Lesson 8: File Handling

In a nutshell, this lesson will cover:

- Read from a File
- Create and Write to a File
- Append data to an existing File
- Delete Files
- Use the '{$I-},{$I+}' compiler directive
- Create and Remove Sub-Directories
- FileSize() - return the size of the file in bytes

Intro

A file contains data which is stored on a permanent medium. When we use variables, their data will be lost when the program terminates. Files have got the advantage of not losing their data after a program terminates (provided that data is saved properly to the file).

A file can contain either text data or binary data. Many text files are saved on permanent storage such as hard disks with the extension **.txt**. Whenever you use a text editor and create a text file, the saved file is written on to the permanent storage. However, some programs have various rich text formats in which text maintains a certain format - such as justification, font-face and font colour. Such files cannot be saved as plain text since extra data is needed to store the formatting. So other files may contain more data other than just text data.

Before we proceed with files, please make sure that you have enough hard disk space for our practice, since I am going to demonstrate to save a file to the hard disk!! :) Just kiddin'.. We only require a few bytes, don't worry!

Back To Top ⇧

Read from a File (file input)

Reading a file in pascal is very easy. There are special functions in Pascal that enable us to read or write a text file. We will still use the readln() and writeln() in order to read and write from/to a text file; The concept of how to read a **text** file is demonstrated below:

```
Program Lesson8_Program1;

Var
        UserFile : Text;
        FileName, TFile : String;

Begin
        Writeln('Enter the file name (including its full path) of the text file (without the
extension):');
        Readln(FileName); { A .txt file will be assigned to a text variable }
        Assign(UserFile, FileName + '.txt');
        Reset(UserFile); { 'Reset(x)' - means open the file x and reset cursor to the
beginning of file }
```

```
        Repeat
                Readln(UserFile,TFile);
                Writeln(TFile);
        Until Eof(UserFile);
        Close(UserFile);
        Readln;
End.
```

It is worth taking a look at various important lines of the program. A new variable of type: `Text` is new to you, and this should be used whenever you are going to edit a **text** file. The variable `FileName` is needed to link to the file indicated by the user. The `Assign()` procedure is used to attach `FileName` + '.txt' to a text file, so that the file could be opened, using - `Reset()`. In order to read from the first line to the very last line of the file, it is recommended to use the repeat-until loop, ending the loop with : `Until Eof(UserFile)`, meaning: 'Until the **E**nd **O**f **F**ile [eof]'.

Back To Top ⇧

Create and Write to a File (file output)

The following program is an example of how to create-and-write or overwrite a file:

```
Program Lesson8_Program2;
Var
        FName, Txt : String[10];
        UserFile   : Text;

Begin
        FName := 'Textfile';
        Assign(UserFile, 'C:\'+FName+'.txt'); {assign a text file}
        Rewrite(UserFile); {open the file 'fname' for writing}
        Writeln(UserFile, 'PASCAL PROGRAMMING');
        Writeln(UserFile, 'if you did not understand something,');
        Writeln(UserFile, 'please send me an email to:');
        Writeln(UserFile, 'victorsaliba@hotmail.com');
        Writeln('Write some text to the file:');
        Readln(Txt);
        Writeln(UserFile, '');
        Writeln(UserFile, 'The user entered this text:');
        Writeln(UserFile, Txt);
        Close(UserFile);
End.
```

In the above program, I am using the `Writeln()` statement so that I can write to the file I have previously assigned. Don't confuse `Writeln()`, with the ordinary stdout. Note that this time the `Writeln()` is taking two parameters instead of one and there will be **no** output to the screen, but the text is transferred to the file instead. The first parameter is the text file and the second one is the text to be written to the file on single line. The cursor moves to the next line just like when outputting on screen.

To check exactly what has just been written to this file, go to C:\, and see if there is a file named **Textfile.txt**. Open it and see what does it contain.

