15. Поток и внешние файлы (Files).

До сих пор ввод значений величин с помощью оператора «Read» осуществлялся с клавиатуры, а вывод результатов (с помощью оператора «Write») — на экран дисплея.

Рассмотрим несколько примеров, когда это обстоятельство может создать неудобства:

1) Входных данных, которые должны быть переданы программе, слишком много;

2) Программу надо запускать несколько раз с одними и теми же начальными данными, например, при тестировке;

3) Результаты выполнения программы нужно хранить долгосрочно или распечатать на принтере;

4) Входные данные для программы были получены в результате выполнения другой программы и хранятся в отдельном файле на диске.

Все эти проблемы в Турбо-Паскале можно решить с помощью использования внешних *файлов* (File), так, чтобы как начальные данные для ввода, так и выходные результаты для вывода хранить в файлах.

Термин файл используется в более широком смысле (иногда употребляется термин *поток* — stream). Он связывается либо с физическим файлом на диске, либо с периферийными устройствами, которые фактически рассматриваются как однотипные логические элементы компьютера, что существенно облегчает обращение к этим элементам.

Клавиатура в Турбо-Паскале именуется как *Input* (устройство для ввода), а экран – *Output* (выходное устройство). В Турбо-Паскале к ним можно обращаться непосредственно, но любой другой поток нужно предварительно объявить в блоке описаний, определяя для него подходящий тип, а затем присоединяя каждому конкретному файлу физическое имя.

Файл – это набор информации, которая в Турбо-Паскале может быть представлена как

- текст (последовательность символов),
- последовательность групп одинакового или различного типа данных,

• произвольный набор данных.

Мы рассмотрим три основных вида файлов, которые поддерживаются в Турбо-Паскале:

	Примеры объявлений
Текстовой	Var f:Text;
Типизированный	Var f_i: File of Integer; Var f_r: File of Real; Var f_st: File of String; Var f_ch: File of Char; Var f_rec: File of Record x,y,z:Real End;
Нетипизированный	Var f:File;

Перед тем, как обратиться к файлу, к нему надо привязать идентификатор-переменную, которому в разделе объявлений назначается тип. Имя файловой переменной конструируется согласно общим правилам формирования идентификаторов.

15.1. Основные процедуры для обработки файлов.

Основными операциями, которые могут быть реализованы на файлах, являются:

Присвоение имени файлу;
Создание файла;
Открытие файла;
Чтение/запись из/в файл;
Закрытие файла.

В общем случае, для различных видов файлов процедуры, реализующие соответствующие функции, могут быть различными. Кроме того, для каждого из видов файлов определены специфические процедуры разного назначения.

Сначала рассмотрим общие для всех видов файлов процедуры и функции.

15.1.1. Назначение имени файлу.

Пусть к физическому файлу присоединена файловая переменная *f*. На этом этапе не имеет значения, какого типа именно данный файл. Назначение имени файлу осуществляется с помощью процедуры:

```
Assign(f,'Имя — физического — файла');
```

После вызова данной процедуры файловая переменная *f* будет ассоциироваться с физическим файлом под именем «Имя—физического—файла». Следовательно, открытие, чтение или запись данных, а также любые модификации будут касаться именно этого файла.

В качестве имени файла может служить как строковая константа, так и предварительно объявленная строковая переменная. Это позволяет выбирать файл для обработки в процессе выполнения программы. Например,

```
Var f:Text;
    name:String[30];
Begin
    Writeln('Введите имя файла:');
    Readln(name);
    Assign(f, name);
    ...
    { Фрагмент для обработки файла}
End.
```

Программный фрагмент объявляет файловую переменную *f* для представления текстового файла, имя которого может быть определено во время каждого запуска программы.

Можно задать имя файла непосредственно в тексте программы, например, Assign (*f*, 'My_ File txt');

Тут же заметим, что оператор: Assign (*f*,' '); привязывает файловую переменную *f* к экрану дисплея.

15.1.2. Создание файла.

В Турбо-Паскале можно создать новый файл программным путем. Для этого используется процедура

```
ReWrite(f);
```

которая *открывает новый файл для записи.* Т.е. после вызова этой процедуры можно уже непосредственно записывать информацию в новый файл, имя которого должно быть уже определено предварительно процедурой «Assign».

