Chapter -11

Data Files

11.1 Introduction

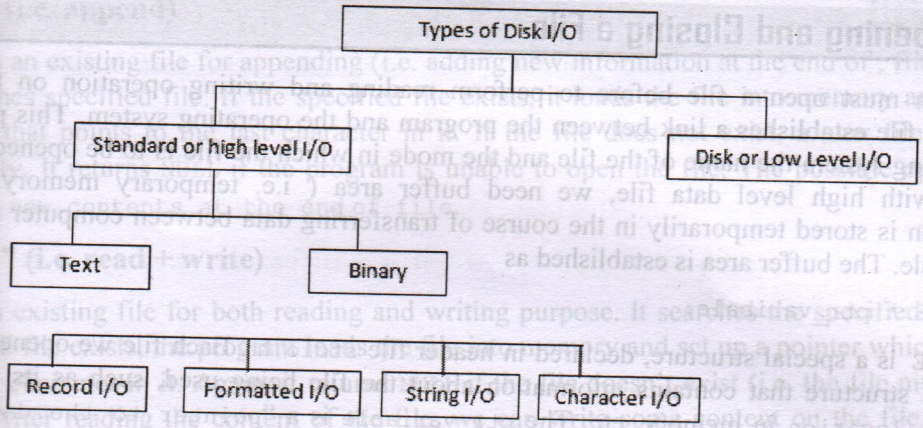
The input output functions like `printf()`, `scanf()`, `getchar()`, `putchar()`, `gets()`, `puts()` are known as console oriented I/O functions which always use keyboard for input device and computer screen or monitor for output device. While using these library functions, the entire data is lost when either the program is terminated or the computer is turned off. Again, it becomes cumbersome and time consuming to handle large volume of data through keyboard. It takes a lot of time to enter the entire data. If the user makes a mistake while entering the data, S/he has to start from the beginning again. If the same data is to be entered again at some later stage, again we have to enter the same data. These problems invite concept of data files. Data can be stored on the disks and read whenever necessary, without destroying it.

A file is a place on the disk where a group of related data is stored. The data file allows us to store information permanently and to access and alter that information whenever necessary. Programming language C has various library functions for creating and processing data files. Mainly, there are two types of data files, one is `stream oriented` or `standard` or `high level` and another is `system oriented` or `low level` data files. In high level data files, the available library functions do their own buffer management whereas the programmer should do it explicitly in the case of system oriented/low level files.

The standard data files are again subdivided into `text files` and `binary files`. The text files consist of consecutive characters and these characters can be interpreted as individual data item. The binary files organize data into blocks containing contiguous bytes of information.

11.2 Disk File I/O

For each, binary and text files, there are a number of formatted and unformatted library functions in C.



Types of I/O

1. Standard I/O (often called high or stream I/O)
2. System-level I/O (often called low-level I/O)

Standard I/O is the most common way of performing I/O in C programs. System I/O provides fewer ways to handle data than standard I/O, and it can be considered a more primitive system and it employs the techniques much like those used by operating system. The character, string and record I/O are often recognized as unformatted I/O.

Standard I/O provides four different ways of reading and writing data whereas system I/O, in contrast, uses record I/O as only one type of reading/writing data.

When data is read or written one character at a time, it is called character I/O. The functions for character I/O are `fgetc()` & `fputc()` which are analogous to `getchar()` and `putchar()` function in console I/O.

When data is read or written as strings using functions `fgets()` & `fputs()`, it is called string I/O. This is analogous to functions like `gets()` & `puts()` to get string from keyboard and screen output.

When data is read or written into disk in a format analogous to that generated by `scanf()` & `printf()` functions as a collection of values that may be mixed types. It includes the functions like `fscanf()` and `fprintf()`. These are called formatted I/O.

Record I/O is the most popular type of I/O generally used to write block of data like array and structure or table of records. The record I/O includes the functions as `fread()` and `fwrite()`.

The types of I/Os are summarized as:

	Record I/O	Formatted I/O	String I/O	Character I/O
Standard I/O	`fread()` `fwrite()`	`fscanf()` `fprintf()`	`fgets()` `fputs()`	`fgetc()` `putc()`
System I/O	`fread()` `fwrite()`			

11.3 Opening and Closing a File

A program must open a file before to perform reading and writing operation on the file. Opening a file establishes a link between the program and the operating system. This provides the operating system the name of the file and the mode in which the file is to be opened. While working with high level data file, we need buffer area (i.e. temporary memory) where information is stored temporarily in the course of transferring data between computer memory and data file. The buffer area is established as

```
FILE * ptr_varibale;
```

Here, `FILE` is a special structure, declared in header file `stdio.h` . Each file we opened has its own `FILE` structure that contains information about the file being used, such as its `current size, its location` in memory, etc. The `ptr_variable` is a "`pointer to the data type FILE`" that stores the beginning address of the buffer area allocated after a file has been opened. This pointer contains all the information about the file and it is used as a communication link between the system and the program. A data file is opened using syntax:

```
ptr_variable=fopen(file_name, file_mode);
```

This associates a file name with the buffer area and specifies how the data file will be utilized (File opening mode). The function `fopen()` returns a pointer to the beginning of the buffer area associated with the file (i.e. the first address of the allocated buffer area is stored in pointer variable `pointer_variable`) . A `NULL` value is returned if the file can not be opened due to some reasons. After opening a file, we can process data files as requirement. Finally, data file must be closed. The closing a file ensures that all outstanding information associated with the file is flushed out from the buffers and all links to the file are broken. It also prevents any accidental misuse of the file. The file is closed using other library function `fclose()` as below

```
fclose(ptr_variable);
```

11.4 File Opening Modes

The file opening mode specifies the way in which a file should be opened (i.e. for reading, writing or both, appending at the end of the file, overwriting the file etc). In other words, it specifies the purpose of opening a file. There are mainly six modes.

a) "r" (i.e. read)

