MODULE – 5

Structure, Union, and Enumerated Data Type: Introduction, structures and functions, Unions, unions inside structures, Enumerated data type.

Files: Introduction to files, using files in C, reading and writing data files, Detecting end of file

Textbook: Chapter 15.1 – 15.10, 16.1-16.5

CHAPTER 2 FILES

2.1 INTRODUCTION TO FILES

- A file is a collection of data stored on a secondary storage device like hard disk.
- Till now, we had been processing data that was entered through the computer's keyboard.
- But this task can become very tedious especially when there is a huge amount of data to be processed.
- A better solution, therefore, is to combine all the input data into a file and then design a C program to read this data from the file whenever required.

The console-oriented I/O operations pose two major problems:

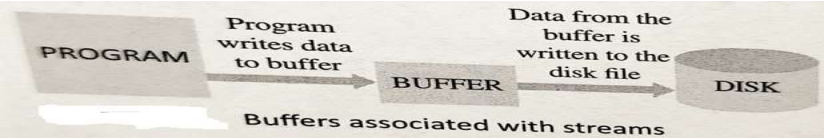
1. First, it becomes cumbersome and time-consuming to handle huge amount of data through terminals.
2. Second, when doing I/O using terminal, the entire data is lost when either the program is terminated or computer is turned off. Therefore, it becomes necessary to store data on a permanent storage device (e.g. hard disks) and read whenever required, without destroying the data.

- In order to use files, we have to learn file input and output operations, i.e., how data is read from or written to a file.
- Although file I/O operations are almost same as terminal I/O, the only difference is that when doing file I/O, the user must specify the name of the file from which data should be read/written.

2.1.1 Buffer Associated with files

A buffer is nothing but a block of memory that is used for temporary storage of data that has to be read from or written to a file.

The creation and operation the buffer is automatically handled by the operating system. However, C provides some functions for buffer manipulation. The data resides in the buffer until the buffer is flushed or written to a file.



2.1.2 Types of Files

In C, the types of files used can be broadly classified into two categories ASCII text files and binary files.

ASCII Text Files

The file that contains ASCII codes of data like digits, alphabets and symbols is called text file (or) ASCII file.

Because text files only process characters, they can only read or write data one character at a time. A line in a text file is not a C string, so it is not terminated by a null character.

The contents of a binary file are not human-readable. If we want the data stored in the file to be human-readable, then store the data in a text file. In a text file, each line of data ends with a newline character. Each file ends with a special character called the end-of-file (EOF) marker.

Binary Files

A binary file may contain any type of data, encoded in binary form for computer storage and processing purposes.

The file that contains data in the form of bytes (0's and 1's) is called as binary file. Generally, the binary files are compiled version of text files. Like text file, binary file also ends with an EOF marker.

2.2 USING FILES IN C

To use files in C, we must follow the steps given below:

- declare a file pointer variable
- open the file
- process the file (Reading from a file and Writing to a file)
- close the file

2.2.1 Declaring a File Pointer Variable

There can be a number of files on the disk. In order to access a particular file, we must specify the name of the file that has to be used. This is accomplished by using a file pointer variable that points to a structure FILE (defined in stdio.h).

The syntax for declaring a file pointer is FILE *file_pointer_name;

FILE *fp;

Then, fp is declared as a file pointer.

2.2.2 Opening a File

A file must be opened before any operation can be performed on it. A file must first be opened before data can be read from it or written to it. In order to open a file, the fopen() function is used. The prototype of fopen() can be given as

```
FILE *file_pointer_name = fopen ("file_name", "Mode");
```

pointer_name can be anything of our choice.

file_name is the name of the file, which we want to open. While opening a file, we need to specify the mode.

Example - FILE * fp;

fp = fopen("Student.txt", "r");

Using the above declaration, the file whose pathname is the string pointed to by file name is opened in the mode specified using the mode. If successful, fopen() returns a pointer-to-file and if it fails, it returns NULL.

File Name

In C, fopen() may contain the path information instead of specifying the filename. The path gives information about the location of the file on the disk. If a filename is specified without a path, it is assumed that the file is located in the current working directory. For example, if a file named student.txt is located on D drive in directory BCA, then the path of the file can be specified by writing, D:\BCA\student.txt

In C, a backslash character has a special meaning with respect to escape sequences when placed in a string. So in order to represent a backslash character in a C program, we must precede it with another backslash. Hence, the above path will be specified as given below in the C program.

