

# Глава 5 Работа с файлами

МГТУ им. Н.Э. Баумана

Факультет Информатика и системы  
управления

Кафедра Компьютерные системы и сети

Лектор: д.т.н., проф.

Иванова Галина Сергеевна

## 5.1 Файловая система

**Файл** – поименованная последовательность элементов данных (компонентов файла), хранящихся, как правило, во внешней памяти.

Как исключение данные файла могут не храниться, а вводиться с внешних устройств (ВУ), например клавиатуры или выводиться на ВУ, например экран.

Полное имя файла включает:

<Имя диска>:<Список имен каталогов>\<Имя файла>.<Расширение>

Имя файла в Windows составляют из строчных и прописных букв латинского и русского алфавитов, арабских цифр и некоторых специальных символов, например, символов подчеркивания «\_» или доллара «\$»

Расширение определяет тип хранящихся данных, например:

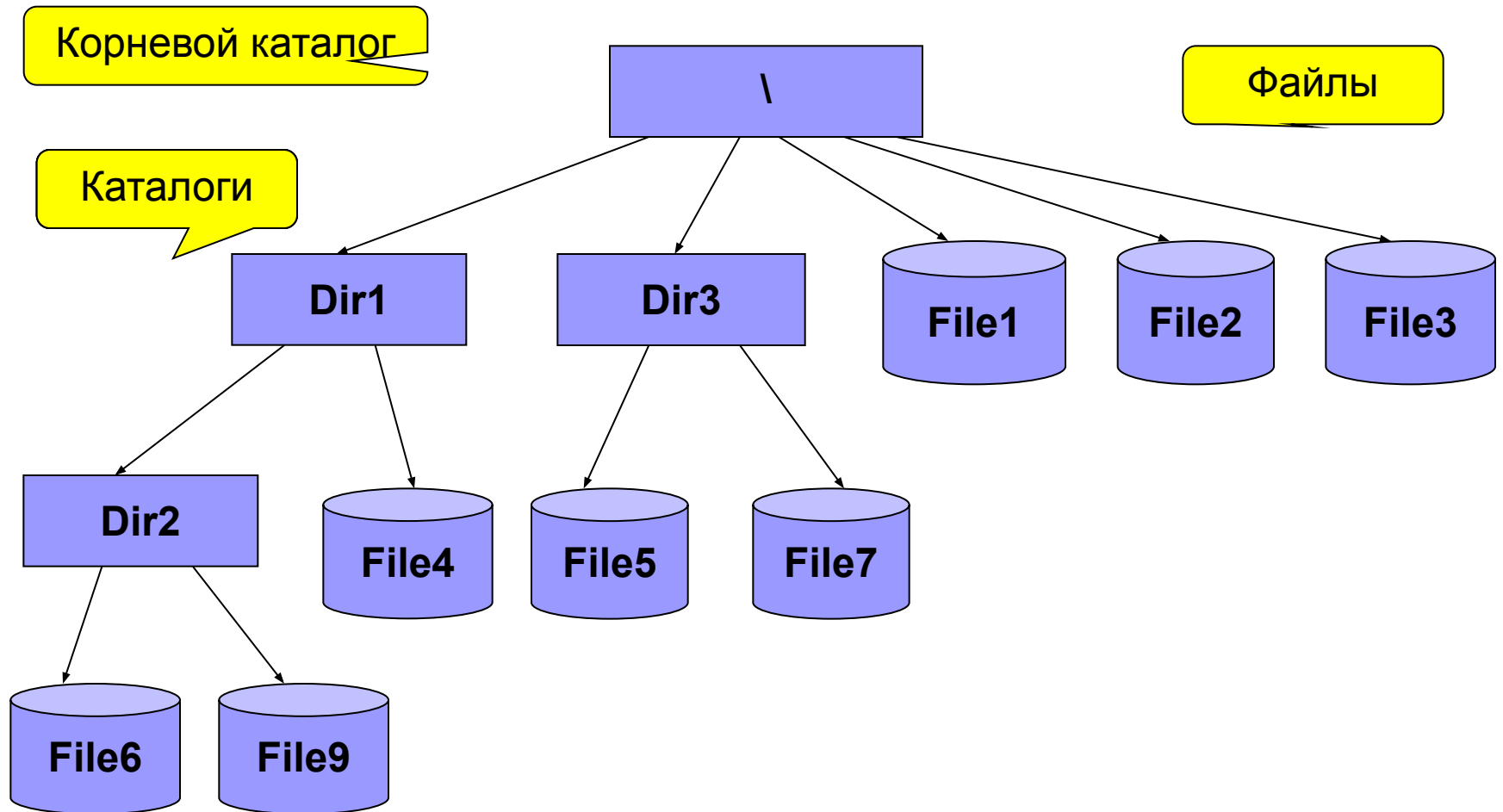
**COM, EXE** – исполняемые файлы (программы);

**PAS, BAS, CPP** – исходные тексты программ на алгоритмических языках ПАСКАЛЬ, БЭЙСИК и С++;

**BMP, JPG, PIC** – графические файлы (рисунки, фотографии);

**WAV, MP3, WMA** – музыкальные файлы.

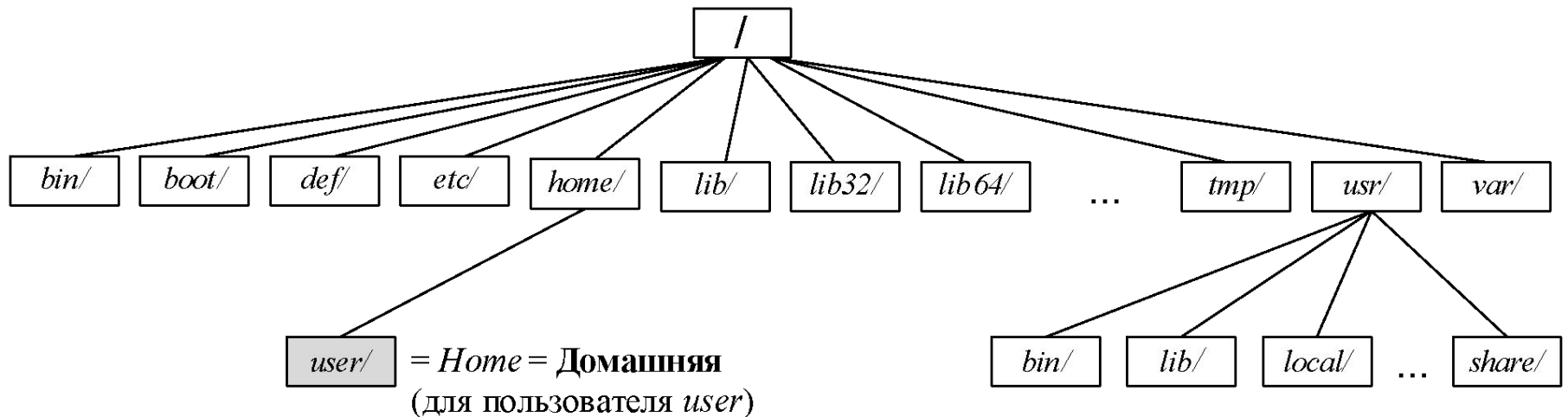
# Организация файлов на внешнем носителе Windows



