Текстовые и нетипизированные файлы

Наумов Д.А., доц. каф. КТ

Программирование и алгоритмические языки, 2020

Содержание лекции

1. Файлы и файловый тип данных
2. Нетипизированные файлы
3. Операции с текстовыми файлами
4. Примеры работы с текстовыми файлами
5. Работа с аргументами программы

Файл

именованная сущность, для которой определены операции ввода и вывода данных.

Для работы с файлами в Pascal предусмотрены **файловые типы данных**:

1. типизированные: информация считывается и записывается в переменные конкретного типа (целые, вещественные, массивы и т. д.);
2. нетипизированные: информация считывается и записывается блоками определённого размера.
3. текстовые: информация обрабатывается посимвольно, но возможно чтение и запись данных в переменные строкового, целого и вещественного типа.

1 Файлы и файловый тип данных

2 Нетипизированные файлы

3 Операции с текстовыми файлами

4 Примеры работы с текстовыми файлами

5 Работа с аргументами программы

Описание переменной файлового типа нетипизированного файла

Для описания файловой переменной нетипизированного файла используется ключевое слово **file** без обозначения типа элементов.

```
//описание переменных файлового типа
1 var
2   F1: file; //это - нетипизированный файл
```

Файл без типа можно представить как последовательность элементов произвольного типа, но заданного размера.

Подпрограммы работы с нетипизированными файлами

Для нетипизированных файлов доступны все те же процедуры и функции, что и для типизированных файлов, за исключением:

- процедуры Reset и Rewrite имеют расширенный синтаксис:
- вместо процедур чтения и записи (Read, Write) необходимо использовать процедуры BlockWrite и BlockRead:

Синтаксис процедур открытия файлов:

```
Reset(var f: File; BuferSize: word);
Rewrite(var f: File; BuferSize: word) ;
```

- Второй параметр BuferSize операторов Reset и Rewrite может быть опущен, что означает задание размера записи в 128 байт.
- Наибольшая скорость обмена данными обеспечивается при длине записи, кратной размеру сектора на диске.

Подпрограммы работы с нетипизированными файлами

Процедура BlockWrite

запись данных в нетипизированный файл.

Синтаксис: BlockWrite(var f: file; var X; Count: word; var WriteCount: word);

Процедура BlockRead(f)

чтение данных из нетипизированного файла.

Синтаксис: BlockRead(var f: file; var X; Count: word; var ReadCount: word);

- f - переменная файлового типа;
- x - переменная произвольного типа;
- Count - количество блоков памяти размером BuferSize;
- ReadCount(WriteCount) - количество считанных (записанных) блоков.

Пример копирования файла при помощи нетипизированных файлов

```
var SourceFile, DestFile: file; //файловые переменные
  SourceFileName, DestFileName: string; //имена файлов
  ReadCount, WriteCount: word;
  Buffer: array[1..BUFFER_SIZE] of byte; // буфер
begin
  ReadLn(SourceFileName, DestFileName);
  Assign(SourceFile, SourceFileName);
  {$I-}Reset(SourceFile, 1);{$I+}
  if IOResult <> 0 then begin
    WriteLn('Error while opening file ', SourceFileName);
    Halt(1);
  end;
  Assign(DestFile, DestFileName);
  Rewrite(DestFile, 1);
```

Пример копирования файла

```
Error := erOK;
repeat
  BlockRead(SourceFile, Buffer, BUFFER_SIZE, ReadCount);
  if ReadCount = 0 then begin
    Error := erReadError;
    break;
  end;
  BlockWrite(DestFile, Buffer, ReadCount, WriteCount);
  if ReadCount <> WriteCount then begin
    Error := erWriteError;
    break;
  end;
until EOF(SourceFile);
Close(SourceFile);
Close(DestFile);
```

1 Файлы и файловый тип данных

2 Нетипизированные файлы

3 Операции с текстовыми файлами

4 Примеры работы с текстовыми файлами

5 Работа с аргументами программы

Описание переменной тестового файла

Для описания файловой переменной текстового файла используется специальный тип данных - **text**.

```
//описание переменных файлового типа
1 var
2   F1: text; //это - текстовый файл
3   F2: file of char; //это - типизированный файл,
                      //он не является текстовым
```

Несмотря на то, что данные из файлов F1 и F2 могут быть считаны посимвольно, для текстовых файлов доступны некоторые специфические функции и процедуры, и по-разному обрабатываются символы перехода на новую строку (символ с кодом 10 или два символа с кодом 10, 13).

