

Файлы

Файловый тип принципиально отличается от других типов данных, с которыми мы познакомились в главе 3. Именно поэтому файлам посвящена отдельная глава.

Что такое файл

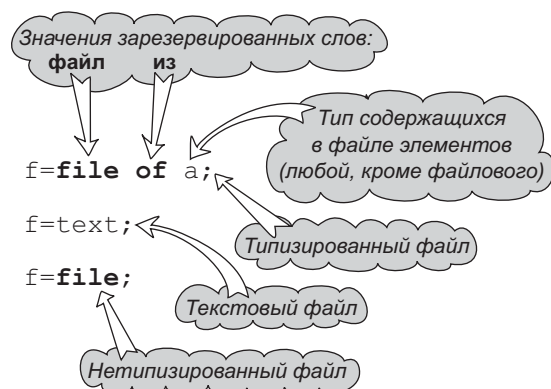
Типы данных, о которых до сих пор шла речь, предназначены для манипулирования информацией, содержащейся в оперативной памяти компьютера. Однако, как известно, оперативная память способна хранить данные только временно — пока компьютер включен. Очевидно, что для эффективной работы нужен какой-то способ продолжительного хранения информации, чтобы ею можно было воспользоваться после выключения и включения компьютера снова. И такой способ существует. Для долговременного хранения информация из оперативной памяти переносится в *файлы*¹. Файл представляет собой некоторое поименованное место на внешнем носителе. В качестве внешнего носителя могут служить разного рода диски (жесткие, гибкие, магнитооптические, компакт-диски, Zip- и Jaz-диски и т.д.).

Кроме “долговременности” у файлов имеется еще одна отличительная особенность: их неопределенный объем (или длина). Если для каждого из прочих структурированных типов всегда точно определено, сколько элементов содержит то или иное значение, то, сколько элементов должно быть в файле, при объявлении файлового типа не указывается. Максимальная длина файла ограничивается только свободным местом на диске, и это является основным отличием файлов от массивов.

¹ Английское слово *File* переводится как архив, подшивка, картотека — иными словами, хранилище информации.

Файловые типы и файловые переменные

Все сказанное выше относится к долговременному хранению информации в компьютерах вообще. А как это реализовано в Turbo Pascal? Для сохранения информации в Turbo Pascal предусмотрена возможность определять файловые типы и файловые переменные. После этого информацию можно перенести в файл на диске (подробностям, как это делается, и посвящена данная глава). Существует три способа определения файлового типа (или файловой переменной).



Эти три способа соответствуют трем видам файлов, которыми позволяет оперировать Turbo Pascal. Речь идет о *типизированных*, *текстовых* и *нетипизированных* файлах.

Text — это не зарезервированное слово, а идентификатор стандартного типа данных — как, например, Integer или Real.

f — имя объявляемого файлового типа или файловой переменной.

a — тип элементов, содержащихся в файле (любой тип, кроме файлового).

Элементы файла могут представлять собой значения любого типа, за исключением файлового либо структурированного типа, элементами которого являются файлы, т.е. существование “файла файлов” не допускается. Вот примеры описаний файловых переменных.

```

type
  arrays=array of integer;
  Date=record
    Day:1..31;           {Число}
    Month:1..12;         {Месяц}
    Year:1900..2000     {Год}
  end;                  {date}
var
  f1:file; ← {Нетипизированный файл}
  f2:text; ← {Текстовый файл}
  f3:file of integer; }
  f4:file of arrays; } ← {Типизированные файлы}
  f5:file of date; }

```

Выше объявлены пять файловых переменных. Причем если среди типизированных файлов элементы файла *f3* относятся к стандартному типу (*Integer*), то элементы файлов *f4* и *f5* — к типам, объявленным пользователем (тип *Arrays* представляет собой массив целых чисел, а тип *Date* — запись).



Файлы, принадлежащие разным видам, имеют свои особенности, однако существует и нечто общее, присущее всем файлам. Так, в файле в каждый момент может быть доступен только один элемент.

Например, если вернуться к объявленным выше файловым переменным, то в файле *f3* каждую минуту можно иметь доступ к одному из целых чисел, из которых состоит файл; в файле *f4* — к единственному массиву; наконец, в файле *f5* — к отдельно взятой записи. Кроме того, файлы всех видов завершаются маркером конца файла (EoF — End of File).

