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 1

Глава 1. Язык Паскаль 3

Основные конструкции и операторы 3
Оператор присваивания 6
Оператор цикла 6
Цикл while do 9
Цикл *repeat ... until* 9
Отличия циклов по условию от цикла с параметром 9
Условный оператор (оператор выбора между несколькими действиями) 9
Структура программы 11
Конструкция *case* 14
Заключение 15
Операции языка Паскаль 16
Арифметические операции 16
Логические операции 18
Заключение 20
Типы данных 20
Массивы 22
Записи 23
Файловые переменные 24
Типы данных, определенные пользователем 25
Константы 27
Объявление константы 27
Использование константы 27
Заключение 28
Строковые величины 29
Что отличает строку от символьного массива? 32
Заключение 33
Массивы 33
Работа с массивом как единым объектом 38
Заключение 40

Множества....41
Заключение....46
Записи....47
Вариантные записи....51
Заключение....52
Графика языка Паскаль....52
Заключение....55
Процедуры и функции....56
Процедуры без параметров....63
О передаваемых параметрах....65
Вызов процедуры/функции....67
Заключение....69
Рекурсия....70
Заключение....74
Файлы данных....74
Заключение....81
Указатели....81
Три полезные процедуры....84
Связные списки....85
Еще одна интересная проблема....86
Указатель на тип....87
Заключение....88

Глава 2. Организация диалога с программой....91

Глава 3. Статическая графика....111

Глава 4. Динамическая графика....127

Глава 5. Вычислительные задачи....147

Глава 6. Работа с массивами....161

Глава 7. Операции со строковыми данными....179

Глава 8. Работа с файлами....195

Глава 9. Динамические величины....211

Предметный указатель....234

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.