Здесь нужно сохранять предельную осторожность, так как если применить процедуру «ReWrite» к уже существующему файлу, вся хранимая в нем информация потеряется.

После открытия файла с помощью процедуры «ReWrite» т.н. *файловый курсор* (аналогично курсору на экране) устанавливается в самом начале файла. При заполнении файла данными, курсор будет продвигаться дальше, точно так же, как при выводе результатов на экран, курсор на дисплее каждый раз перемещается вправо по мере поступления новых данных для вывода. Фактически «ReWrite» «открывает новую пустую тетрадь» для записи информации.

15.1.3. Открытие файла для чтения.

Чтобы получить информацию, хранимую в определенном файле, нужно *открыть файл для чтения.* Это осуществляется процедурой:

```
ReSet(f);
```

При этом файловый курсор устанавливается в начале файла, и продвигается вперед по мере последовательного чтения данных из него.

Если данную процедуру применить к несуществующему файлу, то произойдет логическая ошибка, которая прервет выполнение программы. Но этого можно избежать, если использовать

возможности контролирования режима компилятора с помощью директив компилятора точно так же, как и в случае ввода числовых данных. Директива «{$I-}» устанавливает такой режим компилирования, в котором механизм проверки на существование файла не прерывает выполнения программы. Если физического файла все же не существует, функция *IoResult* будет отличаться от нулевого значения. Эта возможность способствует большей гибкости при обработке файлов. Например, при открытии файла можно каждый раз проверять его на существование и, исходя из результатов проверки, предпринимать те или иные шаги для обработки данного файла. Рассмотрим программный фрагмент:

```
Uses Crt;
Var f:Text;
    name:String[30];
    c:Char;
    error: Word;
BEGIN {Main}
    Writeln('Введите имя файла:');
    Readln(name);
    Assign(f, name);
    {$I-} ReSet(f); {$I+} {Проверка на существование файла}
    error:= IoResult;
If error <> 0
Then Begin
    Writeln(' Такого файла нет!!!');
    Writeln(' Код ошибки =', error,'.');
    c:=ReadKey;
    Halt
    End
Else { Файл существует -> Фрагмент его обработки }
END. {Main}
```

Можно сказать, что с помощью «ReSet» мы как бы «открываем книгу для чтения», которую собираемся читать с самого начала.

15.1.4. Чтение/запись из/в файл.

Для чтения начальных данных из текстового или типизированного файла, а также для вывода в файл результатов вычислений, используются стандартные процедуры «Read» и «Write». Разница в формате между вводом/выводом с клавиатуры на экран и из/в файл заключается только в том, что в первом случае файловая переменная просто отсутствует. Стандартный формат этих операторов можно представить в виде:

```
Read(Файловая — переменная, Список — параметров);
Write(Файловая — переменная, Список — параметров);
```

Чтобы считать данные из файла, его нужно *открыть для чтения*. При этом файловый курсор устанавливается в самом начале файла, а по мере чтения данных курсор перемещается на соответствующее количество позиций,элементов.

Чтобы записать данные в файл, его нужно *открыть для записи*. При этом данные в файл будут заноситься начиная с текущей позиции файлового курсора.

Так как текстовые и типизированные файлы имеют различную организацию хранения данных, процедуры чтения и записи для них несколько отличаются.

15.1.5. Закрытие файла.

После завершения обработки или модификации файла его нужно *закрыть*, наподобие тому, как мы закрываем книгу после завершения чтения или тетрадь после того, как туда записали нужную информацию. В Турбо-Паскале файл закрывается с помощью процедуры:

```
Close(f);
```

которая сообщает программе, что работа с файлом завершена, и, следовательно, освобождается память, отведенная для хранения промежуточных данных, связанных с обработкой файла. Кроме того, она записывает в файл специальный символ-маркер «конец файла».

После вызова этой процедуры данный файл уже не может быть обработан, пока он снова не будет открыт для чтения или записи.

