FILE HANDLING IN C PROGRAMMING

In any programming language it is vital to learn file handling techniques. Many applications will at some point involve accessing folders and files on the hard drive. In C, a stream is associated with a file. Special functions have been designed for handling file operations. Some of them will be discussed in this chapter. The header file stdio.h is required for using these functions.

Opening a file

Before we perform any operations on a file, we need to open it. We do this by using a file pointer. The type *FILE* defined in stdio.h allows us to define a file pointer. Then you use the function *fopen()* for opening a file. Once this is done one can read or write to the file using the *fread()* or *fwrite()* functions, respectively. The fclose() function is used to explicitly close any opened file.

Taking the preceding statements into account let us look at the following example program :

Program 13.1

```
#include <stdio.h>
  main ()
  {
      FILE *fp;
      fp = fopen("data.txt", "r");
```

```
      if (fp == NULL) {
        printf("File does not exist,
please check!\n");
      }
      fclose(fp);
  }
```

fopen()

Let us first discuss *fopen()*. This function accepts two arguments as strings. The first argument denotes the name of the file to be opened and the second signifies the mode in which the file is to be opened. The second argument can be any of the following:

File Mode	Description
r	Open a text file for reading
w	Create a text file for writing, if it exists, it is overwritten.
a	Open a text file and append text to the end of the file.
rb	Open a binary file for reading
wb	Create a binary file for writing, if it exists, it is overwritten.
ab	Open a binary file and append data to the end of the file.

Table 13.1

fclose()

The *fclose()* function is used for closing opened files. The only argument it accepts is the file pointer.

If a program terminates, it automatically closes all opened files. But it is a good programming habit to close any file once it is no longer needed. This helps in better utilization of system resources, and is very useful when you are working on numerous files simultaneously. Some operating systems place a limit on the number of files that can be open at any given point in time.

fscanf() and fprintf()

The functions *fprintf()* and *fscanf()* are similar to *printf()* and *scanf()* except that these functions operate on files and require one additional and first argument to be a file pointer.

Program 13.2

```
#include <stdio.h>
main ()
{
  FILE *fp;
  float total;
  fp = fopen("data.txt", "w+");
  if (fp == NULL) {
    printf("data.txt does not exist, please check!\n");
    exit (1);
```

```
  }
  fprintf(fp, 100);
  fscanf(fp, "%f", &total);
  fclose(fp);
  printf("Value of total is %f\n", total);
}
```

getc() and putc()

The functions *getc()* and *putc()* are equivalent to *getchar()* and *putchar()* functions which you studied in Chapter 12 on Input and Output, except that these functions require an argument which is the file pointer. Function *getc()* reads a single character from the file which has previously been opened using a function like *fopen()*. Function *putc()* does the opposite, it writes a character to the file identified by its second argument. The format of both functions is as follows :

getc(in_file);

putc(c, out_file);

Note: The second argument in the *putc()* function must be a file opened in either write or append mode.

Program 13.3

```
#include <stdio.h>
main ()
```

```
{
  char in_file[30], out_file[30];
  FILE *fpin, *fpout;
  int c;
  printf("This program copies the source file to the destination
file
\n\n");
  printf("Enter name of the source file :");
  scanf("%30s", in_file);
  printf("Enter name of the destination file :");
  scanf("%30s", out_file);
  if((fpin=fopen(in_file, "r")) == NULL)
    printf("Error could not open source file for
reading\n");
  else if ((fpout=fopen(out_file, "w")) == NULL)
    printf("Error could not open destination file for
reading\n");
  else
   {
    while((c =getc(fpin)) != EOF)
      putc(c, fpout);
    printf("Destination file has been copied\n");
  }
}
```

Binary stream input and output

The functions *fread()* and *fwrite()* are a somwhat complex file handling functions used for reading or writing chunks of data containing NULL characters ('\0') terminating strings.

The function prototype of fread() and fwrite() is as below :

```
size_t fread(void *ptr, size_t sz, size_t n, FILE *fp)
size_t fwrite(const void *ptr, size_t sz, size_t n, FILE *fp);
```

You may notice that the return type of *fread()* is size_t which is the number of items read. You will understand this once you understand how fread() works. It reads *n* items, each of size *sz* from a file pointed to by the pointer *fp* into a buffer pointed by a void pointer *ptr* which is nothing but a generic pointer. Function *fread()* reads it as a stream of bytes and advances the file pointer by the number of bytes read. If it encounters an error or end-of-file, it returns a zero, you have to use feof() or ferror() to distinguish between these two. Function *fwrite()* works similarly, it writes *n* objects of *sz* bytes long from a location pointed to by *ptr*, to a file pointed to by *fp*, and returns the number of items written to *fp*.

Let us rewrite program 13.3 using fread() and fwrite() functions .

Program 13.4

```
#include <stdio.h>
#define MAX_SIZE 1024
main ()
{
  FILE *fp, *gp;
  char buf[MAX_SIZE];
```

```
  int i, total = 0;
  if ((fp = fopen("data1.txt", "r") ) == NULL)
    printf("Error in data1.txt file \n");
  else if ((gp=fopen("data2.txt", "w")) == NULL)
    printf("Error in data2.txt file \n");
  else
   {
    while(i=fread(buf, 1, MAX_SIZE, fp))
    {
     fwrite(buf, 1, MAX_SIZE, gp);
     total +=i;
    }
    printf("Total is %d\n", total);
  }
  fclose(fp);
  fclose(gp);
}
```

ftell()

Functions *ftell()* and *fseek()* are important in a program performing file manipulations. Function *ftell()* returns the current position of the file pointer in a stream. The return value is 0 or a positive integer indicating the byte offset from the beginning of an open file. A return value of -1 indicates an error. Prototype of this function is as shown below :

long int ftell(FILE *fp);

fseek()

This function positions the next I/O operation on an open stream to a new position relative to the current position.

int fseek(FILE *fp, long int offset, int origin);

Here *fp* is the file pointer of the stream on which I/O operations are carried on, *offset* is the number of bytes to skip over. The *offset* can be either positive or negative, denting forward or backward movement in the file. *origin* is the position in the stream to which the offset is applied, this can be one of the following constants :

SEEK_SET : offset is relative to beginning of the file
SEEK_CUR : offset is relative to the current position in the file
SEEK_END : offset is relative to end of the file

Program 13.5

```
#include <stdio.h>
#include <stdlib.h>
char buffer[11];
int position;
main ()
{
  FILE *file_ptr;
  int num;
  if ((file_ptr = fopen("test_file", "w+f 10"))
== NULL)
  {
```

```
   printf("Error opening test_file \n");
   exit(1);
  }
  fputs("1111111111", file_ptr);
  fputs("2222222222", file_ptr);
  if ( (position = fseek(file_ptr, 10, SEEK_SET)) != 0)
  {
   printf("Error in seek operation: errno \n");
   exit(1);
  }
  num = 11;
  fgets(buffer, num, file_ptr);
  printf("The record is %s\n", buffer);
  fclose(file_ptr);
}
```

The output from the preceding program will be

The record is 2222222222.

Source : http://www.peoi.org/Courses/Coursesen/cprog/frame13.html