. Для ее решения нужно вспомнить, что число представимо через набор своих цифр следующим образом: $M_0 * 10^0 + M_1 * 10^1 + M_2 * 10^2 + ... + M_n * 10^n$.

Так как число только трехзначное, то достаточно ввести три цифры и вычислить арифметическое выражение согласно приведенной формуле.

Задача 18. Дано действительное число. Округлить его с заданной точностью в меньшую сторону. Точность задана в виде количества цифр после запятой, которые должны остаться после округления.

Подсказка

Предположим, мы решили оставить только две цифры после запятой. Тогда мы умножаем число на 100, и требуемые цифры перейдут в целую часть. Затем полученное число округлим до целого. Этой операцией мы уничтожим все остальные цифры. Затем разделим полученное число на 100, и найденное таким образом число будет результатом.

Задача 19. Дан числовой ряд: 1, 1/2, 1/3, 1/4, ..., 1/*N*. Найти номер элемента, который первым стал меньше заданного числа *E*.

Подсказки

Построим циклический процесс, в котором вычисляется очередной член ряда. И в процессе вычисления будем проверять, стал очередной член меньше E или нет. Если стал меньше, то процесс прекращается и печатается полученный номер.

Задача 20. Провести численный эксперимент, в ходе которого проверить случайность генератора случайных чисел.

Подсказки

Попробуем вычислить много случайных чисел в фиксированном интервале. Предположим это интервал от 1 до 10. Если генератор работает качественно, то число 5 и число 6 должны появляться с примерно одинаковой частотой. Конечно, только по двум числам делать вывод о качестве генератора нельзя. Еще необходимо обдумать, что понимать под частотой.

Задача 21. Вычислить квадратный корень из положительного числа с заданной точностью.

```
Program example;
 Uses crt;
 Var
  a,e,q,min,max:real;
begin
 clrscr;
 read(a);
 min:=0;max:=a;
 repeat
  q:=(max+min)/2;
  if q*q>a then max:=q else min:=q;
 until (max-min<0.001);
 write((max+min)/2)
end.
```

Пояснения

В данной программе реализован обыкновенный метод половинного деления отрезка. Заключается этот метод в следующем:

1. *В* начале расчетов находится отрезок (пусть даже очень большой), в котором гарантированно находится искомый корень.
2. Затем отрезок делится пополам и определяется, в какой половине оказался корень. Таким образом, на каждом шаге отрезок уменьшается в два раза.

Суть процесса решения в стягивании отрезка к корню. Как только отрезок станет достаточно маленьким, задача считается решенной.

Задача 22. Небольшое усложнение предыдущей задачи. Теперь корень не квадратный. Пусть показатель корня любое натуральное число.

Подсказки

В решении предыдущей задачи только в одном месте существенно значима степень корня. Это место выделено жирным шрифтом. Если бы мы извлекали корень третьей степени, то вместо `q*q` надо было бы поставить `q*q*q`. А для произвольной степени лучше написать собственную функцию, которая на вход получает показатель степени и число, которое нужно возвести.

Задача 23. Дана рациональная дробь, представить ее в виде цепной.

```
Program example;
 Uses crt;
 Var
  a,b,c:integer;
begin
 clrscr;
 read(a,b);
 while a>1 do
  begin
   writeln(a div b);
   c:=a mod b;
```

```
    a:=b;
    b:=c;
  end;
end.
```

Пояснения

Цепной дробью называется дробь вида $a_0 + 1/(a_1 + 1/(a_2 + \ldots))$, где величины a_i — называются *знаменателями* цепной дроби. Известно, что любую рациональную дробь можно представить в виде цепной. Очередной *знаменатель* цепной дроби равен результату целочисленного деления очередного числителя на очередной знаменатель.

Чтобы было проще понять процесс образования цепной дроби, приведем пример.

```
Пусть дана рациональная дробь 17/7
Шаг первый: 17/7 = 2 + 3/7 = 2 + 1/(7/3)
Шаг второй: 2 + 1/(7/3) = 2 + 1/(2 + 1/3)
Знаменатели дроби: 2, 2, 3
```

Данная программа не рассчитана на преобразование в цепную дробь целых чисел.

Задача 24. Задача, обратная предыдущей. Дана цепная дробь, представить ее в виде рациональной.

Подсказки

Цепная дробь представляет собой сложное выражение, состоящее из выражений вида: *знаменатель* + *рациональная дробь*. Все, что вам нужно сделать, это научиться данное сложное выражение преобразовывать в *рациональную дробь*, т. е. научиться вычислять новый числитель и новый знаменатель *рациональной дроби*. А проделав такую операцию несколько раз, вы свернете цепную дробь в рациональную. Понятно, что количество операций преобразования равно количеству *знаменателей* цепной дроби.

Задача 25. Дано натуральное число A и число B такое, что $A > B$. Выяснить сколько раз B входит в A как делитель. Иначе говоря, найти такое число N, что A делится на B^N и A не делится на B^{N+1}.

```
Program example;
 Uses crt;
 Var
  a,b,n:integer;
begin
 read(a,b);
 n:=0;
 while a mod b=0 do
  begin
   n:=n+1;
   a:=a div b;
  end;
 write(n);
end.
```

Пояснения

Число *A* надо делить на число *B* до тех пор, пока это возможно. И при каждом делении счетчик делений увеличивать на 1. Таким образом, по окончании работы цикла деления счетчик делений и будет содержать требуемый результат.

Задача 26. Дано натуральное число. Построить его каноническое разложение. Для определения степени делителя числа написать собственную функцию.

Подсказки

Задача сильно похожа на предыдущую. Различие заключается в том, что возможные делители не вводятся извне, их необходимо искать среди простых чисел. Из этого следует, что программа должна уметь искать простые числа и затем для каждого найденного простого применять предыдущую программу, которую разумно было бы оформить как функцию. Если данная функция выдаст значение счетчика, отличное от нуля, то это будет означать, что очередное простое входит в каноническое разложение.

Еще, конечно, надо подумать над тем, какие простые числа могут быть кандидатами на делители.

Задача 27. Вычислить приближенное значение числа Пи, не используя тригонометрические формулы.

Подсказки

Метод приближенного расчета основан на представлении окружности в виде многоугольника, вписанного в эту окружность. С некоторой погрешностью можно предположить, что длина окружности равна периметру данного многоугольника. И чем в многоугольнике больше сторон, тем расчеты будут точнее.

Тригонометрические формулы понадобятся для выражения длины стороны многоугольника через радиус окружности.

Существует одна фигура, для которой эти расчеты очень просты — это квадрат. Осталось научиться строить более сложные многоугольники из квадрата.

Задача 28. Дано натуральное число N. Найти все натуральные числа, меньшие N и являющие суммой двух квадратов натуральных.

Подсказки

Есть два способа решения задач, подобных этой. Можно перебрать все числа в заданном интервале и для каждого проверить, удовлетворяет ли оно заданному свойству. А можно перебрать все числа, являющиеся суммой квадратов, и для каждого посмотреть, входит ли оно в заданный интервал.

Задача 29. Проверить великую теорему Ферма (для $n = 3$) в диапазоне [0, 100] простым перебором.

Подсказки

Решение будет представлять собой три вложенных цикла от 1 до 100 каждый.

Задача 30. Существенное усложнение предыдущей задачи. Пусть также требуется проверить великую теорему Ферма, но в интервале [0, 10 000]. Простой перебор здесь уже не сработает, време-

ни на простой перебор потребуется так много, что программа потеряет смысл. Нужно найти более эффективное решение.

Подсказки

Полный перебор потребует 10 000 * 10 000 * 10 000 операций, что очень много. Но бóльшую часть этих операций можно и не выполнять.

Действительно, если, например, левая часть в формуле Ферма уже больше правой, то еще увеличивать левую часть нет смысла.

Глава 6

Работа с массивами

Массив — это структура данных, без которой не обходится ни одна серьезная программа. Понятие массива существенно сложнее понятия переменной. Как и переменная, элемент массива имеет имя, тип и значение, но в отличие от переменной элемент массива имеет еще значение индекса. Это зачастую создает трудности начинающему программисту. Значение индекса элемента — величина, за которой нужно следить специально, а в сложных алгоритмах индекс может изменяться весьма хитроумным образом. Как правильно отслеживать значения индекса, точных рецептов нет, единственный выход — тренироваться. Вот такой тренировкой мы и займемся.

Задача 1. Дан массив чисел. Найти наибольшее.

```
Program example;
  Var
    i,n,max:integer;
    a:array[1..1000] of integer;
begin
  read(n);
  for i:=1 to n do read(a[i]);
  max:=a[1];
  for i:=2 to n do
   if a[i]>max then max:=a[i];
  write('Максимальное =',max);
end.
```

Пояснения

Внутри второго цикла каждый элемент массива сравнивается с уже найденным наибольшим, и если он больше наибольшего, то его значение становится новым наибольшим.

Задача 2. Найти сумму элементов числового массива.

```
Program example;
  Var
    i,n,s:integer;
    a:array[1..1000] of integer;
begin
  read(n);
  for i:=1 to n do read(a[i]);
  s:=0;
  for i:=1 to n do s:=s+a[i];
  write('Сумма =',s);
end.
```

Пояснения

Перед вычислением суммы переменная `s`, используемая для хранения значения суммы, обнуляется, т. к. значения переменных не обнуляются автоматически.

В объявлении массива указано число 1000. Это не означает, что нам придется вводить тысячу значений. Это означает, что мы можем ввести не более тысячи значений.

Задача 3. Дан числовой массив. Вычислить сумму элементов, имеющих четное значение индекса. Вычислительную часть организовать в виде функции.

```
Program example;
 Uses crt;
 type
  mas=array[1..100] of integer;
 Var
  a:mas;
  i,n:integer;
function calc(b:mas;m:integer):integer;
```

```
var
 i,s:integer;
begin
 s:=0;
 for i:=1 to m do
  if i mod 2=0 then s:=s+b[i];
 calc:=s;
end;
begin
 clrscr;
 read(n);
 for i:=1 to n do
  read(a[i]);
 write(calc(a,n));
end.
```

Пояснения

Необходимо суммировать только каждый второй элемент массива или, иначе говоря, те элементы, индексы которых делятся на два, т. е. остаток от деления на два равен нулю.

В данном решении мы построили функцию, полностью независимую от главной программы. В нее в качестве параметров передается вся информация о массиве и его длина, и он сам.

Задача 4. Вычислить количество элементов числового массива, кратных двойке. Вычислительную часть организовать так же, как и в предыдущей программе.

Подсказки

Эта задача почти точная копия предыдущей. Надо только немного подумать, чем отличаются их вычислительные части.

Задача 5. Дан массив символов. Вычислить, сколько в нем элементов 'a'.

```
Program example;
  Var
    i,n,s:integer;
    a:array[1..1000] of char;
```

```
begin
  readln(n);
  for i:=1 to n do readln(a[i]);
  s:=0;
  for i:=1 to n do
    if a[i]='a' then s:=s+1;
  write('Количество элементов равных "a"=',s);
end.
```

Пояснения

Объявление `a:array[1..1000] of char;` означает, что мы определили массив, элементы которого могут содержать только одиночные символы.

`s` — это счетчик, значение которого увеличивается на единицу каждый раз, когда обнаружится элемент массива, равный 'a'.

Задача 6. Дан числовой массив и два числа *A* и *B*. Вывести на экран монитора все числа, бóльшие *A*, но меньшие *B*.

```
Program example;
  Var
    i,n,s,b:integer;
    a:array[1..1000] of integer;
begin
  read(n);
  for i:=1 to n do read(a[i]);
  read(s,b);
  for i:=1 to n do
    if (a[i]>s)and(a[i]<b) then writeln(a[i]);
end.
```

Пояснения

Ключевое слово **and** переводится как "и", следовательно, в операторе **if** проверяются одновременно два условия: `a[i]` больше *S* и `a[i]` меньше *B*.

