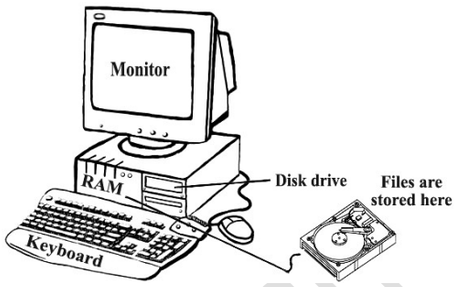
FILE HANDLING –

Definition: A file can be defined as a collection of data / information, which can be stored permanently in a storage device.

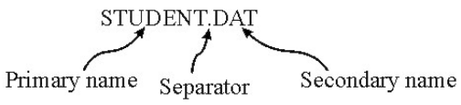
File: A file contains data / information which are stored permanently in a storage device. Floppy disk, compact disk, and hard disk are commonly used to store file information. When large quantity of data is required to be stored and processed, the concept of file is used.

- The data entered through a keyboard are stored, retrieved, processed and the results are printed.
- Consider the following illustration of storage and retrieval.



File Name

- A file stored in a storage device is always identified using a filename (eg. STUDENT.DAT, TEXTINFO.TXT and so on) note that a filename normally has a primary name and a secondary name which are separated by a dot (.).



- The primary and secondary names are assigned by the programmer. Normally the secondary name is used to identify the nature of the file.
- Consider the following secondary names for files -
  - DAT is used to refer to a data file.
  - TXT is used to refer to text file.

Types of files: Files are classified based on the type of data / information stored in them.
The two types of files are –

i. Data file
ii. Text file

- A data file contains data stored in the form of records. A record is a collection of data related to a person or an item.
- e.g - A student record may contain data like rollnumber, students name and marks obtained by him.

Rollno	name	Marks
2201	Arun	78
2202	Deepak	86
.	.	.
.	.	.
.	.	.

2249	Sheriy	98
2250	Sonika	99

- A text file contains information stored in the form of string of characters. The characters entered through the keyboard are stored continuously as illustrated below:

A Text File →

A computer is an electronic device or a multipurpose tool which can be used for data processing, numerical problem solving, graphical 2D or 3D picture generation, multimedia applications and so on.

Sequential access file: In this type, data are kept sequentially. If we want to read the last records of file we need to read all the records before reading that record. It takes more time, or if we desire to access the 10^{th} record, then the first 9 records should be read sequentially for reaching the 10^{th} record.

Random access file: In this type, data can be read and modified randomly. That is if we want to read the last record of the file we can read it directly. It takes less time as compared to a sequential file.

Steps for file operations: There are 3 steps for the operation in C -

a. **Opening a file.**
b. **Reading or writing a file(file processing).**
c. **Closing a file.**

File processing: Data / information stored in a storage device are accessed using a file pointer.

The file processing involves the following operations.

- Creation of file of a type.
- Reading/processing a file.
- Append/add information to a file.
- Modify/edit data in a file.
- Delete items in a file.
- Update the file.

File declaration

: A file is declared and the data is accessed using a file pointer. It has the following –

```
FILE *fp;
```

Where FILE refers to the type name for file declaration,

fp refer to file pointer

Examples:

```
FILE *fp;
FILE *fptr;
FILE *sfile, *tfile;
```

Opening and closing of files:

(1) fopen ( ) function: It is used to open a file and set the file pointer to the beginning/end of a file.

It has the following form.

fp = fopen (“filename”, “mode”);

where, fp refers to file pointer.

filename refers to the name file to be opened.

‘mode’ refers to the operation mode to access data.

The following “mode” are used in data processing -

File access mode string	Meaning	Explanation	Action if file already exists	Action if file does not exist
"r"	Read	Open a file for reading	read from start	failure to open
"w"	Write	Create a file for writing	destroy contents	create new
"a"	Append	Append to a file	write to end	create new
"r+"	read extended	Open a file for read/write	read from start	error
"w+"	write extended	Create a file for read/write	destroy contents	create new
"a+"	append extended	Open a file for read/write	write to end	create new
File access mode flag "b" can optionally be specified to open a file in binary mode. This flag has effect only on Windows systems. On the append file access modes, data is written to the end of the file regardless of the current position of the file position indicator.				
File access mode flag "x" can optionally be appended to "w" or "w+" specifiers. This flag forces the function to fail if the file exists, instead of overwriting it.				

