

Chapter 7

□ TEXT FILES

7.1 INTRODUCTION

The programs we have written so far have been *interactive* in the sense that input came from the keyboard and output went to the screen. Thus the user interacted by typing into the keyboard and looking at the screen. This is fine if the size of the input and output are small. However, suppose we type in 200 names and addresses, sort them, and write out the results to the screen. We use this sorted list to help find particular names quickly. However, once we change to another program or turn off the computer this sorted list is lost. Worse still, all the work we did in typing in the names and address has been lost. One ‘solution’ is to write out the sorted list to a printer so that we have a *hard copy* of the list. In this way we have a permanent record of the list.

This is really not a good solution because such name-and-address lists change over time : names are added or deleted. If a name and address is added or deleted then the whole list must be typed in again and sorted. What we need is a way to permanently store the output of a program. We do this by the use of *files* which are stored in *secondary memory*, i.e., external devices such floppy or hard disks, CDROMs, etc. Instead of being confined to reading from the keyboard and writing to the screen or printer, we can read from and write to files.

7.2 TEXT FILES

Any set of data that is stored on some external device is called a **file**. A program can use a file to input data or to output data. Files are either *external* or *internal*. An external file exists independently of the program and must be connected to an internal file at run-time. An internal file exists only while the program is running whereas an external file can exist before, during or after a program has run. Another way of thinking about this is that an external file exists after the computer has been turned off but an internal file, because it resides in memory, disappears after the computer has been turned off.

A file may be thought of as a sequence or stream of components of a given component type. **We discuss only files whose components are characters.** This type of file is called a *text file* and in Pascal such files have the built-in type TEXT. A text file is declared in Pascal as follows:

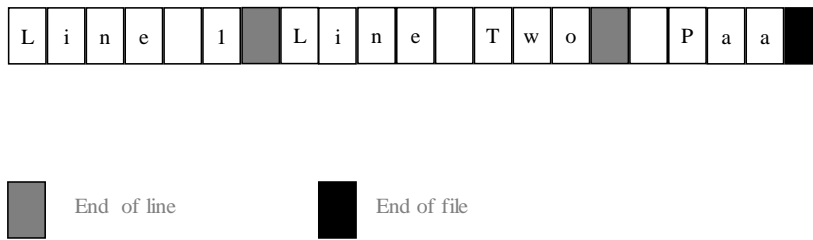


Figure 7.1. Pascal Text File.

A text file is a sequence of characters. Some of these are *end-of-line* characters which separate groups of ordinary characters into *lines*. The final character in a file is a special *end-of-file* character.

Hence we can regard a text file as a sequence of characters grouped into lines. The length of a text file is limited only by the size of external storage or, perhaps, by the operating system. Hence, we may regard a text file as a potentially infinite sequence of characters that flows into a program (input) from an external device or flows out of a program (output) to an external device.

We have already used a text file in the first assignment : your Pascal program file 387-99-1.PAS is a text file which you created in the editor and saved. The executable file 387-99-1.EXE is not a text file. It is a *binary* file.

7.3 CONNECTING INTERNAL AND EXTERNAL FILES

In Pascal all data objects must be declared before they are used in any other part of the program. Hence if we wish to use a file in a program we must first declare it in the **VAR declaration section**. The general form of this declaration is as follows :

```
VAR <list of identifiers> : TEXT;
```

Here are some examples :

```
VAR X,y, File1, FileOut : TEXT;
VAR
  FileIn   : TEXT;
  FileOut  : TEXT;
  TempFile : TEXT;
```

The identifiers `File1`, `X`, `TempFile` etc. are called *internal file names* because they are declared within the program and as such they exist only when the program is running. *External file names*

