**File Handling**

In this section, we will discuss about files which are very important for storing information permanently. We store information in files for many purposes, like data processing by our programs.

**What is a File?**

Abstractly, a file is a collection of bytes stored on a secondary storage device, which is generally a disk of some kind. The collection of bytes may be interpreted, for example, as characters, words, lines, paragraphs and pages from a textual document; fields and records belonging to a database; or pixels from a graphical image. The meaning attached to a particular file is determined entirely by the data structures and operations used by a program to process the file. It is conceivable (and it sometimes happens) that a graphics file will be read and displayed by a program designed to process textual data. The result is that no meaningful output occurs (probably) and this is to be expected. A file is simply a machine decipherable storage media where programs and data are stored for machine usage.

Essentially there are two kinds of files that programmers deal with text files and binary files. These two classes of files will be discussed in the following sections.

**ASCII Text files**

A text file can be a stream of characters that a computer can process sequentially. It is not only processed sequentially but only in forward direction. For this reason a text file is usually opened for only one kind of operation (reading, writing, or appending) at any given time.

Similarly, since text files only process characters, they can only read or write data one character at a time. (In C Programming Language, Functions are provided that deal with lines of text, but these still essentially process data one character at a time.) A text stream in C is a special kind of file. Depending on the requirements of the operating system, newline characters may be converted to or from carriage-return/linefeed combinations depending on whether data is being written to, or read from, the file. Other character conversions may also occur to satisfy the storage requirements of the operating system. These translations occur transparently and they occur because the programmer has signalled the intention to process a text file.

**Binary files**

A binary file is no different to a text file. It is a collection of bytes. In C Programming Language a byte and a character are equivalent. Hence a binary file is also referred to as a character stream, but there are two essential differences.

1. No special processing of the data occurs and each byte of data is transferred to or from the disk unprocessed.
2. C Programming Language places no constructs on the file, and it may be read from, or written to, in any manner chosen by the programmer.

Binary files can be either processed sequentially or, depending on the needs of the application, they can be processed using random access techniques. In C Programming Language, processing a file using random access techniques involves moving the current file position to an appropriate place in the file before reading or writing data. This indicates a second characteristic of binary files.  
They a generally processed using read and write operations simultaneously.

For example, a database file will be created and processed as a binary file. A record update operation will involve locating the appropriate record, reading the record into memory, modifying it in some way, and finally writing the record back to disk at its appropriate location in the file. These kinds of operations are common to many binary files, but are rarely found in applications that process text files.

**Creating a file and output some data**

In order to create files we have to learn about File I/O i.e. how to write data into a file and how to read data from a file. We will start this section with an example of writing data to a file. We begin as before with the include statement for stdio.h, then define some variables for use in the example including a rather strange looking new type.

/\* Program to create a file and write some data the file \*/

#include <stdio.h>

#include <stdio.h>

main( )

{

FILE \*fp;

char stuff[25];

int index;

fp = fopen("TENLINES.TXT","w"); /\* open for writing \*/

strcpy(stuff,"This is an example line.");

for (index = 1; index <= 10; index++)

fprintf(fp,"%s Line number %d\n", stuff, index);

fclose(fp); /\* close the file before ending program \*/

}

The type FILE is used for a file variable and is defined in the stdio.h file. It is used to define a file pointer for use in file operations. Before we can write to a file, we must open it. What this really means is that we must tell the system that we want to write to a file and what the file name is. We do this with the fopen() function illustrated in the first line of the program. The file pointer, fp in our case, points to the file and two arguments are required in the parentheses, the file name first, followed by the file type.

The file name is any valid DOS file name, and can be expressed in upper or lower case letters, or even mixed if you so desire. It is enclosed in double quotes. For this example we have chosen the name TENLINES.TXT. This file should not exist on your disk at this time. If you have a file with this name, you should change its name or move it because when we execute this program, its contents will be erased. If you donâ€™t have a file by this name, that is good because we will create one and put some data into it. You are permitted to include a directory with the file name.The directory must, of course, be a valid directory otherwise an error will occur. Also, because of the way C handles literal strings, the directory separation character â€˜\â€™ must be written twice. For example, if the file is to be stored in the \PROJECTS sub directory then the file name should be entered as “\\PROJECTS\\TENLINES.TXT”. The second parameter is the file attribute and can be any of three letters, r, w, or a, and must be lower case.

**Reading (r)**

When an r is used, the file is opened for reading, a w is used to indicate a file to be used for writing, and an a indicates that you desire to append additional data to the data already in an existing file. Most C compilers have other file attributes available; check your Reference Manual for details. Using the r indicates that the file is assumed to be a text file. Opening a file for reading requires that the file already exist. If it does not exist, the file pointer will be set to NULL and can be checked by the program.