This opens an existing file for reading purpose only. It searches specified file. If the specified file exists, it loads the file into memory and sets up a pointer which points to the first character within it. If the specified file does not exist in specified directory, it returns `NULL`. The possible operation for the "`r`" mode is: `reading from the file.`

b) "w" (i.e. write)

The '`w`' mode opens a file for writing purpose. It searches specified file. If the specified file already exist, the content within the file are overwritten (i.e. content is deleted first and then written). If the file does not exist, a new file is created implicitly. It returns `NULL`, if it is unable to open the file in write mode due to problems in writeing operation for the '`w`' mode is: `writing to the file.`

c) "a" (i.e. append)

It opens an existing file for appending (i.e. adding new information at the end of file) purpose. It searches specified file. If the specified file exists, it loads the file into memory and set up a pointer that points to the last character in it. If the file does not exist, a new file is created implicitly. It returns `NULL` if the program is unable to open the file. The possible operation is: `adding new contents at the end of file.`

d) "r+" (i.e. read + write)

It opens existing file for both reading and writing purpose. It searches the specified file. If the specified file exists, the program loads the file into memory and set up a pointer which points to the first character in it. Again, it returns `NULL` if the file doesn't exist (i.e. the file must already exist). After reading the content of the file, we can write some content on the file. Thus, the possible operations are: `reading existing contents, writing new contents, modifying existing contents of the file.`

e) "w+" (i.e. write + read)

It opens file for reading and writing purpose. If the specified file exists, its contents are destroyed. If the file doesn't exist, the new file is created. It returns the `NULL` character if it is unable to open the file. Although the both modes `'w+'` and `'r+'` are for reading and writing purpose, they are different (i.e. If the file doesn't exist, the file is created implicitly in `'w+'` mode but not in `'r+'` mode). The possible operations are: `writing new contents, reading them back and modifying existing contents of the file.`

f) "a+" (i.e. append+ read):

It opens an existing file for both reading and appending purpose. A new file is created if the specified file doesn't exist. The possible operations are: `reading existing contents, appending new contents to end of file but it can not modify existing content.`

11.5 Library Functions for Reading/Writing from/to a File

11.5.1 String Input/Output

Using string I/O functions, data can be read from a file or written to a file in the form of array of characters.

i. **fgets():** It is used to read string from file.

Syntax: `fgets(string_variable,.int_value, file_ptr_variable);`
Here `int_value` denotes the number of character in a string. The function reads a string from a file representing `file_ptr_variable` and stores in a variable `string_variable`.

ii. **fputs():** It is used to write a string to a file.

Syntax: `fputs(string, file_ptr_variable);`

Here, the `string_variable` is written to a file representing `file_ptr_variable`.

Example 11.1

Write a program to create a file named "test.txt" and write text "Welcome to my College" to this file.

```
#include<stdio.h>
int main()
    {
    FILE *fptr;
    fptr=fopen("c:\\test.txt","w");
    if(fptr==NULL)
        {
        printf("\nFile can not be created");
        exit(1);
        }
    else
        {
        printf("File has been successfully created");
        }
    fputs("Welcome to my College",fptr);
    fclose(fptr);
    return 0;
    }
```

Explanation:

Here, nothing is displayed on the screen. The file "test.txt" is created in c:\ drive and the text "Welcome to my College" is written to the file. [See in c:\ drive, you will find test.txt which has been created by this program]

Example 11.2

Write a program to open the file "test.txt" created in Example 11.1, read its content and display to the screen.

```
#include<stdio.h>
int main()
    {
    FILE *fptr;
    char s[100];
    fptr=fopen("c:\\test.txt","r");
    if(fptr==NULL)
        {
        printf("\nFile can not be opened");
        exit(1);
        }
    fgets(s,100,fptr);
    printf("\nThe text from file is:\t%s",s);
    fclose(fptr);
    getch();
    return 0;
    }
```

Output

```
The text from file is:  Welcome to my College
```

11.5.2 Character Input/Output

Using character I/O functions, data can be read from a file or written to a file one character at a time.

i. **fgetc() :** It is used to read a character from a file.

Syntax: `char_variable=fgetc(file_ptr_variable);`

Here, a character is read from a file representing `file_ptr_variable` and stores in `char_variable`.

ii. **fputc():** It is used to write a character to a file.

Syntax: `fputc(character or char_variable, file_ptr_variable);`
Here, a character is written to a file represented by `file_ptr_variable`.

Example 11.3

Write a program to create file and write some text to it writing one character at a time using `fputc()` function. The program should write until user hits enter key. Read filename from user.

```
#include<stdio.h>
int main()
   {
    FILE *fptr;
    char c;
    char fileName[20];
    printf("Enter the name of file:\t");
    scanf("%s",&fileName);
    fptr=fopen(fileName,"w");
    if(fptr==NULL)
      {
       printf("\nFile can not be created");
       exit(0);
      }
    printf("\nEnter Text. Hit enter key to stop writing\n");
    fflush(stdin);
    while((c=getchar())!='\n')
         {
         fputc(c,fptr);
         }
     fclose(fptr);
     return 0;
   }
```

Output

```
Enter the name of file: c:\student.dat
Enter Text. Hit enter key to stop writing
This is example which reads some text from keyboard character by
character and stores in a file student.dat in c:\drive.
```

Example 11.4

Write a program to open the file created in example 11.3, read its content one character at a time and display it to the screen

```
#include<stdio.h>
int main()
    {
      FILE *fptr;
      char c;
      char fileName[20];
      printf("Enter the name of file:\t");
      scanf("%s",&fileName);
      fptr=fopen(fileName,"r");
      if(fptr==NULL)
        {
         printf("\nFile does not exist!!!!");
         exit(0);
        }
      printf("\nThe Content from File is:\n");
      while((c=fgetc(fptr))!=EOF)
            {
            putchar(c);
            }
       fclose(fptr);
       getch();
       return 0;
      }
```