Если вы забыли закрыть файл, то после выхода из Турбо-Паскалевской среды он автоматически закроется операционной системой. Считается хорошим тоном в обязательном порядке закрывать все открытые файлы, кроме того, если этого не сделать, записываемая в файл информация может частично потеряться. Это происходит из-за того, что процедура «Write» записывает данные не прямо в файл, а во временную динамическую область, которая служит буфером для обмена информацией. Только после закрытия файла буфер освобождается и данные полностью переносятся непосредственно в файл.

15.1.6. Индикация конца файла.

В Турбо-Паскале определена функция логического типа *EoF(f)* для индикации конца файла. Функция принимает значение «истинно» — если файловый курсор находится в конце файла, и «ложно» — в противном случае.

Рассмотрим пример программы, которая открывает текстовой файл для чтения и выводит его содержимое на экран дисплея:

```
Uses Crt;
Var f:Text;
              name:String[30];
              st:String;
              c:Char;
              error: Word;
BEGIN {Main}
    Writeln('Введите имя файла:');
    Readln(name);
    Assign(f, name);
    {$I-} ReSet(f); {$I+}
    error:= loResult;
If error <> 0
Then Begin
    Write ln( Такого файла нет!!! ');
    Write ln(' Код ошибки =', error,'.');
    c:=ReadKey;
```

```
        Halt
        End
    Else Begin
        ClrScr;
        Writeln('Содержимое файла', name);
        Writeln('=================================');
        Repeat
                Readln(f,st);
                Writeln(st);
        Until EoF(f);
        Writeln('=================================');
        c:=ReadKey;
          End;
    END. {Main}
```

Здесь отметим использование процедуры «Readln», которая применяется только к текстовым файлам. Более детально эта и другие специфические процедуры будут рассмотрены ниже.

15.1.7. Другие процедуры для обработки файлов.

При использовании файлов могут потребоваться действия различного назначения. Рассмотрим некоторые из них.

Стирание файла осуществляется процедурой «Erase(f)». После вызова данной процедуры физический файл, связанный с файловой переменной *f*, удаляется с диска.

С помощью процедуры «ReName(f, Новое—имя);» можно *переименовать* физический файл. Следующий фрагмент демонстрирует возможность использования этой процедуры:

```
Program rename_file;
Uses Crt;
Var f:text;
    name,name2: String;
    c:Char;
Begin
    ClrScr;
    Writeln('Enter file name:');
```

```
    Readln(name);
    Assign(f,name);
    {$I-} Reset(f); {$I+}
    If Ioresult<>0
            Then Begin Writeln('There is no such file!!!');
                       Writeln('Press any key!!!');
                       c:=ReadKey;
                       Halt;
                 End
      Else Close(f);
    Writeln('Enter new name for the file:');
    Readln(name2);
    Rename(f,name2);
    Writeln('Press any key!!!');
    c:=ReadKey;
End.
```

Процедура «Flush(f)» используется для *текущего обновления* файла. Она *очищает* буфер, осуществляет запись информации непосредственно в файл и гарантирует сохранность данных на промежуточных этапах выполнения программы.

15.2. Три вида файлов.

В зависимости от типа хранимой информации, различаются три основных вида файлов: текстовой, типизированный и нетипизированный. Для каждого вида, кроме процедур общего назначения, определены специфические процедуры, применимые только к файлам конкретного вида. Мы рассмотрим каждый вид по-отдельности и приведем примеры их использования.

15.2.1. Текстовые файлы.

Текстовые файлы предназначены для хранения текстовой информации, т.е.:

- Прежде всего, это символьная информация;
- Файл организован в виде последовательности строк различной длины (максимальная длина — 128 символов);

- Каждая строка заканчивается специальным символом, называемым «конец строки», или «EoL = End of Line»;
- Последний символ файла — это другой специальный символ — «конец файла», или «EoF = End of File».

Для Турбо-Паскалевского компилятора текстовой файл выглядит так;

```
Это текстовая информация, и она содержится в EOL
в файле под именем «myFile». EOL
Обратите внимание, что каждая строка в EOL
файле прерывается специальным EOL
символом, вместо которого здесь EOL
используется английская аббревиатура его EOL
названия. А в конце текста стоит EOL
аббревиатура конца файла. EOF
```

Мы уже рассмотрели логическую функцию «EoF(f)», которая указывает на состояние «конец файла». Для текстовых файлов определены еще другие логические функции:

EoL(f) — инициирует конец строки, т.е. она принимает значение «истинно» только, если файловый курсор находится в конце строки, иначе ее значение — «ложно».