1. “w” to open a file to write data / information.
2. “r” to open a file to read data / information from the beginning of a file.
3. “a” to open a file to add data / information from at the end of an existing file.
4. “w+” to open a file for writing data can be read after writing using the same mode.
5. “r+” to open a file for reading and writing data.
6. “a+” to open a file to write information at the end or a file. Data can be read after writing using the same mode.

Example:
```
FILE *fptr, *tfile;
fptr = fopen(“STUDENT.DAT”, “w”);
tfile = fopen(“TEXTINFO.TXT”, “r”);
```

Note –

a. If a file is opened in “w” or “w+” mode the contents will be erased automatically. But the “w+” mode will facilitate reading newly entered data by moving the pointer to the previous data in the same file. When a file is created in “w” or “w+” mode, the file pointer is always set at the beginning of a file.
b. When a file is opened in “r” or “r+” mode the file pointer is always set at the beginning of the file.
c. When we try to open a file that does not exist the fopen ( ) function will return a NULL.

Example:
```
fptr = fopen(“STD.DAT”, “r”);
if(fptr == NULL)
printf(“\n the given file is not available”);
```

(2) fclose ( ) functions: It is used to close an active file. It has the following form. -

```
fclose(fp);
```

where fp refers to the file pointer.

Example:
```
fclose(fptr);
fclose(tfile);
```

Simple file operations:

In this section the simple file operation commands using different I/O facilitates of < stdio.h > are explained.

File Input / Output -

- character I/O
- string I/O
- word I/O
- formatted I/O

Character I / O –

The following functions are used to handle the character I/O data-types.

a. getc(fptr)
b. putc(ch , fptr)
c. fgetc(fptr)
d. fputc(ch , fptr)

1. **getc( ) function(macro function):-**

It is used to read a single character from a given file and returns --at the end or file (EOF) or an error is encountered.

The general format of the getc( ) function as follows -

```
ch = getc(fptr)
```

where fptr is a file pointer of the file and ch is a variable receive the character.

Example:
```
FILE *fptr;
char c;
```

c = getc(fptr);

Note that the character read from the file is assigned to the char variable c.

2. **fgetc( ) function[true function]:**

The fgetc( ) is also used to read a single character from a gives file and returns the value of the end of file (EOF) or an error is encountered.

The getc is a macro function while fgetc( ) is true function in the stdio.h library. The operations are otherwise same.

The general format of the fgetc( ) function is as follows -

ch = fgetc(fptr)

where fptr is a file pointer of the file, and ch is a variable which receive the character.

Example:
```
FILE *tfile;
char c;
c = fgetc(tfile);
```

Note that the character read from the file is assigned to the char variable C.

3. **putc( ) function:**

The putc( ) function is used to write a single character into a file.

The general format of the fputc( ) function is as follows:

```
FILE *fptr;
char ch;
putc(ch, fptr);
```

where ch is the character to be written and fptr is a file pointer to the file to receive a character.

Example:
```
FILE *tfile;
char c;
putc(c, tfile);
```

4. **fputc( ) function:**

The fputc( ) function is also used to write a single character onto a given file.

The format of the fputc( ) is same as the putc( ) function. The only difference being that putc( ) is an macro function while fputc( ) is an the function in the stdio.h library.

The general format of the fputc( ) is:

fputc(ch, fptr)

Where ch is a character to be written and fptr is a file pointer to the file to receive the character.

Example:
```
FILE *tfile;
char c;
fputc(c, tfile);
```

write text of the char variable c is written in the file .

Note that the value of the char variable c is written in the file.

Program example: getc() putc()

Que. - Write a C program to create a text file.

Solution:

```
/* text file creation */
#include<stdio.h>
void main( )
{
        char ch;
        FILE *fptr;
        fptr = fopen("SAMPLE.TXT", "w");
       printf("Type the text press enter key at end.");
        /* loop to input the text */
        while((ch = getchar( ) != '\n')
                putc(ch, fptr);
        fclose(fptr);
}
```

Test Run:

Type the text press enter key at end.