Задача 7. Дан числовой массив. Вычислить сумму произведений соседних элементов. Причем каждый элемент может участвовать в произведении не более одного раза.

Подсказки

Произведения вычисляются так: первый элемент умножается на второй, третий на четвертый, пятый на шестой, седьмой на восьмой и т. д. В общем виде произведение выглядит так: `a[i]*a[i+1]`.

Цикл, вычисляющий произведения, должен иметь шаг два, поэтому желательно использовать форму цикла по условию, внутри которого параметр изменяется с помощью следующего оператора: `i:=i+2;`.

Выход из цикла завершается, когда параметр `i` становится равен количеству элементов (для цикла с постусловием), или когда он становится равен `(`*`Количество`*` - 2)` для цикла с предусловием.

Задача 8. Дан массив чисел. Разбросать его положительные и отрицательные элементы по двум разным массивам.

Подсказки

Механизм работы программы может быть следующий: существуют три массива, два из которых пока пустые и один заполнен числами. Обработка заключается в циклическом исполнении следующих команд:

1. Переходим к очередному элементу массива чисел (его индекс увеличивается на 1).
2. Если очередной элемент положителен, то увеличиваем индекс массива положительных чисел и очередной элемент записываем в массив положительных чисел.
3. Иначе, увеличиваем индекс массива отрицательных чисел и очередной элемент записываем в массив отрицательных чисел.

Исходный массив обрабатывается с первого элемента по последний с шагом 1, поэтому обработку можно организовать в виде цикла по параметру, и параметр цикла может играть роль индекса исходного массива. Индексы массивов положительных и отрицательных чисел нужно изменять специально. Их первоначальное значение равно единице.

Задача 9. Дано два массива *A* и *B*. Обнулить в массиве *A* все числа, имеющиеся в массиве *B*.

Подсказки

Программа должна состоять из двух вложенных друг в друга циклов. Внешний цикл просматривает все элементы массива *B* и для каждого текущего элемента *B* запускается внутренний цикл, просматривающий все элементы *A*. Во внутреннем цикле проверяется условие равенства очередного элемента *A* очередному элементу *B*.

B качестве примера организации вложенных циклов вычислим все возможные произведения чисел от 1 до 9.

```
Для i от 1 до 9 делать
  Для j от i до 9 делать Печать(i*j);
```

Если во внешнем цикле есть еще команды кроме внутреннего цикла, то структура должна быть такая:

```
Для i от t до  k делать
  Начало
     ...
     ...
    Для  j от l до p делать
       Начало
          ...
          ...
       Конец цикла
   Конец цикла
```

Задача 10. Дан упорядоченный массив чисел. Переставить его элементы в обратном порядке.

Подсказки

Первый элемент переставляется с последним, второй — с предпоследним и т. д. То есть массив как бы симметрично отображается сам на себя относительно своей середины.

Обозначим количество элементов через `n`. Тогда на каждом шаге перестановки меняются местами элементы `a[i]` и `a[n-i+1]`, `i` пробегает от первого элемента массива до середины.

Если количество элементов нечетно, то срединный элемент переставляется сам с собой.

Перестановку двух элементов можно осуществить следующей последовательностью операций: `c:=a[i];a[i]:=a[n-i+1];a[n-i+1]:=c;`.

Задача 11. Дано два массива *A* и *B*, вставить массив *B* в массив *A*, с позиции *L*.

Подсказки

Работу программы можно организовать в два этапа:

1. Сдвинуть все элементы массива *A* с элемента *L* + 1 на количество позиций, равное количеству элементов массива *B*. Этим этапом мы создадим в массиве *A* дырку длиной в массив *B*.
2. Вставка элементов массива *B*.

Первый этап организуется в виде цикла, параметр которого пробегает массив *A* с последнего элемента по *L* + 1 и каждый *k*-й элемент перемещается в позицию *k* + *L*. Программно это можно организовать в виде цикла по условию или в виде цикла по параметру в форме, в которой параметр изменяется на –1. Приведем пример такого цикла: `for i:=10 downto 1 do s=s+i;`.

Второй этап — это тоже цикл, в котором каждый *i*-й элемент массива *B* вставляется в *L* + *i* – 1 позицию массива *A*.

Задача 12. Дан массив и два числа *K* и *L*. Выбросить из массива *L* элементов, начиная с позиции *K*.

Подсказки

Операция выбрасывания одного элемента с позиции *K* заключается в том, что весь массив, начиная с *K* + 1, сдвигается на одну позицию влево, затирая, таким образом, *K*-й элемент. Тогда операция выбрасывания *L* элементов заключается в повторении *L* раз операции выбрасывания *K*-го элемента.

Сдвиг на одну позицию вправо можно осуществить с помощью следующей конструкции цикла: `for i:=k+1 to n do a[i]:=a[i+1];`. Так как в результате этой операции длина массива уменьшается на 1, то необходимо отнять единицу от переменной, содержащей длину массива.

Очевидно, что обработка будет представлять собой два вложенных цикла. Внешний цикл считает количество выполненных операций выбрасывания одного элемента. Внутренний цикл осуществляет сдвиг элементов массива влево на одну позицию.

Задача 13. Дан массив символов. Переставить попарно его элементы. То есть первый со вторым, третий с четвертым и т. д.

Подсказки

Задача очень похожа на задачу 7, в которой также производится работа с соседними элементами. Данная задача отличается только тем, что с ними делается. А переставить два элемента можно следующим приемом: `c:=a[i]; a[i]:=a[n-i+1]; a[n-i+1]:=c;`.

Задача 14. Дан массив чисел. Упорядочить его по возрастанию.

```
Program example;
 Uses crt;
 Var
  a:array[1..100] of integer;
  i,j,n,c:integer;
begin
 clrscr;
 read(n);
 for i:=1 to n do read(a[i]);
 for i:=1 to n do
  for j:=1 to n-1 do
   if a[j]>a[j+1] then
    begin
     c:=a[j];a[j]:=a[j+1];a[j+1]:=c;
    end;
 for i:=1 to n do
  write(a[i],' ');
end.
```

Пояснения

Введем понятие неправильной пары. Пусть пара рядом стоящих элементов массива называется неправильной, если первый элемент пары больше второго. Каждая такая пара должна быть исправлена, т. е. ее первый и второй элементы поменяны местами.

Главное действие алгоритма состоит в том, чтобы пройти весь массив и исправить все неправильные пары. Конечно, одного прохода массива может оказаться мало, но если в массиве N элементов, то N проходов будет достаточно.

Приведенная программа не оптимальна. Совершенно не обязательно на каждом шаге проходить весь массив. Попробуйте ее улучшить.

Задача 15. Дан двумерный массив целых чисел размерностью $N \times N$. Найти сумму его элементов.

```
Program example;
   Var
     s,i,j,n:integer;
     a:array[1..10,1..10] of integer;
begin
  read(n);
  for i:=1 to n do
    for j:=1 to n do
      read(a[i,j]);
  for i:=1 to n do
    for j:=1 to n do
       s:=s+a[i,j];
  write(s);
end.
```

Пояснения

Обработка двумерного массива отличается от обработки одномерного тем, что для обхода всех элементов двумерного используется конструкция из двух вложенных циклов.

Задача 16. Дан двумерный массив, поменять местами строки и столбцы. Разрешается использовать вспомогательный массив.

Подсказки

Чтобы осуществить требуемую операцию, надо каждую строку записать в столбец, или, что то же самое, записать каждый столбец в строку.

Для массива нет понятий столбец и строка. Есть понятия первый индекс и второй индекс. Следовательно, выполнить поставленную задачу можно, если исходный массив переписать в другой массив, но порядок индексов массивов поменять. Первый индекс исходного массива должен быть вторым у массива-результата, а второй индекс исходного массива должен быть первым для массива-результата.

Задача 17. Дан трехмерный массив размерностью $N \times N \times N$. Геометрически такой массив представляет собой куб. Заполнить единицами его элементы, находящиеся на диагоналях.

Подсказки

У куба четыре диагонали. Поэтому блок установки единиц будет состоять из четырех похожих блоков, в каждом из которых расставляются единицы вдоль одной из диагоналей.

Чтобы понять, как расставлять единицы, необходимо изучить, как изменяются индексы вдоль диагонали. Приведем пример, как решается данная проблема для двумерной матрицы размером 4×4.

A_{11}	A_{12}	A_{13}	A_{14}
A_{21}	A_{22}	A_{23}	A_{24}
A_{31}	A_{32}	A_{33}	A_{34}
A_{41}	A_{42}	A_{43}	A_{44}

В данной матрице две диагонали. В диагонали, идущей из левого верхнего угла в правый, нижние индексы изменяются одновременно от 1 до 4. *В* диагонали, идущей из правого верхнего угла в левый нижний, первый индекс растет от 1 до 4, а второй убывает

от 4 до 1. Программа, расставляющая единицы в обоих диагоналях, будет выглядеть следующим образом:

```
Program example;
  Var
     i,j :integer;
     a:array[1..4,1..4] of integer;
begin
  {Обнуление матрицы}
  for i:=1 to 4 do
    for j:=1 to 4 do a[i,j]:=0;
  {Обработка первой диагонали}
  for i:=1 to 4 do a[i,i]:=1;
  {Обработка второй диагонали}
  for i:=1 to 4 do a[i,5-i]:=1;
end.
```

После того как единицы будут расставлены, вам придется решить еще более важную проблему — как убедиться в том, что единицы расставлены правильно. Проблема в том, как вывести на экран трехмерный массив и сделать это наглядно.

Задача 18. Найти наибольший элемент в двумерном массиве. Вычислительную часть организовать в виде функции или процедуры.

Подсказки

Задача совершенно аналогична задаче поиска наибольшего в одномерном массиве. Двумерность массива приводит лишь к тому, что вместо одного цикла при обработке используется конструкция из двух вложенных циклов.

Задача 19. Найти сумму построчных произведений элементов двумерного массива, т. е. для каждой строки найти произведение ее элементов и полученные произведения просуммировать. Построчные произведения вычислять внутри функции.

Подсказки

Программа должна представлять собой конструкцию из двух вложенных циклов. Внешний цикл будет обходить все строки и суммировать вычисленные произведения. Задача внутреннего цикла — обход всех элементов текущей строки и вычисление их произведения. Вот этот внутренний цикл и необходимо организовать в виде функции.

Задача 20. Дан двумерный массив, состоящий из нулей и единиц. Выделить в нем прямоугольник наибольшей площади, состоящий из одних единиц.

```
Program example;
  Uses crt,graph;
  Var
   a:array[1..70,1..20] of byte;
   c,s,q,x,y,lx,ly:integer;
function square(x,y,lx,ly:integer):integer;
 var
  i,j,s:integer;
  c:char;
begin
 s:=0;
 textcolor(2);
 for i:=x to x+lx do
  for j:=y to y+ly do
   if a[i,j]=0 then s:=1;
 if s=0 then
   for i:=x to x+lx do
     for j:=y to y+ly do
       begin
         gotoxy(i,j);write(a[i,j]);
       end;
 if s=0 then square:=1 else square:=0;
end;
```

```
begin
 textcolor(15);
 clrscr;
 randomize;
 for y:=1 to 20 do
  for x:=1 to 70 do
   begin
   c:=random(40);
   if c=39 then a[x,y]:=0 else a[x,y]:=1;
   end;
 for y:=1 to 20 do
   for x:=1 to 70 do
    begin
     gotoxy(x,y);write(a[x,y]);
    end;
 s:=1400;q:=0;
 repeat
  gotoxy(40,22);write('    ');
  gotoxy(40,22);write(s);
  ly:=1;
  repeat
   lx:=1;
   repeat
    if lx*ly=s then
      begin
       y:=1;
       repeat
        x:=1;
        repeat
         q:=square(x,y,lx,ly);
         x:=x+1;
         if (x+lx>70) then x:=70;
        until (q=1)or(x=70);
        y:=y+1;
```

```
        if y+ly>20 then y:=20;
       until (q=1)or(y=20);
      end;
    lx:=lx+1;
   until (q=1)or(lx=70);
   ly:=ly+1;
  until (q=1)or(ly=20);
  s:=s-1;
 until (s=1)or(q=1);
end.
```

Пояснения

Площадь прямоугольников изменяется от максимальной (весь массив) до минимальной (прямоугольник, состоящий из одной 1). Каждый прямоугольник конкретной площади может быть построен множеством способов. Для площади *S* допустимый прямоугольник — это такой, произведение сторон которого равно *S*. Мы должны для каждого значения площади перебрать все допустимые способы построения прямоугольников. Каждый прямоугольник конкретной площади и формы может располагаться в массиве различным образом. Его левая верхняя вершина может находиться в разных точках массива. Следовательно, для прямоугольника определенной площади и формы мы должны перебрать все возможные расположения его левой верхней вершины.