SeekEoL(f) — возвращает значение «истинно», если от текущего положения файлового курсора до символа конца строки находятся только незначащие символы (т.е. пробелы или знаки табуляции), иначе — «ложно»;

SeekEoF(f) — возвращает значение «истинно», если от текущего положения файлового курсора до символа конца файла находятся только незначащие символы, иначе — «ложно»,

Если мы хотим добавить информацию в текстовой файл, т.е. расширить его, нужно использовать процедуру

```
Append(f);
```

которая открывает файл *f* для записи и устанавливает файловый курсор в конце файла.

Создание нового файла, а также открытие файла для чтения данных, осуществляется процедурами «ReWrite(f)» и «Reset(f)», соответственно.

Информацию в текстовой файл можно записать с помощью процедур «Write» и «WriteLn». Форматы использования могут быть такими:

```
Var f:Text;
    st: String;
    c:Char;
Begin
    { Фрагмент инициации файла}
    st:='Это первая строка'файла';
    c:='!';
    Write(f,st);
    Writeln(f,c);
    WriteLn(f,'Это новая строка.');
    { Фрагмент закрытия файла }
End.
```

В результате в файле запишутся три строки (последняя — пустая):

```
Это первая строка файла!
Это новая строка
```

Таким образом, процедура "Write" записывает данные в файл подряд одной непрерывной цепочкой, "WriteLn" записывает полную строку, т.е. данные +символ конца строки.

Считывание данных из файла осуществляется процедурами "Read" и "ReadLn", при этом "Read" (f, параметры) извлекает данные из файла подряд непрерывным потоком, а "ReadLn" (f, параметры) считывает данные, соответствующие "параметрам", и переводит файловый курсор на следующую строку, даже если в предыдущей строке еше остались другие данные. Если параметры не указываются, тогда процедура "ReadLn" просто переводит курсор на следующую строку без чтения данных. Рассмотрим пример:

```
Use Crt;
Const n=5;m=10;
Var f: Text;
    i,j, sum, columns: Integer;
    massiv:Array [1...n] of Integer;
    c: Char;
Begin
    Assign (f,'TextFile.txt');
    {Формирование файла}
    Rewrite(f);
    For i:=1 to n Do
    Begin For j:=1 to m Do  Begin sum:=i+j; Write(f,sum); End;
            Writeln(f);
End;
columns :=m Div 2;
{Чтение данных из файла}
{и вывод массива на экран дисплея};
Reset(f);
WriteLn('Первые', columns,' колонок:');
WriteLn('==========================');
While not EoF(f) Do
Begin For i:=1 to columns Do Read (f,massiv[i]);
    Readln(f);
    For i;=1 to columns Do Write (massiv[i]:5);
    Writeln;
End; {While}
WriteLn ('==========================');
c:=ReadKey;      Close(f);
End.
```

Обратите внимание, что хотя файл текстовой, т.е. является совокупностью символов, для записи и чтения данных из него используются переменные целого типа. Организация ввода/вывода точно такая же, как и при работе с клавиатурой/экраном. Это и не удивительно, так как эти устройства рассматриваются как абстрактный вид текстовых файлов и обращение к ним осуществляется аналогичным способом. Как при вводе данных с клавиатуры, так и в

случае чтения из файла они должны быть разграничены друг от друга по крайней мере одним пробелом.

Созданный файл будет иметь вид:

```
 2  3  4  5  6  7  8  9 10 11
 3  4  5  6  7  8  9 10 11 12
 4  5  6  7  8  9 10 11 12 13
 5  6  7  8  9 10 11 12 13 14
 6  7  8  9 10 11 12 13 14 15
```

а на экране будет выводиться следующий массив:

```
 2  3  4  5  6
 3  4  5  6  7
 4  5  6  7  8
 5  6  7  8  9
 6  7  8  9 10
```

15.2.2. Типизированные файлы.