Пример полного имени файла:

D:\Dir1\Dir2\File9.pas

# Организация файлов на внешнем носителе Linux



**`/home/user/data.txt`** – абсолютное имя файла, где «`/`» – корневой каталог *Linux*;

**`~/progs/data.txt`** – относительное имя файла, где «`~/`» подразумевает домашний каталог `/home/user` пользователя *user Linux*;

**`../data.txt`** – относительное имя файла, где «`../`» означает родительский для текущего каталог;

**`data.txt`** – файл данных текущего каталога;

**`./prog`** – вызов программы из текущего каталога.

# Файлы Delphi Pascal

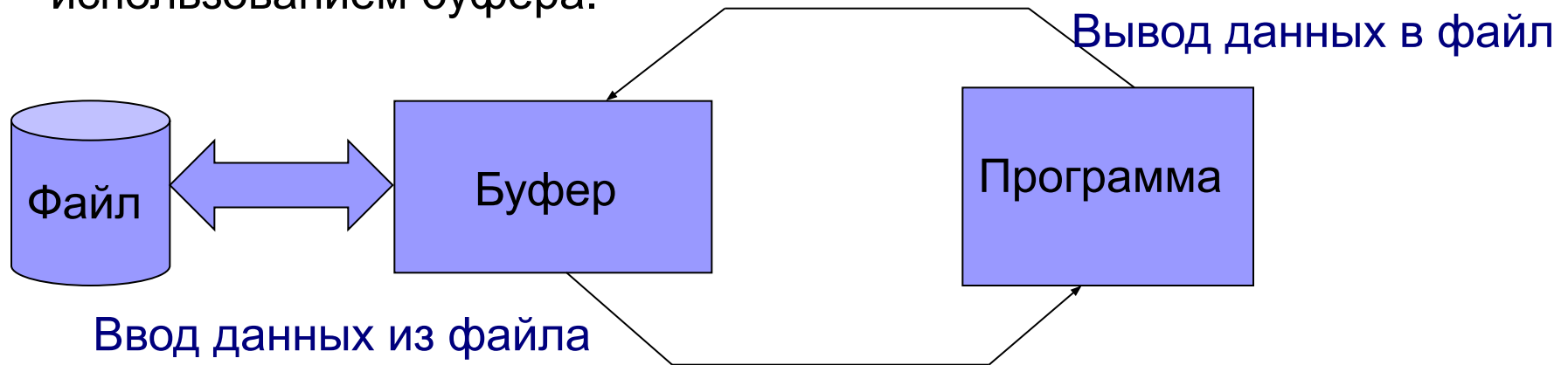
**Файл языка Pascal** – последовательность однотипных компонентов: файл записей, файл целых чисел, файл строк.

В зависимости от типа компонентов различают три типа файлов: *типизированные*, *текстовые* и *нетипизированные*.

Количество компонентов файла при объявлении файловой переменной не указывается.

Максимальный размер файла определяется свободным пространством на устройстве, например, диске.

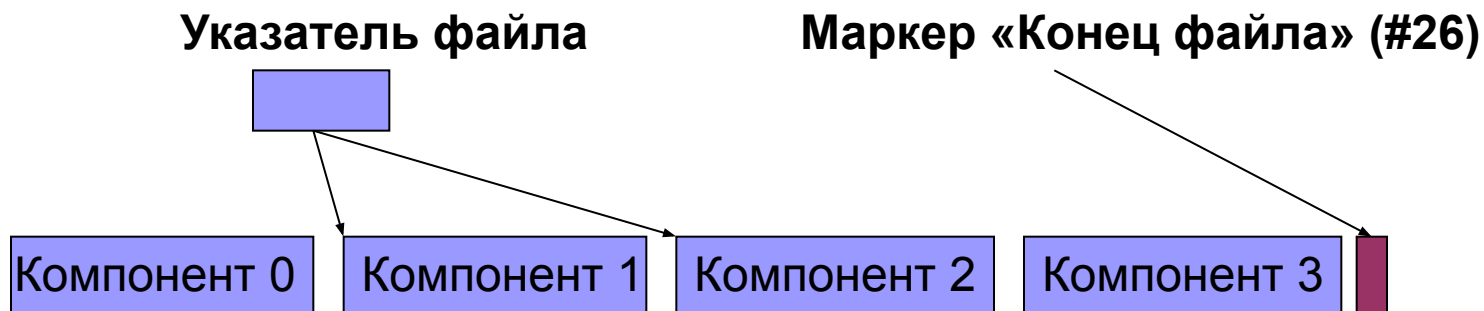
Физически операции ввода-вывода с файлами выполняются с использованием буфера.



Для файлов принципиально возможен не только последовательный, но и **произвольный доступ**, при котором чтение информации осуществляется из указанного места.

# Указатель файла

Доступ к компонентам файла осуществляется через **указатель файла**. При выполнении операции чтения или записи указатель **автоматически** перемещается на следующий компонент.



После вывода последнего компонента файла система пишет специальную запись – **маркер «Конец файла» (байт #26)**.

При обнаружении во время операции чтения маркера конца файла – операция завершается. Попытка читать маркер вызывает прерывание по ошибке чтения.