Computer programming in C language is widely used for science and engineering applications.

Que. - Write a C program to read the text file created in above program (SAMPLE.TXT) and also count the number of vowels present in the file.

Solution:

```
/* text file reading */
#include<stdio.h>
#include<ctype.h>
void main( )
{
        char ch;
        int count = 0;
        FILE *fp;
        fp = fopen("SAMPLE.TEXT", "r");
        printf("\n the content of the text file is :  \n \n");
        /* loop to display the text */

        while(!eof(fp))
        {
                ch = getc(fp);
                printf("%c", ch);
                switch(toupper(ch))
                {
                        case 'A' :
                        case 'E' :
                        case 'I'  :
                        case 'O' :
                        case 'U' :      count++;
                        break;
```

```
                }
            }
            printf(" \n \n No. of vowels present = %d", count);
            fclose(fp);
        }
```

Test Run:

The content of the text file is:

Computer programming in C language is widely used for science and engineering applications.

No. of vowels present = 31

Que– Program to understand the use of putc and getc function.

Program:

```
#include <stdio.h>
#include <conio.h>
main()
{
FILE *fp;
Char ch;
fp = fopen ("one.txt" , "w");
printf("Enter data and press enter key at end");
while((ch=getchar())!='\n')
      putc(ch,fp);
fclose(fp);

printf("Entered  data→\n");
fp = fopen("one.txt" , "r");
while((ch=getc(fp))!=EOF)
     printf("%c",ch);
fclose(fp);
}
```

Program example: fgetc() fputc()

Que. – Program to understand the use of fputc function.

Solution:

```
#include<stdio.h>
void main( )
{
        int ch;
        FILE *fptr;
        fptr = fopen("myfile.txt", "w");
        if (fptr==NULL)
        {
        printf("File does not exist");
        exit(1);
        }
        else
        {
printf("Type the text press ctrl+z key to stop.\n");
        /* loop to input the text */
        while((ch = getchar( ) != EOF)
                fputc(ch, fptr);
        fclose(fptr);
}
```

Test Run:

Type the text press ctrl+z key to stop

Computer programming in C language is widely used for science and engineering applications ^Z

Que. – Program to understand the use of fgetc function.

Solution:

```
#include<stdio.h>
void main( )
{
        int ch;
        FILE *fptr;
        fptr = fopen("myfile.txt", "r");
        if (fptr==NULL)
        {
        printf("File does not exist");
        exit( );
        }
        else
        {
        while((ch = fgetc(fptr ) )!= EOF)
        printf("%c", ch);
        }
        fclose(fptr);
}
```

Test Run:

Computer programming in C language is widely used for science and engineering applications

String I/O -

fgets( ) function: The fgets( ) function is used to read a set of character as a string from a given file and copies the string to a given memory location normally in an array.The general format of the fgets( ) function is as follows:

fgets(sptr, n, fptr) ;

Where sptr is a pointer to the location to receive the string, n is a count of the maximum number of characters to be in the string and fptr is the file pointer to the file to be read.

Example: FILE *fptr;

char a[MAX]; or char a[100];

fgets(a, MAX, fptr);

or

fgets(a, 50, fptr);

Que. – Program to understand the use of fgets function.

Solution:

```
#include<stdio.h>
void main( )
{
        char str[80];
        FILE *fptr;
        fptr = fopen("test.txt", "r");
        if (fptr==NULL)
        {
           printf("File does not exist");
           exit(1);
        }
        else
        {
        while((fgets(str,80,fptr) != NULL)
                puts(str);
      fclose(fptr);
}
```

Test Run:

```
Computer programming in C language is
widely used for science and
engineering applications
```

fputs( ) function: The fputs( ) function is used to write a string to a given file. It is a pointer the string to be written. The general format of the fputs( ) function is as follows:

fputs(sptr,fptr);

Where sptr is a pointer to the string to be witten and fptr is a file pointer to the file.

The fputs( ) function is normally used to copy strings from one file to another.

Example:FILE *fptr;

char test[100];

fputs(test,fptr);

Que. – Program to understand the use of fputs function.