Операции с текстовыми файлами

С текстовыми файлами работают процедуры и функции, описанные в предыдущей лекции:

- Assign(F, s) - связь файла и файловой переменной;
- Reset(F) - открытие файла для чтения;
- Rewrite(F) - открытие файла для перезаписи;
- Close(F) - закрытие файла;
- Rename(F) - переименование файла;
- Erase(F) - удаление файла;
- EOF(F) - проверка на достижения конца файла;

Подпрограммы, которые недоступны для работы с текстовыми файлами:

- Функция Filesize(f)
- Функция Filepos(f)
- Процедура Seek(f, k)
- Процедура Truncate(f)

Открытие файла

Для текстового файла имеется еще одна процедура открытия:

Процедура Append

открывает текстовый файл для дозаписи.

Синтаксис: Append(f)

- f - переменная текстового файла;
- файл должен существовать;
- текущей позицией для записи становится конец файла.

```
1 var
2   F1: text;
3 begin
4   Assign(F1, 'myfile.txt');
5   Append(F1);
```

Чтение из файла

Процедура Read

выполняет чтение информации из текстового или типизированного файла.

Синтаксис: Read(f, x1, x2, x3, ..., xn);

- f - переменная файлового типа;
- x1, x2, x3, ..., xn - переменные.

Для текстового файла можно считывать значения:

- символьного,
- целого,
- вещественного,
- строкового типов
- и отрезков символьного и целого типов.

Запись в файл

Процедура Write

выполняет запись информации в текстовый или типизированный файл.

Синтаксис: Write(f, x1, x2, x3, ..., xn);

- f - переменная файлового типа;
- x1, x2, x3, ..., xn - переменные.

Для текстового файла можно записывать значения символьного, целого, вещественного, строкового типов (и отрезков символьного и целого типов).

Процедура Readln

выполняет чтение из текстового файла до символа конца строки. Символ конца строки считывается из файла, но не добавляется его в считываемые данные.

Синтаксис: Readln(f, x1, x2, x3, ..., xn);

- f - переменная файлового типа;
- x1, x2, x3, ..., xn - переменные.

Процедура Writeln

выполняет запись в текстовый файл и добавляет символ конца строки.

Синтаксис: Writeln(f, x1, x2, x3, ..., xn);

- f - переменная файлового типа;
- x1, x2, x3, ..., xn - переменные.

1 Файлы и файловый тип данных

2 Нетипизированные файлы

3 Операции с текстовыми файлами

4 Примеры работы с текстовыми файлами

5 Работа с аргументами программы

Пусть в текстовом файле содержится информация о двумерном массиве. Первые два числа содержат данные о размерности (4x3), остальные данные - это значения элементов двумерного массива:

```
4 3
-1 6 3
-4 8 12
3 15 9
4 -2 6
```

Удобным механизмом, который можно использовать в задачах, где размерность данных заранее неизвестна, являются открытые массивы, (динамические массивы, для которых не указан тип индекса).

```
type
  TVector = array of real; //одномерный массив TVector
  TMatrix = array of TVector; //массив из векторов
var
  V: TVector;
  M: TMatrix;
```

Так как массивы являются динамическими, то их размер должен быть задан в процессе выполнения при помощи процедуры SetLength() следующего синтаксиса:

```
SetLength(Массив, Размер)
```

- Массив - переменная типа динамического массива;
- Размер - целое число, количество элементов массива.

После того, как размер массива задан, можно получить доступ к его элементам. Доступ осуществляется при помощи операции индексации, элементы нумеруются целыми значениями, начиная с нуля.

```
SetLength(V, 10); //задаем размер, равный 10
for i := 0 to High(V) do
  V[i] := random;
```

Индекс последнего элемента можно узнать при помощи функции High().

