19

DATA FILES

19.1 INTRODUCTION

Many applications require that information be written to or read from an auxillary storage device. Such information is stored on the storage device in the form of data file. Thus, data files allow us to store information permanently and to access later on and alter that information whenever necessary. In C, a large number of library functions is available for creating and processing data files. There are two different types of data files called stream-oriented (or standard) data files and system oriented data files. We shall study stream-orinted data files only in this lesson.

19.2 OBJECTIVES

After going through this lesson you will be able to

- open and close a data file
- create a data file
- process a data file

19.3 OPENING AND CLOSING A DATA FILE

We must open the file before we can write information to a file on a disk or read it. Opening a file establishes a link between the program and the operating system. The link between our program and

the operating system is a structure called FILE which has been defined in the header file “stdio.h”. Therefore, it is always necessary to include this file when we do high level disk I/O. When we use a command to open a file, it will return a pointer to the structure FILE. Therefore, the following declaration will be there before opening the file,

FILE *fp each file will have its own FILE structure. The FILE structure contains information about the file being used, such as its current size, its location in memory etc. Let us consider the following statements,

```
FILE *fp;
fp=fopen(“Sample.C,” “r”);
```

fp is a pointer variables, which contains the address of the structure FILE which has been defined in the header file “stdio.h”. fopen() will oepn a file “sample.c” in ‘read’ mode, which tells the C compiler that we would be reading the contents of the file. Here, “r” is a string and not a character.

When fopen() is used to open a file then, it searches on the disk the file to be opened. If the file is present, it loads the file from the disk into memory. But if the file is absent, fopen() returns a NULL. It also sets up a character pointer which points to the first character of the chunk of memory where the file has been loaded.

Reading A file:

To read the file’s contents from memory there exists a function called fgetc(). This is used as:

```
s=fgetc(fp);
```

fgetc() reads the character from current pointer position, advances the pointer position so that it now points to the next character, and returns the character that is read, which we collected in the variable s. This fgetc() is used within an indefinite while loop, for end of file. End of file is signified by a special character, whose ascii value is 26.

While reading from the file, when fgetc() encounters this Ascii special character, instead of returning the characters that it has read, it returns the macro EOF. The EOF macro has been defined in the file “stdio.h”.

When we finished reading from the file, there is need to close it.

This is done using the function fclose() through the following statement:

```
fclose(fp);
```

This command deactivates the file and hence it can no longer be accessed using getc().

'C' provides many different file opening modes which are as follows:

1. "r" Open the file for reading only.
2. "w" Open the file for writing only.
3. "a" Open the file for appending (or adding) data to it.
4. "r+" The existing file is opened to the beginning for both reading and writing.
5. "w+" Same as "w" except both for reading and writing.
6. "a+" Same as "a" except both for reading and writing.

INTEXT QUESTIONS

1. Distinguish briefly between input and output.
2. What is the major difference between the end of a string and the end of a file ?
3. What is EOF, and what value does it usually have ?
4. What must be done to a file before it can be used ?
5. How does the fopen function work?

19.4 CREATING A DATA FILE

A data file must be created before it can be processed. A stream-oriented data file can be created in two ways. The first one is to create the file directly using a text editor or word processor. The second one is to write a program that generates information in a computer and then writes it out to the data file. We create unformatted data file with the second method.

Let us consider following program in C.

```
#include "stdio.h"

main()

{ FILE *fs,
```

```
char ch;

fs=fopen(“sample1.c”, “w”);

do

putc (toupper (ch=getchar (,), fs);

while (ch!= '\h');

fclose (fs);
```

This program starts with definning the stream pointer fs, indicating the beginning of the data–file buffer area. A new data file called sample1.c is then opened for writing only. Next a do–while loop reads a series of characters from the keyboard and writes their uppercase equivalents to the data file. The putc function is used to write each character to the data file. Notice that putc requires specification of the stream pointer fs as an argument.

The loop continues as long as a newline character (\n) is not entered from the keyboard. Once a newline character is detected, loop comes to an end and data file is closed. After executing the program, the data file sample1.C will contain an uppercase equivalent to the line of text entered into the computer from the keyboard. For example, if the orgional file contains the following text "param" is a good boy. Then the data file sample1.c will contain the following text.

```
PARAM IS A GOOD BOY.
```

19.5 PROCESSING A DATA FILE

To execute any program we are required to first enter the program, compile it, and then execute it. Instead of the program prompting for entering the source and target filenames it can be through command prompt in the form:

```
C> filecopy sample1.c sample2.c
```

where sample1.c is the source filename and sample2.c is target filename.

The second option is possible by passing the source filename and target filename to the function main(). Let us first consider an example:

```
#include “stdio.h>
main( int argc,char * argv [])
```

```
{
FILE *fs , *ft;
```

The arguments which are passed on to main() at the command prompt are called command line arguments. These are named as argc & argv. argv is an array of pointer to strings and argc is an int whose value is equal to the number of strings to which argv points.

When the program is executed, the strings on the command line are passed to main(). The string at the command line are stored in memory and address of the first string is stored in argv[0], address of the second string is stored in argv[1] and so on. The no. of strings given on the command line are stored in argc.