Solution:

```
#include<stdio.h>
void main( )
{
        char str[80];
        FILE *fptr;
        fptr = fopen("test.txt", "w");
        if (fptr==NULL)
        {
        printf("File does not exist");
        exit( );
        }
        else
        {
printf("Enter the text to stop entering  press ctrl+z .\n");
        /* loop to input the text */
        while((gets(str) != NULL)
                fputs(str, fptr);
       fclose(fptr);
}
```

Test Run:

```
Enter the text to stop entering  press ctrl+z
Computer programming in C language is
widely used for science and
engineering applications ^Z
```

Word I/O - The stdio.h supports to read and write integer value on a given file. The following functions are used to process the integer quantities.

a. getw( )
b. putw( )

getw( ): The getw( ) function is used to read an integer value from a given file. It returns the next integer from the input file, or EOF, an error encountered.

The general format of the getw( ) is as follows:

```
int getw(FILE *fptr)
```

where fptr is a pointer to a file to receive an integer value.

Example:
```
FILE *fptr;
int c;
c = getw(fptr);
```

putw( ): The putw( ) function is used to write an integer quantity on to the specified file.

The syntax of the putw( ) is:

```
int putw(int value, FILE *fptr)
```

where value is an integer value to be written on a given file and fptr is a file pointer to a given file.

Example:
```
FILE *fptr;
int i;
putw(i, fptr);
```

Programming Example putw( ) and getw( )

Que. – Program to understand the use of putw function.

Solution:

```
#include<stdio.h>
void main( )
{
    int i;
    FILE *fptr;
    fptr = fopen("num.dat", "w");
    if (fptr==NULL)
    {
    printf("File does not exist");
    exit( );
    }
    else
    {
    for(i=0;i<=30;i++)
        putw(i, fptr);

    fclose(fptr);
}
```

Que. – Program to understand the use of getw( ) function.

Solution:

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

```
{
        int i;
        FILE *fptr;
        fptr = fopen("num.dat", "r");
        if (fptr==NULL)
         {
           printf("File does not exist");
           exit(1);
         }
        else
          {
             while (i=getw(fptr)!=EOF)
                printf("%d",i);
          }
 fclose(fptr);
}
```

Formatted I/O -

The stdio.h library provides all types data handling features for the requirement of a programmer. Sometimes one may be requires to store and retrieve only formatted data. The formatted data can also be processed with a file handling. The following functions are use to access and process the formatted data.

a. fscanf( )
b. fprintf( )

fscanf( ): It is used to read a formatted data from a specified file.

The controlled string to read a format data is same as the scanf( ) function. The scanf( ) function reads from the keyboard while fscanf( ) reads from a given file.

The general format of the fscanf( ) function is as follows:

```
fscanf(fptr, "control string", &list)
```

Where - fptr is a file pointer to receive a formatted data
control/string is a different format commands such as %d, %f etc. and
list is a variable parameter list to be read from the specified file and normally the list is used along with address operations.

Example:

```
void main( )
{
        FILE *fptr;
        int x, y;
        fptr = fopen(name, "r");
        - - - - -
        - - - - -
        while((fscanf(fptr, %d %d", &x, &y))! = EOF)
        {
                - - - - -
                - - - - -
        }
}
```

fprintf( ): The fpintf( ) function is used to write a formatted data into a given file. The printf( ) function is used to write or display the formatted data on the video screen while the fprintf( ) is used to write a specified file. The control string is same for both printf( ) and fprintf( ) functions.

The general format of the fprintf( ) is as follows:

fprintf(fptr, "control string", list)

Where - fptr is a file pointer to write a formatted data,
control string is a format command such as %d %c %f etc and
list is the variable list to be written in to the file

Example:

```
void main ( )
{
    FILE *fptr;
    int x, y;
    fptr = fopen(name, "w");
    - - - - -
    - - - - -
    fprintf(fptr, "%d %d \n", x, y);
    - - - - -
    - - - - -
    fclose(fptr);
}
```

Que. - A file called "STUDENT.DAT" contains information such as student rollnumber, name and total marks. Write a C program to create a file to store details of n students.