Output

```
Enter the name of file:  c:\student.dat
The Content from File is:
This is example which reads some text from keyboard character by
charactores in a file student.dat in c:\drive.
```

Example 11.5

Write a program to append some text to a file. Your program should take name of the file from user.

```
#include<stdio.h>
int main()
    {
     FILE *f;
     char c,fileName[20];
     printf("\nEnter file name in which You want to append text:\t");
     gets(fileName);
     f=fopen(fileName,"a");
     if(f==NULL)
          {
           printf("\nFile can not be created or opened");
          }
     printf("Enter Text to be appended:\n");
```

```
    while((c=getchar())!='\n')
        {
         fputc(c,f);
        }
    fclose(f);
    return 0;
   }
```

Output

```
Enter file name in which You want to append text:  c:\student.dat
Enter Text to be appended:
This is appended text. This program does not delete previous data of this
file. It adds this text at the end of the file.
```

Here, the above program adds the given text at the end of file `c:\student.dat` file. See the content of file `student.dat` in `c:\` drive.

Example 11.6

Create a program which asks file name from user and create it to write some lines of text. Your program should write the text until the user presses an enter key at the beginning of line. Read the content of file and display to the screen of computer.

```
#include<stdio.h>
int main()
   {
    FILE *fptr;
    char s[200],fileName[20],c;
    printf("\nEnter name of file:\t");
    gets(fileName);
    fptr=fopen(fileName,"w");
    if(fptr==NULL)
       {
        printf("\nFile can not be created");
        exit(0);
       }
     printf("\nStart writing. . .\n");
     while(strlen(gets(s))!=0)
        {
        fputs(s,fptr);
        fputs("\n",fptr);
        }
      rewind(fptr);
      printf("Reading from file.............");
      printf("\nThe Content From File is:\n");
      while((c=fgetc(fptr))!=EOF)
        {
        putchar(c);
        }
     fclose(fptr);
     getch();
     return 0;
   }
```

Output

```
Enter name of file:    c:\example.txt
Start writing. . .
Hello! this is first line
Hello!!! this is second line
Reading from file.............
The Content From File is:
Hello! this is first line
Hello!!! this is second line
```

Example 11.7

Write a program that opens a file and copies all its content to another file. Your program should read name of source and destination file from user.

```
#include<stdio.h>
int main()
      {
       FILE *fptrSource,*fptrDest;
       char ch,sourceFileName[20],destFileName[20];
       printf("\nEnter Source File name:\t");
       gets(sourceFileName);
       printf("\nEnter Destination File:\n");
       gets(destFileName);
       fptrSource=fopen(sourceFileName,"r");
       if(fptrSource==NULL)
         {
         printf("\nSource File can not be opened");
         exit(1);
         }
       fptrDest=fopen(destFileName,"w");
       if(fptrDest==NULL)
         {
          printf("\nDestination File can not be created\n");
          exit(1);
         }
       while((ch=fgetc(fptrSource))!=EOF)
         {
         fputc(ch,fptrDest);
         }
        printf("Successfully copied!!!!!!!!!");
        fclose(fptrSource);
        fclose(fptrDest);
        getch();
        return 0;
   }
```

Output

```
Enter Source File name:  c:\example.txt
Enter Destination File:   c:\example.dat
Successfully copied!!!!!!!!!
```

Here, the content of `example.txt` file in `c:\` is copied into `example1.dat`. See effect of this program in `c:\` drive. For this, `example.txt` file must exist in `c:\` drive.

11.5.3 Formatted Input/Output

These functions are used to read numbers, characters or string from a file or write them to a file in format as our requirement.

i. **fprintf():** It is formatted output function which is used to write integer, float, char or string data to a file.

Syntax: `fprintf(file_ptr_variable, "control_string", list_variables);`

ii. **fscanf():** It is formatted input function which is used to read integer, float, char or string value from a file.

Syntax: `fscanf(file_ptr_variable, "control_string", &list_variables);`

Example 11.8

Write a program to create a file named `student.txt` in `D:\` drive and write `name`, `roll`, `address` and `marks` of a student to the file.

```
#include<stdio.h>
int main()
{
FILE *f;
char name[20],address[20];
int roll;
float marks;
f=fopen("d:\student.txt","w");
if(f==NULL)
    {
    printf("\nFile can not be created!!!");
    exit(1);
    }
printf("\nEnter name of student:\t");
gets(name);
printf("\nEnter roll number:\t");
scanf("%d",&roll);
fflush(stdin);
printf("\nEnter address of student:\t");
gets(address);
printf("\nEnter marks of the student:\t");
scanf("%f",&marks);
printf("\nThe data is being written in file.....");
fprintf(f,"Name=%s\nRoll=%d\nAddress=%s\nMarks=%.2f",name,roll,address,marks);
fclose(f);
getch();
return 0;
}
```

Output

```
Enter name of student:  Ram Pd Sharam
Enter roll number:      1001
Enter address of student:       Kathmandu
Enter marks of the student:     89.5
The data is being written in file.....
```

Example 11.9

Write a program to write even numbers between 10 and 50 to a file named evenNumbers.txt in D:\ drive.

```
#include<stdio.h>
int main()
 {
   FILE *fp;
   int i;
   fp=fopen("d:\\evenNums.txt","w");
   if(fp==NULL)
      {
        printf("\nFile can not be created");
        exit(1);
      }
    else
       printf("File has been successfully created");
    fprintf(fp,"The Even numbers between 10 and 50 are\n");
    for(i=10;i<=50;i++)
      {
       if(i%2==0)
          fprintf(fp,"%d\n",i);
      }
    fclose(fp);
    getch();
    return 0;
  }
```

Example 11.10

Write a program to create a data file and write the natural numbers from 1 to 20 to the file and then read the numbers from the file to display the twice of stored numbers.

```
#include<stdio.h>
int main()
   {
    FILE *fptr;
    int i,numFromFile;
    fptr=fopen("test.txt","w+");
    if(fptr==NULL)
       {
        printf("\nFile can not be created");
```

```
        exit(1);
        }
    for(i=1;i<=20;i++)
        {
        fprintf(fptr,"%d\n",i);
        }
    rewind(fptr);
    printf("\nThe twice of natural numbers stored in file are\n");
    for(i=1;i<=20;i++)
        {
        fscanf(fptr,"%d",&numFromFile);
        numFromFile=2*numFromFile;
        printf("%d\t",numFromFile);
        }
    fclose(fptr);
    getch();
    return 0;
    }
```

Output

```
The twice of natural numbers stored in file are
2       4       6       8       10      12      14      16      18
20 22   24      26      28      30      32      34      36      38
40
```

Example 11.11

Write a program to open a file named `inventory.txt` and store the following data to the file.