Может показаться, что программа для большого массива будет работать слишком долго, но есть серьезные возможности для ее ускорения. А именно:

1. Если площадь перебирать от максимальной к минимальной, то первый найденный прямоугольник и будет искомым.
2. Прямоугольник конкретной площади и формы не поместится в любом положении в массив.

Учет этих утверждений ведет к очень серьезному ускорению программы.

Задача 21. Дано две прямоугольных матрицы. Одна из них существенно больше другой. Выяснить, содержится ли меньшая в большей.

Подсказки

Задача от предыдущей отличается только тем, что имеется шаблон, который нужно найти. А процедура поиска такая же.

Задача 22. Свернуть одномерный массив по следующему правилу: на каждом шаге очередной элемент нового массива получается суммированием двух симметричных элементов массива, полученного на предыдущем шаге. Процесс заканчивается, когда остается одно число.

Подсказки

Программа, очевидно, будет состоять из двух циклов. Внешний цикл считает свертки массива и работает до тех пор, пока в массиве более чем одно число. Внутренний цикл пробегает половину массива, и для каждого очередного элемента суммирует его со своим симметричным.

Задача 23. Дан символьный массив. Определить, содержит ли он хотя бы один палиндром заданной длины. Задачу можно решить простым перебором.

```
Program example;
 Uses crt;
 Var
  a:array[1..100] of char;
  n,i,k,l:integer;
  q:boolean;
begin
 clrscr;
 readln(n);
 for i:=1 to n do
  readln(a[i]);
 read(L);
 i:=1;
 repeat
  q:=true;
  for k:=i to i+L div 2+1 do
```

```
   if a[k]<>a[2*i+L-1-k] then q:=false;
  if not q then i:=i+1;
 until (i>n-L+1) or q;
 if q then
 for k:=i to i+L-1 do
  write(a[k])
 else write('no');
end.
```

Пояснения

Палиндром — это такая последовательность символов, которая в обратном порядке читается так же, как и в прямом.

В программе есть два довольно длинных выражения, которые надо пояснить.

`i+L div 2+1` — заголовок цикла. Палиндром симметричен, на каждом шаге проверки мы сравниваем два симметричных относительно середины элемента, поэтому цикл проверки должен работать только до середины. `i` — это начальная позиция предполагаемого палиндрома. Прибавив к этой искомой длине `1`, мы и получим середину палиндрома, если точнее, ближайший к середине элемент.

`2*i+L-1-k` — элемент симметричный `k`-му. Первый элемент, с которого начинается работа проверки, имеет индекс `i`, ему симметричный отстоит на длину палиндрома, т. е. `i+L-1`. Вычесть `1` необходимо для учета `i`-ой позиции. Поясним: пусть $L = 5$ и $i = 3$, тогда симметричный ему элемент будет $3 + 5 - 1 = 7$, и сам палиндром состоит из следующих позиций 3, 4, 5, 6, 7 (как раз пять позиций).

Далее при увеличении индекса `k` на `1` индекс его симметричного элемента должен уменьшаться на единицу, т. е. нам нужна величина, которая будет равна на первом шаге 0 и затем на каждом шаге будет увеличиваться на `1`. Выразим эту величину через `k`. Получим следующее выражение: `k-i`. Осталось вычесть из индекса эту величину из найденного ранее значения индекса первого симметричного элемента. Получим `i+L-1-(k-i)=2*i+L-1-k`.

Задача 24. Дан символьный массив. Найти наибольший содержащийся в нем палиндром. Для определения палиндрома использовать функцию.

Подсказки

Самый большой палиндром, который может содержаться в массиве, совпадает с массивом, т. е. его центр совпадает с центром массива. Если же весь массив не является палиндромом, то следующий наибольший палиндром, возможно, имеет центр на один элемент левее или на один элемент правее середины массива и т. д.

Функцию целесообразно использовать для поиска палиндрома от известного центра.

Задача 25. Дан квадратный числовой массив. Выяснить, является ли он магическим квадратом.

Подсказки

Необходимо вычислить сумму элементов любой строки или любого столбца. Назовем эту сумму *суммой*.

Организуем два цикла. Пусть первый считает суммы столбцов, а второй суммы строк. Каждую найденную сумму необходимо сравнивать с *суммой*. Если хотя бы один раз результат сравнения даст значение "ложь", то это и будет означать, что квадрат не является магическим.

Задача 26. Дан двумерный массив. Найти в нем все седловые точки.

Подсказки

Существует два типа седловых точек. Это элементы массива, являющиеся наибольшими в столбце и наименьшими в строке, и наоборот, наименьшими в столбце и наибольшими в строке.

Из определения следует и способ проверки. Для поиска точек первого типа необходимо для каждого столбца найти наибольшее значение, а затем проверить, является оно наибольшим в своей строке или нет.

Задача 27. Дана двумерная таблица, заполненная нулями и единицами. Необходимо выяснить, существует ли путь по единицам из левого верхнего угла в правый нижний, если можно идти только по горизонталям и вертикалям.

Подсказки

Некоторые дороги могут вести в тупик, т. е. в такую точку, вокруг которой одни нули, за исключением, конечно, точки, из которой пришли. Необходимо придумать способ так выходить из тупиков, чтобы не возвращаться в них снова. Простейший способ заключается в том, чтобы единицу тупика заменить на ноль.

Если рисунок из единиц будет сложным, то вполне возможна ситуация, когда мы пойдем по кругу. С этим тоже не сложно бороться, достаточно помечать свой путь, например, двойками, которые при возвращении можно заменять нулями, что даст возможность не идти второй раз той же дорогой.

И последнее, в каждой точке возможно, как максимум три варианта дальнейшего пути, т. к. вокруг точки может быть 3 единицы. Необходим какой-то порядок выбора, в принципе любой.

Задача 28. Двумерная таблица заполнена случайными числами. Необходимо найти путь из левого верхнего угла в правый нижний такой, что сумма чисел, встреченных на пути, будет максимально большой. Двигаться можно только вправо и вниз.

Подсказки

Над матрицей можно выполнить следующее интересное преобразование: двигаясь от левой верхней точки к правой нижней, каждое значение заменить на сумму текущего значения точки матрицы и наибольшего значения из двух: левого и верхнего. Запишем это формулой. $A_{ij} = A_{ij} + \max(A_{i-1j}, A_{ij-1})$. Подумайте, что это даст.

Задача 29. Реализовать решето Эратосфена.

Подсказки

Решето Эратосфена — это способ получения простых чисел. В качестве пояснения и подсказки запишем следующий очень общий алгоритм:

```
Начиная с первого элемента массива и пока не достигнут
его конец, делаем
  Если очередной элемент не равен нулю,
    то обнуляем все элементы, большие очередного и
    кратные ему.
```

Глава 7

Операции со строковыми данными

Строки — это специальным образом устроенные символьные массивы. Для них компиляторы обычно имеют специальные строковые операции, позволяющие выполнять достаточно сложные действия одной командой. Правда, все это можно делать, используя обычные символьные массивы, но мы потренируемся в использовании специальных процедур и функций.

Задача 1. Даны две строки. Составить третью строку, являющуюся их суммой.

```
Program example;
  Var
    a1,a2:string[100];
    a3:string[200];
begin
  a1:='1111111111111';
  a2:='2222222222222';
  a3:=a1+a2;
  write(a3);
end.
```

Пояснения

Операция сложения добавляет вторую строку-слагаемое к концу первой строки-слагаемого.

Задача 2. Вводится строка символов. Вычислить, сколько в ней символов "a".

```
Program example;
  Var
   a:string[200];
   i,n,s:integer;
begin
  readln(a);
  n:=length(a);s:=0;
  for i:=1 to n do
    if copy(a,i,1)='a' then s:=s+1;
  write(s);
end.
```

Пояснения

Функция **length** вычисляет длину строки.

Функция **copy** копирует из нашей строки `i`-й символ. Если он представляет собой символ "a", то счетчик `s` увеличивается на единицу. Таким образом, по окончании работы цикла `s` равно количеству символов "a", содержащихся в строке.

Задача 3. Вводится строка символов. Вырезать из нее все символы "a".

Подсказки

Как обнаруживать символы, можно посмотреть в предыдущей задаче.

Вырезать символ, находящийся в позиции `i`, можно следующей операцией:

```
a:= copy(a,1,i-1)+copy(a,i+1,length(a)-i);
```

Задача 4. Даны две строки *A* и *B*. Выяснить, является ли *B* подстрокой *A*.

```
Program example;
  Var
    a,b:string;
```

```
begin
  readln(a);readln(b);
  if pos(b,a)>0 then write ('B является подстрокой A')
     else  write ('B не является подстрокой A');
end.
```

Пояснения

Функция **pos** выясняет, содержит ли вторая строка первую. Если вторая строка содержит первую, то функция возвращает номер позиции второй строки, с которой начинаются символы первой строки, если же вторая строка не содержит первую, то результатом будет ноль.

Задача 5. Дан массив строк и строка с алфавитом. Выяснить, состоят ли строки только из элементов алфавита. Проверку текущей строки осуществлять посредством функции.

Подсказки

В программе должен быть цикл проверки одной строки с целью выяснения, состоит ли она только из символов алфавита. Проверка массива строк — это тоже циклическая операция. Таким образом, программа представляет собой два вложенных цикла. Внешний отвечает за проверку всех строк, и внутренний — за проверку всех символов текущей проверяемой строки. Вот этот внутренний цикл можно написать в теле требуемой функции.

Проверку одного символа можно осуществлять с помощью функции **pos**. Пусть строка имеет имя `stroka`, и алфавит имеет имя `alfavit`, проверяемый символ находится в `stroka` в позиции `i`. Тогда проверка может выглядеть следующим образом: `if pos(copy(stroka,i,1), alfavit) = 0 then` *необходимая группа операторов*.

Задача 6. Дана строка символов. Заменить в ней латинские заглавные буквы на латинские строчные.

Подсказки

Можно создать две строки, одна из которых содержит все заглавные латинские символы, вторая содержит все латинские символы в том же порядке, но строчные. Затем организуем цикл, пробе-

гающий все символы анализируемой строки, и для каждого символа выполним две операции:

1. Выясним, в какой позиции он находится в строке заглавных латинских букв.
2. Если номер позиции не ноль, то возьмем символ, находящийся в той же позиции в строке строчных символов, и вставим вместо найденного заглавного в анализируемую строку.

Проверка номера позиции на равенство нулю необходима, т. к. в анализируемой строке не обязательно все символы латинские заглавные. Если же будет обнаружен такой неправильный символ, то его номер вхождения в строку заглавных будет нулем.

Задача 7. Дана строка символов, распечатать все участки строки, состоящие из одинаковых символов.

Подсказки

Прежде всего заметим, что длина участка не оговаривается, следовательно, это может быть участок, состоящий только из одного символа. Следовательно, любой символ строки представляет собой начало такого участка, если только слева от него нет такого же.

Если мы будем просматривать все символы строки по порядку, то начало нового участка можно считать обнаруженным, если текущий символ не равен предыдущему.

Перед началом распечатки символов нового участка курсор следует перевести на новую строку, таким образом, каждый новый участок будет распечатываться на новой строке. (Перевести курсор на новую строку можно командой `writeln`.)