Доступ к элементам файла обычно осуществляется последовательно. Иными словами, для того чтобы “добраться” до последнего элемента, прежде придется обработать (считать или записать) все предыдущие. Кроме того, для типизированных и нетипизированных файлов возможен переход к определенному элементу (минуя предыдущие — об этом далее).



Число элементов файла определяет его объем, или длину. Как уже отмечалось, длина файла при его объявлении не фиксируется. При этом допускается существование файлов, не содержащих ни одного элемента — такие файлы называются пустыми (сравните: пустое множество).

Выше указывалось, что информация (как правило) записывается на дисках. На поверхности диска имеются дорожки, расположенные по концентрическим окружностям, которые и содержат информацию (рис. 5.1). (Разумеется, если взглянуть на поверхность реального диска, никакой информации и дорожек разглядеть не удастся — так же, как на магнитофонной ленте не видно музыки.)

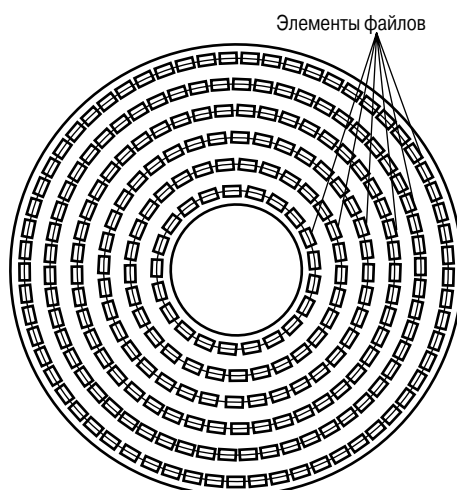


Рис. 5.1. Поверхность диска и дорожки

Хорошо бы так же наглядно изобразить файл с информацией. Как указывалось выше, информация на диске записывается на дорожках, располагающихся по концентрическим окружностям. Поэтому логично представить файл в виде части дорожки — сектора (рис. 5.2).

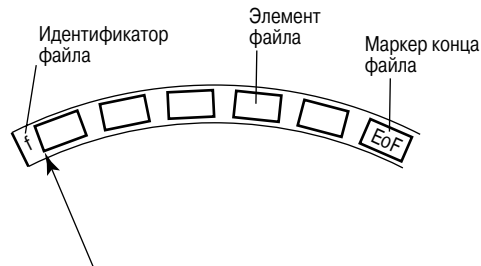


Рис. 5.2. Визуальное представление файла

Изображенный файл содержит пять элементов. Начинается файл идентификатором, который обозначен буквой *f*. Далее следуют несколько элементов информации, которая содержится в файле, и завершается файл маркером конца файла (EoF). Элементы, образующие файл, представляют собой значения, принадлежащие любому простому или структурированному типу, за исключением файлового. Стрелка на рисунке представляет собой *указатель текущей позиции* файла. Если инициировать запись в файл или считывание из файла, запись будет проводиться начиная с элемента, перед которым расположен указатель текущей позиции. Это же касается и считывания. Чтобы подчеркнуть, что длина файла не ограничена, изображенный здесь сектор справа не замкнут.

Разумеется, данное визуальное представление файла очень условно. Часто файл занимает не часть дорожки, а несколько дорожек. Кроме того, никакой стрелки в реальности не существует. Тем не менее представленные изображения должны помочь читателю лучше понять принципы сохранения данных в файлах на диске.

Операции над файлами

В отличие от других типов данных, в Turbo Pascal нет встроенных операторов, предназначенных для манипулирования файлами. Например, не удастся с помощью оператора присваивания присвоить файловой переменной некоторое значение. В силу этого соответствующие операции реализованы в виде процедур и функций.

Организация доступа к файлам

Поскольку файлы, в отличие от данных других типов, содержатся в так называемой *вторичной памяти* (т.е. на дисках), для доступа к файлам недостаточно только объявить файловую переменную.