Solution:

```
/* file creation */
#include<stdio.h>
void main( )
{
    int rno, total, i, n;
    char sname[20];
    FILE *fp;
    fp = fopen("STUDENT.DAT", "w");
    clrscr( );
    printf("\n How many students ? ");
    scanf("%d", &n);
    /*  loop to read n students details */
    for(i=0; i<n; i++)
    {
        printf("\n student rollno., name, total ?");
        scanf("%d %s %d", &rno, name, &total);
        fprintf(fp, "%d %s %d \n", rno, name, total);
    }
    fclose(fp);
}
```

Test Run:

```
How many students ?

Student rollno., name, total ? 101, A, 78
```

Student rollno., name, total ? 102, B, 79

Que. - Write a C program to read information from a file "STUDENT.DAT" which is created in STUDENT.DAT. Also display the list of students who have scored 75 marks and above.

Solution:

```
/* file processing */
#include<stdio.h>
void main( )
{
        int rno., tot;
        char sname[20];
        FILE *fptr;
        fptr = fopen("STUDENT.DAT", "r");

/* list of students who have scored 75 marks and above */
        printf("\n - - - - - - - - - - - - - - - - - - - - - - -")
        printf("Roll no          Name          Total");
        printf("\n - - - - - - - - - - - - - - - - - - - - - - - ");
        while(!feof(fptr))
        {
                fscanf(fptr, "%d %s %d\n", &rno., name, &tot);
                if(tot >= 75)
                printf("\n %6d * %-20s %6d", rno., name, tot);
        }
        printf("\n- - - - - - ------------------------------ ");
        fclose(fptr);
}
```

Que. - Write a C program to add / append information to the file "STUDENT.DAT" which is created in STUDENT.DAT. Also display the details of all the students after appending.

Solution:

```
/* Append / Add records to a file */
#include<stdio.h>
#include<ctype.h>
void main( )
{
        int rno., tot;
        char sname[20], ch = 'Y';
        FILE *fptr;
        fp = fopen("STUDENT.DAT", "a+");
        /* loop to add students details */
        while(toupper(ch) = = 'Y')
        {
                printf("\n student rollno., sname, total ?");
                scanf("%d %s %d\n", &rno., sname, &tot);
                fprintf(fp,"%d %s %d\n", rno., sname, tot);

                printf("\n press y to add more records");
                printf("\n any other key to stop")
                ch = getche( );
```

```
        }

            /* loop to print the details of all students */
            rewind(fp);
            printf("\n - - - - - - ");
            printf("\n Roll No Name Total");
            printf("\n - - - - - -");
            while(!feof(fp))
            {
                    fscanf(fp, "%d %s %d", &rno., sname, &tot);
                    printf("\n %6d \t %-20s %6d", rno., sname, tot);
            }
            printf("\n - - - - - -");
            fclose(fp);
    }
```

Test Run:

Student roll no., name, total ? 103 R 67
press y to add more records
any other key to stop. . . .n

O/P -

Roll no.	Name	Total
101	A	78
102	B	79
103	R	67

}

/* Record I/O in Files writing structure In to file */

```
#include<stdio.h>
#include<conio.h>
int main()
{
FILE *fp;
char another='Y';
struct emp
{
char name[40];
int age;
float bs;
};
struct emp e;
fp=fopen("employee.dat","w");
if (fp==NULL)
{
puts("cannot open file");
exit(1);
}
```

```
while(another==’Y’)
{
printf(“\n Enter name age and basic salary:”);
scanf(“%s%d%f”,e.name,&e.age,&e.bs);
fprinf(fp ,“%s %d %f” , e.name , e.age , e.bs);
printf(“Add another record(Y/N)”);
fflush(stdin);
another=getche();
}
fclose(fp);
return 0;
} } }
```

Output:
Enter name , age and basic salary : sunil 34 1250.50
Add another record (Y/N) Y
Enter name , age and basic salary : sameer 31 1300.50
Add another record (Y/N) Y
Enter name , age and basic salary : rahul 34 1400.55
Add another record (Y/N) N

/* Record I/O in Files reading structure from file */

```
#include<stdio.h>
#include<conio.h>
int main()
{
FILE *fp;
char another=’Y’;
struct emp
{
char name[40];
int age;
float bs;
};
struct emp e;
fp=fopen(“employee.dat”,”r”);
if (fp==NULL)
{
puts(“cannot open file”);
exit(1);
}
printf(“\n name age and basic salary:”);

while((fscanf(fp ,“%s %d %f” , e.name , &e.age , &e.bs)!=null)
{
printf(“%s %d %f”, e.name , e.age , e.bs);
}
fclose(fp);
return 0;
} } }
```

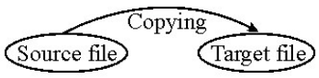
Output:

name	age	basic salary :
sunil	34	1250.50
sameer	31	1300.50
rahul	34	1400.55

Que. - Write a C program to copy the contents of the one file to another.