# Описание файловых переменных

1. *Типизированные файлы:* `file of <Тип компонента>`,  
где <Тип компонента> – любой тип данных, кроме файлового.
2. *Текстовые файлы:* `text`
3. *Нетипизированные файлы:* `file`

## Примеры:

- 1) 

```
Var F1: file of real;  
    F2: file;  
    F3: Text; ...
```
- 2) 

```
Type FF = file of integer;  
    FR = file;  
    FC = text;  
Var F1:FF;  
    F2,F3:FC;  
    F4:FC; ...
```

# Использование файлов в качестве параметров подпрограмм

Файлы можно передавать в подпрограмму только через **параметры-переменные**.

**Пример:**

```
Type FF = file of integer;  
Procedure Print(Var F1:FF);
```



# Работа с файлами

Работа с файлами включает:

- ***инициализацию файловой переменной*** – установление связи файловой переменной с файлом;
- ***открытие файла*** – подготовку файла для выполнения операций ввода-вывода;
- ***обработку компонентов файла*** – выполнение операций ввода-вывода элементов данных;
- ***заккрытие файла.***

# Инициализация файловой переменной

Процедура `Assign` или `AssignFile (Var f; st:string)` – связывает файловую переменную `f` с файлом, имя которого указано в строке `st`.

Если файл находится в текущем каталоге, то достаточно задать имя файла и его расширение. В противном случае необходимо указать полное имя файла.

Пример:

```
Type FI1 = file of integer;
```

```
Var f1, f2: FI1;
```

```
...
```

```
AssignFile (f1, 'F1.dat'); {файл в текущем каталоге}
```

```
AssignFile (f2, 'd:\iva\a.dat'); {файл в другом каталоге}
```

# Открытие файла

При открытии файла необходимо задать направление передачи данных: запись или чтение. Кроме того текстовый файл можно открыть для добавления компонентов.

1. Процедура **ReSet (Var f)** – открывает файл для чтения данных. Устанавливает указатель файла на первый компонент. Если файл не существует, выдается сообщение об ошибке.
2. Процедура **ReWrite (Var f)** – открывает файл для записи. Если указанный файл существовал, то он *уничтожается* без выдачи предупреждения пользователю, иначе он *создается* и указатель устанавливается на начало.
3. Процедура **AppEnd (Var f: text)** – открывает текстовый файл для добавления данных. Указатель файла устанавливается на конец файла.

# Контроль операций ввода-вывода

4. Функция `IOResult:Word` – возвращает код завершения операции ввода-вывода: 0 – если операция прошла нормально, код ошибки, если нет. *Функция применяется при отключенном контроле операций ввода-вывода `{$I-}`.*

**Пример.** Проверка существования файла.

```
Var f:file of char;
Begin
  AssignFile(f,'a.dat'); {инициализация ф. п.}
  {$I-} {отключение контроля ошибок ввода-вывода}
  ReSet(f); {открытие файла для чтения}
  {$I+} {включение контроля ошибок ввода-вывода}
  if IOResult<>0 then WriteLn('File was not found')
  else WriteLn('File was found'); ...
```

# Обработка компонентов файла

Основные операции над компонентами – операции записи и чтения. На базе этих операций выполняют более сложные операции:

- *создание файла* – занесение в файл требуемых записей;
- *модификация файла* – изменение всех или нескольких записей, добавление и удаление записей;
- *поиск* нужной информации в файле.

Операции записи и чтения для каждого типа файла осуществляется по-своему.

# Заккрытие файла

Процедура `Close` или `CloseFile (Var f)` – выполняет закрытие файла. При этом вновь созданный файл регистрируется в каталоге.

Процедура закрытия файла обеспечивает **ВЫВОД ОСТАВШИХСЯ КОМПОНЕНТОВ ИЗ БУФЕРА В ФАЙЛ.**

Связь файловой переменной с файлом при закрытии сохраняется, поэтому при продолжении обработки повторно процедуру `AssignFile()` можно не выполнять.

**Пример:**

```
CloseFile (f) ;
```

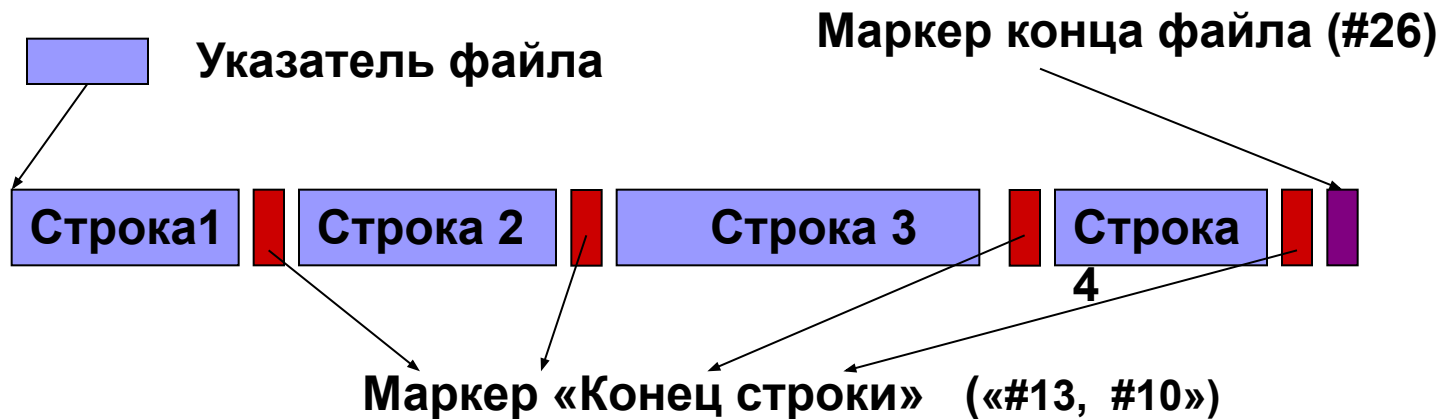
# Стандартные процедуры и функции обслуживания файлов (библ. System)

1. Процедура **ReName (Var f; name:string)** – выполняет переименование файла **f**, где **name** – новое имя файла.  
При совпадении нового имени файла с ранее существовавшим выдается сообщение об ошибке.
2. Процедура **Erase (Var f)** – выполняет удаление файла. Перед уничтожением файл должен быть закрыт.
3. Функция **EOF (Var f) :boolean** – проверяет конец файла, возвращает **TRUE**, если указатель стоит после последней записи и **FALSE**, если нет.