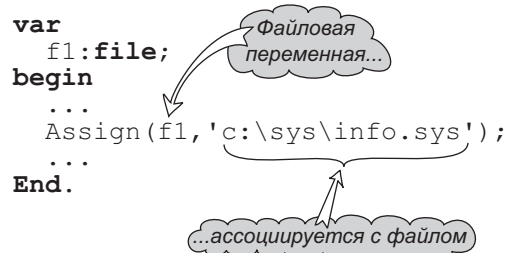
Файловые переменные и реальные файлы

Присмотримся к объявленным выше файловым переменным *f1-f5*. Здесь для каждой переменной указан идентификатор, вид соответствующего файла и тип данных, которому должны принадлежать элементы того или иного файла (если данный файл относится к типизированным). Достаточно ли этого для манипулирования содержимым файлов? И если да, то каким образом действия над файловой переменной (которая представляет некоторую область памяти) смогут повлиять на конкретный файл на диске? Ведь при объявлении файловой переменной не указывается ни имя файла, ни в каком каталоге он содержится.



Для того чтобы конкретный файл стал доступным, его необходимо как-то связать (или ассоциировать) с ранее объявленной файловой переменной. Такое связывание осуществляется с помощью процедуры Assign (см. приложение В), которая является стандартной процедурой Turbo Pascal.

Вот как можно в программе на Turbo Pascal ассоциировать файл на диске с объявленной файловой переменной.



Здесь файловая переменная *f1* ассоциируется с файлом INFO.SYS, содержащимся в каталоге C:\SYS.



До выполнения над файлом каких-либо действий всегда необходимо обратиться к процедуре Assign. После вызова процедуры Assign связь между указанными файлом и файловой переменной существует вплоть до завершения работы программы либо пока к этой файловой переменной снова не будет применена процедура Assign.

Открытие, закрытие и удаление файла

После связывания некоторого реального файла с файловой переменной, чтобы получить доступ к содержимому этого файла, его необходимо *открыть*. Открыть файл в Turbo Pascal можно для чтения, записи, а также для чтения и записи одновременно. При открытии файла ищется уже существующий файл на диске либо создается новый (если файл открывается для записи), и указатель текущей позиции устанавливается в начало файла.

Для открытия файлов предназначены процедуры Reset и Rewrite (см. приложение В); данные процедуры применимы к файлам любого вида.



Файл, открываемый с помощью процедуры Reset, непременно должен уже существовать. Если файла с таким именем и в указанном каталоге не окажется, имеет место ошибка.

Файл, открытый с помощью процедуры Reset, изображен на рис. 5.3.

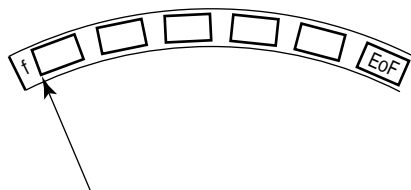


Рис. 5.3. После открытия файла с помощью процедуры Reset, указатель текущей позиции расположен перед первым элементом



Чем процедура Rewrite отличается от Reset применительно к типизированным или нетипизированным файлам? Если процедуру Reset применить к несуществующему файлу, будет иметь место ошибка (процедура Rewrite в этом случае создаст новый файл). Если же процедуру Reset применить к существующему на диске файлу, файл будет открыт (процедура Rewrite при этом удалит содержимое старого файла и создаст новый (пустой) файл с тем же именем).



Поскольку при применении процедуры Rewrite к уже существующему файлу его содержимое удаляется (без всякого предупреждающего сообщения), а затем создается новый (пустой) файл с тем же именем, пользоваться данной процедурой следует осторожно.

Файл, открытый с помощью процедуры Rewrite, изображен на рис. 5.4.

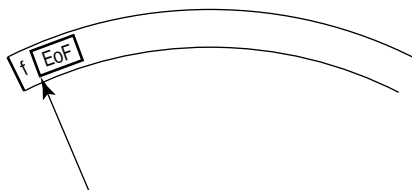


Рис. 5.4. После применения процедуры Rewrite открывается пустой файл, в котором указатель текущей позиции расположен перед маркером конца файла