Solution: The existing file is a source file.

All the character of a source file are read one by one and written as another file called target file.




```
/* text file copying */
#include<stdio.h>
main( )
{
        char ch, source_name[12], target_name[12];
        FILE *source_fptr, *target_fptr;
        printf("\n Enter the source file name");
        scanf("%s", source_name);
        source_fptr = fopen(source_name, "r");
        if(source_fptr==NULL)
         {
            printf("file not exist");
             exit(1);
        }
        printf("\n Enter the target file name:");
        scanf("%s", target_name);
        target_ptr = fopen(target_name, "w+");
        if (target_fptr = =NULL)
                {
                        printf("\n Insufficient memory !!!");
                        fclose(source_name);
                        printf("\n press any key. . . ");
                        getch( );
                        exit( );
                }
/* loop to copy the text */
while(! feof(source_fptr))
{
        ch = getc(source_fptr);
        putc(ch, target_fptr);
}
printf("\n The contents of the target file are : ");
printf("\n - - - - - - -");
rewind(target_fptr);
while(!feof(target_fptr))
{
```

```
        ch = getc(target_fptr);
        printf("%c", ch);
    }
    fclose(source_fptr);
    fclose("target_fptr");
    printf("\n \n press any key. . . ");
    }
```

Test Run:

Enter the source file name: SAMPLE.TXT

Enter the target file name: SAMPLE.BAK

The contents of the target file are

Computer programming in C language is widely used for science and engineering applications.

Press any key

fseek( )function:

fseek( ) function is used to move the file pointer to any position in a file from a given reference position. It has the following from:

fseek(fp, m, ref_pos);

where, fp-refers to file pointer

m-refers to the number of bytes to be skipped.

ref_pos refers to the reference position from where n no. of bytes are to be skipped.

Following are symbols for reference positions -

Value	**Symbol**	**Reference**
0	SEEK_SET	from the beginning of a file.
1	SEEK_CUR	from the current position of file pointer.
2	SEEK_END	from the end of a file.

Example:

1. fseek(sptr, sizeof(rec), SEEK_SET);
   will first set the file pointer to the first record and then move the file pointer n number of bytes forward.

2. fseek(efile, -n, SEEk_cur);
   will move the file pointer n number of byte backward. If n is equal to the size of structure, then the file pointer is set to the previous record in the file.
   Example :

```
#include <stdio.h>
int main ()
{
  FILE *fp;
  fp = fopen("file.txt","w+");
  fputs("This is Siddhartha choubey", fp);
```

```
    fseek( fp, 7, SEEK_SET );
    fputs(" C Programming Langauge", fp);
    fclose(fp);

    return(0);
}
```

Output:
This is C Programming Langauge

fread( ) function:
fread( ) function is used to read a structure from a file. Note that the values of the data members in the structure are read together.
This has the following form:

fread(&s, m, n, fp);

where, s - refers to the structure variable.
m - refers to the size of the structure
n - refers to the number of structures to be read from a file usually it is assigned 1.
fp - refers to file pointer.
Example: fread(&srec, sizeof(srec), 1,sptr);
fread(&emprec, 24, 1, efile);

fwrite( ) function:
fwrite function is used to write a structure to a file. The value of the data members in the structure are written together.
This has the following form:

fwrite(&s, m, n, fp);

where, s - refers to the structure variable.
m - refers to the size of the structure
n - refer to the number of structures to be written to a file. Usually it is assigned 1.
fp - refers to file pointer.

