Lesson 10 USING FILES IN C

Many programs are required to read and write data to a persistent storage device, usually a disk. C provides input/output functions for carrying out such tasks. Disk input and output functions typically operate on files. A file is just a collection of bytes that is given a name.

File Output

The type **FILE** is a structure and is defined in stdio.h. It is used to define a file pointer for use in file operations. The definition of C requires a pointer to a **FILE** type to access a file. The name can be any valid variable name.

Opening a File

Before we can write to a file, we must open it. This really means that we must tell the system that we want to write to a file and what the actual filename is. We can do this with the **fopen()** function. This function must be supplied with the name of the file and the file attribute. The filename should not be an existing file if you do not want the contents to be overwritten. When the program is run the file will be created.

The file attribute can be either "r", "w", or "a". There are additional attributes available in C to allow more flexible input/output, check your compiler for specific details. 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.

Opening a file for reading requires that the file already exists. If it does not exist, the file pointer will be set to NULL and can be checked by the program. The following lines can do this.

```
     if(fp == NULL)
       {
        printf("File failed to open\n");
        exit (1);
       }
```

It is good programming practice to make sure that all file pointers are checked to assure proper file opening in a manner similar to the above code. 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 deletion of any data already there. If the file fails to open for any reason, a NULL will be returned so the pointer should be tested as above.

Appending a File

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 set to the end of the data already contained in the file so that new data will be added to any data that already exists in the file.

Outputting to the File

The job of actually outputting to a file is nearly the same to what we have already seen, outputting to the standard output device i.e. the screen. 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 and the file pointer defined earlier is the first argument within the parentheses. The remainder of the statement looks like the printf() statement.

Closing a file

To close a file, use the function **fclose()** with the file pointer in the parentheses. In this simple program, it is not necessary to close the file because the system will close all open files before returning to the operating system. It is good programming practice for you to get in the habit of closing 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 a simply a tool that you use to point to a file and you decide what file it will point to.

```
#include<stdio.h>
int main()
     {
       FILE *fp;
       char hold[45];
       int  i;

       fp = fopen("test.txt", "w");   /* open for writing */
       strcpy(hold, "HELLO WORLD, AGAIN.");

       for(i = 1 ; i < 20 ; i++)
          fprintf(fp, "%s  Line number %d\n", hold, i);
       fclose(fp);
       return(0);
      }
```

The putc() function

The **putc()** function outputs one character at a time, the character is the first argument in the parentheses and the file pointer is the second argument.

```
#include<stdio.h>
int main()
     {
       FILE *fp;
       char hold[45];
       int  i;

       fp = fopen("test2.txt", "w");   /* open for writing */
       strcpy(hold, "HELLO WORLD, AGAIN AND AGAIN");

       for(i = 1 ; i < 20 ; i++)
          putc(hold[i],fp);
       fclose(fp);
       return(0);
       }
```

Reading a File

We can read from a file using the **getc()** function. Consider the following code.

```
#include<stdio.h>
int main()
       {
        FILE *fptr;
        int k;
        fptr = fopen("test.txt", "r");
        if(fptr == NULL)
           {
           printf("File NOT Found\n");
           exit(1);
           }
         else
           {
              do
               {
                k=getc(fptr);
                 putchar(k);
               } while (k != EOF);
            }
                 fclose(fptr);
        }
```

The main body of the program is one *do while* loop in which a single character is read from the file and outputted to the screen until an EOF is detected from the input file. The file is then closed and the program terminated.

Suppose that we wanted to read in a word instead of a character, we can use the **fscanf()** function. This function stops reading when it finds a space or a newline character, it will read a word at a time, and display the results one word to a line.

To read an entire line, we can use the function **fgets().** This reads the line and also the `\n` character into a buffer. The first argument of the function call contains the name of the buffer, the second argument contains the maximum number of characters to be read into the buffer and the third argument contains the file pointer. Note that the function will read characters into the buffer until the newline character is meet or until it reads the maximum number of characters allowed minus one. If the function finds an EOF, then it will return a value of NULL. See the code examples in the lesson for fgets().

```
#include<stdio.h>
#include<stdlib.h>
int main()
  {
   FILE *fptr;
   char buffer[80];
   int k;
   fptr = fopen("test.txt", "r");
   do
     {
       k = fscanf(fptr, "%s", buffer);
       if(k != EOF)
```

```
         printf("%s\n", buffer);
     } while (k != EOF);
    fclose(fptr);
    return(0);
   }
```

Using a variable filename

In most circumstances we want our program to accept any filename for manipulation type in by the user. Thus we need to declare a variable filename. The following code allows for this.

```
#include<stdio.h>
#include<stdlib.h>
int main()
   {
   FILE *fptr;
   char buffer[80], file[20];
   char *k;
   printf("Enter filename -> ");
   scanf("%s", file);
   fptr = fopen(file, "r");
   if(fptr == NULL)
     {
     printf("File NOT found \n");
     exit(1);
     }
   do
     {
     k = fgets(buffer, 80, fptr);
     if(k != NULL)
       printf("%s", buffer);
     }while (k != NULL);
   fclose(fptr);
   return(0);
   }
```

fopen() attributes

"r"	Open for reading. File already exists
"w"	Open for writing. If file exists it will be overwritten, if not it will be created
"a"	Open for append. Existing file is appended or new file will be created
"r+"	Open for reading and writing, file must exist
"w+"	Open for reading and writing, existing file will be overwritten
"a+"	Open for reading and appending. File will be created if it dosnt exist

NOTE: The above table is handling files in text mode (by default). By writing the letter `b` after the letters above, example "a+b" we can handling a file in binary mode.

Programming Exercise

1. Write up your own notes on the following functions open(), fclose(), putc(), printf(), fprintf(), scanf().
2. Write a program that will search for a given character in a given file.

3. Write a program that will prompt for a filename for an input file, send its contents to the printer.
4. Write a program that counts the number of lines in a file.(HINT \n characters)

5. Write a program that will get rid of blank lines in an input file and write the processed file to an output file. Example input and output are as follows

FREQUENCY	IMPEDENCE
100	101.1
.	.
.	.
150	149.2
200	300.6

FREQUENCY	IMPEDENCE
100	101.1
150	149.2
200	300.6