Product Name	Quantity	Rate
AAA	3	50
BBB	2	100
CCC	4	40

```
#include<stdio.h>
int main()
    {
    FILE *fptr;
    char items[10];
    int qty,rate,i;
    fptr=fopen("c:\\inventory.txt","w");
    if(fptr==NULL)
        {
        printf("\nFile can not be created");
        exit(1);
        }
    for(i=0;i<3;i++)
        {
        printf("Enter product name, qty and rate:\t");
        scanf("%s",items);
```

```
        scanf("%d%d",&qty,&rate);
        fprintf(fptr,"%s\t%d\t%d\n",items,qty,rate);
      }
    fclose(fptr);
    getch();
    return 0;
   }
```

Output

```
Enter product name, qty and rate:    AAA  3  50
Enter product name, qty and rate:    BBB  2  100
Enter product name, qty and rate:    CCC  4  40
```

Example 11.12

Write a program to open a file named `inventory.txt` created in above program and read data from the file and display inventory table with the total amount [i.e. `rate*qty`] of each item.

```
#include<stdio.h>
#include<stdlib.h>
 int main()
  {
    FILE *fptr;
    char items[10];
    int qty,rate,i;
    fptr=fopen("c:\\inventory.txt","r");
    if(fptr==NULL)
      {
       printf("\nFile can not be opened");
       exit(0);
      }
    printf("Item\tQty\tRate\tTotal Amount\n");
    printf("---------------------------------\n");
    for(i=0;i<3;i++)
      {
      fscanf(fptr,"%s",items);
      fscanf(fptr,"%d%d",&qty,&rate);
      printf("%s\t%d\t%d\t%d\n",items,qty,rate,qty*rate);
      }
    fclose(fptr);
    getch();
    return 0;
     }
```

Output

```
Product Name    Quantity      Rate    Total Amount
-------------------------------------------------------------
AAA             3             50          150
BBB             2             100         200
CCC             4             40          160
```

11.6 End Of File (EOF)

The constant EOF represents an integer and it determines whether the file associated with a file pointer has reached end of file or not. This integer is sent to the program by the operating system and is defined in a header file stdio.h. While creating a file, the operating system transmits the EOF signal when it finds the last character to the file. In text mode, a special character, 1A hex (decimal 26) is inserted after the last character in the file. This character is used to indicate EOF. If this character is encountered at any point in the file, the read function will return the EOF signal to the program. An attempt to read content after EOF might either cause the program to terminate with an error or result in an infinite loop situation. Thus, the last point of file is detected using EOF while reading data from file. Without this type of mark or signal, we can't detect the end character at the file such that it is difficult to find up to what time the character is to be read from the file in reading process.

Example 11.13

Write a program to read the content of file student.txt stored in C:\ drive. Use EOF to detect the end of file.

```
#include<stdio.h>
int main()
{
FILE *fptr;
char c;
fptr=fopen("c:\\student.dat","r");
if(fptr==NULL)
     {
    printf("\nFile does not exist!!!!!");
     }
while((c=fgetc(fptr))!=EOF)
     {
      printf("%c",c);
     }
fclose(fptr);
return 0;
}
```

11.7 Binary Data Files

The binary files organize data into blocks containing contiguous bytes of information. A binary file is a file of any length that holds bytes with values in the range 0 to 0xff. (0 to 255). The binary files are recognized as a stream of bytes and programming languages tend to work with streams rather than files. In binary file, the opening mode of text mode file is appended by a character 'b' i.e.

i. "r" is replace by "rb"

ii. "w" is replace by "wb"

iii. "a" is replace by "ab"

iv. "r+" is replace by "r+b"

v. "w+" is replace by "w+b"

vi. "a+" is replaced by "a+b"

Thus, the statement `fptr=fopen("student.txt","wb");` opens file `"student.txt"` for writing in binary mode. The default mode is text mode. Thus when a charcter "r" is written as mode of opening, this is assumed that the file is opened in text mode. We can write "rt" in place of "r" and so on in the case of text file. The syntax for all above mode in the case of binary mode is similar to the previous one (i.e. text mode).

Example 11.14

Write a C program to write some text "Welcome to my College" to a data file in binary mode. Read its content and display it.

```
#include<stdio.h>
int main()
  {
  FILE *fptr;
  char c;
  fptr=fopen("test.txt","w+b");
  if(fptr==NULL)
    {
     printf("\nFile can not be created");
    }
  fputs("Welcome to my College",fptr);
  rewind(fptr);
  printf("\The content from file:\n");
  while((c=fgetc(fptr))!=EOF)
    {
     printf("%c",c);
    }
  fclose(fptr);
  return 0;
}
```

Output

```
The content from file:
Welcome to my College
```

11.8 Differences between Binary Mode and Text Mode

There are mainly three differences between binary and text mode. They are as follows.

i. In text mode, a special character EOF whose ASCII value is 1A hex (26 in decimal) is inserted after the last character in the file to mark the end of file. But, there is no such special character present in the binary mode. The binary mode files keep track of the end of file from the number of characters presented in the directory entry of the file. In this mode, the numbers are stored in binary format. Therefore, a number stored in memory may happen to be 1A hex. But, in text mode it represents the end of file.