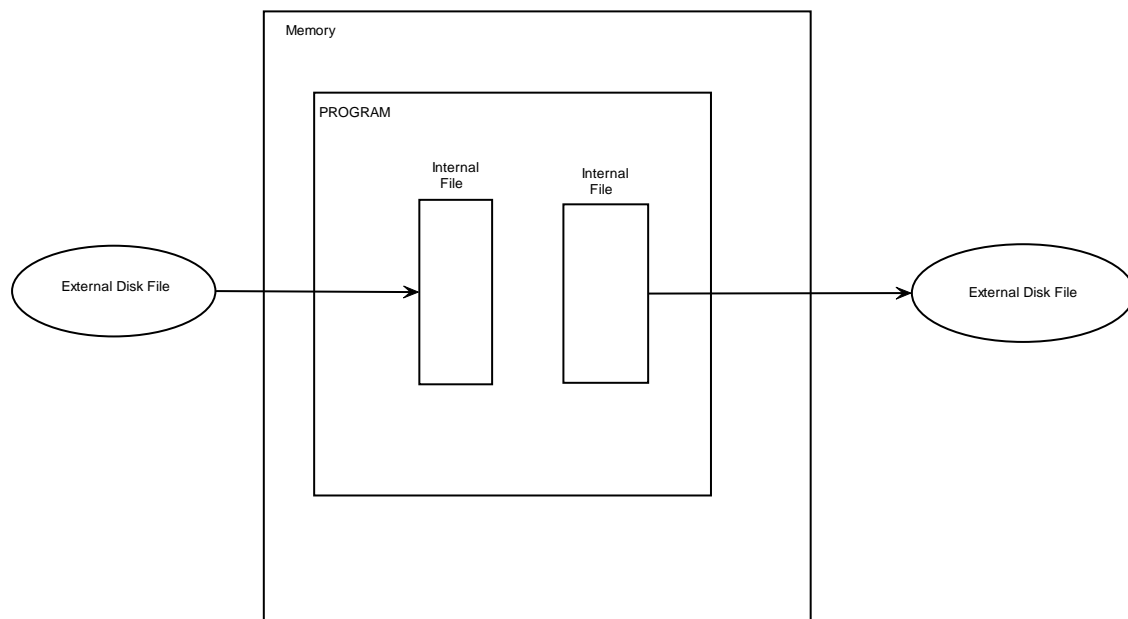


Figure 7.2. External and Internal Files.

are those used by the operating system for actual files that exist on a hard disk (say) whether the program is running or not. Indeed, they exist even when the computer itself is not running. (see Figure 2)

Before an external file can be used by a program it must be connected to an internal file. After this connection is made the program uses the internal file to read in or write out data. This is because, to the program, the external file name is merely a string of characters that is used by the operating system to identify files.

This is somewhat similar to actual and formal parameters in procedures : formal parameters are internal to the procedure and actual parameters are external to the procedure. The connection between the two sets of parameters is made when the procedure is called (used).

Turbo Pascal provides the `Assign` procedure to connect internal to external files. It has the general form :

```
Assign(<internal file name>, <external file name>);
```

where `<internal file name>` is a declared Pascal file variable and `<external file name>` is a string literal or variable that contains the name of the actual file on the hard disk.

Here are some examples :

```
Assign(FileXYZ, 'Student.DAT');
Assign(NewFile, 'C:\Fortran\Progs\Brent.FOR');
ExtFile := 'Ass-98-1.PAS';
Assign(FProg, ExtFile);
```

NOTE : If the external file named in the `Assign` procedure does not exist then a new, empty file is created.

7.4 OPENING AND CLOSING INTERNAL FILES

Once we have made the connection between the internal and external file we work with the internal file only. However, before we do anything with the internal file we must tell the program how we intend to use it : (1) for *Input* or reading, or (2) for *Output* or writing. In other words we must tell the program if a sequence of characters is to flow in from the file (input) or flow out to the file (output). Pascal provides two procedures to do this :

```
Reset (<internal file name>);    {Open for Reading}
ReWrite(<internal file name>);    {Open for Writing}
```

Caution : If the `ReWrite` procedure is used on an existing file then any data in it is destroyed.

When we are finished reading or writing a file we must *Close* it. For various technical reasons we may lose data or a complete file if a program ‘crashes’ and a file is open. Hence it is important to close a file as soon as we are finished with it.

The Pascal procedure for closing a file is :

```
Close(<internal file name>);    {Closes any open file}
```

7.5 READING AND WRITING TEXT FILES

Reading and writing to files is very similar to reading from the keyboard and writing to the screen. In fact the same procedures are used. However, we must tell the procedures the name of the file to use. The general form of calling these procedures is as follows :

```
Read ( <internal file name>, <list of variables> );
ReadLn( <internal file name>, <list of variables> );
```

and

```
Write ( <internal file name>, <list of variables> );
WriteLn( <internal file name>, <list of variables> );
```

Rather than give abstract descriptions of operation of these procedures we will give examples of how they are used and how they work.