Here is a small program that reads a file and display its contents on screen.

/\* Program to display the contents of a file on screen \*/

#include <stdio.h>

void main()

{

FILE \*fopen(), \*fp;

int c;

fp = fopen("prog.c","r");

c = getc(fp) ;

while (c!= EOF)

{

putchar(c);

c = getc(fp);

}

fclose(fp);

}

**Writing (w)**

When a file is opened for writing, it will be created if it does not already exist and it will be reset if it does, resulting in the deletion of any data already there. Using the w indicates that the file is assumed to be a text file.

Here is the program to create a file and write some data into the file.

#include <stdio.h>

int main()

{

FILE \*fp;

file = fopen("file.txt","w");

/\*Create a file and add text\*/

fprintf(fp,"%s","This is just an example :)"); /\*writes data to the file\*/

fclose(fp); /\*done!\*/

return 0;

}

**Appending (a)**

When a file is opened for appending, it will be created if it does not already exist and it will be initially empty. If it does exist, the data input point will be positioned at the end of the present data so that any new data will be added to any data that already exists in the file. Using the a indicates that the file is assumed to be a text file.

Here is a program that will add text to a file which already exists and there is some text in the file.

#include <stdio.h>

int main()

{

FILE \*fp

file = fopen("file.txt","a");

fprintf(fp,"%s","This is just an example :)"); /\*append some text\*/

fclose(fp);

return 0;

}

**Outputting to the file**

The job of actually outputting to the file is nearly identical to the outputting we have already done to the standard output device. The only real differences are the new function names and the addition of the file pointer as one of the function arguments. In the example program, fprintf replaces our familiar printf function name, and the file pointer defined earlier is the first argument within the parentheses. The remainder of the statement looks like, and in fact is identical to, the printf statement.

**Closing a file**

To close a file you simply use the function fclose with the file pointer in the parentheses. Actually, in this simple program, it is not necessary to close the file because the system will close all open files before returning to DOS, but it is good programming practice for you to close all files in spite of the fact that they will be closed automatically, because that would act as a reminder to you of what files are open at the end of each program.

You can open a file for writing, close it, and reopen it for reading, then close it, and open it again for appending, etc. Each time you open it, you could use the same file pointer, or you could use a different one. The file pointer is simply a tool that you use to point to a file and you decide what file it will point to. Compile and run this program. When you run it, you will not get any output to the monitor because it doesnâ€™t generate any. After running it, look at your directory for a file named TENLINES.TXT and type it; that is where your output will be. Compare the output with that specified in the program; they should agree! Do not erase the file named TENLINES.TXT yet; we will use it in  
some of the other examples in this section.

Reading from a text file

Now for our first program that reads from a file. This program begins with the familiar include, some data definitions, and the file opening statement which should require no explanation except for the fact that an r is used here because we want to read it.

#include <stdio.h>

main( )

{

FILE \*fp;

char c;

funny = fopen("TENLINES.TXT", "r");

if (fp == NULL)

printf("File doesn't exist\n");

else {

do {

c = getc(fp); /\* get one character from the file

\*/

putchar(c); /\* display it on the monitor

\*/

} while (c != EOF); /\* repeat until EOF (end of file)

\*/

}

fclose(fp);

}

In this program we check to see that the file exists, and if it does, we execute the main body of the program. If it doesn’t, we print a message and quit. If the file does not exist, the system will set the pointer equal to NULL which we can test. The main body of the program is one do while loop in which a single character is read from the file and output to the monitor until an EOF (end of file) is detected from the input file. The file is then closed and the program is terminated. At this point, we have the potential for one of the most common and most perplexing problems of programming in C. The variable returned from the getc function is a character, so we can use a char variable for this purpose. There is a problem that could develop here if we happened to use an unsigned char however, because C usually returns a minus one for an EOF – which an unsigned char type variable is not  
capable of containing. An unsigned char type variable can only have the values of zero to 255, so it will return a 255 for a minus one in C. This is a very frustrating problem to try to find. The program can never find the EOF and will therefore never terminate the loop. This is easy to prevent: always have a char or int type variable for use in returning an EOF. There is another problem with this program but we will worry about it when we get to the next program and solve it with the one following that.

After you compile and run this program and are satisfied with the results, it would be a good exercise to change the name of TENLINES.TXT and run the program again to see that the NULL test actually works as stated. Be sure to change the name back because we are still not finished with TENLINES.TXT