



МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Прикладная математика»

## **Программирование в Delphi: файлы**

Методические указания к лабораторной работе № 10  
по курсам «Информатика», «Алгоритмические языки  
и программирование»

Автор  
Е.Н. Ладоса, Д.С. Цымбалов, О.В. Яценко,  
А.П.Мул, В.И.Снигирева

Ростов-на-Дону, 2018

## Аннотация

Описывается создание и преобразование файлов в Object Pascal. Целью работы ставится рационализация обработки структурированной гетерогенной информации средствами Delphi. Предназначены для студентов всех специальностей факультета «Информатика и вычислительная техника».

## Автор

Доцент, к.т.н.  
Ладоса Е.Н.

Старший преподаватель кафедры  
«Электроника и электротехника»  
Цымбалов Д.С.

Доцент, к.ф.-м.н.  
Яценко О.В.

Старший преподаватель кафедры  
«Прикладная математика»  
Мул А.П.

Студент ДГТУ  
Снигирева В.И.



## Цель работы

Цель работы – изучить функциональность Delphi в части обработки структурированных гетерогенных данных на уровне операционной системы, приобрести навыки создания, преобразования и сопоставления **файлов**. Рассмотреть основные средства работы с **файлами** и их практическое применение.

## Файлы

Компьютерные программы должны иметь возможность сохранять данные на диске и читать их с диска. Представим себе, например, гипотетическую программу **PIM (Personal Information Manager – личный информационный помощник)**, с помощью которой пользователь может хранить и просматривать имена, телефоны и адреса своих знакомых. Все эти данные, естественно, должны храниться на диске в виде файлов. Если бы при каждом запуске программы их приходилось вводить заново, программа была бы бесполезной. Поэтому почти в каждой программе должны быть предусмотрены операции **ввода и вывода файлов**.

**Ввод файла** – это операция чтения информации из файла, хранящегося на диске.

**Вывод файла** – это операция записи информации в хранящийся на диске файл.

В Pascal поддерживаются три типа файлов данных: текстовые, типизированные и нетипизированные.

- File – текстовый файл, т.е. набор символьных строк переменной длины,
- File of <тип> - типизированный файл, т.е. набор данных указанного типа,
- File – нетипизированный файл, т.е. набор неструктурированных данных.

Например,

Var

A1: text; //описываются файловые переменные

A2: file of integer;

A3: file;

## Текстовые файлы

Текстовые файлы являются **последовательными файлами**. Доступ к их элементам может быть получен только последовательно, т.е. сначала предоставляется доступ к первому элементу файла, затем ко второму и так далее до конца файла. Термин **доступ** означает как чтение из файла, так и запись в файл. Таким образом, процесс чтения данных из текстового файла или записи в него всегда начинается с начала файла и продолжается до конца файла.

## Работа с текстовыми файлами

Для получения доступа к текстовому файлу в Pascal нужно сначала объявить переменную типа `Text`, которая называется дескриптором файла. В дескрипторе хранится **указатель файла**, который похож на курсор в текстовом редакторе. Как и курсор, указатель файла обозначает текущую позицию в открытом текстовом файле. В режиме ввода указатель файла определяет следующий элемент данных, который будет считан из файла. В случае текстового файла таким элементом данных является символ. В режиме вывода указатель файла определяет позицию, в которую будет записан следующий элемент данных.

Объявив дескриптор файла, нужно связать его с файлом данных с помощью процедуры `Assign()`, синтаксис которой имеет вид

**Assign** (*дескриптор\_файла*, *имя\_файла*) ;

Строковое выражение *имя\_файла* должно содержать любое правильное имя файла. Если файл находится не в текущем каталоге, в выражении *имя\_файла* должен быть указан его полный маршрут, включая имя диска и всех подкаталогов.

И наконец, для получения доступа к файлу его нужно **открыть**. Текстовый файл можно открыть или для ввода, или для вывода но не для обеих операций одновременно. Процедура `Reset()` открывает или повторно открывает текстовый файл в режиме ввода, а процедуры `Rewrite()` и `Append()` открывают или повторно открывают текстовый файл в режиме вывода. Синтаксис этих процедур имеет вид

**Reset** (*дескриптор\_файла*) ;

**Rewrite** (*дескриптор\_файла*) ;

`Append` (*дескриптор\_файла*) ;

Процедура `Reset()` открывает существующий файл данных, ассоциированный с заданным дескриптором, и устанавливает указатель файла в его начало. Если файл уже открыт, то сначала он закрывается, а затем открывается повторно. Если файла данных с указанным именем не существует, то генерируется ошибка `file not found` (файл не найден).