ii. In text mode, the numbers are stored as string of characters whereas in binary format they are stored the same way as they are stored in computer's main memory. For example, the number 1234 occupies two bytes primary memory but it occupies 4 bytes (one byte per character) of disk memory in the file of text mode. But, in binary mode, the number 1234 occupies only 2 bytes (i.e. same as it occupies in the primary memory). Hence, if a large amount of numerical data is to be stored in a disk file, text mode will be inefficient. The solution is to open the file in binary mode and use the functions fread() and fwrite() instead of fprintf(), fscanf().

iii. In C language, the end of a line is signaled by a single character, the new line character '\n' with ASCII value 10 in decimal. Whereas in DOS, the end of line is marked by two characters, the carriage return (CR) with ASCII value 13 in decimal and the line feed (LF) with the ASCII value 10 in decimal, which is same as of C's new line character. In text mode, a new line character is automatically converted into the carriage return-linefeed combination before being written to the disk or file. Likewise, the carriage return-line feed combination on the disk is converted into a new line when the file is read by a C program. However, these conversions don't take place in the case of binary mode. Here, the new line character '\n' is written as only line feed (LF) character. Similarly, while reading the combination of CR/LF will be treated as two separate characters as 'r' and 'n' characters.

11.9 Record Input/Output

It is clear that character I/O and string I/O permit reading and writing of character data only, whereas the formatted I/O permits reading and writing of characters data as well as numeric data. However, the numbers are stored as sequence of characters, not the way they are stored in memory. As a result, they take a lot of disk space. In addition to that, these methods provide no direct way to read and write complex data types such as array and structures. Though, the arrays and structures can be handled by reading and writing each array element one at a time, but this approach is very inefficient.

The solution to these problems are records I/O, also known as block I/O. Record I/O writes numbers to files in binary format so that integers are stored in 2 bytes, long integers are stored in 4 bytes and single precision floating point numbers are stored in 4 bytes- the same format used to store numeric data in primary memory. It permits reading or writing multiple data values in the form of arrays, structures and array of structure once. Thus, the arrays, structures, array of structures, etc can be read from a file or written to a file as a single unit using record I/O.

The functions `fwrite()` is used for record output while `fread()` is used for record input. The syntax for these are as below.

```
fwrite(&ptr, size_of_array_or_structure, number_of_structure_or_array, fptr);
fread(&ptr, size_of_array_or_structure, number_of_structure_or_array, fptr);
```

Where

i. `ptr` is the address of an array or a structure to be written

ii. `size_of_array_or_structure` is an integer value that shows the size of structure or array which is being read or written.

iii. `number_of_structure_or_array` is an integer value that indicates number of arrays or structures to be written to file or read from file.

iv. `fptr` is a file pointer of a file opened in binary mode.

Example 11.15

Write a program that opens an existing file `nums.txt` , read an array stored in it and display to the screen.

```
#include<stdio.h>
int main()
 {
  FILE *f;
  int nums[10],i;
  f=fopen("c:\\nums.txt","rb");
  if(f==NULL)
     {
      printf("\nFile can not opened; ");
     }
  printf("\nReading array from file ..........\n");
  fread(&nums,sizeof(nums),1,f);
  for(i=0;i<10;i++)
     printf("%d\t",nums[i]);
  fclose(f);
  getch();
  return 0;
 }
```

Output

```
Reading array from file ..........
12      34      56      78      54      22      11      66      7
45
```

Example 11.16

Create a structure named `employee` having members: `empName`, `age` and `salary`. Use this structure to read the `name`, `age` and `salary` of employee and write entered information to a file `employee.dat` in `D:\` drive.

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

```
{
struct employee
   {
   char empName[20];
   int age;
   float salary;
   };
struct employee emp;
FILE *fptr;
fptr=fopen("d:\\employee.dat","wb");
if(fptr==NULL)
      {
      printf("File can not be created!!!!");
      exit(1);
      }
printf("Employee Name:\t");
scanf("%s",emp.empName);
printf("Employee age:\t");
scanf("%d",&emp.age);
printf("Salary of the Employee:\t");
scanf("%f",&emp.salary);
printf("\nWriting this information to a file.....\n");
fwrite(&emp,sizeof(emp),1,fptr);
getch();
return 0;
}
```

Output

```
Employee Name:  Ram
Employee age:   34
Salary of the Employee: 1200.50
Writing this information to a file.....
```

Alternative method

```
#include<stdio.h>
int main()
{
struct employee
   {
   char empName[20];
   int age;
   float salary;
   };
struct employee emp;
FILE *fptr;
fptr=fopen("d:\\employee.dat","wb");
if(fptr==NULL)
   {
    printf("File can not be created!!!!");
    exit(1);
   }
```

```
    printf("Employee Name:\t");
    scanf("%s",emp.empName);
    printf("Employee age:\t");
    scanf("%d",&emp.age);
    printf("Salary of the Employee:\t");
    scanf("%f",&emp.salary);
    printf("\nWriting this information to a file.....\n");

fprintf(fptr,"EmpName=%s\tAge=%d\tSalary=%f",emp.empName,emp.age,emp.salary);
  getch();
  return 0;
 }
```

But this method is not good as explained above.