Другие функции обслуживания файлов см. Help.

## 5.2 Текстовые файлы

**Текстовый файл** – файл, компонентами которого являются символьные строки переменной длины, заканчивающиеся специальным маркером – маркером «Конец строки».



Текстовые файлы используют для хранения и обработки символов, строк, символьных массивов. Числовые и логические данные при записи в текстовые файлы должны преобразовываться в символьные строки.

Текстовый файл можно открыть для записи, чтения и добавления записей в конец. Файл, открытый для записи, не может использоваться для чтения и наоборот.





# Текстовый файл в шестнадцатеричном представлении

```
view abc.txt - Far 2.0.1661 x86
F:\...сты1\Глава 5\abc.txt 1251 451 col 0 0%
000000000: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAA
000000010: 41 41 0D 0A 42 42 42 42 42 42 42 42 42 42 42 42 42 42 AA♫BBB
000000020: 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 BBBBBBB
000000030: 0D 0A 43 43 43 43 43 43 43 43 43 43 0D 0A 44 44 44 44 ♫CCCC
000000040: 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 DDDDDDD
000000050: 44 44 0D 0A 45 45 45 45 45 45 45 45 45 45 45 45 45 45 DD♫EEE
000000060: 46 46 46 46 46 46 46 46 46 46 46 46 0D 0A 47 47 47 47 FFFFFFF
000000070: 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 47 GGGGGGG
000000080: 0A 48 48 48 48 48 48 48 48 48 48 48 48 48 48 48 48 48 HHHHHH
000000090: 48 48 48 48 48 48 48 48 48 48 48 48 48 48 48 48 0D 49 HHHHHH
0000000A0: 0A 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 49 IIIIII
0000000B0: 0D 0A 4A 4A 4A 4A 4A 4A 4A 4A 4A 4A 4A 4A 4A 4A 4A 4A ♫JJJJJ
0000000C0: 4A 4A 4A 4A 4A 4A 0D 0A 4B 4B 4B 4B 4B 4B 4B 4B 4B 4B JJJJJJ
0000000D0: 4B 4B 4B 4B 4B 4B 4B 4B 0D 0A 4C 4C 4C 4C 4C 4C 4C 4C KKKKKKK
0000000E0: 4C 4C 4C 4C 4C 4C 4C 4C 4C 4C 4C 4C 4C 4C 4C 4C 4C 4C LLLLLLL
0000000F0: 4C 4C 4C 4C 4C 4C 4C 4C 0D 0A 4D 4D 4D 4D 4D 4D 4D 4D LLLLLLL
000000100: 4D 4D 4D 4D 4D 4D 4D 4D 0D 0A 4E 4E 4E 4E 0D 0A 4F 4F MMMMMMM
000000110: 4F 4F 4F 4F 0D 0A 50 50 50 50 50 50 50 50 0D 0A 51 51 OOOO♫P
000000120: 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 QQQQQQQ
000000130: 51 51 51 51 51 51 51 51 51 51 51 0D 0A 52 52 52 52 52 QQQQQQQ
000000140: 52 52 52 52 52 52 52 52 52 52 52 52 52 52 52 52 52 52 RRRRRRR
000000150: 52 52 52 52 52 52 52 52 0D 0A 53 53 53 53 53 53 53 53 RRRRRRR
1 2wWrap 3 4 5 6 7Next 8CodePg
```



# Типизированный файл

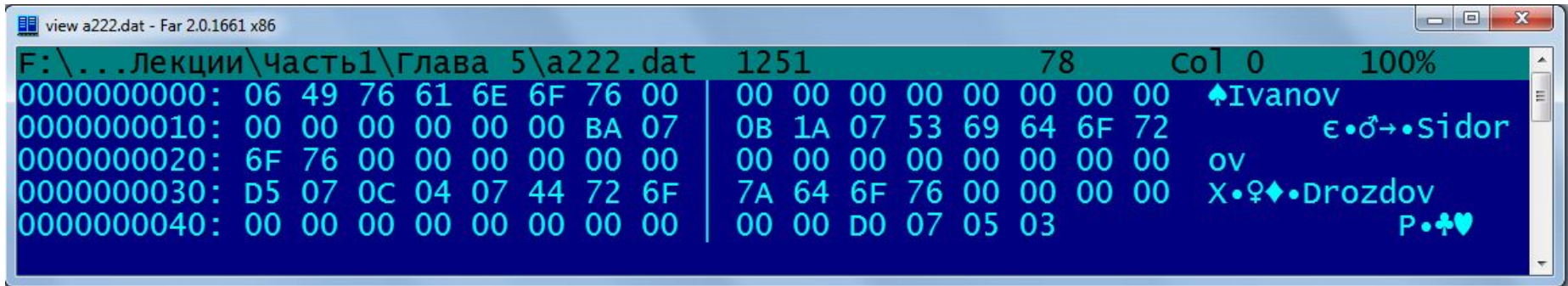
Символьное представление



view a222.dat - Far 2.0.1661 x86

```
F:\...\ты1\Глава 5\a222.dat 1251 78 col 0 100%
♠Ivanov                €•♂→•Sidorov          X•♀♦•Drozdov
  P•♣♥
```

Шестнадцатеричное представление



view a222.dat - Far 2.0.1661 x86

```
F:\...Лекции\Часть1\Глава 5\a222.dat 1251 78 col 0 100%
000000000: 06 49 76 61 6E 6F 76 00 | 00 00 00 00 00 00 00 00  ♠Ivanov
000000010: 00 00 00 00 00 00 BA 07  | 0B 1A 07 53 69 64 6F 72   €•♂→•Sidor
000000020: 6F 76 00 00 00 00 00 00  | 00 00 00 00 00 00 00 00   ov
000000030: D5 07 0C 04 07 44 72 6F  | 7A 64 6F 76 00 00 00 00   X•♀♦•Drozdov
000000040: 00 00 00 00 00 00 00 00  | 00 00 D0 07 05 03         P•♣♥
```

# Стандартные текстовые файлы

Программе, работающей в консольном режиме, без объявления, инициализации файловой переменной и открытия доступны два стандартных текстовых файла, связанных с логическими устройствами ввода и вывода:

**INPUT** – файловая переменная для обозначения файла данных, вводимых с клавиатуры;

**OUTPUT** – файловая переменная для обозначения файла данных, выводимых на экран.

# Процедуры и функции обработки текстовых файлов

1. Функция `EOLn ([Var f]) : Boolean` – возвращает `TRUE`, если во входном текстовом файле достигнут маркер конца строки; при отсутствии файловой переменной проверяется файл `INPUT`, связанный с клавиатурой.
  - При работе с клавиатурой функция `EOLn` возвращает `TRUE`, если *последним* считанным был символ `#13`.
  - При работе с диском функция `EOLn` возвращает `TRUE`, если *следующим* считанным будет символ `#13`.
2. Процедура `Read ([Var f:text;] v1,v2,...vn)` – обеспечивает ввод символов, строк и чисел. При вводе чисел пробелы и символы табуляции игнорируются. Если файловая переменная не указана, то ввод осуществляется из файла `INPUT`.