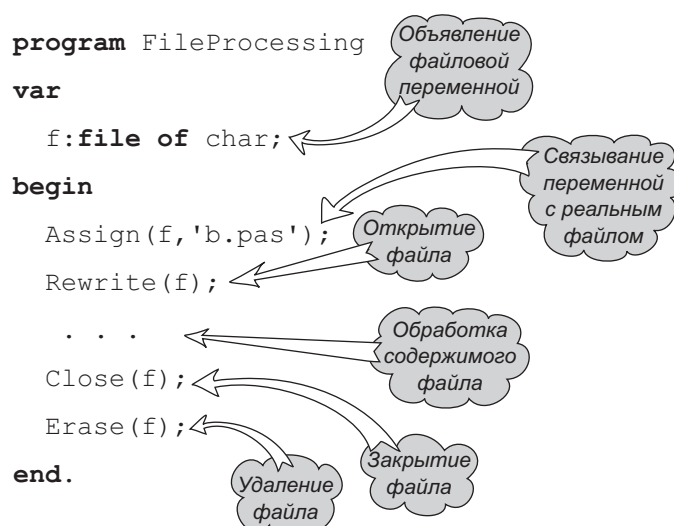
После открытия файла (с помощью процедуры Reset или Rewrite) осуществляется обработка его содержимого, по завершении которой файл закрывается и, иногда, удаляется (если данный файл вспомогательный).



Для закрытия файлов предназначена процедура Close, а для удаления — процедура Erase (см. приложение В); данные процедуры применимы к файлам любого вида.

В принципе, процедуру Close для закрытия файлов применять не обязательно — все открытые файлы при нормальном завершении программы закрываются автоматически, однако использование процедуры Close все же считается признаком хорошего тона. Кроме того, может понадобиться закрыть файл с помощью процедуры Close, чтобы затем применить для его открытия иную процедуру (вместо Reset — Rewrite либо наоборот).

Вот как схематически можно представить программу, в которой открывается (вернее, создается) некоторый файл, осуществляется его обработка, затем файл закрывается и удаляется.



Здесь объявляется файловая переменная f , принадлежащая типу FILE OF CHAR (соответствующий файл содержит последовательность символов). После этого данная переменная ассоциируется с файлом В.PAS (произвольный файл). Затем файл открывается (с помощью процедуры Rewrite). Далее содержимое файла подвергается некоторой обработке. Наконец, файл В.PAS закрывается (процедура Close) и удаляется (процедура Erase).

Действия, представленные на схеме программы многоточием, являются операциями записи-считывания или иными операциями, обрабатывающими содержимое файла (см. далее).

Запись-считывание

В Turbo Pascal для осуществления записи информации в файл и считывания из файла для текстовых и типизированных файлов предназначены процедуры `Read` и `Write`, а для нетипизированных — `BlockRead` и `BlockWrite` (см. приложение В). Причем использование процедур `Read` и `Write` имеет особенности, зависящие от вида файла (текстовый это файл или типизированный). Поэтому использование упомянутых процедур рассматривается в разделах, посвященных соответствующим видам файлов.

Помимо `Read` и `Write`, к текстовым файлам для считывания и записи применимы процедуры `ReadLn` и `WriteLn` (см. приложение В). Эти процедуры рассматриваются в разделе, посвященном текстовым файлам.

Манипулирование содержимым файла

Предположим, объявлена файловая переменная, которая затем ассоциирована с некоторым файлом на диске. После этого соответствующий файл был открыт с помощью процедуры `Reset` или `Rewrite`. Что дальше? Что можно делать с содержимым файла помимо записи или считывания (см. предыдущий раздел)? С некоторыми операциями, применимыми к содержимому файлов, мы познакомимся в данном разделе.

Примечание. Рассматриваемые в данном разделе стандартные подпрограммы применимы ко всем видам файлов (или их большинству). Исключения указаны явно. Подпрограммы, осуществляющие обработку содержимого файла и применимые к какому-то одному виду файлов либо применение которых отличается в зависимости от вида файла, рассматриваются в разделе, посвященном соответствующему виду, далее в этой главе.

Поэлементная обработка файла

Предположим, каждый элемент некоторого файла нужно как-то обработать. Например, если элементы файла — числа, то нужно умножить каждое из них на некоторый коэффициент, а если это строки — добавить в конец каждой строки некоторую подстроку. Для решения подобной задачи необходимо открыть файл, а затем осуществить обработку каждого его элемента, пока не будет достигнута метка конца файла. Но как определить, когда дос-

тигнут конец файла (т.е. когда указатель текущей позиции расположен перед меткой конца)? Для этого в Turbo Pascal предназначена стандартная функция EoF (см. приложение В).