We can use fputc() in while loop as follows:

```
while(!feof(fs))
{
ch=fgetc(fs);
fputc(ch, ft);
}
```

Here, feof() is a macro which returns a 0 if end of file is not reached. When the end of file is reached feof() returns a non-zero value, ! makes it 0.

You can read or write strings of characters from and to files instead of single character by the help of fputs(). Let us consider an example.

```
#include “stdio.h”
main()
{
        FILE * fp;
        char str[80];
        fp=fopen(“sample.txt” , “w”) ;
        if ( fp = = NULL)
{
        puts(“cannot open file”);
        exit();
```

```
}
        printf("\n enter a few lines of text :\n");
        while(  strlen(gets(str))>0)
{
        fputs(str, fp);
 fputs("\n");
}
fclose(fp);
}
```

fputs () function writes the contents of the array to the disk . Since fputs() does not automatically add a new line character the end of the string, it must be explicitly done. Let us consider an example of reading strings from a disk file.

```
#include "stdio.h"
main()
{
        FILE *fp;
        char str[80];
fp=fopen("sample.txt,"r");
If (fp= = NULL)
{
        puts(" Cannot open file");
exit();
}
while(fgets(str,  79,fp)!=NULL)
printf("%s",s);
fclose(fp);
}
```

The function fputs() takes three arguments. The first is the address where the string is stored, second is the maximum length of the string. The third argument is the pointer to the structure FILE.

For formatted reading and writing of characters, strings, integers, floats, there exists two functions, fscanf and fprintf(). Let us con-

sider an example:

```
#include “stdio.h”
main()
{
FILE *fp;
char choice= ‘y’;
char name[20];
int age;
float basic;
fp=fopen (“Emp.dat”, “w”);
if(fp= =NULL)
{
puts(“Cannot open file”);
exit();
}
while (choice = = ‘y’)
{
printf(“\n” Enter name, age and basic salary \n”);
scanf(“\ %s%d %f,”  name, &age, & basic);
fprintf(fp,”%s %d %f \n”,name, &age, &basic );
printf(“Another details(y/n)”);
fflush(stdin);
ch= getche() ;
}
 fclose (fp);
}
```

fpritnf() , writes the values of three variables to the file. If we want to read the details which we had entered through fprintf() then we have to use fscanf() within the loop. Let us consider an example:

```
#include “stdio.h”
main()
```

```
{
        FILE* fp;
        char name (4ϕ);
        int age;
float basic;
fp=fopen("Emp.dat", "r");
if (fp = = NULL)
{
puts("Cannot open file");
exit ();
}
while (fscanf(fp, "%s %d %f", name, &age, &basic)!=EOF)
printf("\n %s %d %f", name,&age, &basic);
fclose(fp);
}
```

fscanf() function is used to read the data from the disk.

The program which we have used till now, used text modes in which the file is opened. But now we will discuss the binary mode also which are different from text mode in the following manner:

- Handling of newlines
- representation of end of file
- storage of numbers

The format is

e.g. fp=fopen("Sample.txt", "rb");

In case of binary modes, there is no special character present to mark the end of file. The binary mode files keep track of the end of file from the number of characters present in the directory entry of the file. The file that has been written in binary mode must be read back only in binary mode.

If large amount of numerical data is to be stored in a disk file, using text mode may turn out to be inefficient. The solution is to open the file in binary mode. Then, only each number would occupy same number of bytes on disk as it occupies in memory.

RECORD I/O IN FILES

If you want to write a combination of dissimilar data types, you have to use structures let us consider an example

```
        #include “stdio.h”
main()
{
        FILE *fp;
        char ch= ‘y’;
        struct emp
{
char name[40];
int age;
float basic;
};
struct emp el;
fp= fopen( “Emp.dat:”,”w”);
if(fp = = NULL)
{
        puts(“Cannot oepn file”);
        exit();
}
        while(ch= = ‘y’ )
{
        printf(“\n Enter name, age, and basic salary”);
        scanf(     “%s%d%f”, e.name, &e.age, &e.basic);
        fprintf(fp, “%s%d%f\n, e.name, e.age, e.basic);
printf(“want to add more record (y/n)”);
fflush(stdin);
ch=getch();
}
fclose(fp);
}
```

In this program, the variable(basic) which is number would occupy more number of bytes, since the file has been opened in text mode. This is because when the file is opened in text mode, each number is stored as a character string. Let us consider the following examples which is receiving records from keyboard and writing them to a file in binary mode.

```
# include “stdio.h”
main()
{
FILE *fp;
char ch=’y’;
struct emp
{
char name[40];
int age;
float basic;
};
struct  emp e;
fp=fopen(“emp.dat”, “wb”);
if (fp= = NULL)
{
puts(“Cannot open file”);
exit();
}
while(ch= =’y’)
{
printf(“\n Enter name, age and basic salary”);
scanf(“%s%d%f”, e.name, &e.age,&e.basic);
fwrite (&e, sizeof (e), 1,fp);
printf(“want to continue (y/n)”);
fflush(stdi^);
ch= getch();
}
```

```
fclose(fp);
}
```