## Процедуры и функции обработки текстовых файлов (2)

3. Процедура `ReadLn([Var f;][v1,v2,...,vn])` – осуществляет ввод символов, строк и чисел. После чтения последней переменной оставшаяся часть строки до маркера конца строки пропускается так, что следующее обращение к `ReadLn` или `Read` начинается с первого символа новой строки.
4. Процедура `Write([Var f;]v1,v2, ...,vn )` – осуществляет вывод одного или более выражений типа `CHAR`, `STRING`, `BOOLEAN`, а также целого или вещественного типов. При выводе числовых значений последние преобразуются в символьное представление. Если файловая переменная не указана, то вывод осуществляется в файл `OUTPUT`.

Любой параметр из списка вывода может иметь формат:  
<Параметр> [: <Целое1> [: < Целое2> ]].

# Процедуры и функции обработки текстовых файлов (3)

5. Процедура **WriteLn**(**[Var f;]** **[v1, v2, ..., vn]**) – осуществляет вывод в текстовый файл. Если файловая переменная не указана, то вывод осуществляется в файл **OUTPUT**.  
Выводимая строка символов завершается маркером конца строки. Если список вывода не указан, то в файл передается только маркер конца строки.
6. Функция **SeekEOLn**(**[Var f]**) **:boolean** – пропускает пробелы и знаки табуляции до маркера конца строки или до первого значащего символа и возвращает **TRUE**, при обнаружении маркера. Если файловая переменная не указана, то функция проверяет файл **INPUT**.
7. Функция **SeekEOF**(**[Var f]**) **:boolean** – пропускает все пробелы, знаки табуляции и маркеры конца строки до маркера конца файла или до первого значащего символа и возвращает **TRUE** при обнаружении маркера. Если файловая переменная отсутствует, то функция проверяет файл **INPUT**.

# Формирование текстового файла

**Пример.** Разработать программу, которая формирует текстовый файл из 26 строк, содержащих случайное количество соответствующих прописных букв латинского алфавита, например:

AAAAA

BBBBBBB

CCCC

DDDDDDDDDD и т.д.

```
program EX5_1;  
  {$APPTYPE CONSOLE}  
uses SysUtils;  
Var f:text;  
    a:char; n,i:integer;  
    fname,st:string[30];
```



## Формирование текстового файла (2)

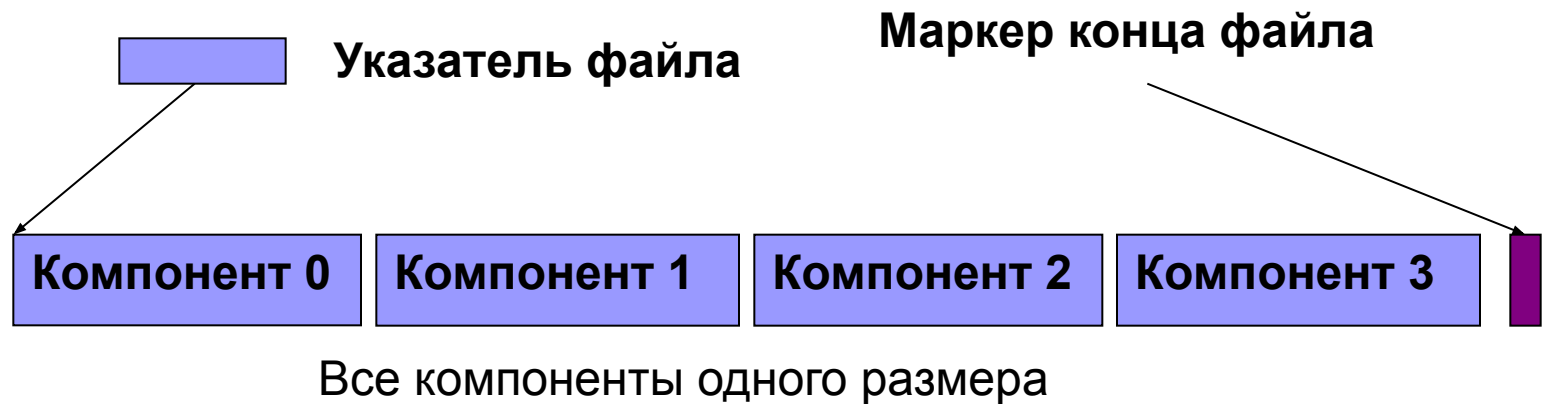
Begin

```
WriteLn('input File name');
ReadLn(fname);
Assign(f, fname);
ReWrite(f);
Randomize;
for a:='A' to 'Z' do
  begin
    st:='';
    n:=Random(30)+1;
    for i:=1 to n do st:=st+a;
    WriteLn(f, st);
    WriteLn(st);
  end;
Close(f);
ReadLn;
end.
```

## 5.3 Типизированные файлы

**Типизированный файл** – файл, все компоненты которого одного типа, заданного при объявлении файловой переменной.

Компоненты хранятся на диске во *внутреннем* (двоичном) формате. При этом числа хранятся в формате, в котором над ними выполняются операции.



Типизированный файл можно открыть для записи и чтения. Файл, открытый для записи, может использоваться для чтения. В файл, открытый для чтения, можно писать.

Поскольку размер компонентов **одинаков**, принципиально возможен не только последовательный, но и **прямой доступ**.