Append text to an existing File

When appending text to an existing file, it means that the file will be opened to write additional data, and exiting contents will not be overwritten.

```
Var
          UFile : Text;

Begin
          Assign(UFile, 'C:\ADDTEXT.TXT');
          ReWrite(UFile);
          Writeln(UFile, 'How many sentences ' +
                    + 'are present in this file?');
          Close(UFile);
End.
```

Try this program by copying into your Pascal compiler. Run this program for two times or more. After you run this program several times, find the file named 'addtext.txt'. Check whether there is more than one sentence or not.

There is another function which works with files addresses the problem of appending text to existing text files. This function is called 'append(f)', where ***f*** is a variable of type text. This can be done by simply replacing the `ReWrite(UFile)` to `Append(UFile)`, and the file contents will not be overwritten, but added at the end of the file.

The above program can be changed to the following.

```
Var
          UFile : Text;

Begin
          Assign(UFile, 'C:\ADDTEXT.TXT');
          Append(UFile);
          Writeln(UFile, 'How many sentences, '+
                    +'are present in this file?');
          Close(UFile);
End.
```

Run the program two times or more to see the changes...

Delete Files

In Pascal, the function used to delete files from a particular storage medium is the `Erase(UFile)` where ***f*** is a variable of data types **Text** or **File**. The effect of this function is to completely erase the file from the permanent storage.

Unlike any other file functions, the `Erase()` function does not open the file to delete it, so you don't need to call `Close()` after `Erase()`.

Demonstration below:

```
Var
```

```
          UFile : Text; { or it could be of 'file' type}

Begin
          Assign(UFile, 'C:\ADDTEXT.TXT');
          Erase(UFile);
End.
```

The '{$I-},{$I+}' compiler directives

Compiler directives are used to adjust the compiler settings during the compilation process. Compiler directives are declared using the symbols '{', '$' and '}'. Note that the braces are used for comments, but still, '{$*directive*}' is taken to be a directive.

We will be focusing our attention on a special directive used with file handling. The `{$I+/-}` is used to signal the compiler not to take into consideration I/O errors including file operations during runtime. Say, for example, if a file is trying to be opened, but it does not exist, the program does not halt execution with a runtime error, saying: 'File not found' or whatever annoying error! When a user is asked to input a filename and this file does not exist, the program should not crash! Makes sense? So, if you apply this compiler directive before you open the file, either for writing or reading, I/O errors will not cause the program to raise a runtime error, crashing the program.

Now, I you should learn how to use this compiler directive by knowing where to put it and how to handle the error that might occur. This should be placed where the file is going to be opened or perhaps deleted. This concept is illustrated below:

```
{$I-}
...actions on file...
{$I+}
{ enable the I/O error check again }
```

It is important to enable again the I/O error check, after you have disabled it, so that any unknown future errors would be handled internally, through runtime errors unfortunately. However, if you think that it would be better to disable I/O error checking for the entire program, you can! Just apply the directive at the beginning. However this is *not recommended* since during the runtime of your program you will not be notfied of IO errors that might occur and you might not be able to identify the errors and debug them! After you control a file action, you must check whether an error has occured or not by using the system function `IOResult`, which returns IO error information. This is typical traditional exception handling. You should test for errors using the statement:

```
If (IOResult <> 0) Then ...
```

The `IOResult` function should be explicitly used after an IO error check so that it will automatically clear the error flag of the system otherwise, the IO error causes a 'mutation' to other IO processes resulting into a runtime error. You are not required to grasp each word perfectly from what have been said here, but most importantly is that you understand the concept. Do include that statement if the IO directives are to be used and everything would be fine. Now I should show you how to use it through a simple example:

```
Program Lesson8_Program3;
Uses Crt;

Var
          t : Text;
          s : String;

Begin
          Assign(t, 'C:\ABC.DEF');
          {$I-}   { disable i/o error checking }
          Reset(t);
          {$I+}   { enable again i/o error checking - important }
          If (IOResult <> 0) Then
          Begin
                    Writeln('The file required to be opened is not found!');
                    Readln;
          End Else
          Begin
                    Readln(t,s);
                    Writeln('The first line of the file reads: ',s);
                    Close(t);
          End;
End.
```

If the required file is found the IOResult returns a 0 value, meaning no errors; ELSE if not found (IOResult returns a non-0 value) display an error message!