Процедура `Rewrite()` создает новый файл данных с именем, ассоциированным с дескриптором, и устанавливает указатель файла на его начало. Если файл с этим именем уже существует, то он удаляется и вместо него создается новый файл с этим же именем. Если файл уже открыт, то он закрывается и удаляется, а вместо него создается новый файл. Таким образом, процедура `Rewrite()` или создает новый файл, или перезаписывает существующий файл заново.

Для добавления данных в конец файла используется процедура

`Append()`. Она открывает существующий файл, имя которого ассоциировано с дескриптором, и устанавливает указатель файла в *конец* файла. Если файл уже открыт, то процедура `Append()` закрывает его, а затем открывает повторно. Если файла с этим именем не существует, то генерируется ошибка `file not found`.

Когда текстовый файл открыт в режиме чтения, из него можно читать данные с помощью процедуры `Read()`, синтаксис которой имеет вид

`Read (дескриптор_файла, переменная);`

Процедура `Read()` выполняет следующие операции:

1. Считывает из файла, ассоциированного с дескриптором, порцию данных, на которую показывает указатель файла.
2. Сохраняет считанные данные в переменной.
3. Передвигает указатель файла на следующую порцию данных.

Если, например, в текущей позиции файла хранится целое число, то оно должно быть считано в переменную целого типа. Присутствующие в текстовом файле **символы-разделители** (пробелы, символы табуляции, символы перехода на новую строку) отделяют друг от друга числа, записанные в десятичном формате. Таким образом, процедура `Read()` одновременно со считыванием преобразует числовые данные из десятичного формата в двоичный.

Несколько вызовов процедуры `Read()` можно объединить в один вызов.

Например, группа операторов

```
Read(myFile, variable1) ;
```

```
Read(myFile, variable2);
```

```
Read(myFile, variables);
```

эквивалентна одному оператору

```
Read(myFile, variable1, variable2, variables);
```

Общий синтаксис процедуры `Read()` имеет вид

```
Read (дескриптор_файла, переменная1 [,переменная2,  
...]);
```

В отличие от `Read()`, процедура `Readln()` читает данные с новой строки. Синтаксис процедуры `Readln()` имеет вид

`Readln (дескриптор_файла, переменная) ;`

Процедура `Readln()` работает аналогично `Read()`, за исключением того, что `Readln()` начинает читать данные не с позиции, заданной указателем файла, а со следующей новой строки.

Одним вызовом процедуры `Readln()` можно прочитать несколько переменных, причем они не обязательно должны находиться в одной строке. Если строка закончилась, а список переменных еще не исчерпался, то начинают счи-

ываться переменные из следующей строки. Синтаксис такого вызова имеет вид `Readln (дескриптор_файла, переменная1 [, переменная2, ...])`;

В отличие от `Read()`, несколько вызовов `Readln()` не эквивалентны одному вызову с объединенным списком переменных. Например, группа операторов

```
Readln(myFile, variable1);  
Readln(myFile, variable2);  
Readln(myFile, variable3);
```

не эквивалентна одному оператору

```
Readln(myFile, variable1, variable2, variable3);
```

Это объясняется тем, что один оператор переходит на следующую строку, только если в текущей закончились переменные, а при использовании трех операторов переход на новую строку происходит при каждом вызове `Readln()`.

Если с помощью процедуры `Rewrite()` или `Append()` файл открыт в режиме вывода, то в него можно записывать данные с помощью процедуры `Write()`, общий синтаксис которой имеет вид

```
Write (дескриптор_файла [, выражение [  
:минимальная_ширина  
[:длина_дробной_части] ] ] );
```