## Процедуры и функции обработки типизированных файлов

1. Процедура **Read (Var f; c1, c2, ..., cn)** – осуществляет чтение компонентов типизированного файла. Список ввода содержит одну или несколько переменных того же типа, что и компоненты файла. Если файл исчерпан, обращение к процедуре вызывает ошибку ввода-вывода.
2. Процедура **Write (Var f; c1, c2, ..., cn)** – осуществляет запись компонентов в типизированный файл. Список вывода содержит одно или более выражений того же типа, что и компоненты файла.
3. Процедура **Seek (Var f; numcomp:longint)** – осуществляет установку указателя файла на компонент с номером numcomp.
4. Функция **FileSize (Var f):longint** – возвращает количество компонентов файла. Может использоваться для установки на конец файла совместно с Seek():  

```
Seek (f, FileSize (f) );
```
5. Функция **FilePos (Var f):longint** – возвращает порядковый номер компонента, который будет обрабатываться следующим.
6. Процедура **Truncate (Var f)** – выполняет «усечение» файла.

# Обработка типизированных файлов

**Пример 1.** Разработать программу, которая создает файл, компонентами которого являются символы, введенные с клавиатуры, изменяет символы, записанные в файл, организует чтение символов из файла попеременно сначала и с конца (прямой доступ), затем находит указанный символ в файле и удаляет его из файла.

```
program Ex5_2;  
  {$APPTYPE CONSOLE}  
  uses SysUtils;  
  Var f, f1:file of char;  
      ch,i:Ansichar;  
      j:longint;  
      name:string[8];  
begin  
  WriteLn('Input file name:');  
  ReadLn(name);
```

# Создание файла

{открытие и создание файла}

```
AssignFile(f, name+'.dat');
```

```
ReWrite(f); {открываем файл для записи }
```

{занесение записей в файл}

```
WriteLn('Input symbol or «#»:');
```

```
ReadLn(ch);
```

```
while ch<>'#' do {пока не введено «#» с клавиатуры}
```

```
begin
```

```
    Write(f, ch); {записываем символ в файл}
```

```
    ReadLn(ch); {вводим символ с клавиатуры}
```

```
end;
```

```
WriteLn;
```

# Последовательное чтение записей из файла

{последовательное чтение записей из файла}

`ReSet (f) ;`            {открываем файл для чтения}

`while not EOF (f) do`    {до конца файла}

`begin`

`Read (f, ch) ;`    {читаем символ из файла}

`Write (ch, ' ' ) ;`    {выводим символ на экран}

`end;`

`WriteLn;`

# Изменение записей в файле

{изменение записей в файле}

```
ReSet (f) ;           {открываем файла для чтения}
while not EOF (f) do {пока не достигнут конец файла}
  begin
    Read (f, i) ;     {читаем символ из файла}
    Write (i, ' ');   {выводим символ на экран}
    i := chr (ord (i) + 10) ; {изменяем символ}
    WriteLn (i) ;     {выводим на экран измененный символ}
    Seek (f, FilePos (f) - 1) ; {возвращаемся на один
                                КОМПОНЕНТ}
    Write (f, i) ;    {перезаписываем символ}
  end;
WriteLn;
```

# Прямой доступ к записям файла

{попеременное чтение записей с начала и конца файла}

```
ReSet (f) ;      {открываем файл для чтения}
j:=0;           {устанавливаем номер компонента равным 0}
while not EOF(f) do  {пока не достигнут конец файла}
  begin
    Read(f,i) ;    {читаем символ из начала файла}
    Write(i) ;     {выводим символ на экран}
    Seek(f,FileSize(f)-FilePos(f)) ; {устанавливаем
                                     указатель для чтения из конца файла}
    Read(f,i) ;   {читаем символ из конца файла}
    Write(i) ;    {выводим символ на экран}
    j:=j+1;      {увеличиваем номер компонента}
    Seek(f,j) ;  {устанавливаем указатель на следующий от
                                     начала компонент}
  end;
WriteLn;
WriteLn('Input symbol for erase:');
ReadLn(ch) ;
```



# Удаление записей из файла

```
{подготовка к удалению записей: переименование исходного
файла и открытие нового файла с тем же именем}
CloseFile (f) ;                {закрываем файл}
ReName (f , name+ ' .bak ' ) ; {переименовываем файл}
ReSet (f) ;                    {открываем файл для чтения}
AssignFile (f1 , name+ ' .dat ' ) ;
ReWrite (f1) ;                {открываем новый файл для записи}
{удаление записей - перепись остающихся записей в другой файл}
while not EOF(f) do
  begin
    Read(f,i) ;                {читаем символ из файла}
    if i<>ch then Write(f1,i) ; {если символ не подлежит
                                удалению, то записываем его в новый файл}
  end;
CloseFile (f) ;
Erase (f) ;    {удаляем старый файл, после закрытия в нем ничего
не изменилось, поэтому повторно его можно не закрывать}
```

# Последовательное чтение записей из файла

{последовательное чтение записей из нового файла}

ReSet (f1) ; {открываем новый файл для чтения}

while not EOF (f1) do

begin

Read (f1 , ch) ; {читаем из файла}

Write (ch , ' ');

end;

CloseFile (f1) ;

WriteLn;

ReadLn;

end.

# Создание файла букв алфавита

**Пример 2.** Создать файл букв латинского алфавита и удалить буквы, код которых не кратен 2.

{Создание файла}

```
program Ex5_3a;  
  {$APPTYPE CONSOLE}  
uses SysUtils;  
var f:file of Char;  
    i:Char;  
begin  
  AssignFile(f,'a.dat');  
  Rewrite(f);  
  For i:='A' to 'Z' do Write(f,i);  
  CloseFile(f);  
end.
```

# Удаление букв с нечетными кодами

```
program Ex5_3b;                                     j – номер компонента, в который пишем;
{$APPTYPE CONSOLE}                                j1 – номер компонента, из которого читаем
uses SysUtils;
Var f:file of Char;
    i:Integer;n:Char; j,j1:longInt;
begin
    AssignFile(f,'a.dat');
    Reset(f); j:=0;
    while not EOF(f) do
        begin
            Read(f,n);
            if (ord(n) mod 2) = 0 then
                begin
                    j1:=FilePos(f);
                    Seek(f,j); Write(f,n); inc(j); Seek(f,j1);
                end
            end;
        Seek(f,j); truncate(f);
    end.
```