Задача 8. Даны две строки A и B одинаковой длины. Составить третью строку, в которую символы первых двух будут входить попеременно.

Подсказки

В начале работы третья строка должна быть пустая. Необходимо организовать цикл, внутри которого будет формироваться третья строка, верхняя граница изменения параметра цикла равна длине строк A и B. На k-ом шаге цикла из строк A и B копируются функцией `copy` k-ые элементы и вставляются в формируемую строку.

Задача 9. Дана строка символов. Вывести на экран монитора все символы, имеющие повторение, но каждый символ выводить не более одного раза.

Подсказки

Программа будет представлять собой цикл, пробегающий все символы строки. Для каждого необходимо выяснить, печатался он уже или нет, и если не печатался, то печатать этот символ.

Выяснить, печатался символ или нет, можно на основании следующего факта. Если подобный символ имеется левее текущей позиции, то он уже был напечатан. Следовательно, для проверки необходимо организовать еще один цикл внутри данного, в котором будут проверяться все позиции, от первой до текущей, на предмет того, не стоит ли в них такой же символ.

Задача 10. Дана строка символов. Выяснить, сколько в ней различных элементов.

Подсказки

Задача совершенно аналогична предыдущей с той лишь разницей, что в предыдущей задаче символы надо было распечатывать, а в данной только считать. Для подсчета заведем счетчик, значение которого будет увеличиваться на единицу каждый раз при обнаружении нового символа.

Задача 11. Дана строка символов. Построить вторую строку, содержащую все элементы первой, в таком же порядке, но без повторений.

Подсказки

Данная задача аналогична предыдущим двум. Разница заключается в том, что делать при обнаружении нового символа. В данной задаче его требуется записать в новую строку, которая в начале работы пуста.

Задача 12. Дано две строки символов. Выяснить, можно ли одну строку получить перестановкой элементов другой строки.

Подсказки

Во-первых, необходимо проверить, одинаковая ли у них длина. Если нет, то очевидно одну строку нельзя получить из другой.

Если длина строк одинакова, то можно поступить следующим образом: пройдем все символы одной из строк и для каждого символа будем осуществлять поиск такого же во второй строке. Если такой же символ будет найден, то будем его вырезать из второй строки. Очевидно, если мы сможем пройти первую строку до конца (т. е. для каждого символа первой будет найден такой же во второй), то это значит, что строки имеют одинаковый состав. Программа же будет состоять из двух вложенных циклов. Внешний цикл будет пробегать все элементы первой строки, а внутренний будет искать во второй строке элемент, равный текущему элементу первой строки. От внутреннего цикла можно избавиться, если использовать функцию `pos`. Необходимо только заметить, что в описанном методе есть важный подводный камень. Будьте внимательны.

Задача 13. Дан массив строк. Упорядочить его в порядке возрастания длин. Использовать процедуру.

Подсказки

Сформируем числовой массив, в который занесем длины строк исходного массива. Далее реализуем процедуру упорядочивания его в порядке возрастания.

В алгоритме упорядочивания есть блок, в котором переставляются элементы неупорядоченной пары, т. е. такой пары, в которой предыдущий элемент больше последующего. Если в этом блоке одновременно с перестановкой неупорядоченной пары массива длин переставлять соответствующую пару элементов массива строк, то по окончании работы алгоритма мы получим массив строк, упорядоченный по возрастанию длин.

Задача 14. Дана строка символов. Сформировать новую строку, в которой элементы первой будут отделены друг от друга пробелами.

Подсказки

Пусть строка имеет имя `strok`, и пусть требуется вставить один пробел в позицию `i`. Эту операцию можно осуществить так:

```
strok:=copy(strok,1,i-1)+'
'+copy(strok,i,length(strok)+1);
```

Для решения поставленной задачи операцию вставки необходимо повторить для каждой позиции строки.

Есть в этой задаче небольшая хитрость. Длина строки после каждой вставки пробела увеличивается на единицу. То есть если мы, например, вставим пробел во вторую позицию и затем начнем вставлять следующий пробел в третью позицию, то фактически пробел встанет не после следующего символа, а после пробела. Чтобы справиться с этой проблемой, достаточно увеличивать номер позиции, в которую вставляется пробел, не на единицу, а на два.

А можно поступить еще проще. Будем формировать вторую строку следующим оператором присвоения `b:=a[I]+' ';`. Если эту операцию выполнить для всех элементов строки `a`, то это и будет решение задачи.

Задача 15. Дан массив строк. Выровнять строки массива по самой длинной строке точками. Точки добавлять в конец строки. Добавку точек осуществлять внутри процедуры.

Подсказки

Строки требуется выровнять по наидлиннейшей. Следовательно, первым делом необходимо найти самую большую длину. Эта задача сводится к задаче о поиске наибольшего.

Далее организуется цикл, внутри которого к строкам справа добавляются строки, состоящие из точек. Добавка строки из точек осуществляется простой операцией сложения (в первой задаче приведен пример того, как это выполняется).

Строку из точек можно сформировать так:

1. Вычисляем разность между длиной текущей строки и максимальной длиной. Полученное число равно количеству точек, которые необходимо добавить к текущей строке.
2. Пусть полученное число обозначается через `N`. Сформировать строку можно следующим программным блоком:

```
A:='';
For i:=1 to n do A:=A+'.';
```

Задача 16. Дана строка символов. Распечатать символы строки в обратном порядке, начиная с последнего.

Подсказки

Программа будет представлять собой цикл, параметр которого должен пробегать от последнего элемента до первого, копировать их из строки и распечатывать.

Можно воспользоваться следующей формой цикла по параметру `for i:=k downto t do`. Такой цикл изменяет свой параметр от большего значения к меньшему с шагом единица.

Задача 17. Дана строка символов с длиной, кратной трем. Разрезать ее на строки по три символа.

Подсказки

Программу удобно организовать в виде цикла по параметру. Начальное значение параметра — единица. Конечное значение равно длине строки, поделенной на три.

Величина параметра играет роль номера строки из трех символов, в которую будет заноситься очередной кусок большой строки. Кроме того, величина параметра будет определять номер позиции, начиная с которой вырезается очередной кусок из трех символов. А именно, если величину параметра обозначить через `i`, а номер позиции через `L`, то `L:=3*(i-1)+1`.

Задача 18. Дана строка символов и число N. Вырезать N символов с правого конца строки.

Подсказки

Некоторые компиляторы предоставляют специальную процедуру, выполняющую требуемую операцию. Предположим, что мы такой процедуры не имеем. Проблема заключается в том, что процедура **`copy`** копирует слева направо. То есть известна самая левая позиция копируемого фрагмента и его длина. В нашем случае известна самая правая позиция копируемого фрагмента (длина строки) и его длина. Используя эти данные, можно вычислить длину левого конца копируемого фрагмента, после чего уже несложно применить процедуру **`copy`**.

Нетрудно заметить, что операция вырезания заменена на операцию копирования тех элементов строки, которые должны остаться.

Задача 19. Дана строка. Вырезать фрагмент от начала до позиции, в которой находится пробел.

Подсказки

Номер позиции, в которой находится пробел, несложно получить с помощью функции **`pos`**. Тогда длина вырезаемого фрагмента бу-

дет равна полученному номеру позиции минус 1 (если не вычесть 1, то мы вырежем фрагмент вместе с пробелом).

Задача 20. Вычислить, сколько раз строка *A* входит в строку B.

Подсказки

Чтобы понять сложность задачи, достаточно рассмотреть следующий простой пример. Пусть *A* = "аааааа" и *B* = "ааа". Строка *B* входит в *A* четыре раза.

Программа будет представлять собой цикл по условию, завершающий свое выполнение тогда, когда функция `pos` не обнаружит *B* в *A*. На каждом шаге цикла программа будет пытаться обнаружить *B* в *A*, и если это удастся, то далее программа будет вычеркивать из *A* первый символ найденной подстроки. Эта операция даст нам гарантию того, что программа при следующей проверке не обнаружит еще раз то же вхождение.

Конечно, каждый раз, когда будет обнаружено очередное вхождение, некая величина, играющая роль счетчика вхождений, должна увеличиваться на единицу.

Метод, предложенный ранее, содержит одну ошибку, в которой вам придется разобраться самостоятельно. А для того чтобы ее увидеть, попробуйте проанализировать следующий пример: пусть *A* = 'aasasass' и *B* = 'as'.

Задача 21. Дан массив строк. Упорядочить его лексикографически. Порядковым номером считать его номер в таблице ASCII.

```
Program example;
 Uses crt;
 Var
  a:array[1..100] of string;
  L,i,n:integer;
procedure work(m:integer);
var
 i,j:integer;
 s:string;
begin
 if m<L then work(m+1);
 for i:=1 to n do
```

```
  for j:=1 to n-1 do
   if copy(a[j],m,1)>copy(a[j+1],m,1) then
    begin
     s:=a[j];
     a[j]:=a[j+1];
     a[j+1]:=s;
    end;
end;
begin
 clrscr;
 readln(n);
 for i:=1 to n do
  readln(a[i]);
 L:=length(a[1]);
 writeln('Result');
 work(1);
 for i:=1 to n do
  writeln(a[i]);
end.
```

Пояснения

Обратите внимание на то, как построена рекурсия. Рекурсивные вызовы не прекращаются до тех пор, пока не будет достигнут самый глубокий уровень, и только на этапе возвратов каждый вызов выполняет содержательную работу. За счет этого первый вызов делает свою работу последним, и, стало быть, первая буква приобретает наибольший приоритет.

Данная программа работает из предположения, что все строки имеют одинаковую длину. Вы можете подумать о том, как ее усовершенствовать, чтобы это допущение стало необязательным.

Задача 22. Дана строка строчных латинских символов. Упорядочить элементы строки в алфавитном порядке. Алфавит произвольный.

Подсказки

Можно составить массив, каждый элемент которого будет содержать номер элемента строки в заданном алфавите.

После этого работу по упорядочению можно будет выполнять над новым массивом. И если каждую операцию перестановки осуществлять как с элементами числового массива, так и с элементами строки, то задача окажется решенной.

Задача 23. Дано несколько положительных целых чисел, возможно различной длины. Упаковать их в строку так, чтобы каждое из них занимало в строке одинаковое количество позиций.

```
Program example;
 Uses crt;
 Var
  a:array[1..100] of integer;
  stroka,s:string;
  k,i,n:integer;
begin
 clrscr;
 readln(n);
 for i:=1 to n do
  readln(a[i]);
 stroka:='';
 for i:=1 to n do
  begin
   str(a[i],s);
   for k:=1 to 6-length(s) do
    s:=s+' ';
   stroka:=stroka+s;
  end;
 writeln(stroka);
end.
```

Пояснения

Для упаковки числа должны иметь одинаковую длину. Для этого недостаточно длинные числа можно дополнить пробелами.

Самое длинное целое число содержит не более 5 знаков. В программе на каждое число отводится 6 знаков.

Задача 24. В строке упакованы целые числа, возможно, различной длины и различного знака. Каждое число отделяется от своего соседа не менее чем одним пробелом, но возможно между некоторыми из них несколько пробелов. Необходимо выделить эти числа.

Подсказки

При чтении символов строки могут быть только две ситуации: прочитан пробел и прочитан не пробел. Если прочитан пробел, то нужно продолжать чтение до тех пор, пока пробелы не закончатся. Пока читаются не пробелы, их нужно записывать в очередное число. Как только будет прочитан пробел, это будет знаком, что число закончилось.

Задача 25. Дана строка символов, представляющая собой правильно построенное арифметическое выражение, состоящее из целых положительных чисел, открывающих и закрывающих скобок и знаков сложения. Вычислить данное выражение.

Подсказки

Прежде всего нужно научиться правильно обрабатывать элементарные выражения, т. е. выражения без скобок. Это просто. Цикл обработки выполняется до тех пор, пока в элементарном выражении находятся знаки плюс. Цикл обработки состоит из следующих операций:

1. Поиск очередного плюса.
2. Выделение правого аргумента.
3. Выделение левого аргумента.
4. Выполнение операции сложения.
5. Подстановка результата в элементарное выражение.