IMPORTANT: if the file is successfully found, the file is opened and you should close it as shown in the program above. However, if it is not found there is no need to include a call to `Close()` on an unopened file and this is done conditionally as shown in the example. Study carefully the program above and run it several times with valid paths and non-valid ones to distinguish well the difference as well as how the compiler directive works!!

The compiler directive could be applied to different file-related functions, similarly as in the program above. That is, you can use it with 'Rewrite()', 'Append()', 'Erase()', and also 'FSearch()'. You can try as many programs as you wish and practice the IO directive with trial and error. I know, that this directive is somewhat complicated and hard to be understood but nonetheless useful!

Create and Remove Sub-Directories

In Pascal, there are functions with which you can either create or remove a directory from the hard disk. To create a directoy, we use the function 'CreateDir(c)' where 'c' is of type `PChar`. Ohh, that sounds weird... What the heck does `PChar` mean? `PChar` is a pointer variable which holds the address of a dynamic variable of a specified type. It is a sort of a special kind of pointer used for characters but you can still pass a value of type String because they are identical data types. Before you create the directory, however, you have to check if the directory exists, else a runtime error shows up fiddling in front of your monitor!! :-) The directory is created as follows:

```
NewDir := FSearch('C:\Pascal Programming', GetEnv(''));
If NewDir = '' Then
          CreateDir('C:\Pascal Programming');
```

From only three lines of code, we have been able to create a dir. But, before you go practicing on your own, you should read the following fully documented example: (I will explain in detail the `FSearch()` function later on)

```
Program Lesson8_Program4;
Uses
          WinDos, Dos;  { note the inclusion of the 'windos.tpu' library }

Var
          NewDir : PathStr; { for searching the dir and create a new one, if it does not exist
}
          F : Text;

Begin
          { search for the dir }
          NewDir := FSearch('C:\Pascal Programming', GetEnv(''));
          { create a new one, if it does not exist }
          If NewDir = '' Then
                    CreateDir('C:\Pascal Programming');
          Assign(F,'C:\Pascal Programming\pascal-programming.txt');
          {$I-} ReWrite(F); {$I+} { disable and enable back again I/O error checking }
          { write to text file }
          Writeln(F,'http://pascal-programming.info/');
          {$I-} Close(F); {$I+}
End.
```

The variable type, `PathStr`, is new to you, and this is a variable defined in the 'dos.tpu' library. So, you have to include the 'dos' library at the beginning of your program. The `FSearch()` function is implemented also in the windos library to search in a dir list. So, it was useful in our program and any other program to search if the directory exists or not. It's not important to know exactly what does 'GetEnv()' mean, which is the second parameter of FSearch(). The Borland Turbo Pascal reference says that its function is to return the value of the specified function. Note that 'FSearch()' is found in the 'dos.tpu' library, while CreateDir()' is found in the 'windos.tpu' library.

To remove a directory, it is quite simple. Just add 'remove()' at the end of your program! :-) Your operating system will automatically erase the dir if it exists. It doesn't matter if you 'remove' a dir which does not exist. Hopefully, no runtime errors, will show up! The 'remove(Pch)' function is also implemented in the windos library, where 'Pch' is a variable of type `PChar`, again.

FileSize() - Return the Size of a file in bytes

Now, for the fun stuff!! To return the size of a file, simply declare a variable of `LongInt` data type, and assign it to the `FileSize()` function of the file. A file variable of type byte should be assigned to the file for which you would like to return its size, using `Assign()`.

```
Program Lesson8_Program4;
Var
          f : File of Byte; { file var of type byte }
          sz : LongInt;  { var for the size }

Begin
          Assign(f,'C:\anyfile.txt');
          {$I-} Reset(f); {$I+}
          If (IOResult <> 0) Then
          Begin     { file found? }
                    Writeln('File not found.. exiting');
```

```
                    Readln;
          End Else
          Begin
                              { Return the file size in Kilobytes }
                    sz := Round(FileSize(f)/1024);
                    Writeln('Size of the file in Kilobytes: ',sz,' Kb');
                    Readln;
                    Close(f);
          End;
End.
```