Функция EoF возвращает значение TRUE, если указатель текущей позиции находится за последним элементом файла либо если файл пустой. В противном случае функция возвращает значение FALSE.

Присмотримся к трем изображениям на рис. 5.5.

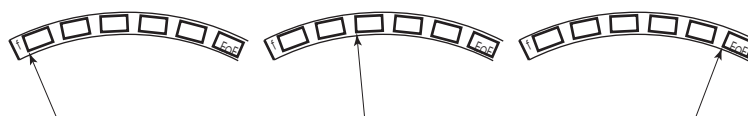


Рис. 5.5. Указатель текущей позиции расположен в начале, в середине и в конце файла

Для представленных здесь трех ситуаций функция EoF в первом и во втором случаях вернет значение FALSE, а в третьем — TRUE.

Усечение файла

Иногда возникает задача удалить конец файла начиная с определенной позиции. Для подобных задач в Turbo Pascal существует стандартная процедура Truncate (см. приложение В).



Процедура Truncate, будучи применена к открытому файлу, удаляет его содержимое от указателя текущей позиции и до конца.

Внимательный читатель может заметить, что должна существовать возможность позиционировать указатель в файле перед определенным элементом. Вообще-то, в Turbo Pascal для этого предназначена специальная процедура Seek (см. следующий раздел). Однако также можно с помощью оператора цикла с параметром (FOR) организовать считывание определенного числа элементов в файле (типизированном или нетипизированном), а затем применить процедуру Truncate. Например, если требуется, чтобы в файле содержались только 30 элементов, а оставшаяся часть файла, начиная с 31-го элемента, нужно удалить, это можно осуществить с помощью следующих операторов:

```
for i:=1 to 31 do
  Read(f, a);
Truncate(f);
```

Размер файла, положение и перемещение указателя в файле

Часто для решения различных задач необходимо выяснить размер файла, определить, перед каким элементом в файле расположен указатель текущей позиции, либо переместить этот указатель к определенному элементу. Осуществить это можно с помощью соответственно функций `FileSize` и `FilePos` либо процедуры `Seek` (см. приложение В).

Например, для того чтобы переместить указатель текущей позиции к третьему элементу файла, ассоциированного с файловой переменной `f`, процедуру `Seek` следует применить так.

```
Seek(f, 3);
```

Файл, к которому таким образом применена процедура `Seek`, изображен на рис. 5.6.

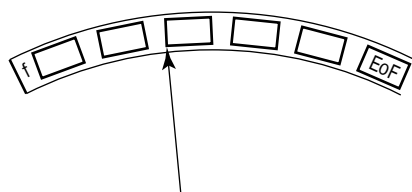


Рис. 5.6. Процедура `Seek` позиционировала указатель к заданному элементу файла

А если к файлу, изображенному на рис. 5.6, применить функцию `FilePos`, данная функция возвратит значение 3. (Функция `FileSize`, примененная к этому же файлу, вернет значение 5.)

Виды файлов

В начале данной главы указывалось, что Turbo Pascal оперирует файлами трех видов: типизированными, текстовыми и нетипизированными. Примеры объявления файловых переменных для каждого из трех видов представлены в табл. 5.1.

Таблица 5.1. Объявление файловых переменных для различных видов файлов

Вид файла	Содержимое файла	Примеры описания переменных
Типизированный	Совокупность элементов, принадлежащих любому типу, за исключением файлового	<code>f1:file of Integer;</code> <code>f2:file of Real;</code> <code>f3:file of Boolean;</code>
Текстовый	Совокупность строк	<code>f:Text;</code>
Нетипизированный	Последовательность элементов произвольного типа	<code>f:file;</code>

Типизированные файлы

Итак, типизированный файл содержит совокупность элементов, принадлежащих типу, который определен при объявлении файловой переменной. Доступ к элементам файла, как правило, организуется последовательно. Можно также переместить указатель текущей позиции (с помощью процедуры `Seek`) к любому элементу файла.

Для того чтобы файл открыть как типизированный, его необходимо ассоциировать с файловой переменной, имеющей тип `FILE OF...`. Вместо трех точек здесь может присутствовать любой из простых или структурированных типов, как стандартных, так и созданных пользователем, за исключением файлового либо структурированного типа, элементами которого являются файлы (т.е. существование “файла файлов” не допускается).