# Создание файла Таблица дней рождения

**Пример 3.** Разработать программу, которая создает файл, содержащий список фамилий и даты рождения. Осуществить поиск в этом файле даты рождения по заданной фамилии.

```
program Ex5_4a; {Создание файла}
{$APPTYPE CONSOLE}
uses SysUtils;
Type fam=record
    ff:string[22]; {фамилия}
    year:word;    {год рождения}
    month:1..12;  {месяц рождения}
    day:1..31     {день рождения}
end;
Var f:file of fam;
    fb:fam;
begin
    AssignFile(f, 'fam.dat');
    Rewrite(f);
```

# Создание файла

```
WriteLn('Input family or empty string');
ReadLn(fb.ff);
while length(fb.ff) <> 0 do {пока строка не пустая}
begin
    WriteLn('Input birthday: year month day');
    ReadLn(fb.year, fb.month, fb.day); {фамилию вводим
                                        в отдельной строке, так как ввод строки
                                        завершается нажатием клавиши Enter}

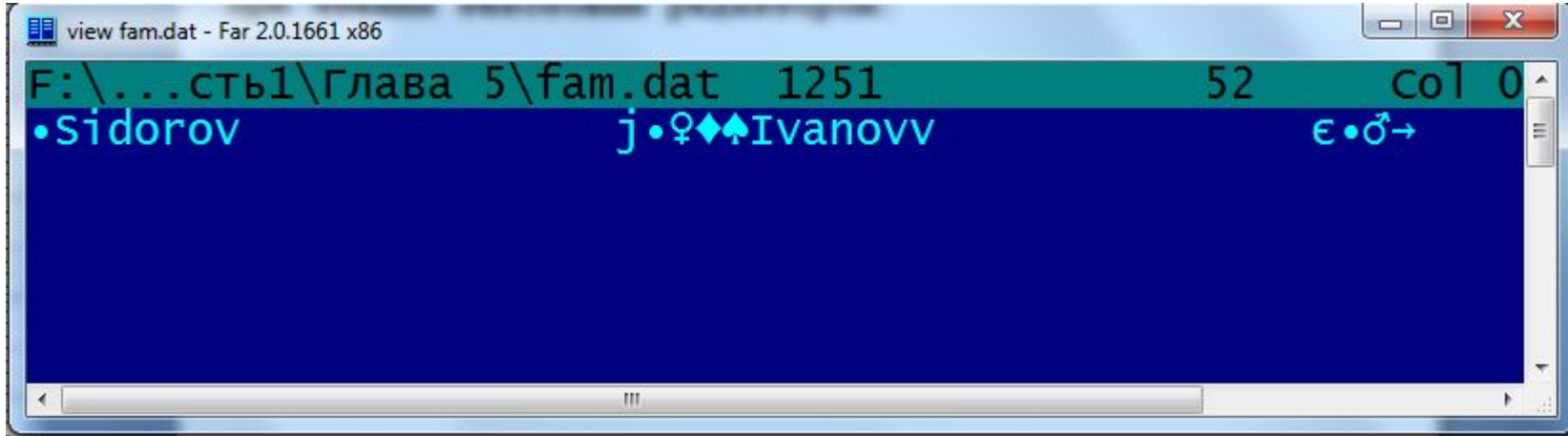
    Write(f, fb);
    WriteLn('Input family or empty string');
    ReadLn(fb.ff);
end;
CloseFile(f);
End.
```

```
Sidorov
1980 12 4
Ivanov
1978 11 26
```

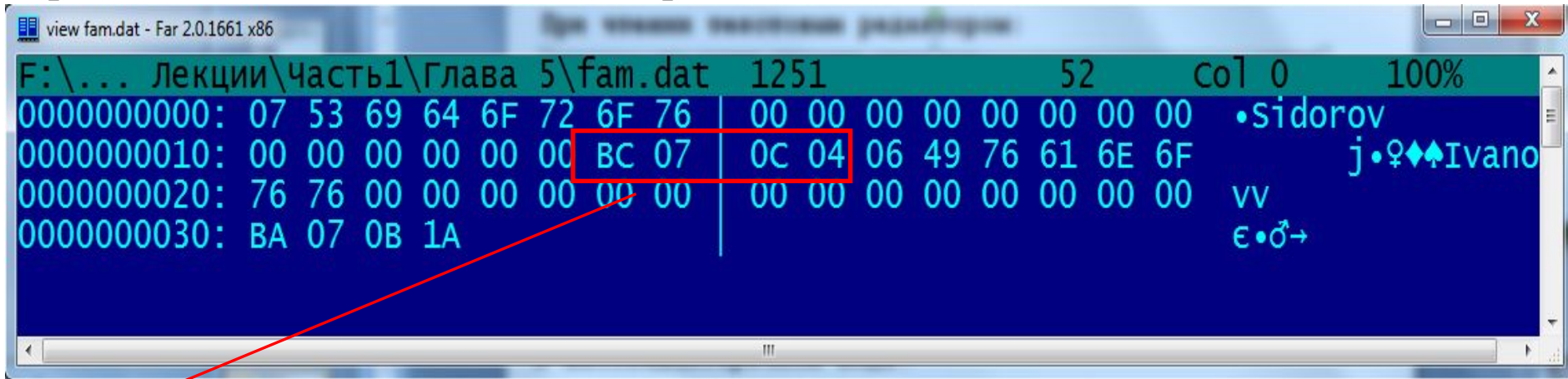
# Содержимое файла

Sidorov  
1980 12 4  
Ivanov  
1978 11 26

При чтении текстовым редактором:



При чтении в шестнадцатеричном виде:



$BC\ 07\ 0C\ 04: 07BC_{16} = 1980_{10};\ 0C_{16} = 12_{10};\ 04_{16} = 4_{10}$

# Поиск данных в файле

```
program Ex5_4b;
{$APPTYPE CONSOLE}
uses SysUtils;
Type fam=record
    ff:string[22]; {фамилия}
    year:word;     {год рождения}
    month:1..12;   {месяц рождения}
    day:1..31      {день рождения}
end;
Var f:file of fam;
    fb:fam;
    fff:string;
    key:boolean;
begin
    Write('Input family: ');
    Readln(fff);
    AssignFile(f, 'fam.dat');
```