Рассмотрим чтение данных из текстового файла в двумерный массив.

```
var
  F: Text; //текстовый файл входных данных
  var M:TMatrix; //массив
  i, j: TIndex; //индексы
  Rows, Cols: TIndex; //количество строк и столбцов
begin
  //связываем файл и файловую переменную
  Assign(F, FileName);
  {$I-}
  Reset(F); //открываем файл для чтения
  {$I+}
  if IOResult <> 0 then //если возникла ошибка
  begin
    //то файл не существует, завершаем работу
    WriteLn('Error: file not found');
    halt;
  end;
```

```
//первые два элемента - это количество строк и столбов
{$I-}
Readln(F, Rows, Cols);
{$I+}
//если не удалось считать значения
if IOResult <> 0 then
begin
  WriteLn('Error: Data error');
  halt;
end;
//если размеры меньше 1 - ошибка
if (Rows < 1) or (Cols < 1) then
begin
  WriteLn('Error: Bad array size');
  halt;
end;
```

```
//задаем размер массива
SetLength(M, Rows);
//задаем размер всех строк
for i := 0 to Rows-1 do
  SetLength(M[i], Cols);
//читаем данные из файла
for i := 0 to Rows-1 do
begin
  for j := 0 to Cols-1 do
  {$I-}Read(F, M[i][j]);{$I+}
  //если не удалось считать значение
  if IOResult <> 0 then
  begin
    WriteLn('Error: Data error');
    halt;
  end;
end;
Close(F); //закрываем файл
```

1 Файлы и файловый тип данных

2 Нетипизированные файлы

3 Операции с текстовыми файлами

4 Примеры работы с текстовыми файлами

5 Работа с аргументами программы

Работа с аргументами программы

1. При запуске (через командную строку или из другой программы) программе могут передаваться аргументы, которые затем программа может обработать и использовать.
2. В аргументах могут быть заданы настройки, имена файлов входных и выходных данных и другие опции.
3. Большинство утилит операционной системы (как семейства Windows, так и семества Linux) запускаются из командной строки и настраиваются при помощи задаваемых аргументов.

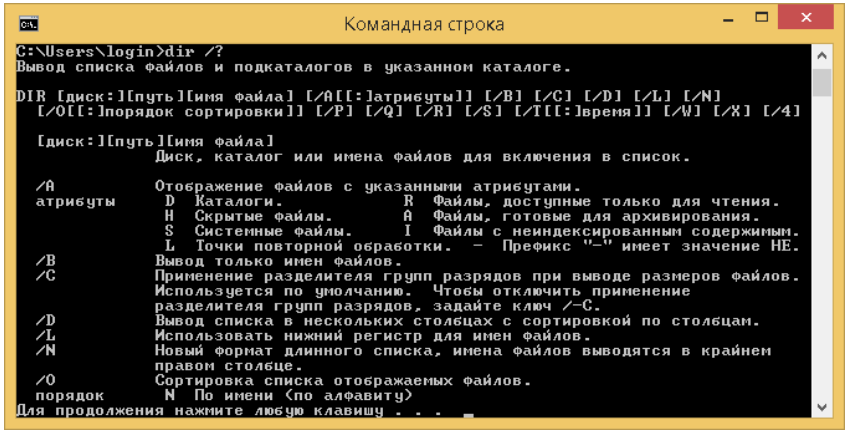

Рис. 1: Пример списка аргументов утилиты DIR

Работа с аргументами программы

Пусть программа (scalar_prod.exe) должна считать из файлов два вектора, найти их скалярное произведение и вывести результат в текстовый файл.
Тогда, необходимы три аргумента:

- имя первого входного файла;
- имя второго входного файла;
- имя файла результатов.

Запуск может осуществляться следующим образом:

```
scalar_prod.exe file1.txt file2.txt result.txt
```

Работа с аргументами программы

Для доступа к аргументам программы в языке Pascal используются следующие функции:

- function ParamCount: integer - получить количество аргументов программы;
- function ParamStr(ParamNumber: integer): string - получить значение аргумента с номером ParamNumber.