The first argument is the address of the structure to be write to the disk. The second argument is the size of the structure in bytes. sizeof() operator gives the size of the variable in bytes. The third argument is the number of such structures that we want to write at one time. The last argument is the pointer to the file we want to write to. Let us consider an example of reading records from binary file.

```
#include "stdio.h"
main()
{
    FILE *fp;
    struct emp
    {
    char name [40];
int age;
float basic;
};
struct emp e;
    fp =fopen ("Emp. Dat", "rb");
if(fp= = NULL)
{
puts("Cannot open file");
exit();
}
while (fread(&e, sizeof (e),1,fp) = =1)
printf ("\n %s%d%f," e.name, e.age, e.basic);
fclose (fp);
}
```

The function fread() returns the number of records read. If end of file reaches, it returns a 0.

INTEXT QUESTIONS

6. Distinguish between printf and fprintf ?

7. What is append mode, and what letter is used to specify if?

8. How does write mode differ from append mode when an existing file is being written to ?

Rewind() & fseek()

The rewind() function places the pointer to the beginning of the file, irrespective of where it is present right now. The syntax is

rewind(fp);

Where fp is the file pointer.

Pointer movement is of utmost importance since fread() always reads that record where the pointer is currently placed. Similarly , fwrite() always writes the record where the pointer is currently placed.

The fseek() function move the pointer from one record to another.

The syntax is

fseek(fp,no-of-bytes, position);

Here, fp is the file pointer, no-of-bytes is the integer number that indicates how many bytes you want to move & position is the position of the pointer from where you want to move e.g. it may be current position, beginning of file position, & End of file position.

e.g. to move the pointer to the previous record from its current position, the function is

fseek(fp, -recsize, SEEK-CUR);

Here no-of-bytes is stored in variable recsize which itself is a record size, SEEK_CUR is a macro defined in “stdio.h”. Similarly SEEK_END, SEEK_SET can be used for end of file and beginning of file respectively

If we wish to know where the pointer is positioned right now, we can use the function ftell(). It returns this position as a long int which is an offset from the beginning of the file. The value returned by ftell()

can be used in subsequent calls to fseek(). The sample call to ftell() is show below:

```
p=ftell(fp);
```

DETECTING ERRORS

To detect any error that might have occurred during a read/write operation on a file the standard library function ferror() can be used. e.g.

```
        while(!feof(fp))
        {
           ch=fgetc(fp);
        if (ferror())
        {
        printf("Error in reading file");
        break;
        }
        - - - -
- - - -
```

If error occurs, ferror() returns a non-zero value & the if block gets executed.

To redirect the output of a program, from the screen to a file let us see an example:

```
#include "stdio.h"
main()
{
char ch;
while((ch=getc(stdin))!=EOF)
putc(ch,stdout);
}
```

On compiling this program we would get an executable file sample exe. Normally when we execute this file, the putc() function will cause what ever we type to be printed on screen, until you don't type ctrl+z. But through DOS this can be done using redirection.

c>sample.exe>sample1.txt.

Here, the output of sample.exe is redirecting to the file sample1.txt. So, '>' symbol is used for redirection (output). Similarly you can use '<' for input redirection e.g c>smple.exe< Newsample.txt

19.6 WHAT YOU HAVE LEARNT

In this lesson you have learnt about different file opening modes and before that commands to open and close a file. You are now very well know about file pointer. You can write or read to or from the file with the help of functions fread (and fwrite). You can also change the position of file pointer with the help of command fseek(). The function ftell() tells you the current position of file pointer.

You can also use symbol '>' and '<' for output redirection and input redirection respectively.

19.7 TERMINAL QUESTIONS

1. In which file the macro FILES are defined?
2. If a file is opened for reading, it is necessary that the file must exist. Is it true or false?
3. Write a program to count the number of words in a given text file.
4. While using the statement,

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

   What happens if my.c exists on the disk?
5. Write a program which will append or add the records in a student file, and also modifies the details of any student.
6. What role does the fseek function play, and how many arguments does it have?
7. If a C library does not contain a rewind function, how can it always be implemented?

19.8 KEY TO INTEXT QUESTIONS

1. Information that enters computer from the outside is called input, whereas information produced by the computer and sent to outside world is known as output.

2. The end of a string is always indicated by a null character, whereas the manner in which the end of file is indicated depends on the storage device and the computer.

3. EOF is a constant returned by many I/O functions to indicate that the end of an input file has been reached. Its value on most computers is –1.

4. It must be opened by the program.

5. The fopen function has two parameters. The first is a string specifying the file name. The second is a string that specifies the mode in which the file is to be opened. The function returns a FILE pointer associated with the opened file.

6. The print ϕ function can send output only to the standard output file, whereas fprintf can send its output to any opened output file.

7. Append mode is used to add characters to the end of a file that already exists. It is specified by using the mode string “a” as the second parameter to fopen.

8. When write mode is used, the former contents of the file are erased. With append mode, the contents of the file are returned and new information is added to the end of the file.