Для манипулирования типизированными файлами подходят все процедуры и функции, о которых шла речь выше. Здесь необходимо заметить, что использование процедур `Read` и `Write` для типизированных файлов имеет особенности.

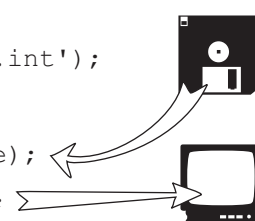


При использовании для типизированных файлов, процедура `Read` (см. приложение В) считывает один или несколько компонентов файла, а затем считанное значение (значения) присваивается некоторой переменной (переменным).

```

var a,b,c,d,e,f:integer;
    f1,f2:file of integer;
begin
  Assign(f1,'file1.int');
  Reset(f1);
  Read(f1,a,b,c,d,e);
  Write(a,b,c,d,e);

```



(Здесь пять первых компонентов файла $f1$ считываются и присваиваются в качестве значений переменным a, b, c, d и e , а затем значения этих переменных выводятся на экран.)

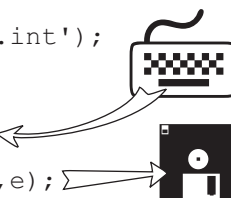


А при использовании для типизированных файлов процедуры `Write`, значение некоторой переменной (или переменных) записывается в файл в качестве еще одного компонента (нескольких новых компонентов).

```

Assign(f2,'file2.int');
Rewrite(f2);
Read(a,b,c,d,e);
Write(f2,a,b,c,d,e);

```



(Здесь значения для переменных a, b, c, d и e считываются с клавиатуры, а затем записываются в файл $f2$.)

После записи каждой переменной, представленной в качестве параметра процедуры `Write` (очередному элементу файла при этом присваивается значение переменной, т.е. старое значение элемента заменяется новым), указатель текущей позиции файла перемещается к следующему элементу. Если указатель находится в конце файла, то при записи очередного элемента этот элемент дополняет файл.


Текстовые файлы

Текстовый файл представляет собой последовательность строк разной длины, состоящих из символов. Каждая строка текстового файла оканчивается маркером конца строки (End of Line — EoLn), а завершает текстовый файл, как и любой другой, маркер конца файла (End of File — EoF). К элементам текстового файла возможен только последовательный доступ, начиная с первого. Для того чтобы файл открыть как текстовый, его необходимо ассоциировать с файловой переменной, имеющей тип Text.


Для манипулирования текстовыми файлами подходят только те из рассматриваемых процедур, в описании которых (см. приложение В) это указывается прямо. Необходимо заметить, что использование процедур Read и Write с текстовыми файлами имеет особенности, которые мы выясним ниже. Кроме того, к текстовым файлам (и только к ним) применимы процедуры ReadLn и WriteLn, которые также будут рассмотрены. Однако прежде познакомимся с так называемыми *стандартными текстовыми файлами*.

Стандартные текстовые файлы

В приведенных выше двух примерах использования процедур Read и Write с типизированными файлами, также используются операторы вывода на экран и ввода с клавиатуры, которые выглядят так.

`Write(a, b, c, d, e);` 

и

`Read(a, b, c, d, e);` 

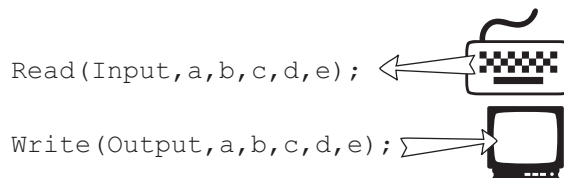
Первый оператор выводит на экран значения переменных *a*, *b*, *c*, *d* и *e*. Второй оператор позволяет ввести с клавиатуры значения, которые присваиваются переменным *a*, *b*, *c*, *d* и *e*.

Возникает вопрос, не имеют ли эти операторы какого-либо отношения к процедурам ввода-вывода, применимым к файлам? Оказывается, имеют, и самое непосредственное.