Example 11.17

Create a structure named `student` that has `name`, `roll` and `marks` as members. Assume appropriate types and size of members. Use this structure to read and display records of 3 students. Write array of structure to a file and then read its content to display to the screen.

```
#include<stdio.h>
int main()
  {
  struct student
       {
        char name[30];
        int roll;
        float marks;
       };
   struct student s[3],st[3];
   int i;
   float tempForMarks;
   FILE *fptr;
   fptr=fopen("d:\student.txt","w+b");
   if(fptr==NULL)
        {
         printf("File can not be created!!!!!!!");
         exit(1);
         }
    for(i=0;i<3;i++)
       {
        printf("\n\nEnter Information of Student No %d\n",i+1);
        printf("Name:\t");
        scanf("%s",s[i].name);
        printf("\nRoll:\t");
        scanf("%d",&s[i].roll);
        printf("\nMarks:\t");
        scanf("%f",&tempForMarks);
        s[i].marks=tempForMarks;
        }
```

```
printf("\n\nWriting Information to file.....\n");
fwrite(&s,sizeof(s),3,fptr);
rewind(fptr);
printf("\nReading same content from file.....\n");
fread(&st,sizeof(st),3,fptr);
printf("Student Name\tRoll\tMarks");
printf("\n-------------------------------------------------\n");
for(i=0;i<3;i++)
    printf("%s\t\t%d\t%.2f\n",st[i].name,st[i].roll,st[i].marks);
fclose(fptr);
getch();
return 0;
}
```

Output

```
Enter Information of Student No 1
Name:   Saroj
Roll:   100
Marks:  12
Enter Information of Student No 2
Name:   Jagdish
Roll:   101
Marks:  78
Enter Information of Student No 3
Name:   Harish
Roll:   102
Marks:  90
Writing Information to file.....
Reading same content from file.....
Student Name            Roll            Marks
-------------------------------------------------
Saroj                   100             12.00
Jagdish                 101             78.00
Harish                  102             90.00
```

Example 11.18

A book record consists of its *title*, *author*, *pages* and *price*. Write a program to perform following operations

a. **Read the records of 13 books**
b. **Create at least one structure pointer to display the records of 13 books**
c. **Store records of all 13 books in the file "booklist.dat"**
d. **Read only the information of 9 books from "booklist.dat" skipping 2 books from first and 2 books from last and display in terminal**

```
#include<stdio.h>
#define N 13
struct Book
{
char title[50],author[50];
int pages;
float price;
```

```
};
int main()
{
 struct Book b[N],*bptr;
 int i;
 FILE *fp;
 float temp;
 for(i=0;i<N;i++)
  {
   printf("\nRead record #%d",i+1);
   fflush(stdin);
   printf("\nTitle=?") ;
   gets(b[i].title);
   fflush(stdin);
   printf("\nAuthor Name=?");
   gets(b[i].author);
   printf("\nNo of pages=?");
   scanf("%d",&b[i].pages);
   printf("Price=?");
   scanf("%f",&temp);
   b[i].price=temp;
   }
bptr=&b[0];
printf("\nRecords of book using structure pointer");
printf("\nTitle \tAuthor\tNo pages\tPrice");
for(i=0;i<N;i++)
   printf("\n%s\t%s\t%d\t%f",(bptr+i)->title, (bptr+i)->author,
        (bptr+i)->pages,(bptr+i)->price);

printf("\nNow the records are being written in file.....");
fp=fopen("booklist.dat","w");
for(i=0;i<N;i++)
  fwrite(&b[i],sizeof(b[i]),1,fp);
printf("\nBook records skipping first 2 and last 2 records");
printf("\nTitle \tAuthor\tNo pages\tPrice");
for(i=2;i<N-2;i++)
  printf("\n%s\t%s\t%d\t%f",(bptr+i)->title,(bptr+i)->author,
    (bptr+i)->pages,(bptr+i)->price);
getch();
return 0;
}
```

11.10 Direct/Random Access

The reading and writing operation in all previous programs were sequential. While reading data from a file, the data items are read from the beginning of the file in sequence until end of the file. Similarly, while writing data to a file, data items are placed one after the other in a sequence. But, sometimes we have to access a particular data item placed in any location without starting from the beginning. Such type of access to a data item is called direct or random access.

For random access in a file, the knowledge of file pointer is necessary. A `file pointer` is a pointer to a particular byte in a file. While opening a file in write mode, the file pointer is at the beginning of file. Every time, when we write to a file, the file pointer moves to the end of the data items written so that writing process can continue from that point. While opening a file in read mode, the file pointer is at the beginning of file. After reading a data item, the file pointer moves to the beginning of the next data item, which can subsequently be read. However, if the file is opened in append mode, then the file pointer will be positioned at the end of the existing file, so that new data items can be written from there onwards. Therefore, we can read any data item from a file or write to a file randomly if we would be successful to move this file pointer as our requirement. The frequently used functions for random access are :

i. fseek() function

It sets the file pointer associated with a stream/file handle to a new position. This function is used to move the file pointer to different positions.

Syntax: `fseek(fptr, offset, mode);`

where `fptr` is a file pointer, `offset` is an integer that specifies the number of bytes by which file pointer is moved and `mode` specifies from which position the offset is measured. The value of `mode` may be `0`, `1` or `2`. Here, `0` represents beginning of the file (`SEEK_SET`), `1` represents current position of the file (`SEEK_CUR`) and `2` represents end of the file (`SEEK_END`).

Examples:

a) `fseek(fptr, 0, SEEK_SET)` => moves file pointer at beginning of the file

b) `fseek(fptr, 0, SEEK_END)` => moves the file pointer at end of the file

c) `fseek(fptr, 10, SEEK_SET)` => moves the file pointer at 10 bytes right from the beginning of file

d) `fseek(fptr, -2, SEEK_CUR)` =>moves the file pointer at 2 bytes left from current position.

ii. rewind() function

One way of positioning the file pointer to the beginning of the file is to close the file and then re-open it again. The same task can be accomplished using `rewind()` function. This function positions the file pointer in the beginning of the file. The use of function `rewind()` is equivalent of using `fseek(fptr, 0, SEEK_SET)`.

iii. ftell() function

This function determines the current location of the file pointer within the file. It returns integer value.

Syntax: `ftell(fptr);`

where `fptr` is a file pointer for the currently opened file.