Если мы умеем работать с элементарными выражениями, то работа с большой строкой, содержащей скобки, будет выглядеть следующим образом:

1. Ищем очередную закрывающую скобку.
2. Ищем соответствующую ей открывающую скобку.
3. Делаем копию элементарного выражения, находящегося между этими двумя скобками.

4. Обрабатываем полученное элементарное так, как это описано ранее, и результат вставляем вместо своего оригинала.

Учтите, что в любом месте строки может быть произвольное количество пробелов и их нужно уметь игнорировать.

Задача 26. Дано предложение, состоящее из слов различной длины, разделенных пробелами (возможно, между словами не один пробел). Необходимо распечатать это предложение в строке экрана так, чтобы выполнялись следующие условия:

1. Слов на одной экранной строке должно быть как можно больше.
2. Слова не должны разрываться. Известно, что ширина строки заведомо больше любого слова предложения.

```
Program example;
 Uses crt;
 Var
  slovo,stroka:string;
  i,L,n,k,ost:integer;
begin
 clrscr;
 readln(L);
 readln(stroka);
 n:=length(stroka);
 i:=1;
 ost:=L;
 repeat
  slovo:='';
  while (stroka[i]<>' ') and (i<=length(stroka)) do
   begin
    slovo:=slovo+stroka[i];
    i:=i+1;
   end;
   if slovo<>'' then
   if length(slovo)<ost-1 then
```

```
    begin
     write(slovo,' ');
     ost:=ost-length(slovo);
    end
   else
    begin
     writeln;
     write(slovo,' ');
     ost:=L-length(slovo);
    end;
   while (stroka[i]=' ') and (i<=length(stroka)) do i:=i+1;
 until i>=n;
end.
```

Пояснения

Слова, конечно, должны разделяться одним пробелом. Из этого следует, что, напечатав слово, надо напечатать один пробел.

После этого необходимо принять решение о том, печатать следующее слово или нет. Его нельзя печатать, если длина слова больше, чем оставшееся количество допустимых знаков на экранной строке. Отсюда следует, что прежде чем печатать слово, его нужно выделить из символьной строки и определить его длину.

Задача 27. Дан массив строк небольшого размера (слов), упаковать его в массив строк большего по ширине размера. Перенос слов не допускается.

Подсказки

Необходимо сделать примерно то же самое, что делает MS Word при форматировании абзаца по ширине.

Конечно, необходимо попытаться вставить в каждую строку как можно больше слов, и, во-вторых, каждая строка абзаца не должна заканчиваться пробелами.

Понятно, что программа должна как-то рассчитывать количество используемых пробелов в каждой строке. Это можно делать исходя из следующих соображений:

1. Между двумя словами должен быть хотя бы один пробел. То есть если в строку умещается 5 слов, то в этой строке должно быть как минимум 4 пробела.

2. Если же после расчета количества вмещающихся слов оказывается, что остался остаток строки, в который уже не вмещается ни одно слово, то этот остаток необходимо равномерно поделить между всеми словарными промежутками.

Задача 28. Дан небольшой набор слов. Упаковать его в длинную строку. Известно, что на каждое слово выделено одинаковое количество позиций, эти позиции называются словарной ячейкой. Слово внутри своей словарной ячейки должно быть расположено по центру.

Подсказки

Мы уже делали почти то же самое, когда упаковывали в строку числа. Теперь разница в том, что пробелы нужно добавлять и справа и слева от слова для того, чтобы слово оказалось по центру своей ячейки.

Задача 29. Даны две строки. Первая из них отличается от второй на один символ. А именно, во второй строке один символ лишний. Выяснить какой именно.

Подсказки

Необходимо сравнивать подстроки, взятые из первой строки и второй. На *i*-ом шаге будем копировать и сравнивать подстроки, состоящие из *i* последовательных символов. И на каком-то шаге окажется, что подстроки не равны.

Задача 30. Дана строка, состоящая из слов, разделенных пробелами (возможно, некоторые слова разделяются несколькими пробелами). Необходимо выяснить, нет ли в этой строке одинаковых слов.

Подсказки

Чтобы выделить очередное слово, надо запомнить позицию, на которой закончилось предыдущее слово.

Очевидно, что циклов должно быть два, т. к. каждое слово нужно сравнить с остальными словами.

Глава 8

Работа с файлами

Если говорить точнее, то речь пойдет о файлах данных. Использование файлов данных — это единственная возможность хранить и обрабатывать практически неограниченные объемы данных. Техника работы с файлами не сложна, но надо хорошо понять разницу между файловой переменной и файлом. Язык Паскаль работает с файловой переменной так, как будто она и есть сам файл, но это все-таки разные вещи.

Задача 1. Ввести *N* чисел, записать их в файл, затем прочитать из файла и вывести на экран.

```
Program example;
  Uses crt;
  Var
     i,a,n:integer;
     f:file of integer; {Обьявляется файловая переменная,
                         далее с ней будет связан файл целых
                         чисел}
begin
   read(n);
  assign(f,'file'); {Файловая переменная связывается
                     с файлом}
  rewrite(f); {Файл создается}
  for i:=1 to n do {Файл заполняется числами, вводимыми
                    с клавиатуры}
    begin
       read(a);
```

```
      write(f,a);
    end;
  close(f); {Файл закрывается}
  reset(f);  {Файл открывается как существующий}
  for i:=1 to n do {Файл читается и считываемые значения
                    выводятся на экран}
    begin
       read(f,a);
       writeln(a);
    end;
end.
```

Задача 2. Имеется файл символов. Выяснить, сколько символов хранится в файле.

```
Program example;
  Var
    c:char;
    f:file of char;
    i:integer;
begin
  assign(f,'file');reset(f);i:=0;
  while not eof(f)  do
    begin
      read(f,c);i:=i+1;
    end;
  write(i);
  close(f);
end.
```

Пояснения

В данной программе выполняются следующие действия:

1. С файловой переменной связывается имя файла.
2. Файл открывается для работы. Процедура **reset** открывает только уже существующие файлы.

3. Пока не достигнут конец файла (проверку конца осуществляет функция **eof**, она принимает значение "истина", если конец файла достигнут), читается очередной символ, и счетчик символов увеличивается на 1.
4. По окончании работы цикла файл закрывается процедурой **close**.

Задача 3. Дан файл целых чисел. Вычислить их сумму.

Подсказки

Задача аналогична предыдущей, разница лишь в том, что здесь суммируются сами прочитанные элементы.

Задача 4. Дан массив целых чисел. Сохранить их в файле.

```
Program example;
  Uses crt;
   Var
     a:array[1..100] of integer;
     i,n:integer;
     f:file of integer;
begin
  read(n);
  for i:=1 to N do read(a[i]);
  assign(f,'name');rewrite(f);
  for i:=1 to n do write(f,a[i]);
  close(f);
end.
```

Пояснения

В данной программе для открытия файла используется процедура **rewrite**, т. к. она создает еще не существующий файл. А в нашем случае файл еще не существует.

Задача 5. С клавиатуры вводится некоторое количество целых чисел (понятие массива не используется). Записать в файл только положительные.

Подсказки

Так как массивов нет, то нельзя сначала ввести числа, а затем записывать их в файл. Следовательно, все три операции (ввод, проверка на положительность и запись в файл) должны выполняться в одном цикле.

Задача 6. Дан файл действительных чисел. Добавить одно число в конец файла.

Подсказки

С помощью функции **filesize** можно определить размер файла. С помощью процедуры **seek** указатель на элемент файла можно установить в конец файла. Последнее действие — запись числа.

Задача 7. Дан файл символов. Найти все символы "а" и записать вместо них символы "в". Для записи символа "в" написать процедуру, которая в качестве аргумента должна получить файловую позицию.

Подсказки

Программа будет представлять собой цикл по условию, выполнение которого прекращается по достижении конца файла. На каждом шаге цикла программа читает очередной элемент, и если окажется, что он равен "а", то вместо него записывается символ "в".

Необходимо помнить, что каждая операция чтения/записи файла сдвигает указатель позиции файла на одну позицию в сторону конца файла. Поясним ситуацию следующим примером: `Seek(f,5); Read(f,g); write(f,9);`. В данном примере число `9` будет записано не в пятую позицию, а в шестую. Если мы желаем записать `9` в пятую позицию, это можно сделать так: `Seek(f,5); Read(f,g); seek(f,5);write(f,9);`.

Не забудьте, что файловые позиции нумеруются с нуля.

Задача 8. Дан массив целых чисел. Требуется записать их в файл в обратном порядке. То есть последнее число на первое место, предпоследнее на второе и т. д.

Подсказки

Необходимо завести две переменные. Одна из них будет играть роль индекса массива, а вторая пусть будет номером позиции файла.

Затем можно организовать цикл по параметру, роль параметра может играть переменная, указывающая на позицию файла. Переменная-индекс массива также должна изменяться в этом же цикле, но в обратном порядке: от величины количества элементов массива до единицы. На каждом шаге цикла в файл записывается очередной элемент массива.

Задача 9. Дано два числовых файла одинаковой длины. Построить третий файл, содержащий попарные суммы.

Подсказки

Так как в задаче говорится о трех файлах, то нужно завести три файловые переменные и, соответственно, открыть три файла. Причем два файла уже существуют и, следовательно, их можно открыть процедурой `reset`, один файл новый, поэтому его следует открыть процедурой `rewrite`.

Так как файлы одинаковой длины, то программа будет представлять собой цикл по условию, а условием выхода будет достижение конца файла.

На каждом шаге цикла необходимо прочитать по одному числу из каждого файла, сложить их и полученную сумму записать в третий файл.

Задача 10. Дан числовой файл, упорядочить его содержимое по возрастанию.

Подсказки

В *главе 6* есть задача упорядочивания массива по возрастанию. Файл — это также своего рода массив, только хранящийся не в оперативной памяти, а на постоянном носителе. Надо подумать о различии в доступе к элементу массива и элементу файла.

Задача 11. Дан файл символов и массив целых чисел. Распечатать элементы файла, стоящие в позициях, номер которых является элементом массива.

Подсказки

Необходимо организовать цикл (можно по параметру), параметр которого будет пробегать все элементы массива и на каждом шаге цикла выполнять следующие операции:

1. Установить указатель на элемент файла, номер позиции которого равен значению текущего элемента массива. Сделать это можно так: `seek(f,a[i]);`.
2. Прочитать элемент файла и распечатать его значение.

Задача 12. Дан большой символьный файл из 100 элементов. Разбить его на два файла по 50 элементов.

Подсказки

Из условия задачи ясно, что требуется открыть три файла. Один, содержащий элементы, нужен на протяжении всей работы программы. Два других можно открыть сразу, а можно открывать их поочередно. Сначала открыть первый, заполнить его элементами исходного, закрыть, затем открыть второй.

Действие программы будет состоять из двух циклов, в каждом из которых читается очередной элемент исходного файла. В первом цикле прочитанный элемент необходимо записывать в первый файл результата, а во втором цикле прочитанный элемент необходимо записывать во второй файл результата.

Задача 13. Дана группа файлов целых чисел с серийными именами. Под серийными именами мы будем понимать имена, состоящие из одинакового имени и добавленной цифры, например: file1, file2, file3, file4, ... Напечатать имена файлов и номера позиций, в которых находится заданный элемент. Всего дано 9 серийных файлов.

```
Program example;
  Uses crt;
   Var
     s,s1,w:string;
     f:file of intger;
     i,a,g,k:integer;
```

```
begin
  s:='file';
  read(a);
  for i:=1 to 9 do
   begin
     str(i,w)
     s1:=s+w; assign(f,s1);reset(f);
     k:=1;
     while not eof(f) do
       begin
         read(f,g);
         if g=a then writeln(s1,' ',k);
         k:=k+1;
       end;
    close(f);
   end;
end.
```

Пояснения

В переменной `s1` хранится имя файла, состоящее из двух частей: постоянной строки `'file'` и добавочной части, состоящей из цифры, преобразованной в строку.