Дело в том, что в Turbo Pascal существует два стандартных идентификатора, играющих роль текстовых файловых переменных, ассоциируемых вместо файлов с конкретными физическими устройствами компьютера. Речь идет об идентификаторах INPUT и OUTPUT, которые ассоциируются соответственно с клавиатурой и экраном компьютера. Указанные файлы считаются постоянно открытыми (т.е. для них не нужно использовать процедуры Rewrite и Reset), и эти идентификаторы можно использовать в программах.

Так, операторы ввода-вывода, о которых шла речь выше, можно представить и иначе.



Первый оператор считывает из файла INPUT (т.е. позволяет ввести с клавиатуры) значения, которые будут присвоены переменным *a*, *b*, *c*, *d* и *e*. Второй оператор значения тех же переменных записывает в файл OUTPUT (т.е. выводит на экран). Первая и вторая пары операторов эквивалентны.



Иными словами, в процедурах Read и Write, когда требуется осуществить ввод с клавиатуры или вывод на экран, указывать файл (INPUT или OUTPUT) необязательно. Если в процедуре Read или Write файл не указан, по умолчанию подразумевается файл INPUT или OUTPUT, в зависимости от того, ввод или вывод инициируется.

Текстовые файлы: запись-считывание

Для того чтобы лучше понять, как функционирует процедура Read с текстовыми файлами, целесообразно воспользоваться файлом INPUT. (Мы помним, что это такой же текстовый файл, как и любой другой, только его не нужно открывать. И, кроме того, считывание из файла INPUT — это не что иное, как ввод с клавиатуры.)



Процедура `Read`, будучи применена к текстовому файлу, считывает одно или несколько значений файла и присваивает считанное значение (значения) переменной (переменным), указанной при вызове процедуры. Значения, о которых идет речь, могут принадлежать типам `Char`, `STRING`, а также любым целочисленным или вещественным типам.

Вот как происходит считывание из текстового файла.

- **Считывание значений типа `Char`.** Если это переменная (переменные) типа `Char`, из файла считывается символ и присваивается переменной, затем считывается следующий символ и присваивается следующей переменной — и так до тех пор, пока всем переменным типа `Char`, указанным при вызове процедуры `Read`, не будут присвоены считанные из файла значения. (При вводе с клавиатуры между вводимыми значениями требуется некоторый разделитель — пробел, символ табуляции (клавиша `<Tab>`) или конца строки (клавиша `<Enter>`.) Если очередной считанный символ окажется маркером конца строки `EoLn` (при вводе с клавиатуры для этого достаточно нажать клавишу `<Enter>`), считывание будет продолжено из новой строки. Если очередной считанный символ окажется маркером конца файла `EoF` (при вводе с клавиатуры для этого достаточно воспользоваться комбинацией клавиш `<Ctrl+Z>`), выполнение процедуры `Read` будет прекращено и программа перейдет к следующему оператору.
- **Считывание значений типа `STRING`.** Если это строковая переменная (переменные), из файла будут считаны все символы до ближайшего маркера конца строки. Если длина считанной строки превзойдет допустимую для значений типа `STRING` величину (255 символов), все оставшиеся до конца строки байты отбрасываются. Затем при каждом новом обращении к процедуре `Read` будет считываться маркер конца строки, т.е. строка нулевой длины. Иными словами, с помощью процедуры `Read` не удастся считать несколько строк подряд. Для того чтобы считать несколько строковых значений, следует несколько раз обратиться к процедуре `ReadLn` (об этой процедуре речь пойдет дальше в этой главе).

- **Считывание числовых значений.** Если это значение типа Integer или Real, процедура Read будет пропускать любые пробелы, символы табуляции или маркеры конца строки, предшествующие числовой строке. Когда будет обнаружена первая цифра, процедура Read начнет формировать числовое значение, пока не встретится один из перечисленных символов (пробел, символ табуляции или маркер конца строки). Считанная таким образом последовательность цифр рассматривается как символьное представление соответствующего числа, и полученное значение присваивается соответствующей переменной. Если числовая строка не соответствует ожидаемому формату, имеет место ошибка ввода-вывода. Следующая процедура Read начнет считывание с пробела, символа табуляции или маркера конца строки, которым была завершена предыдущая числовая строка.



Процедура ReadLn от Read отличается только тем, что после выполнения действий, присущих процедуре Read, осуществляется переход к следующей строке файла.