Пример вывода всех аргументов командной строки:

```
for i := 1 to ParamCount do
  WriteLn('Param number ', i, ' is ', ParamStr(i));
```

При обработке аргументов командной строке желательно предусмотреть возможность запуска программы с аргументом $'\backslash h'$ или $'\backslash ?'$ для вывода подсказки в случае, если пользователь не знает, как именно необходимо запускать программу, а также вывод справки при запуске программы без аргументов.

Проверка аргументов командной строки для программы scalar_prod:

```
if ParamCount = 1 then
  if (ParamStr(1)= '\h') or (ParamStr(1)= '\?') then begin
    Writeln('Использование программы:');
    Writeln('   scalar_prod file1 file2 result');
    Writeln('   file1 - имя входного файла для вектора V1');
    Writeln('   file2 - имя входного файла для вектора V2');
    Writeln('   result - имя файла результатов');
  end
  else Writeln('Запустите программу с ключом '\?'')
else
  if ParamCount <> 3 then
    Writeln('Запустите программу с ключом '\?'')
  else begin
    FileName1 := ParamStr(1);
    FileName2 := ParamStr(2);
    ResultFileName := ParamStr(3);
  end;
```

Для запуска программы из среды Lazarus необходимо настроить параметры командной строки при помощи окна "Параметры запуска вызываемого при помощи меню "Запуск - Параметры запуска"

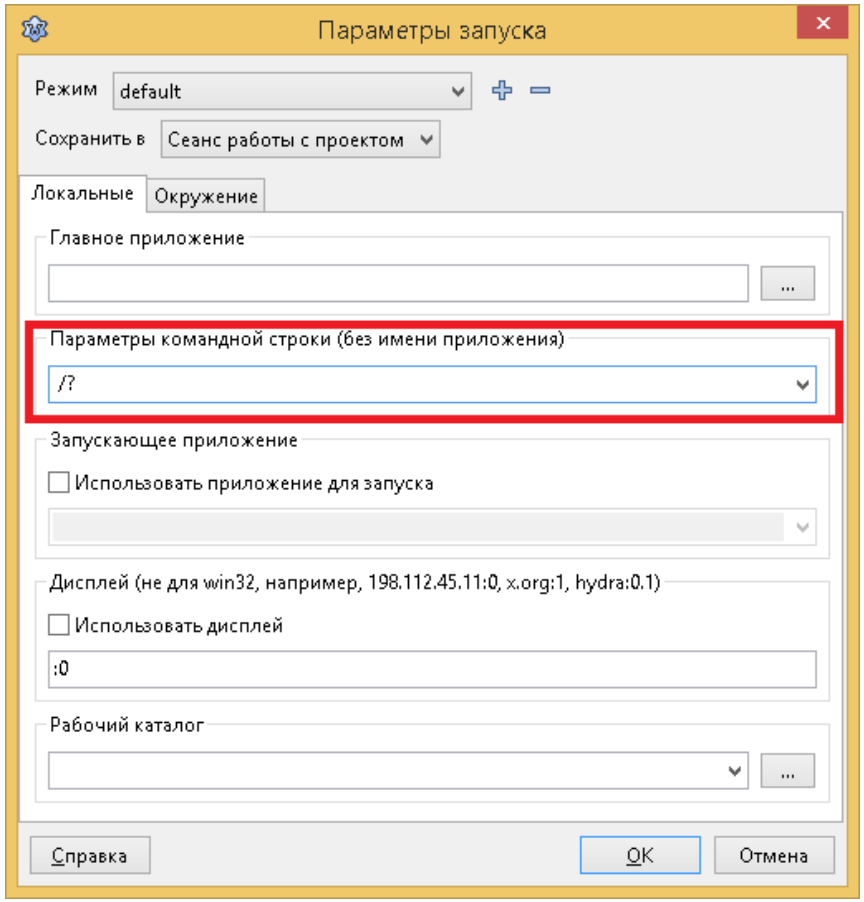

Рис. 2: Окно настройки параметров командной строки в Lazarus