Первая позиция каждого числа — это знак. Если число положительное, то знака нет.

После того как определено имя файла, он открывается, прочитывается, и в случае нахождения искомого элемента номер его позиции распечатывается.

Задача 14. Дана группа серийных файлов символов. Объединить их в один. Имена отличаются друг от друга номером. Нумерация начинается с единицы.

Подсказки

Что такое файлы с серийным именем и как их открывать, обсуждалось в предыдущей задаче.

Для решения этой задачи необходимо знать, сколько всего файлов имеется. В программе должна быть конструкция цикла, внутри

которого открываются поочередно серийные файлы, читается их содержимое и записывается в большой файл, созданный в начале работы программы.

Задача 15. Дано два файла. Один файл строк и один файл целых чисел. Вывести строки в порядке, на который указывают числа числового файла.

Подсказки

Другими словами, числовой файл содержит номера строк, хранящихся в файле строк.

В начале программы необходимо открыть оба файла, затем выполнить следующие операции:

1. Прочитать очередной элемент числового файла. Обозначим его, например, через *A*.
2. Указатель строкового файла установить в позицию *A*. Это можно сделать с помощью процедуры **seek**.
3. Прочитать и распечатать очередную строку.

Задача 16. Дан файл чисел. Найти в нем наибольший элемент и распечатать его номер.

Подсказки

В *главе 6* имеется задача о поиске наибольшего в массиве чисел. Данная задача отличается только тем, что место массива занимает файл. Логика программы от этого не изменяется. Место индекса массива займет номер позиции файла.

Задача 17. Даны файл символов и строка. Выяснить, содержится ли данная строка в файле.

Подсказки

В задаче мы имеем дело с файлом символов. Файл символов — это большая символьная строка. Однако файловая природа этой строки мешает нам воспользоваться функцией **pos**.

Воспользуемся следующим алгоритмом: заведем две переменные `s1` и `s2`. Пусть `s1` указывает на номер элемента в строке, а `s2` — на номер позиции в файле. Перед началом работы алгоритма

`s1=1` и `s2=0`. Далее запустим циклический процесс, на каждом шаге которого рассмотрим следующие ситуации:

1. Текущий элемент строки не равен элементу, расположенному в текущей позиции файла. Тогда `s2` увеличивается на единицу, а `s1` приравнивается к единице.
2. Текущий элемент строки равен элементу, расположенному в текущей позиции файла. Тогда и `s1` и `s2` увеличиваются на единицу.

Циклический процесс останавливается в двух случаях:

1. `s1` стало равно длине строки. Это означает, что строка содержится в файле.
2. `s2` дошла до конца файла (т. е. значение `s2` стало равно количеству элементов файла). Это означает, что строка не содержится в файле.

Задача 18. Дан файл строк. Создать символьные файлы с именами, которые являются элементами строкового файла.

Подсказки

Программа представляет собой циклическую конструкцию, внутри которой выполняются следующие операции:

1. Читается очередной элемент строкового файла.
2. Создается файл с именем, равным значению прочитанного строкового элемента.
3. Вновь созданный файл закрывается.

Задача 19. Дан файл строк. Пусть все элементы данного файла — это имена файлов. Для каждой строки выполнить следующее действие: если файла с таким именем нет, то его создать, если же он есть, то его уничтожить. Проверка осуществляется относительно текущего каталога.

Подсказки

Файл, содержащий имена файлов, — это файл строк. Таким образом, данная задача сильно похожа на предыдущую.

Программа представляет собой циклическую конструкцию, внутри которой выполняются следующие операции:

1. Прочитывается очередной элемент строкового файла.

2. Выясняется, есть или нет файла с таким именем. Такую проверку можно провести с помощью функции **fsearch**.
3. Как создать новый файл, рассматривалось ранее. Уничтожить файл можно процедурой **erase**.

Задача 20. Дан файл строк. Пусть все элементы данного файла — это имена файлов. Нечетные строки представляют собой имена существующих файлов, четные — это новые имена. Требуется переименовать файлы. Перед операцией переименовывания проверять, есть такой файл или нет.

Подсказки

Для решения необходимо построить цикл, внутри которого выполнялись бы следующие действия:

1. Читается пара строк исходного файла.
2. Проверяется, есть ли файл с именем, равным первой строке. Если есть, то файл переименовывается. Переименовать файл можно с помощью процедуры **rename**. Новое имя — это вторая строка.

Задача 21. Дан файл символов. Переставить его элементы в обратном порядке, используя два файловых указателя.

```
Program example;
 Uses crt;
 Var
  i,n,a,b:integer;
  f1,f2:file of integer;
begin
 clrscr;
 readln(n);
 assign(f1,'data');rewrite(f1);
 for i:=1 to n do
  write(f1,i);
 close(f1);
 reset(f1);
```

```
 assign(f2,'data');reset(f2);
 seek(f2,filesize(f2)-1);
 for i:=1 to n div 2 do
  begin
   read(f1,a);
   read(f2,b);
   seek(f1,filepos(f1)-1);
   seek(f2,filepos(f2)-1);
   write(f1,b);write(f2,a);
   seek(f2,filepos(f2)-2);
  end;
 close(f2);
 reset(f1);
 for i:=1 to n do
  begin
   read(f1,a);
   write(a,' ');
  end;
end.
```

Задача 22. Дан файл целых положительных чисел. Выяснить средствами дихотомического поиска позицию числа с заданным значением. Известно, что числа в файле не повторяются и расположены в порядке возрастания.

Подсказки

Хорошим примером дихотомии является метод половинного деления отрезка для поиска корней уравнения. Этот пример можно посмотреть в *главе 5*.

Первый шаг дихотомического поиска в нашей задаче может выглядеть так:

1. Прочитаем число из середины файла.
2. Если это число больше заданного *числа*, то *число* находится в первой половине файла, иначе оно находится во второй половине.

Задача 23. Дан файл чисел. Поменять местами пары соседей. То есть первое со вторым, третье с четвертым и т. д.

Подсказки

Задача в общем простая, но нужно не забывать, что в результате выполнения операции чтения файловый указатель смещается на одну позицию в сторону конца файла.

Задача 24. В файле сохранен двумерный массив записей. Первый элемент хранит символьное значение, второй элемент — номер следующего элемента связного списка. Распечатать значения списка по порядку следования.

Подсказки

Решением будет циклический процесс, на каждом шаге которого выполняются следующие операции:

1. Чтение очередной записи.
2. Распечатка символьного компонента записи.
3. Переход на новую позицию в файле.

Задача 25. В файле сохранен двумерный массив символов. Организовать навигацию по файлу с использованием клавиш управления курсора.

Подсказки

Представьте себе таблицу в одну колонку, напечтанную на экране, по которой нужно осуществить движение курсора. Почти то же самое необходимо организовать в файле. Разница в том, что порядок элементов в файле линейный.

Так как строки в файле располагаются по порядку, и их длина одинакова, то для перехода на новую строку величину файлового указателя необходимо менять скачком на длину строки.

Задача 26. Дан текстовый файл, в котором строки имеют возможно различную длину. Прочитать этот файл и вывести его содержимое на экран.

```
Program example;
 Uses crt;
```

```
 Var
   f:text;
   s:string;
begin
 clrscr;
 assign(f,'text.txt');reset(f);
 while not eof(f) do
  begin
   readln(f,s);
   writeln(s);
  end;
end.
```

Пояснения

Программа несложная и в пояснениях не нуждается, но для лучшего понимания работы с текстовыми файлами средствами языка Паскаль попробуйте заменить `readln(f,s);` на `read(f,s);`. Приставка `ln` нужна для перехода на новую строку.

Задача 27. В текстовом файле хранится текст. В этом тексте многократно упоминается некоторое ключевое слово (например, Stoimost), после которого через произвольное количество пробелов стоит число. Необходимо просуммировать все эти числа.

```
Program example;
 Uses crt;
 Var
   f:text;
   s,c:char;
   stroka:string;
   i,sum,a:longint;
   cod:integer;
begin
 clrscr;
 assign(f,'text.txt');reset(f);
 sum:=0;
 while not eof(f) do
```

```
  begin
   read(f,s);
   if s='S' then
    begin
      stroka:='S';
      for i:=1 to 7 do
       begin
        if not eof(f) then read(f,s);
        stroka:=stroka+s;
       end;
      if stroka='Stoimost' then
       begin
        s:=' ';
        while s=' ' do read(f,s);
        stroka:=s;
        while s<>' ' do
         begin
          read(f,s);
          if s<>' ' then   stroka:=stroka+s;
         end;
        val(stroka,a,cod);
        sum:=sum+a;
       end;
    end;
  end;
  write('sum=',sum);
end.
```

Пояснения

Программа не учитывает тот факт, что файл текстовый, и, следовательно, в нем есть управляющие символы перехода на новую строку. Однако если эти символы не будут разрывать искомый текст, то программа будет работать корректно.

Общая идея вполне видна. Необходимо искать буквы "S" и, как только очередная буква найдена, необходимо проверить, не является ли она началом искомого слова, и если да, то, пропустив все пробелы, добраться до числа.

Задача 28. Дан текстовый файл, выбрать из него все числа с учетом того, что некоторые из них могут иметь знак.

Подсказки

Числа могут не отделяться пробелами, поэтому числом назовем последовательность цифр. Отсюда ясно и как их выделять. А именно, читаем до тех пор, пока не встретим цифру, и далее читаем цифры, пока не встретим иной символ.

После того как формирование числа завершено, необходимо посмотреть, есть ли знак операции впереди числа.

Задача 29. В файле целых чисел записана таблица целых чисел, состоящая из нескольких столбцов. Просуммировать столбцы.

Подсказки

Конечно, должно быть известно количество столбцов. Пусть их, например, 5. Тогда нужно читать файл блоками по 5 целых чисел и соответствующие суммировать.

Задача 30. В файле записана таблица в текстовом виде. Просуммировать столбцы.

Подсказки

Как прочитать число из текста, мы уже знаем (задача 28), как читать столбцы — также понятно (задача 29). Осталось совместить эти действия в одной программе.

Глава 9

Динамические величины

Самая сложная тема. Во-первых, сложно само понятие указателя, а во-вторых, на это понятие опираются такие серьезные структуры данных, как связные списки и деревья. И, кроме того, использование указателей — это единственная возможность получить под DOS полный доступ к оперативной памяти компьютера.

Задача 1. Дан массив чисел. Найти их сумму.

```
Program example;
 Uses crt;
 Var
  a,b:^integer;
  i,n,s:integer;
begin
 read(n); new(a);
 b:=a; {Запоминаем начало области массива}
 for i:=1 to n do
  begin
   read(a^); {Вводится значение в очередную ячейку памяти}
   inc(a); {Переход на следующую ячейку}
  end;
 s:=0;
 a:=b; {Вспоминаем начало области массива}
 for i:=1 to n do
```

```
  begin
   s:=s+a^; inc(a);
  end;
 write('s=',s);
end.
```

Пояснения

Для создания массива используем указатель на целое число. Конечно, не очевидно, что созданный указатель будет указывать на область памяти, достаточную для размещения желаемого количества данных. Но если вводимый массив не слишком велик (для массива не более чем 100 чисел плохие ситуации нам не встречались), то скорее всего проблем не возникнет. Для перехода между ячейками памяти следует использовать процедуры **inc** и **dec**. Операция сложения в языке Паскаль для указателей не определена.

При повторном обходе указателем `a`, конечно, можно и не пользоваться. И тогда оператор `a:=b;` можно убрать. Наличие данного оператора в нашей программе — это момент стиля программирования. Мы думаем, что полезно не менять смысл используемых переменных. Указатель `a` мы используем только для прохода области массива, а указатель `b` для хранения начала. Правила стиля не обязательны, но они упорядочивают мышление и стандартизируют текст программы.