Example 11.19

Create a structure named `Employee` having `empName`, `age` and `salary` as its members. Read `name`, `age` and `salary` of a number of employees and write these data to a file named `employee.txt`. Read employee information again and again until user wants to add more employees. Finally, write a program to search information of a particular employee from the file.

```
#include<stdio.h>
#include<string.h>
int main()
 {
 struct employee
    {
     char empName[20];
     int age;
     float salary;
    };
 struct employee emp;
 FILE *fptr;
 char yes_no, name[15];
 int dataFound=0;
 fptr=fopen("c:\\employee.txt","w+b");
 if(fptr==NULL)
    {
     printf("File can not be created!!!!");
     exit(1);
    }
 do
  {
   printf("Employee Name:\t");
   scanf(" %s",&emp.empName);
   printf("Employee age:\t");
   scanf("%d",&emp.age);
   printf("Salary of the Employee:\t");
   scanf("%f",&emp.salary);
   fwrite(&emp,sizeof(emp),1,fptr);
   printf("Do you want to add another employee?? Press y or Y:\t");
   fflush(stdin);
   yes_no=getchar();
   }while(yes_no=='Y' || yes_no=='y');
printf("Enter the name of employee which is to be searched!!!!\t");
fflush(stdin);
gets(name);
rewind(fptr);
while(fread(&emp,sizeof(emp),1,fptr)==1)
        {
         if(strcmp(emp.empName,name)==0)
            {
             dataFound=1;
             printf("Name\tAge\tSalary\n");
```

```
            printf("-------------------------\n");
            printf("%s\t%d\t%.2f",emp.empName,emp.age,emp.salary);
            }
        }
if(dataFound==0)
    printf("\nMatching data not found!!!!!!");
getch();
return 0;
```

Output

```
Employee Name:    Ram
Employee age:   24
Salary of the Employee: 12000
Do you want to add another employee?? Press y or Y:   y
Employee Name:  Bina
Employee age:   23
Salary of the Employee: 23080
Do you want to add another employee?? Press y or Y:   y
Employee Name:  Harish
Employee age:   26
Salary of the Employee: 20000
Do you want to add another employee?? Press y or Y:   n
Enter the name of employee which is to be searched!!!!  Bina
Name    Age     Salary
-------------------------
Bina    23      23080.00
```

Example 11.20

Write a program to open a file `employee.txt` created in above program and edit/modify the detail of a particular employee.

```
#include<stdio.h>
#include<string.h>
int main()
{
 struct employee
        {
        char empName[20];
        int age,recordNo=0;
        float salary;
        };
  struct employee emp;
  FILE *fptr;
  char yes_no, name[15];
  int dataFound=0;
  fptr=fopen("c:\\employee.txt","r+b");
```

```
    if(fptr==NULL)
      {
        printf("File does not exist!!!!");
        exit(1);
      }
    printf("Enter the name of employee which is to be modified!!!!\t");
    gets(name);
    while(fread(&emp,sizeof(emp),1,fptr)==1)
      {
        if(strcmp(emp.empName,name)==0)
          {
           dataFound=1;
           printf("\nThe old record is:\n");
           printf("Name\tAge\tSalary\n");
           printf("-------------------------\n");
           printf("%s\t%d\t%.2f",emp.empName,emp.age,emp.salary);
           printf("\nEnter new name, age and salary:\t");
           scanf("%s%d%f",emp.empName,&emp.age,&emp.salary);
           fseek(fptr,sizeof(emp)*recordNo,SEEK_SET);
           if(fwrite(&emp,sizeof(emp),1,fptr)==1)
            printf("\nThe record has been successfully modified!!!!\n");
          }
        recordNo++;
      }
    if(dataFound==0)
      printf("\nMatching data not found!!!!!!");
    fclose(fptr);
    getch();
    return 0;
    }
```

Output

```
Enter the name of employee which is to be modified!!!!  Bina
The old record is:
Name    Age     Salary
-------------------------
Bina    23      23080.00
Enter new name, age and salary:  Beena   25    40000
The record has been successfully modified!!!!
```