D:\\BCA\\student.txt

File Mode

The second argument in fopen() is the mode. Mode conveys to C the type of processing that will be done with the file.

Sl.No	Mode	Description
1	r	Opens a text file in **reading** mode. Only reading possible. Does not create the file if it does not exist.
2	w	Only writing is possible. Create the file if it does not exist; otherwise, erase the old content of the file and open a blank file.
3	a	Only writing is possible. Create a file; if it does not exist, otherwise open the file and write from the end of the file. (Does not erase the old content).
4	r+	Reading and writing are possible. Create a file if it does not exist, overwriting existing data. Used for modifying content.
5	w+	Reading and writing are possible. Create a file if it does not exist. Erase old content.
6	a+	Reading and writing are possible. Create a file if it does not exist. Append content at the end of the file.

The above modes are used with text files only. If we want to work with binary files we use **rb, wb, ab, rb+, wb+ and ab+**.

Example Code -

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
        FILE *fp=fopen("C:\\Users\\sonim\\Documents\\zoom\\stu.txt","r");
        if(fp==NULL)
        {
        printf("\n The file could not be opened");
        exit(1);
        }
}
```

The fopen() function can fail to open the specified file if we attempt to open a non-existent file for reading.

2.2.3 Closing a File Using fclose()

To close an open file, the fclose() function is used which disconnects a file pointer from a file. After fclose() has disconnected the file pointer from the file, the pointer can be used to access a different file or the same file but in a different mode. The fclose() function not only closes the tile, but also flushes all the buffers that are maintained for that file. If we do not close a file after using it, the system closes it automatically when the program exits. However, since there is a limit on the number of files which can be opened simultaneously, we must close a file after it is used.

Syntax – int fclose(FILE *fp);

Example – fclose(fp);

fp is the file pointer which points to the file that has to be closed. The function returns an integer value which indicates fclose() was successful or not. A zero is returned if the function was successful and a non-zero is returned if an error occurred.

If a file's buffer has to be flushed without closing it then use fflush() function.

2.3 READING DATA FROM FILES

The reading from a file operation is performed using the following pre-defined file handling methods.

1. getc()
2. getw()
3. fscanf()
4. fgets()
5. fread()

1. **getc(*file_pointer) -** This function is used to read a character from specified file which is opened in reading mode. It reads from the current position of the cursor. After reading the character the cursor will be at next character.

```
#include<stdio.h>

int main()

{

  FILE *fp;
```

```
  char ch;
  fp = fopen("MySample.txt","r");
  ch = getc(fp);
  printf("ch = %c", ch);
  fclose(fp);
  return 0;
}
```

2. **getw(*file_pointer) -** This function is used to read an integer value form the specified file which is opened in reading mode. If the data in file is set of characters then it reads ASCII values of those characters.

```
#include<stdio.h>
int main()
{
  FILE *fp;
  int i,j;
  fp = fopen("MySample.txt","w");
  putw(65,fp);  // inserts A
  putw(97,fp);  // inserts a
  fclose(fp);
  fp = fopen("MySample.txt","r");
  i = getw(fp);   // reads 65 - ASCII value of A
  j = getw(fp);   // reads 97 - ASCII value of a
  printf("SUM of the integer values stored in file = %d", i+j);  // 65 + 97 = 162
  fclose(fp);
```

```
    return 0;
}
```

3. **fscanf(*file_pointer, typeSpecifier, &variableName) -** This function is used to read multiple datatype values from specified file which is opened in reading mode.

```
#include<stdio.h>
int main()
{
  char str1[10], str2[10], str3[10];
  int year;
  FILE * fp;
  fp = fopen ("file.txt", "w+");
  fputs("We are in 2016", fp);
rewind(fp);    // moves the cursor to begining of the file
fscanf(fp, "%s %s %s %d", str1, str2, str3, &year);
printf("Read String1 - %s\n", str1);
printf("Read String2 - %s\n", str2);
printf("Read String3 - %s\n", str3);
printf("Read Integer - %d", year);
fclose(fp);
return 0;
}
```

4. **fgets(variableName, numberOfCharacters, *file_pointer) -** This method is used for reading a set of characters from a file which is opened in reading mode starting from the

current cursor position. The fgets() function reading terminates with reading NULL character.

```
#include<stdio.h>
int main()
{
        FILE *fp;
        char str[20];
        fp = fopen ("file.txt", "r");
        fgets(str,6,fp);
        printf("str = %s", str);
        fclose(fp);
        return 0;
}
```

5. **fread(source, sizeofReadingElement, numberOfCharacters, FILE *pointer) -** This function is used to read specific number of sequence of characters from the specified file which is opened in reading mode.