Задача 2. Дан массив. Найти в нем наименьшее.

```
Program example;
 Uses crt;
 Var
  a,b:^integer;
  i,n,min:integer;
begin
 read(n); new(a);b:=a;
 for i:=1 to n do
  begin
   read(a^); inc(a);
  end;
 a:=b; min:=a^;
 for i:=2 to n do
```

```
  begin
   inc(a);
   if min>a^ then min:=a^;
  end;
 write('min=',min );
end.
```

Пояснения

О способе прохода области массива больше говорить не будем. Он всегда один и тот же. Здесь отличие от предыдущей программы только одно — это операция определения наименьшего. В данной программе текущий элемент, взятый из области памяти, сравнивается с уже найденным минимальным, и если текущий элемент еще меньше, то уже он становится минимальным.

Задача 3. Дан массив. Упорядочить его по возрастанию.

Подсказки

Задача упорядочивания массива по возрастанию уже решалась. Поэтому алгоритм обсуждать не будем. Существенное отличие от ранее решенной задачи заключается в том, что изменяется способ прохода по массиву. Вспомним, что в процессе прохода массива мы работаем с парой чисел-соседей. Чтобы обеспечить такую пару соседей, лучше всего создать два указателя, каждый из которых будет пробегать всю область массива, но один из них пусть бежит на один адрес вперед. Далее записан фрагмент программы, показывающий такой проход области массива. Здесь указатель `a` содержит адрес начала области массива, а указатели `a1` и `a2` на каждом шаге указывают на соседей, `a2` бежит на один адрес вперед. Всего в массиве `n` чисел.

```
a1:=a;
inc(a);
a2:=a;
for i:=1 to n-1 do
 begin
   inc(a1);
   inc(a2);
 end;
```

Задача 4. Дан двумерный массив. Ввести его и вывести на экран в виде таблицы.

Подсказки

В этой программе используется более сложная конструкция, нежели просто указатель. Здесь мы будем пользоваться массивом указателей на тип "указатель на целое число". То есть каждый элемент массива — это указатель на указатель. Физически это означает, что первый указатель содержит адрес памяти, по которому располагается адрес памяти, по которому содержится адрес памяти, начиная с которого расположено данное. В принципе язык Паскаль не запрещает строить сколь угодно длинные цепочки таких указателей. Чтобы добраться до данного, надо применить несколько раз операцию получения данного по адресу. В случае двойного указателя для доступа к данным можно записать, например, так: `C^^`.

```
Program example;
 Uses crt;
 Type
  a=^integer;
  b=array[1..5] of ^a;
 Var
  c:b;
  g:a;
  i,j,n:integer;
begin
 clrscr;
 read(n);
 for i:=1 to n do
  begin
   new(c[i]); new(c[i]^);g:=c[i]^;
   for j:=1 to n do
    begin
     read(c[i]^^); inc(c[i]^);
    end;
   c[i]^:=g;
  end;
```

```
 clrscr;
 for i:=1 to n do
  begin
   for j:=1 to n do
    begin
     gotoxy(i*3,j);write(c[i]^^);
     inc(c[i]^);
    end;
  end;
end.
```

Важное замечание

Создание массива указателей на целое, т. е. объявление вида `b:array[1..10] of ^integer;` не привело бы к желаемому результату. В первых задачах мы действительно пользовались одним указателем для создания целого массива, но это был один массив. В случае с несколькими массивами нет никакой гарантии, что никакие два не будут использовать общей области памяти, скорее наоборот пересечение практически гарантировано.

Задача 5. Дан двумерный массив. Вычислить сумму построчных произведений. Для расчета построчного произведения использовать функцию либо процедуру.

Подсказки

Техника создания двумерного массива и его обхода дана в предыдущей задаче. Остается предыдущую задачу немного модифицировать и вместо вывода элементов массива на экран вычислять построчные произведения. Кстати, в *главе 6* задача с таким же условием уже решалась.

Задача 6. Реализовать метод решета Эратосфена.

Подсказки

Данная задача также уже решалась в *главе 6*. Сейчас остается только применить к ней технику работы с указателями.

Задача 7. Дан символьный массив. Определить, сколько в нем различных символов.

Подсказки

Техника создания и прохождения массива та же, что и в первых задачах. Идея определения количества различных элементов звучит так: очередной элемент является новым, если слева от него нет такого же. Количество различных элементов равно количеству таким образом определенных новых.

```
Program example;
 Uses crt;
 Type
  a=^char;
 Var
  g,h,w:a;
  i,n,s,j:integer;
  q:boolean;
begin
 clrscr;
 readln(n);
 new(g);h:=g;
 for i:=1 to n do
  begin
   readln(g^);
   inc(g);
  end;
 g:=h;
 s:=1;
 for i:=2 to n do
  begin
   inc(g);
   q:=true;
   w:=h;
   for j:=1 to i-1 do
    begin
     if w^=g^ then q:=false;
```

```
     inc(w);
    end;
   if q then s:=s+1;
  end;
 write('s=',s);
end.
```

Задача 8. Дан символьный массив. Вставить в массив некоторое количество символов.

Подсказки

Для того чтобы вставить в массив символы, в нем необходимо сделать дырку размером, равным количеству вставляемых символов. Делать эту дыру необходимо перемещением символов на свободное место за последним символом массива. Так вот, перемещать символы нужно, начиная с последнего. Эта задача противоположна следующей. Если данная задача долго будет создавать для вас проблемы, просмотрите решение следующей, а потом вернитесь к решению этой.

Задача 9. Дан символьный массив. Выбросить из него все символы "а".

Подсказки

Задача в каком-то смысле противоположна предыдущей. Если в задаче 8 мы перемещали элементы массива на свободное место, чтобы вставить новые, то сейчас будем выполнять противоположную операцию: по обнаружении элемента, содержащего символ "а", мы будем смещать элементы массива справа налево, тем самым затирая обнаруженный элемент.

Здесь мы будем использовать специальный технический прием. Его суть в двух рядом идущих указателях, один из которых показывает на текущий элемент, а второй — на следующий. Эти два указателя дают возможность легко организовать перемещение элементов массива.

```
Program example;
 Uses crt;
 Type
  a=^char;
```

```
 Var
  g,h,w,f:a;
  i,n,j:integer;
  c:char;
begin
 clrscr; readln(n);
 new(g);h:=g;
 for i:=1 to n do
  begin
   readln(g^); inc(g);
  end;
 g:=h;c:='a';
 i:=0;
 while i<=n do
  begin
   i:=i+1;
   if g^='a' then
    begin
     w:=g; f:=g; inc(f);
     for j:=i to n-1 do
      begin
       w^:=f^; inc(w);inc(f);
      end;
      n:=n-1; i:=i-1; dec(g);
    end;
   inc(g);
  end;
 g:=h;
 for i:=1 to n do
  begin
   write(g^); inc(g);
  end;
end.
```

Задача 10. Дан массив строк. Выровнять строки точками по длиннейшей.

Подсказки

Программа, выполняющая условие задачи, может состоять из двух частей: в первой части определяется наибольшая длина строки, а во второй части к каждой строке добавляется необходимое количество точек.

Техническая деталь: есть ограничения в объявлениях указателя на строку. Далее приведены два правильных варианта, а неверный выделен жирным шрифтом:

```
type
  ss=string[50];
var
 str1:^ss;
 str2:^string;
 str3:^string[50];
```

Задача 11. Построить связный список и пройти его из начала в конец.

Подсказки

Данная задача уже разбиралась в *главе 1*, и здесь она только для напоминания. Разберите ее еще раз подробно, это основа для работы со связными списками.

```
Program example;
 Uses crt;
 Type
  shablon=^zapis;
  zapis=record
         a:integer;
         next:shablon;
        end;
 Var
  uk1,uk2:shablon;
  i:integer;
```

```
begin
 clrscr;
 new(uk1);uk2:=uk1;
 uk1^.a:=1;
 for i:=2 to 10 do
  begin
   new(uk1^.next);
   uk1:=uk1^.next;
   uk1^.a:=i;
  end;
 uk1:=uk2;
 for i:=1 to 10 do
  begin
   write(uk1^.a,' ');
   uk1:=uk1^.next;
  end;
end.
```

Задача 12. Построить связный список и создать возможность его произвольного просмотра с использованием клавиш управления курсором.

Подсказки

Связный список, создаваемый в этой задаче, должен в отличие от задачи 11 иметь два указателя: на следующую запись и предыдущую. Программу целесообразно построить как цикл, внутри которого производится опрос клавиатуры, и в зависимости от нажатой клавиши выбирается указатель для перехода либо на следующую запись, либо на предыдущую.

Задача 13. Дан массив чисел. Найти их сумму. Массив устроен в виде связного списка.

Подсказки

Алгоритм прохождения массива известен. Остается при прохождении массива считать сумму элементов.

Задача 14. Дан массив. Найти в нем наибольшее. Массив устроен в виде связного списка.

Подсказки

Данная задача мало отличается от предыдущей. Такой же обход массива, только здесь мы не добавляем в сумму очередной элемент списка, а проверяем, не больше ли он уже найденного наибольшего.

Задача 15. Дан массив. Упорядочить его по возрастанию. Массив устроен в виде связного списка.

Подсказки

Данная задача решалась уже дважды, с обычными массивами и с массивами, созданными посредством указателей. Поэтому все, что нужно, это вспомнить предыдущие задачи и подумать, какие проблемы создает новая структура данных. Главная проблема, пожалуй, такая же, как и в массивах, созданных посредством указателей. Элементы этих массивов не имеют индексов.

Задача 16. Дан массив. Распечатать все нечетные палиндромы. Массив устроен в виде связного списка. Для поиска палиндромов использовать процедуры, желательно применить рекурсию.

```
Program example;
 Uses crt;
 Type
 shablon=^zapis;
 zapis=record
        a,num:integer;
        next,pred:shablon;
       end;
 Var
  uk1,uk2,uk,left,right:shablon;
  i,n:integer;
begin
 clrscr;
 read(n);
```

```
new(uk1);uk2:=uk1;
for i:=1 to n do
 begin
  read(uk1^.a);
  uk1^.num:=i;
  new(uk1^.next);
  uk:=uk1;
  uk1:=uk1^.next;
  uk1^.pred:=uk;
 end;
uk1:=uk2;
uk1:=uk1^.next;
for i:=2 to n-1 do
 begin
  left:=uk1;right:=uk1;
  repeat
   left:=left^.pred;
   right:=right^.next;
  until (left^.a<>right^.a) or (left^.num=1) or (right^.num=n);
  if (left^.a<>right^.a) then
   begin
    left:=left^.next;
    right:=right^.pred;
   end;
  if (left<>right) then
   begin
    uk:=left;
    while uk<>right do
     begin
      write(uk^.a,' ');
      uk:=uk^.next;
      delay(10000);
     end;
```

```
      write(uk^.a);
      writeln;
     end;
    uk1:=uk1^.next;
   end;
end.
```

Пояснения

Нечетный палидром — это массив, элементы которого симметричны относительно центрального. Логика алгоритма здесь такова: для каждого элемента, начиная со второго и заканчивая предпоследним, мы предполагаем, что он центр палиндрома. Затем проверяем это предположение. Для того чтобы проверка была легкой, каждая запись связного списка снабжается двумя указателями: вправо (следующий) и влево (предыдущий).

В качестве проверки мы пытаемся идти по связному списку одновременно вправо и влево до тех пор, пока текущие правый и левый элементы одинаковы или пока не достигнут конец или начало списка. Текущий элемент действительно центр палиндрома, если нам удалось отойти от него хотя бы на одну позицию. Найденный палиндром располагается от правого указателя до левого.

Техническая деталь: язык Паскаль допускает сравнение указателей, но только двумя способами. Можно проверить два указателя на равенство или неравенство. Сравнения больше и меньше для указателей не определены.

Задача 17. Дан массив строк. Выровнять строки точками по длиннейшей.