7.5.1 Opening And Reading An Existing Text File

Let us look at an example which (1) opens a text file for reading, (2) reads it in line-by-line, and (3) Writes each line out to the screen.

```

PROGRAM ShowFile;

    VAR
        IntFile : TEXT;
        ExtFile : STRING;
        Line    : STRING;

    BEGIN
        Write('Type in name of file to read :');
        Readln(ExtFile);
        Assign(IntFile, ExtFile);
        Reset(IntFile);

        WHILE NOT(Eof(IntFile)) DO BEGIN
            Readln(IntFile, Line);
            Writeln(Line);
        END; { while }

        Close(IntFile);
    END. {Prog ShowFile}

```

The first thing this program does is declare `IntFile` as a text file. Then it declares `ExtFile` as a string: this string will hold the actual DOS or Windows file name, for example, `C:\SUB\test.dat`.

The program then begins by asking the user for the external file name. Next, it *connects* the internal to the external file using the

```
Assign(<Internal filename>, <External file name>)
```

procedure. Then it *opens* the file by using the

```
Reset(< Internal file name>)
```

procedure. The program is now ready to read in the file.

The size of the file to read in is not known in advance so a **while** -loop is used which tests for the end of the file with the

```
Eof(<Internal file name>)
```

When the file is initially opened a *file pointer* is set to point to the first character in the file. This pointer is hidden and is used internally by the `Read` and `Write` and `Eof` procedures.

When the `Readln(IntFile, Line)` is executed it reads into the string variable `Line` all the characters up to the first end-of-line character. It then moves the file pointer past the end-of-line character to the first character on the next line.

The `Writeln(Line)` writes out the characters in `Line` to the screen.

The program then goes to the top of the **while** -loop, tests if the file pointer is pointing to an end-of-file character and if not reads in another line and writes it out to the screen. This is repeated until the **while** test finds the file pointer is pointing the an end-of file character.

7.5.2 Opening and Writing to a New Text File

In this example we will (1) open a new text file for writing, and, (2) write some results out to it.

```
PROGRAM WriteToNewFile;

  VAR
    OutFile : TEXT;
    NewExtF : STRING;
    i       : INTEGER;

  BEGIN
    Write('Type in name of new file :');
    Readln(NewExtF);
    Assign(OutFile, NewExtF);
    Rewrite(OutFile);

    FOR i := 1 TO 100 DO BEGIN
      Writeln(OutFile, i:5, i*i:5);
    END; { FOR }

    Close(OutFile);
  END. {Prog WriteToNewFile}
```

This simple program writes out a table of the first 100 integers and their squares.

Exercise 7.5.1 : Compile and run the two previous programs and see how they work. In the first program open an existing text file, e.g., the .PAS file of the first assignment. In the second, use the external file name 'SQUARES.DAT'. You will not see any screen output because it all goes to the file. When the program is finished use some text editor to look at 'SQUARES.DAT'.

See what happens when you change the `WriteLn` to a `Write`.

Add a statement to the second program so that you get output to the screen as well as the file. □

Exercise 7.5.1 : Modify the `ShowFile` program in 7.5.1 so that the file is read in character-by-character. Keep a count of the characters that are read in and at the end of the program write out the number of characters read in. This is the size of the file in bytes, because each character is stored in a byte. □

7.6 GENERAL PROCEDURES FOR OPENING FILES

We can see in the programs above that we have to do a few standard things each time we wish to open a file :

1. Get an external file name from the keyboard.
2. Assign it to an internal file name.
3. Open the internal file for reading or writing.

We will now write two procedures that do these three steps so that we don't have to repeat this code each time we open a file.

Open a File for Reading

```
PROCEDURE OpenFileR (VAR InFile : TEXT);
  VAR
    ExtFile : STRING;
  BEGIN
    Write('Type in name of file to read :');
    Readln(ExtFile);
    Assign(InFile, ExtFile);
    Reset(InFile);
  END; {PROC OpenFileR }
```

Open a File for Writing

```
PROCEDURE OpenFileW (VAR OutFile : TEXT);
  VAR
    ExtFile : STRING;
  BEGIN
    Write('Type in name of file to write :');
    Readln(ExtFile);
    Assign(OutFile, ExtFile);
    Rewrite(OutFile);
  END; {PROC OpenFileW }
```

We can now use these procedures in any program. Let us use `OpenFileW` in the program `WriteToNewFile`.