*Выражение* может иметь любой числовой или строковый тип, а *минимальная\_ширина* и *длина\_дробной\_части* должны иметь целочисленный тип. Необязательный параметр *минимальная\_ширина* задает количество символов текстового файла, отводимых для выводимого выражения. Если длина выводимого выражения меньше указанной, то процедура `Write()` дополняет его пробелами слева, а если больше, то в файл выводится больше символов, чем задано выражением *минимальная\_ширина*, т.е. выводятся все необходимые символы. Если выводится число вещественного типа, то необязательный параметр *длина\_дробной\_части* задает количество цифр после десятичной точки. Одним вызовом процедуры `Write()` в файл можно вывести произвольное количество выражений (включая нулевое). Выводимые выражения отделяются друг от друга запятыми. Например, оператор

```
Write(myFile, 10:5, 10.47589:8:2);
```

выводит в текстовый файл, ассоциированный с дескриптором `myFile`, следующий текст:

```
10    10.48
```

Процедура `Writeln()` работает аналогично `Write()` за исключением того, что после вывода всех указанных в ее списке выражений `Writeln()` за-



писывает в файл управляющие символы возврата каретки и начала новой строки (<CR><LF>). Например, операторы

```
Write(myFile, 'Hello ');  
Write(myFile, 'and Good-bye');  
Writeln(myFile);      {Переход на новую строку}  
Writeln(myFile, 'Hello ');  
Writeln(myFile, 'and Good-bye');
```

выводят в файл следующий текст:

```
Hello and Good-bye  
Hello  
and Good-bye
```

Как и в случае с Read() и Readln(), операторы Write () можно объединять, однако объединение операторов Writeln() приведет к объединению строк в результирующем файле. Следовательно, предыдущий пример можно переписать так:

```
Write(myFile, 'Hello ', 'and Good-bye');  
Writeln(myFile);      {Переход на новую строку}  
Writeln(myFile, 'Hello ');  
Writeln(myFile, 'and Good-bye');
```

Когда программа закончила работу с файлом, его необходимо закрыть. Процедура Close () разрывает связь дескриптора с файлом, возвращая эти ресурсы системе. Если файл был в режиме вывода, то процедура Close () перед закрытием файла записывает в его конец символ конца файла <EOF>. Синтаксис процедуры Close () имеет вид

`Close (дескриптор_файла);`

#### **Программирование в Pascal: обратите внимание**

Встроенный в ВР компилятор Pascal поддерживает два стандартных дескриптора текстовых файлов: input и Output. Файлом дескриптора input, работающим в режиме чтения, считается стандартное устройство ввода операционной системы (обычно это клавиатура).

Файлом дескриптора Output, работающим в режиме записи, является стандартное устройство вывода операционной системы (обычно это консоль MS DOS). Перед запуском консольного приложения файлы, ассоциированные с дескрипторами Input и output, автоматически открываются. Это эквивалентно выполнению следующих операторов:

```
Assign(Input, '');  
Reset(Input);  
Assign(Output, '');  
Rewrite(Output);
```

Любая попытка доступа к файлам с помощью дескрипторов input или Output из приложения Windows (не консольного) генерирует сообщение об ошибке ввода-вывода.

Некоторые встроенные подпрограммы ввода-вывода текстовых файлов не требуют явного задания дескриптора файла.

Если дескриптор опущен, то по умолчанию предполагается, что дескриптором яв-

ляется Input для процедур ввода и Output для процедур вывода. Например, вызов Read(value) эквивалентен вызову Read(Input, value), а Write(value) - вызову Write(Output, value).

### Подпрограммы обработки текстовых файлов

В Pascal предусмотрены две очень полезные встроенные функции для работы с файлами: Eof() (end-of-file — конец файла) и Eoln() (end-of-line — конец строки). Функция Eof() возвращает булево значение, сообщающее о том, достигнут ли конец файла в режиме чтения. Если значение Eof(дескриптор\_файла) равно True, значит, указатель файла находится за последним символом. Функция Eof() используется в недетерминированных циклах, выход из которых выполняется при достижении конца файла. Функция Eoln(дескриптор\_файла) возвращает булево значение, сообщающее, находится ли указатель файла в конце текущей строки. Для файла в режиме чтения значение Eoln() равно True, если указатель находится на символе конца строки или если Eof() равно True. Функция Eoln() также используется в недетерминированных циклах для посимвольной обработки текстовых файлов.

Ниже приведен список встроенных подпрограмм Pascal, предназначенных для работы с текстовыми файлами.