Процедура Write, будучи применена к текстовому файлу, осуществляет запись в файл значения (значений) одной или нескольких переменных, указанных при вызове процедуры. Значения, о которых идет речь, могут принадлежать типам Char, STRING, Boolean, а также любым целочисленным или вещественным типам. Кроме того, это могут быть значения перечислимых типов, созданных на основе Char или одного из целочисленных типов. При этом файл непременно должен быть открыт для записи.

Если указатель текущей позиции расположен в начале файла, новая информация записывается поверх старой. Если же указатель находится в конце файла, то новая информация дополняет содержимое файла.

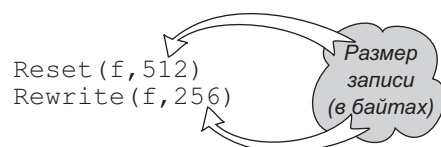


Процедура WriteLn отличается от Write тем, что после выполнения действий, присущих процедуре Write, осуществляется запись в файл маркера конца строки. При этом файл, к которому применяется процедура WriteLn, должен быть открыт для записи.

Нетипизированные файлы

Отличие таких файлов в том, что при их объявлении не определяется тип компонентов. Это позволяет получать доступ к файлам с любой структурой. При открытии файла (процедурой `Reset` или `Rewrite`) необходимо только указать размер элементов (записей²), которыми будет проводиться обмен с файлом. При этом файл трактуется как последовательность этих записей (т.е. элементов произвольного типа).

Вот как может выглядеть обращение к процедурам `Reset` и `Rewrite` для открытия нетипизированного файла.



Здесь f — файловая переменная, имеющая тип `FILE` (что соответствует нетипизированным файлам). А в качестве второго параметра процедур `Reset` и `Rewrite` указаны размеры записей (в байтах), которыми будет производиться считывание из файла или запись в файл данных. Поскольку это значение представляет собой выражение типа `Word`, его максимальная величина равна 65535 байт (предельное значение для этого целочисленного типа). Если второй параметр процедур `Reset` и `Rewrite` не задан, по умолчанию длина записи считается равной 128 байт.



При определении длины записи следует принять во внимание, что длина файла может оказаться не кратна этой величине. В этом случае последняя запись будет неполной и несколько последних байтов файла могут быть не считаны. Чтобы этого не случилось, можно установить длину записи равной единице.

² Записи (*blocks*), о которых здесь идет речь, не имеют ничего общего со структурированным типом, с которым мы познакомились в главе 4. Те записи (*records*) представляют собой совокупность полей, принадлежащих разным типам. Записи, применительно к нетипизированным файлам, — это некоторые условные элементы, из которых якобы состоит файл. Этот условный элемент определяется при открытии файла процедурой `Reset` или `Rewrite`.

Для манипулирования нетипизированными файлами подходят те же процедуры, что и для типизированных, за исключением Read и Write. Для обмена данными с нетипизированными файлами применяются специальные процедуры: BlockRead и BlockWrite (см. приложение В).



Процедура BlockRead считывает из файла указанное количество записей. При этом количество считываемых байтов равно числу считываемых записей, умноженному на размер записи, определенный при открытии файла процедурой Reset или Rewrite (если размер записи определен не был, по умолчанию он принимается равным 128 байт). В случае, если это значение окажется больше 64 Кбайт (или 65535 байт — предел для значений типа Word), будет зафиксирована ошибка. Иными словами, если при открытии файла длина записи была определена максимальной (65535 байт), то считывание возможно только по одной записи. Файл, из которого происходит считывание, предварительно должен быть открыт.



Процедура BlockWrite добавляет в файл одну или несколько записей. При этом, как и для процедуры BlockRead, количество записываемых байтов равно числу добавляемых записей, умноженному на размер записи, определенный при открытии файла процедурой Reset или Rewrite.

Нетипизированный файл, из которого осуществляется считывание или в который выполняется запись, предварительно должен быть открыт.

Резюме

Для долговременного хранения информация из оперативной памяти переносится в файлы. В этой главе читатель познакомился с принятыми в Turbo Pascal видами файлов (типизированными, текстовыми и нетипизированными), научился открывать и закрывать их, осуществлять запись-считывание, а также иным образом манипулировать содержимым файлов.