EXAMPLE : *Using a File-Opening Procedure.*

```
PROGRAM WriteToNewFile2;
  VAR
    OutFile : TEXT;
    i       : INTEGER;
  PROCEDURE OpenFileW (VAR OutFile : TEXT);
    VAR
      ExtFile : STRING;
    BEGIN
      Write('Type in name of file to write :');
      Readln(ExtFile);
      Assign(OutFile, ExtFile);
      Rewrite(OutFile);
    END; {PROC OpenFileW }
  { Start of Main Program }
  BEGIN
    OpenFileW(OutFile);
    FOR i := 1 TO 100 DO BEGIN
      Writeln(OutFile, i:5, i*i:5);
    END; { FOR }
    Close(OutFile);
  END. {Prog WriteToNewFile}
```

□

Exercise 7.6.1 : Write a single general file-opening procedure that opens a file for reading or writing.

□

The Files INPUT and OUTPUT We note that the `Read` `Readln` procedures can be used to input data from text files or the keyboard. Likewise the `Write` `Writeln` procedures can be used to output data to text files or the screen. In fact the keyboard and the screen are treated as text files that are automatically opened with the names `INPUT` and `OUTPUT`. Hence `Read(<variable list>)` is the same as `Read(INPUT, <variable list>)`, where `INPUT` is an internal file name automatically connected to the keyboard at run-time. Likewise, `Write(<variable list>)` is the same as `Write(OUTPUT, <variable list>)`, where `OUTPUT` is an internal file name automatically connected to the screen at run-time.

7.7 SUMMARY OF TEXT FILE PROCEDURES AND FUNCTIONS

We now summarize all the procedures and functions that can be applied to text files.

1. `Assign(<IntFile>, <ExtFile>)` Connects an internal file to an external file.
2. `Reset(<IntFile>)` Opens a file for reading.
3. `Rewrite(<IntFile>)` Opens a file for writing.
4. `Read(<IntFile>, <variable list>)` Reads from a file into a list of variables.
5. `Readln(<IntFile>, <variable list>)` Reads from a file into a list of variables and moves file pointer to the start of the next line.
6. `Write(<IntFile>, <variable list>)` Writes from a list of variables out to a file.
7. `Writeln(<IntFile>, <variable list>)` Write from a list of variables out to a file and moves file pointer to the start of the next (blank) line.
8. `Close(<IntFile>)` Closes a file that has been opened for reading or writing.
9. `Eof(<IntFile>)` Returns `true` if the file pointer is pointing to an end-of-file character in a file.
10. The pre-defined internal files `INPUT` and `OUTPUT` are connected to the keyboard and screen respectively and are automatically opened when the program runs and closed when the program is finished.

EXAMPLE : *File Copying*. The following program shows how all the above procedures and functions are used to copy one file to another.

```
PROGRAM CopyFile;
  PROCEDURE OpenFileR (VAR InFile : TEXT);
    VAR
      ExtFile : STRING;
    BEGIN
      Write('Type in name of file to read :');
      Readln(ExtFile);
      Assign(InFile, ExtFile);
      Reset(InFile);
    END; {PROC OpenFileR }
  PROCEDURE OpenFileW (VAR OutFile : TEXT);
    VAR
      ExtFile : STRING;
    BEGIN
      Write('Type in name of file to write :');
      Readln(ExtFile);
      Assign(OutFile, ExtFile);
      Rewrite(OutFile);
    END; {PROC OpenFileW }
  PROCEDURE Main;
    VAR
      IntFileR,
      IntFileW : TEXT;
      CharCount : 0..maxint;
      ch       : CHAR;
    BEGIN
      OpenFileR(IntFileR);
      OpenFileW(IntFileW);

      count := 0;
      WHILE (NOT(Eof(IntFileR))) DO BEGIN
        Read (IntFileR, ch);
        Write(IntFileW, ch);
        CharCount := CharCount + 1;
      END; { while }

      Writeln('The size of file copied = ', CharCount, ' bytes');

      Close(IntFileR);
      Close(IntFileW);
    END; {PROC Main}
  BEGIN
    Main;
  END. {Prog CopyFile}
```

□

Note : In the program above the integer count is incremented each time a character is read from the file. Thus, at the end of the program this variable contains the number of characters and, therefore,

bytes in the file. Further note that the character count includes all blanks, end-of-lines, and other non-printable characters.

Exercise 7.7.1 : Re-write the program above so that it reads and writes a line at a time rather than a character at a time. □

Here is the ASCII character code table, with NP for not-printable and \square for blank. Note that codes 32 to 126 are keyboard characters.