## Поиск данных в файле (2)

```
key:=false; {признак "запись найдена"}
```

```
ReSet(f);
```

```
while (not EOF(f)) and (not key) do {пока не  
обнаружен конец файла и не найдена запись}
```

```
begin
```

```
  Read(f,fb);
```

```
  if fb.ff=fff then
```

```
    begin
```

```
      WriteLn('DATA: ',fb.year,fb.month:3,fb.day:3);
```

```
      key:=true; {"запись найдена"}
```

```
    end;
```

```
  end;
```

```
  if not key then WriteLn('No information');
```

```
  CloseFile(f);
```

```
  ReadLn;
```

```
end.
```

# Чтение текстового файла как типизированного

**Пример 4.** Разработать программу, которая открывает текстовый файл как типизированный с компонентом типа CHAR и читает его по СИМВОЛУ.

```
program Ex5_5;  
  {$APPTYPE CONSOLE}  
uses SysUtils;  
Type ff=file of char;  
Var f:ff;  
    a:char;      n,i:integer;  
    fname,st:string[30];  
begin  
  Write('Input file name:');  
  ReadLn(fname);  
  AssignFile(f,fname);  
  ReSet(f);
```

# Чтение текстового файла как типизированного(2)

```
while not EOF(f) do
begin
```

```
    st:='';
```

```
    Read(f,a);
```

```
    while (a<>#13) and not EOF(f) do {до маркера конца
                                     строки или конца файла}
```

```
        begin
```

```
            st:=st+a;
```

```
            Read(f,a);
```

```
        end;
```

```
        if not EOF(f) then Read(f,a); {пропускаем символ #10}
```

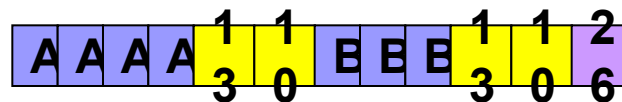
```
        WriteLn(st);
```

```
    end;
```

```
    CloseFile(f);
```

```
    ReadLn;
```

```
end.
```



## 5.4 Нетипизированные файлы

*Нетипизированными* называют файлы, объявленные без указания типа компонентов.

Операции чтения и записи с такими файлами осуществляются блоками, что позволяет организовать высокоскоростной обмен данными между диском и памятью. Отсутствие типа делает эти файлы совместимыми с любыми другими, однако обрабатывать такие файлы существенно сложнее.

Нетипизированные файлы, как и типизированные, допускают организацию прямого доступа, но к записям.

Нетипизированный файл можно открыть для записи и для чтения:

```
ReSet (Var f; [recsize:word]);
```

```
ReWrite (Var f; [recsize:word]);
```

где `recsize` – размер записи файла в байтах. Длину записи задают кратной 512 байт, например: 1024, 2048. Если длина записи не указана, она принимается равной 128.

# Процедуры и функции обработки нетипизированных файлов

## 1. Процедура

**BlockRead(Var f:file; Var buf;Count:word[;Var res:word])** – осуществляет чтение блока записей из файла в буфер buf.

Параметр res будет содержать количество фактически обработанных записей. Если последняя запись – неполная, то значение параметра res ее не учтет.

## 2. Процедура

**BlockWrite(Var f:file;Var buf;Count:word[;Var res:word])** – осуществляет запись блока из буфера buf в файл.

# Копирование файлов

Пример. Разработать программу копирования файлов

```
program Ex5_6;  
{$APPTYPE CONSOLE}  
Uses SysUtils;  
Const recs=1024;  
Var fi,fo:file;  
    buf: array [1..2*recs] of byte;  
    i:integer;    namein,nameout: string;  
begin  
    WriteLn('Input file name:');  
    ReadLn(namein);  
    AssignFile(fi,namein);  
    {$I-} ReSet(fi,1); {$I+}  
    if IOResult <> 0 then  
        begin  
            WriteLn('File with name ',namein,' not found');  
            ReadLn;    Halt;  
        end;  
end;
```

Размер записи  
равен 1, чтобы  
обеспечить  
кратность любой  
длине файла

## Копирование файлов (2)

```
WriteLn('Input name Output_file:');
ReadLn(nameout);
AssignFile(fo,nameout);
ReWrite(fo,1);
while not EOF(fi) do
  begin
    BlockRead(fi,buf,sizeof(buf),i);
    BlockWrite(fo,buf,i);
  end;
CloseFile(fi);
CloseFile(fo);
ReadLn;
end.
```

# Дополнительные процедуры и функции для работы с файлами

1. `Function ChangeFileExt(const FileName, Extension: string): string` – изменяет существующее расширение файла на указанное.
2. `Procedure ChDir(const S:string); overload;`  
`Procedure ChDir(P:PChar); overload;` – изменяет текущий каталог (каталог по умолчанию).
3. `Function CreateDir(const Dir: string): Boolean` – создает новый каталог.
4. `Function DeleteFile(const FileName: string): Boolean` – удаляет указанный файл.
5. `Function DirectoryExists(const Directory: string): Boolean` – проверяет существование каталога по указанному адресу.
6. `Function DiskFree(Drive: Byte): Int64` – возвращает объем в байтах свободного пространства на указанном диске:  
0 – устройства по умолчанию; 1 – диск А; 2 – диск В и т.д. Функция возвращает -1, если указанный диск не существует.



## Дополнительные процедуры и функции для работы с файлами (2)

7. `Function DiskSize(Drive: Byte): Int64` – возвращает объем памяти указанного диска.
8. `Function FileExists(const FileName: string): Boolean` – проверяет существование файла по указанному адресу;
9. `Function FileSearch(const Name, DirList: string): string` – ищет файл в указанных через точку с запятой каталогах, если не находит, то возвращает пустую строку.
10. `Function FindFirst(const Path: string; Attr: Integer; var F: TSearchRec): Integer` – ищет в каталоге первый файл с указанной маской и атрибутами;
11. `Function FindNext(var F: TSearchRec): Integer` – ищет следующие файлы.
12. `Function GetCurrentDir: string` – возвращает имя текущего каталога.
13. `Function ForceDirectories(Dir: string): Boolean` – создает каталоги и подкаталоги.
14. `Function RemoveDir(const Dir: string): Boolean` – удаляет указанный пустой каталог.
15. `Function SetCurrentDir(const Dir: string): Boolean` – устанавливает текущий каталог.