Функция	Назначение
Eof()	Проверяет, находится ли указатель файла в конце или за пределами файла
Eoln()	Проверяет, находится ли указатель файла в конце строки
SeekEof()	Возвращает True, если до конца файла остались только символы-разделители
SeekEoln()	Возвращает True, если до конца текущей строки остались только символы-разделители

Процедура	Назначение
Append()	Открывает существующий файл для добавления текста в его конец
Assign()	Связывает имя файла с дескриптором
AssignPrn()	Присваивает дескриптор текстового файла принтеру
Close()	Разрывает связь между файлом и дескриптором
Erase()	Удаляет файл
Flush()	Очищает буфер текстового файла, открытого для вывода
Read()	Читает данные из файла



Readln()	Читает данные из файла с новой строки
Reset()	Открывает существующий файл для чтения
Rewrite()	Создает и открывает новый файл
SetTextBuf()	Присваивает текстовому файлу буфер ввода-вывода
Write()	Записывает данные в текстовый файл
Writeln()	Записывает в текстовый файл данные и символы конца строки

### Не типизированные файлы.

Последовательность байтов, содержащие данные произвольного типа и структуры. Основное назначение – обеспечение совместимости с любыми типами файлов ОС.

Для связи файла с переменной используется процедура Assign.

В процедурах Reset и Rewrite для нетипизированных файлов указывается дополнительно параметр RecSize, чтобы задать размер записи, использующейся при передачи файлов. Procedure res(var F:file;RecSize:word); если этот параметр не указан, то по умолчанию длина записи 128 байт. Если 512, то скорость обмена максимальная.

Допускается применение любой стандартной процедуры для типизированных файлов, кроме read и write, они заменяются на

**Procedure BlockRead (var F:file; var Buf; Count:integer [; var AmtTransferred:integer]);** F- файловая переменная, Buf – переменная в которую будут помещаться данные из файла, count - размер этой переменной, AmtTransferred – количество реально прочитанных блоков.

**Procedure BlockWrite(var F:file; var Buf; Count:integer [; var AmtTransferred:integer]);**- аналогична BlockRead.

Пример. Программа выполняет ввод вещественных чисел из текстового файла и запись их в нетепезированный файл блоками по четыре числа.

```

Program create_bfile;
Var buf  : array[1..4] of real;
    F_in : text;
    F_out: file;
    i,k   : integer;
    name_in, name_out: string;
begin
    {$I-} (*отключаем контроль ошибок ввода/вывода*)
    Writeln('Введите имя входного файла'); readln(name_in);
    Assign(f_in,name_in);
    Reset(f_in);

```



```
If IOResult<>0 then begin
  Writeln('Файл ', name_in, ' не найден'); exit end;
Writeln('Введите имя выходного файла'); readln(name_out);
Assign(f_out, name_out);
Rewrite(f_out, sizeof(real)*4);
{$I+}
i:=0;
while not eof(f_in) do begin
  inc(i);
  read(f_in, buf[i]);
  if i=4 then begin
    blockwrite(f_out, buf, 1); i:=0; end;
end;
if i <> 0 then begin
  for k:=i+1 to 4 do buf[k]:=0;
  blockwrite(f_out, buf, 1);
end;
close(f_in); close(f_out);
end.
```

## Типизированные файлы.

Содержат компоненты одного типа, тип любой кроме файлового.

Для чтения из типизированного файла применяется процедуры **read readln**, для записи **write writeln**. Список параметров должен быть того же типа что и файл.

Для работы с типизированными файлами применяются следующие процедуры и функции

1. **procedure Seek(var F; N:LongInt);** перемещает указатель в типизированном файле, связанном с файловой переменной F к требуемому компоненту N (нумерация с нуля).
2. **function FilePos (var F):LongInt;** - возвращает номер текущего компонента в файле F
3. **function FileSize (var F):Integer;** - возвращает количество компонентов.

Пример. Программа, которая выводит на экран заданную по номеру запись из файла, сформированного в программе create\_bfile.

```
Program get_bfile;
Type Tbuf = array[1..4] of real;
var
```

```

buf:Tbuf;
  F: file of tbuf;
  i,k  : integer;
  filename: string;
begin
{$I-} (*отключаем контроль ошибок ввода/вывода*)
  Writeln('Введите имя входного файла'); readln(filename);
  Assign(f, filename);
  Reset(f);
  If IOResult<>0 then begin
    Writeln('Файл ',filename, ' не найден'); exit end;
{$I+}
  While true do begin
    Writeln ('введите номер записи или -1 для окончания');
    Readln(k);
    If (k > filesize(f)) or (k<0) then begin
      Writeln('Такой записи в файле нет'); exit end;
    Seek(f, k);
    read(f, buf);
    For i:=1 to 4 do writeln(buf[i]:6:1);
  End;
  Close(f);
End.