ASCII CODE NUMBERS AND CHARACTERS							
0	NP	32	\square	64	@	96	'
1	NP	33	!	65	A	97	a
2	NP	34	"	66	B	98	b
3	NP	35	#	67	C	99	c
4	NP	36	\$	68	D	100	d
5	NP	37	%	69	E	101	e
6	NP	38	&	70	F	102	f
7	NP	39	'	71	G	103	g
8	NP	40	(72	H	104	h
9	NP	41)	73	I	105	i
10	NP	42	*	74	J	106	j
11	NP	43	+	75	K	107	k
12	NP	44	,	76	L	108	l
13	NP	45	-	77	M	109	m
14	NP	46	.	78	N	110	n
15	NP	47	/	79	O	111	o
16	NP	48	0	80	P	112	p
17	NP	49	1	81	Q	113	q
18	NP	50	2	82	R	114	r
19	NP	51	3	83	S	115	s
20	NP	52	4	84	T	116	t
21	NP	53	5	85	U	117	u
22	NP	54	6	86	V	118	v
23	NP	55	7	87	W	119	w
24	NP	56	8	88	X	120	x
25	NP	57	9	89	Y	121	y
26	NP	58	:	90	Z	122	z
27	NP	59	;	91	[123	{
28	NP	60	<	92	\	124	
29	NP	61	=	93]	125	}
30	NP	62	>	94	^	126	~
31	NP	63	?	95	_	127	NP

Laboratory Exercise No. 7: *Encrypting Text Files* .

The use of the Internet and Email has increased the demand for encryption programs that take a plain text file and encrypt it (scramble) it so that it cannot be read by anyone during its transmission.

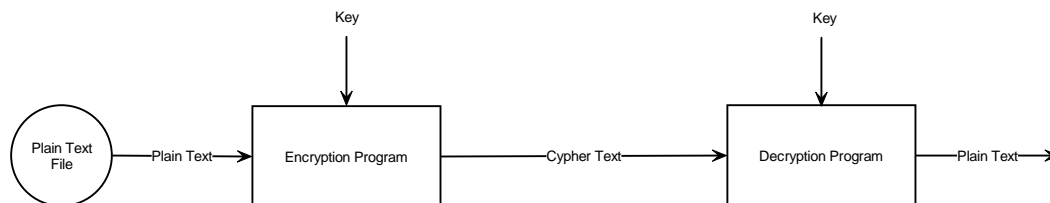


Figure 7.3. Encryption and Decryption.

We can use the CopyFile program above to make a scrambled copy of a file. Instead of writing out each character that we read in we write out an encrypted character as follows :

$$\text{EnCh} := \text{Chr}(\text{Ord}(\text{Ch}) + \text{Random}(127)) \text{ MOD } 127)$$

Remember that $\text{Ord}(\text{Ch})$ transforms an ASCII character Ch to its ASCII code number and $\text{Chr}(i)$ transforms an ASCII code number i to its ASCII character. $\text{Random}(n)$ is a built-in function that returns a non-negative integer, uniformly distributed between 0 and n . The encryption works as follows :

1. $\text{Ord}(\text{Ch})$ gives the character code between 0 and 127.
2. $\text{Ord}(\text{Ch}) + \text{Random}(1000)$ adds a random number uniformly distributed between 0 and 1000. The result is a random number between 0 and 1127.
3. $(\text{Ord}(\text{Ch}) + \text{Random}(1000)) \text{ MOD } 127$ gives a random number between 0 and 127.
4. $\text{Chr}((\text{Ord}(\text{Ch}) + \text{Random}(1000)) \text{ MOD } 127)$ give an ASCII character corresponding to this random number.

For example, the encryption of the character 'M' is

1. $\text{Ord}('M') = 109$.
2. $\text{Ord}('M') + \text{Random}(1000) = 109 + 576 = 685$.
3. $685 \text{ MOD } 127 = 50$.
4. $\text{Chr}(50) = '2'$.

In this lab exercise do the following :

1. Get the CopyFile program working. Use the file EncBr11.TXT as the test input file. This is available on the Class Web Page. Download it and save to your hard disk.
2. Add the encryption statement above and write the encrypted character to the output file (call it EncBr11.ENC).
3. View the encrypted file with some editor.

Exercise 7.7.1 : How do you *decrypt* a file you have encrypted in lab exercise 7? □