Подсказки

Задачи 17, 18, 19 уже решались с обычными указателями. Следовательно, необходимо просто вспомнить уже решенное и подумать, что нового вносят связные списки.

Задача 18. Реализовать метод решета Эратосфена. Массив устроен в виде связного списка.

Задача 19. Дан символьный массив. Определить, сколько в нем различных символов. Массив устроен в виде связного списка.

Задача 20. Дано два связных списка, вставить один в другой с некоторой заданной позиции.

```
Program example;
 Uses crt;
 Type
 shablon=^zapis;
 zapis=record
        a:integer;
        next:shablon;
       end;
 Var
  u,uk,uk1_start,uk1_end,uk2_start,uk2_end:shablon;
  i,n,m,l:integer;
begin
 clrscr;
 read(n);
 new(uk1_end);uk1_start:=uk1_end;
 read(uk1_end^.a);
 for i:=1 to n-1 do
  begin
   new(uk1_end^.next);
   uk1_end:=uk1_end^.next;
   read(uk1_end^.a);
  end;
 write('m=');read(m);
 new(uk2_end);uk2_start:=uk2_end;
 read(uk2_end^.a);
 for i:=1 to m-1 do
  begin
   new(uk2_end^.next);
   uk2_end:=uk2_end^.next;
   read(uk2_end^.a);
  end;
```

```
 write('l=');read(l);
 uk:=uk1_start;
 for i:=1 to l-1 do
  uk:=uk^.next;
 u:=uk^.next;
 uk^.next:=uk2_start;
 uk2_end^.next:=u;
 for i:=1 to n+m do
  begin
   write(uk1_start^.a,'  ');
   uk1_start:=uk1_start^.next;
  end;
end.
```

Пояснения

Решение состоит в следующем:

1. Мы разрываем исходный связный список на два куска.
2. К концу первого куска приклеиваем второй связный список.
3. К концу второго связного списка приклеиваем начало второго куска от первого списка.

Операция склейки связных списков реализуется простым присваиванием адресов. А для того чтобы разорвать связный список, мы выполняем две операции:

1. Проходим связный список до той записи, с которой должна начаться вставка.
2. Запоминаем адрес следующей записи в дополнительном указателе.

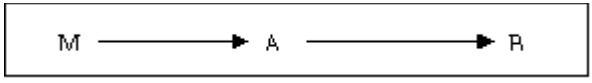
Задача 21. Дан символьный массив. Выбросить из него все символы "а". Массив устроен в виде связного списка.

Подсказки

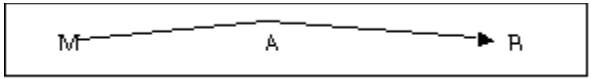
Такая задача уже решалась, но то решение мы использовать не будем. Структура связного списка позволяет не двигать весь массив. Можно переназначать адреса.

Более ничего говорить не будем, а поясним эту операцию картинкой:

Исходная ситуация



После переназначения



Задача 22. Дан связный список записей, в каждой из которых два поля: числовое, содержащее позицию записи в списке, и второе символьное, содержащее произвольный символ. Построить второй связный список, в котором:

- присутствовали бы все записи первого;
- записи, содержащие один и тот же символ, находились бы рядом;
- порядок записей, содержащих одинаковый символ, сохранялся бы.

Для того чтобы условие было более понятно, приведем пример — табл. 9.1 и 9.2.

Таблица 9.1. *Исходные данные*

1	2	3	4	5	6	7
а	а	и	а	с	и	с

Таблица 9.2. *Результат*

1	2	4	3	6	5	7
а	а	а	и	и	с	с

Подсказки

Данную задачу можно решить различными способами. Мы предложим вам следующий алгоритм: переходим на начало первого списка; первый элемент второго списка равен первому элементу первого; для всех элементов первого списка делать: переходим к следующему элементу. Если слева от данного элемента нет такого же, то продолжаем формирование второго связного списка. Для этого данный элемент заносим во второй связный список для всех элементов первого связного списка, начиная с данного. Если те-

кущий элемент равен данному, то текущий заносим во второй связный список.

Эта задача допускает более экономное решение, позволяющее обойтись без второго связного списка. Оно использует принцип из задачи 21. Суть решения в переопределении адресов. Немного его опишем. Пусть в процессе прохода связного списка мы нашли неповторяющийся слева элемент (назовем его *первый*). В тот момент, когда он был обнаружен, он связан со следующим. Разорвем эту связь и свяжем его со следующим таким же. С этим вновь найденным выполним операцию установления новой связи еще раз. Когда операцию установления новой связи выполнить не удастся, т. к. справа в связном списке таких же больше нет, то последний найденный связываем с тем, который находится за *первым*.

Это не алгоритм, это неформальное описание метода, если вам будет интересно, попробуйте довести его до четкого алгоритма и написать программу.

Задача 23. Построить двоичное дерево. Заполнить его узлы случайными числами и вывести полученное дерево на экран.

```
Program example;
 Uses crt;
 Type shablon=^zapis;
 zapis=record
        a:integer;
        left:shablon;
        right:shablon;
       end;
 Var
  tree1,tree2:shablon;
  max:integer;
procedure Create_Tree(tree:shablon;n:integer);
var
 new_tree:shablon;
begin
 if n<=max then
  begin
   new(tree^.left);
```

```
   new_tree:=tree^.left;
   new_tree^.a:=random(10);
   Create_Tree(New_tree,n+1);
   new(tree^.right);
   new_tree:=tree^.right;
   new_tree^.a:=random(10);
   Create_Tree(New_tree,n+1);
  end;
end;
procedure View_tree(tree:shablon;x,n:integer);
begin
 gotoxy(x,2*n);write(tree^.a);
 if n<=max then
  begin
   View_tree(tree^.left,x-(20 div n),n+1);
   View_tree(tree^.right,x+(20 div n),n+1);
  end;
end;
begin
 randomize;
 clrscr;
 read(max);
 new(tree1);tree1^.a:=random(10);
 tree2:=tree1;
 Create_Tree(tree1,1);
 View_Tree(Tree2,40,1);
end.
```

Пояснения

Дерево от линейного связного списка отличается тем, что записи узлов дерева связаны не с одной записью, а с двумя или больше. Если каждый узел дерева указывает на два следующих, то такое дерево называется *двоичным*. Все примеры, разобранные далее, посвящены только двоичным деревьям.

Двоичное дерево можно определить, как корневой узел, который порождает два двоичных дерева, левое и правое, поэтому естественно строить двоичное дерево посредством рекурсивной проце-

дуры, в которой создается очередной узел и процедура вызвается еще два раза для левой и правой ссылок.

Задача 24. Узлы двоичного дерева заполнены случайными числами, найти их сумму.

Подсказки

Технология построения двоичного дерева представлена ранее. Остальное просто.

Задача 25. В дереве числовых данных найти наибольшее.

Подсказки

Технически эта задача ничем не отличается от предыдущей, поэтому подсказку смотрите ранее.

Задача 26. Узлы двоичного дерева заполнены случайными числами. Найти ветвь (ветвь — это линейная последовательность узлов), сумма узлов которой наибольшая. На экран распечатать дерево и найденное значение суммы.

Подсказки

В этой задаче недостаточно уметь совершать полный обход двоичного дерева. Надо научиться каким-то образом отличать одну ветвь от другой. Визуально это просто, но мы работаем не с визуальной картинкой, что и создает затруднения. Основную работу по решению этой проблемы вам придется выполнить самостоятельно. Заметим только что:

1. При рекурсивном построении программы новый вызов рекурсивной процедуры обхода означает шаг вниз.
2. Завершение текущего вызова процедуры обхода означает шаг вверх.
3. Вызов внутри процедуры какой-либо процедуры, в том числе и рекурсивный вызов ее самой, не означает завершения данного вызова.

Задача 27. Дано двоичное дерево, узлы которого заполнены случайными числами. Для каждого узла, имеющего дочерние, поменять значения его дочерних узлов местами.

```
Program example;
 Uses crt;
 Type
 shablon=^zapis;
 zapis=record
        a:integer;
        left:shablon;
        right:shablon;
       end;
 Var
  tree1,tree2:shablon;
  max:integer;
procedure Create_Tree(tree:shablon;n:integer);
var
 new_tree:shablon;
begin
 if n<=max then
  begin
   new(tree^.left);
   new_tree:=tree^.left;
   new_tree^.a:=random(10);
   Create_Tree(New_tree,n+1);
   new(tree^.right);
   new_tree:=tree^.right;
   new_tree^.a:=random(10);
   Create_Tree(New_tree,n+1);
  end;
end;
procedure View_tree(tree:shablon;x,n,y:integer);
begin
 gotoxy(x,y);write(tree^.a);
 if n<=max then
  begin
   View_tree(tree^.left,x-(20 div n),n+1,y+3);
```

```
   View_tree(tree^.right,x+(20 div n),n+1,y+3);
  end;
end;
procedure change(tree:shablon;n:integer);
var
 left,right:shablon;
 a:integer;
begin
 left:=tree^.left;
 right:=tree^.right;
 a:=left^.a;
 left^.a:=right^.a;
 right^.a:=a;
 if n<max then
  begin
   change(left,n+1);
   change(right,n+1);
  end;
end;
begin
 randomize;
 clrscr;
 read(max);
 new(tree1);tree1^.a:=random(10);
 tree2:=tree1;
 Create_Tree(tree1,1);
 View_Tree(Tree2,40,1,1);
 change(Tree2,1);
 View_Tree(Tree2,40,1,12);
end.
```

Пояснения

Не перепутайте задачи: поменять местами значения узлов и поменять местами узлы — это разные задачи. Поменять местами узлы от вас потребуется в задаче 30.

Здесь же для каждого узла дерева необходимо выполнить следующие операции:

1. Прочитать значения левого и правого узла.
2. Значение левого узла присвоить правому. Значение правого присвоить левому.

Напомним, что обмен значениями между двумя величинами происходит так: `c:=a; a:=b; b:=c;`.

Задача 28. Записать некоторое количество двоичных чисел в двоичное дерево.

Подсказки

Предположим, у нас есть несколько двоичных чисел и несколько первых цифр у них совпадает. Это означает, что мы можем сформировать из этих цифр общую ветвь дерева. Далее несовпадение цифр означает ответвление. Покажем это на примере. Предположим, даны три числа: 110011, 111001, 111100. В виде дерева их можно записать так:

					1				
				1					
			0		1	0	0	1	
		0			1				
	1				0				
1					0				

Переходя по дереву вниз, влево или вправо, от цифры к цифре, мы получим все три числа. Заметьте, что построенное дерево не двоичное, из некоторых узлов идет два ответвления, а из корневого — только одно. Но его можно вполне организовать как двоичное. Для этого достаточно в тех узлах, в которых только одно ответвление, второе помечать как **NIL**. **NIL** — это адрес в никуда. Тогда при обходе дерева достаточно будет перед переходом по новому адресу проверять, не указывает ли он в никуда, т. е. не равен ли он **NIL**. Вывод дерева можно организовать так же, как это делалось в первых программах, посвященных работе с деревьями.

Задача 29. Решить задачу, обратную предыдущей. То есть двоичное дерево заполнено нулями и единицами. Необходимо считать из него двоичные числа.

Подсказки

Собственно эту задачу вы уже практически решили. Нужно только обдумать процедуру вывода. Сейчас требуется вывести числа, т. к. мы числа привыкли записывать и не в виде дерева.

Задача 30. Дано двоичное дерево. Поменять местами левые и правые ветви.

Подсказки

Вот сейчас нужно поменять не значения, а именно узлы. Подобную работу мы уже выполняли. Пример нужной нам операции дан в задаче 20. Идея следующая: для каждого узла дерева выполняем следующие операции:

1. Адресу левого указателя присваиваем адрес правого.
2. Адресу правого указателя присваиваем адрес левого.

Напомним, что обмен значениями между двумя величинами происходит так: `c:=a; a:=b; b:=c;`. Только в этот раз речь идет об указателях, что, впрочем, непринципиально.

Предметный указатель

У

Ф

Ц