Example 11.21

Write a program that allows a user to enter name, age, sex, number of songs recorded for duet singers using a file. The program should also have the provision for editing, searching, deleting and listing the details of singer. The program should be menu driven.

```
#include<stdio.h>
#include<string.h>
int main()
    {
```

```
struct singer
    {
     char name[20];
     int age;
     char sex;
     int noOfSongs;
     };

struct singer si;
FILE *fptr,*fptrTemp;
char yes_no, singer_name[15];
char choice,fileName[15]="d:\\singert.txt";
int dataFound=0,recordNo=0;
fptr=fopen(fileName,"r+b");
if(fptr==NULL)
   {
    fptr=fopen(fileName,"w+b");
    if(fptr==NULL)
       {
       printf("File can not be created!!!!!\n");
       exit(1);
       }
    }
 while(1)
    {
    printf("\n\n1.Add Record\n");
    printf("2.List Record\n");
    printf("3.Modify Record\n");
    printf("4.Search Record\n");
    printf("5.Delete Record\n");
    printf("6.Exit\n");
    printf("Enter Your choice\t");
    fflush(stdin);
    choice=getchar();
    printf("\n");
    switch(choice)
     {
      case '1':
        fseek(fptr,0,SEEK_END);
        printf("Singer's Name:\t");
        fflush(stdin);
        gets(si.name);
        printf("\nSinger's age:\t");
        scanf("%d",&si.age);
        printf("\nSex of Singer:\t");
        flush(stdin);
        si.sex=getchar();
        printf("\nNumber of Songs:\t");
        scanf("%d",&si.noOfSongs);
        fwrite(&si,sizeof(si),1,fptr);
        break;
      case '2':
```

```
            fseek(fptr,0,SEEK_SET);
            printf("Name\tAge\tSex\tnSongs\n");
            while(fread(&si,sizeof(si),1,fptr)==1)
             {
              printf("-------------------------\n");
              printf("%s\t%d\t%c\t%d\n",si.name,si.age,si.sex,si.noOfSongs);
             }
            printf("-------------------------------\n")
           break;
      case '3':
         printf("Enter the name of singer which is to be modified!!!!\t");
         fflush(stdin);
         gets(singer_name);
         rewind(fptr);
         dataFound=0;
         recordNo=0;
         while(fread(&si,sizeof(si),1,fptr)==1)
           {
           if(strcmp(si.name,singer_name)==0)
            {
            dataFound=1;
            printf("\nThe old record is:\n");
            printf("Name\tAge\tSex\tNo Of Songs\n");
            printf("---------------------------\n");
            printf("%s\t%d\t%c\t%d",si.name,si.age,
           si.sex,si.noOfSongs);
           printf("\nEnter new name, age,sex and number of songs\t");
           gets(si.name);
           scanf("%d",&si.age);
           fflush(stdin);
           si.sex=getchar();
           flush(stdin);
           scanf("%d",&si.noOfSongs);
           fseek(fptr,sizeof(si)*(recordNo),SEEK_SET);
           if(fwrite(&si,sizeof(si),1,fptr)==1)
            {
            printf("\nThe record has been modified!!!!\n");
            break;
           }
          }
          recordNo++;
          }
          if(dataFound==0)
          printf("\nMatching data not found!!!!!!");
          break;
      case '4':
         dataFound=0;
         printf("Enter the name of singer which is to be Searched!!!!\t");
         fflush(stdin);
         gets(singer_name);
         fseek(fptr,0,SEEK_SET);
         while(fread(&si,sizeof(si),1,fptr)==1)
```

```
        {
         if(strcmp(si.name,singer_name)==0)
          {
          dataFound=1;
          printf("Name\tAge\tSex\tNo Of Songs\n");
          printf("-------------------------------\n");
          printf("%s\t%d\t%c\t%d\n",si.name,si.age,si.sex,si.noOfSongs);
          }
        }
        if(dataFound==0)
         printf("No Data Found\n");
        break;
    case '5':
     printf("\nEnter name of singer to be deleted!!!!");
     scanf("%s",singer_name);
     fptrTemp=fopen("d:\\temp.dat","wb");
     rewind(fptr);
     while(fread(&si,sizeof(si),1,fptr)==1)
         {
          if(strcmp(si.name,singer_name)!=0)
             {
              fwrite(&si,sizeof(si),1,fptrTemp);
              printf("\nWriting.........\n");
             }
         }
     fclose(fptr);
     fclose(fptrTemp);
     remove(fileName);
     rename("d:\\temp.dat",fileName);
     fptr=fopen(fileName,"rb+");
     printf("\nSuccessfully deleted.......\n");
     break;
   case '6':
     fclose(fptr);
     printf("\nExiting.............");
     getch();
     exit(1);
     break;
   default:
     printf("\nInvalid Character!!!!!!!!!!");
    }
  }
 return 0;
}
```

Output

```
1.Add Record
2.List Record
3.Modify Record
4.Search Record
5.Delete Record
```

```
6.Exit
Enter Your choice      1
Singer's Name:  Narayan Gopal
Singer's age:   45
Sex of Singer:  M
Number of Songs:        800
1.Add Record
2.List Record
3.Modify Record
4.Search Record
5.Delete Record
6.Exit
Enter Your choice      1
Singer's Name:  Koili Devi
Singer's age:   89
Sex of Singer:  F
Number of Songs:        300
1.Add Record
2.List Record
3.Modify Record
4.Search Record
5.Delete Record
6.Exit
Enter Your choice  2
Name          Age     Sex     nSongs
-------------------------------------------------------
Narayan Gopal         45      M       800
-------------------------------------------------------
Koili Devi            89      F       300
-------------------------------------------------------
1.Add Record
2.List Record
3.Modify Record
4.Search Record
5.Delete Record
6.Exit
Enter Your choice      3
Enter the name of singer which is to be modified!!!!    Koili Devi
The old record is:
Name       Age      Sex     No Of Songs
-------------------------------------------------
Koili Devi      89      F       300
Enter new name, age,sex and number of songs     Koili Devi  60  F  600
The record has been  modified!!!!
1.Add Record
2.List Record
3.Modify Record
4.Search Record
5.Delete Record
6.Exit
```

```
Enter Your choice      2
Name            Age     Sex  nSongs
-----------------------------------------------
Narayan Gopal       45      M       800
-----------------------------------------------
Koili Devi          60      F       600
-----------------------------------------------
1.Add Record
2.List Record
3.Modify Record
4.Search Record
5.Delete Record
6.Exit
Enter Your choice      4
Enter the name of singer which is to be Searched!!!!  Koili Devi
Name          Age     Sex     No Of Songs
-----------------------------------------------
Koili Devi      60      F       600
1.Add Record
2.List Record
3.Modify Record
4.Search Record
5.Delete Record
6.Exit
Enter Your choice  1
Singer's Name:  Ram
Singer's age:   27
Sex of Singer:  M
Number of Songs:      55
1.Add Record
2.List Record
3.Modify Record
4.Search Record
5.Delete Record
6.Exit
Enter Your choice      2
Name    Age     Sex     nSongs
-----------------------------------------
Narayan Gopal   45      M       800
-----------------------------------------
Koili Devi      60      F       600
-----------------------------------------
Ram     27      M       55
-----------------------------------------
1.Add Record
2.List Record
3.Modify Record
4.Search Record
5.Delete Record
```

```
6.Exit
Enter Your choice        5
Enter name of singer to be deleted!!!! Ram
Writing.........
Writing.........
Successfully deleted.......

1.Add Record
2.List Record
3.Modify Record
4.Search Record
5.Delete Record
6.Exit
Enter Your choice        2
Name     Age      Sex      nSongs
-----------------------------------------
Narayan Gopal    45       M        800
-----------------------------------------
Koili Devi       60       F        600
-----------------------------------------
```