```

## Подпрограммы управления файлами

Система Pascal предоставляет программисту также встроенные подпрограммы, выполняющие различные операции над файлами, такие как создание каталогов или переименование файлов. Ниже перечислены встроенные подпрограммы управления файлами.

Функция	Назначение
CreateDirO	Создает новый каталог
DeleteFile()	Удаляет файл с диска
DirectoryExists()	Определяет, существует ли заданный каталог
DiskFree()	Возвращает количество байтов свободной памяти на заданном диске
DiskSize()	Возвращает размер в байтах заданного диска

FileDateToDateTime()	Преобразует значение даты/времени системы DOS в значение типа TDateTime
FileExists()	Проверяет, существует ли заданный файл
FileGetAttr()	Возвращает атрибуты заданного файла
FileGetDate()	Возвращает печать даты/времени DOS
FileOpen ()	Открывает файл с указанным методом доступа
FileRead()	Читает заданное количество байтов из файла
FileSearch()	Находит маршрут файла
FileSeek()	Размещает указатель текущего файла в предварительно открытом файле
FileSetAttr()	Устанавливает атрибуты заданного файла
FileSetDate()	Устанавливает для заданного файла печать времени DOS
FileWrite()	Записывает содержимое буфера в текущую позицию файла
FindFirst()	Находит первый экземпляр файла с указанными атрибутами в заданном каталоге
FindNext()	Возвращает следующий экземпляр файла с указанными атрибутами в заданном каталоге
ForceDirectories()	Создает все каталоги заданного маршрута, если их еще не существует
GetCurrentDir()	Возвращает имя текущего каталога
RemoveDir()	Удаляет существующий пустой каталог
RenameFile()	Изменяет имя файла
SetCurrentDir()	Устанавливает текущий каталог

Процедура	Назначение
ChDir()	Изменяет текущий каталог
FileClose()	Закрывает указанный файл
FindClose()	Освобождает память, выделенную процедурой FindFirst()
GetDir()	Возвращает текущий каталог заданного диска

### Контрольные вопросы

1. Для чего используются файлы данных?



2. Назовите три типа файлов данных в Object Pascal.
3. Что такое последовательный файл? Как его создать?
4. Что такое управляющие символы?
5. Что такое указатель файла?
6. Напишите код Object Pascal, выполняющий над текстовым файлом MyData.txt следующие операции:

- а) открытие файла в режиме ввода;
- б) открытие файла в режиме вывода (с уничтожением старых данных);
- в) открытие файла в режиме вывода (без уничтожения старых данных).

7. Какая разница между следующими двумя фрагментами:

```
Read(inFile,    intVar1);  
Read(inFile,    intVar2);
```

и

```
Readln(inFile, intVar1, intVar2);
```

8. Какая разница между следующими двумя фрагментами:

```
Write(outFile, intVar1, ' ');  
Write(outFile, intVar2, ' ');
```

и

```
Writeln(outFile, intVar1, ' ', intVar2, ' ');
```

9. Что делает процедура CloseFile()?
10. Опишите назначение функций Eof () и Eoln (). Какие операции они выполняют?

### Задание

1. Дан файл, содержащий последовательность слов, число букв в которых от 1 до 8. Слова разделены запятыми. Вывести на экран слова, имеющие наименьшую длину.
2. В текстовом файле f1 записана последовательность целых чисел, разделенных пробелами. Записать в текстовый файл f2 все положительные числа.

### Список использованной литературы

1. Фаронов В.В. Delphi 3. Учебный курс. М.: «Нолидж», 1998. 400 с.
2. Галисеев Г.В. Программирование в среде Delphi 8 for .NET. М.: Издательский дом «Вильямс», 2004. 304 с.
3. Павловска Т.А. Паскаль. Программирование на языке высокого уровня. СПб.: Питер, 2003. 393 с.

4. Абрамов С.А. и др. Задачи по программированию