```
#include<stdio.h>
#include<string.h>
int main()
{
  FILE *fp;
  char str[20];
  fp = fopen ("file.txt", "r");
  fread(str,sizeof(char),5,fp);
  printf("str = %s", str);
  fclose(fp);
  return 0;
```

}

2.4 WRITING DATA TO FILES

The writing into a file operation is performed using the following pre-defined file handling methods.

1. putc()
2. putw()
3. fprintf()
4. fputs()
5. fwrite()

1. **putc(char, *file_pointer) -** This function is used to write/insert a character to the specified file when the file is opened in writing mode.

```
#include<stdio.h>
int main(){
  FILE *fp;
  char ch;
  fp = fopen("MySample.txt","w");
  putc('A',fp);
  ch = 'B';
  putc(ch,fp);
  fclose(fp);
  return 0;
}
```

2. **putw(int, *file_pointer) -** This function is used to writes/inserts an integer value to the specified file when the file is opened in writing mode.

   Same Program under getw()

```
#include<stdio.h>
int main()
{
  FILE *fp;
  int i,j;
  fp = fopen("MySample.txt","w");
  putw(65,fp);  // inserts A
  putw(97,fp);  // inserts a
  fclose(fp);
  fp = fopen("MySample.txt","r");
  i = getw(fp);   // reads 65 - ASCII value of A
  j = getw(fp);   // reads 97 - ASCII value of a
  printf("SUM of the integer values stored in file = %d", i+j);  // 65 + 97 = 162
  fclose(fp);
  return 0;
}
```

3. **fprintf(*file_pointer, "text") -** This function is used to writes/inserts multiple lines of text with mixed data types (char, int, float, double) into specified file which is opened in writing mode.

```
#include<stdio.h>
int main()
{
FILE *fp;
char *text = "\nthis is example text";
```

```
int i = 10;
fp = fopen("MySample.txt","w");
fprintf(fp,"This is line1\nThis is line2\n%d", i);
fprintf(fp,text);
fclose(fp);
return 0;
}
```

4. **fputs("string", *file_pointer) -** TThis method is used to insert string data into specified file which is opened in writing mode.

```
#include<stdio.h>
int main()
{
  FILE *fp;
  fp = fopen("MySample.txt","w");
  fputs("Hi!\nHow are you?",fp);
  fclose(fp);
  return 0;
}
```

5. **fwrite("StringData", sizeof(char), numberOfCharacters, FILE *pointer) -** This function is used to insert specified number of characters into a file which is opened in writing mode.

```
#include<stdio.h>
```

```
int main()
{
  FILE *fp;
  char *text = "Welcome to C Language";
  fp = fopen("MySample.txt","w");
  fwrite(text,sizeof(char),5,fp);
  fclose(fp);
  return 0;
}
```

2.5 DETECTING THE END-OF-FILE

When reading or writing data to files, we often do not know exactly how long the file is. For example, while reading the file, we usually start reading from the beginning and proceed towards the end of the file. In C, there are two ways to detect the end of file.

1. EOF
2. feof

EOF

While reading the file, character by character, the programmer can compare the character that has been read with EOF, which is a symbolic constant defined in stdio.h.

Let's say we have "new.txt" file with the following content.

This is demo!

This is demo!

Example

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

```
  int c = getc(f);
  while (c != EOF) {
    putchar(c);
    c = getc(f);
  }
  fclose(f);
  return 0;
}
```

Output

This is demo!

This is demo!

In the above program, file is opened by using fopen(). When integer variable c is not equal to EOF, it will read the file.

feof()

We can use the standard library function feof() which is defined in stdio.h.

The function feof() takes the file pointer as an argument and returns zero (false) when the end of file has not been reached and a one (true) if the end of file has been reached.

Syntax of feof() - int feof(FILE *file_pointer)

Example – Let's say we have "new.txt" file with the following content.

This is demo!

This is demo!

```
#include <stdio.h>

#include<stdlib.h>

void main()
```

```
{
  FILE *fp = fopen("new.txt", "r");
  char str[100];
  if(fp==NULL)
  {
    printf("The file could not be opened");
   exit(1);
  }
  while(1)
  {
        fgets(str,79,fp);
        if(feof(fp))
              break;
        printf("%s",str);
  }
  fclose(fp);
}
```

Output

This is demo!

This is demo!

The function feof() is checking that pointer has reached to the end of file or not.