Турбо-Паскаль позволяет работать с т.н. *типизированными* файлами. Они состоят из совокупности однотипных данных, и этот тип называется *базовым*. Базовым для типизированных файлов может служить любой легальный тип – скалярный или структурированный. В первом случае элементом, компонентом файла будет одна величина, а во втором случае – совокупность однотипных или разнотипных данных, составляющих структурированную величину. Приведем примеры объявления файлов различных типов:

```
Type fArray = Array [1..5] of Real;
                testRecord = Record i,j:Integer
                                    x,y:Real;
                                    name: String
    End;
Var fileInteger: File of Integer;
    fileReal:File of Real;
```

```
fileChar:File of Char;
fileString:File of String;
fileRecord:File of testRecord;
fileArray:File of Array;
```

Таким образом, типизированные файлы состоят из неявно пронумерованных компонентов, первому из которых соответствует нулевой номер. Когда файл открывается для чтения, файловый курсор устанавливается на нулевой компонент. По мере чтения данных курсор перемещается по направлению к концу файла. При чтении из файла параметры оператора Read должны по типу совпадать с типом компонентов файла. Например,

```
Var f:File of Real;
    rl,r2,r:Real;
    ...
    Assign(f,'my_file');
    ReSet(f);

Read(f,rl,r2);
```

Здесь оператор Read считывает сразу два компонента, следовательно, файловый курсор переместится на две позиции и установится на третьем компоненте. Такой доступ называется последовательным. Если мы хотим считать какой-нибудь конкретный компонент, сначала файловый курсор надо установить на нем, а затем считать. Это можно реализовать с помощью процедуры Seek(f, component), где параметр component: LongInt — номер компонента. Например, пятый элемент файла можно считать с помощью фрагмента:

```
...
ReSet(f);
Seek(f, 4);
Read(f,r);
```

так как первый элемент идет под нулевым номером.

Если курсор дошел до конца файла, при попытке чтения возникнет логическая ошибка. Следовательно, корректность обращения к файлу нужно контролировать программным путем.

Запись информации в файл осуществляется только с помощью процедуры «Write», при этом в файл можно занести только величины такого типа, какого типа сам файл. Например,

```
r:=5.5;
Write(f, r);
```

При этом, значение параметра *r* в файл запишется на текущей позиции курсора, и если на этом месте уже была занесена информация, она потеряется. Если мы хотим просто добавить компоненты к файлу, то после его открытия курсор нужно переместить в конец файла, а затем уже осуществить запись. Можно считывать компоненты из файла до тех пор, пока не будет достигнут его конец, а затем начать запись. Но такой подход довольно неудобный. Функция FileSize(f) может организовать возможность расширения файла более компактно — она возвращает количество компонентов в файле. Следующий фрагмент дает такой же эффект, что и процедура Append(f) для текстовых файлов:

```
...
ReSet(f);
Seek(f,FileSize(f));
```

15.2.3. Нетипизированные файлы.

Нетипизированные файлы используются обычно тогда, когда файл содержит разнотипные элементы. Объявление таких файлов происходит через ключевое слово *FILE* без определения типа.

При открытии такого файла обычно указывается длина записи, напр.:

```
Assign(f,'file16');
Reset(f,32);
```

При отсутствии длины подразумевается 128. Надо отметить, что при таком способе работы с файлом скорость обмена данных значительно больше, чем с типизированными файлами, так как в этом случае не происходит преобразования данных.

Все процедуры и функции, которые используются для типизированных файлов, также могут быть использованы в этом случае, за исключением процедур *READ* и *WRITE*. Вместо них используются процедуры *BLOCKREAD* и *BLOCKWRITE*:

BLOCKREAD(<ф.п>,<б.о>,<N1>[,<N2>])
BLOCK\WRITE(<ф.п>,<б.о>,<N1>[,<N2>])

где <ф.п> — файловая переменная, <б.о> — переменная, буфер обмена, в которую считываются данные с диска, или из которой данные записываются на диск, N1 – количество записей, которые надо записать или считать за одно обращение, N2 – необязательный параметр, определяющий количество реально обработанных записей.