Example: fwrite(&srec, sizeof(srec), 1, fptr);
fwrite(&emprec, 26, 1, efile);

/* Record I/O in Files writing structure In to file using fwrite*/

```
#include<stdio.h>
#include<conio.h>
int main()
{
FILE *fp;
char another='Y';
struct emp
{
char name[40];
int age;
float bs;
```

```
};
struct emp e;
fp=fopen("employee.dat","wb");
if (fp==NULL)
{
puts("cannot open file");
exit(1);
}

while(another=='Y')
{
printf("\n Enter name age and basic salary:");
scanf("%s%d%f",e.name,&e.age,&e.bs);
fwrite(&e,sizeof(e),1,fp);
printf("Add another record(Y/N)");
fflush(stdin);
another=getche();
}
fclose(fp);
return 0;
} } }
```

Output:

```
Enter name , age and basic salary : sunil 34 1250.50
Add another record (Y/N) Y
Enter name , age and basic salary : sameer 31 1300.50
Add another record (Y/N) Y
Enter name , age and basic salary : rahul 34 1400.55
Add another record (Y/N) N
```

/* Record I/O in Files reading structure from file */

```
#include<stdio.h>
#include<conio.h>
int main()
{
FILE *fp;
char another='Y';
struct emp
{
char name[40];
int age;
float bs;
};
struct emp e;
fp=fopen("employee.dat","rb");
if (fp==NULL)
{
puts("cannot open file");
exit(1);
}
```

```
printf("\n name age and basic salary:");

while((fread(&e , sizeof(e) , 1 , fp )==1)
        printf("%s %d %f", e.name , e.age , e.bs);

fclose(fp);
return 0;
} } }
```

Output:

```
 name     age   basic salary :
  sunil     34    1250.50
  sameer  31    1300.50
  rahul     34    1400.55
```

/* Data base programming */

```
/* A menu-driven program for elementary database management */
#include "stdio.h"
main( )
{
FILE *fp, *ft ;
char another, choice ;
struct emp
{
char name[40] ;
int age ;
float bs ;
} ;
struct emp e ;
char empname[40] ;
long int recsize ;
fp = fopen ( "EMP.DAT", "rb+" ) ;
if ( fp == NULL )
{
fp = fopen ( "EMP.DAT", "wb+" ) ;
if ( fp == NULL )
{
puts ( "Cannot open file" ) ;
exit( ) ;
}

}
recsize = sizeof ( e ) ;
```

```
while ( 1 )
{
clrscr( ) ;
printf ( "\n MENU \n" ) ;
printf ( "\n1. Add Records\n") ;
printf ( "\n2. List Records\n") ;
printf ( "\n3. Modify Records\n") ;
printf ( "\n4. Delete Records\n") ;
printf ( "\n0. Exit\n" ) ;
printf ( "Your choice" ) ;
fflush ( stdin ) ;
choice = getche( ) ;
switch ( choice )
{
case '1' :
fseek ( fp, 0 , SEEK_END ) ;
another = 'Y' ;
while ( another == 'Y' )
{
printf ( "\nEnter name, age and basic sal. " ) ;
scanf ( "%s %d %f", e.name, &e.age, &e.bs ) ;
fwrite ( &e, recsize, 1, fp ) ;
printf ( "\nAdd another Record (Y/N) " ) ;
fflush ( stdin ) ;
another = getche( ) ;
}
break ;
case '2' :
rewind ( fp ) ;
while ( fread ( &e, recsize, 1, fp ) == 1 )
printf ( "\n%s %d %f", e.name, e.age, e.bs ) ;
break ;
case '3' :
another = 'Y' ;
while ( another == 'Y' )
{
printf ( "\nEnter name of employee to modify " ) ;
scanf ( "%s", empname ) ;
rewind ( fp ) ;
while ( fread ( &e, recsize, 1, fp ) == 1 )
 {
if ( strcmp ( e.name, empname ) == 0 )
{
printf ( "\nEnter new name, age & bs" ) ;
scanf ( "%s %d %f", e.name, &e.age,
&e.bs ) ;
fseek ( fp, - recsize, SEEK_CUR ) ;
fwrite ( &e, recsize, 1, fp ) ;
break ;
}
}
printf ( "\nModify another Record (Y/N) " ) ;
```

```
fflush ( stdin ) ;
another = getche( ) ;
}
break ;
case '4' :
another = 'Y' ;
while ( another == 'Y' )
{
printf ( "\nEnter name of employee to delete " ) ;
scanf ( "%s", empname ) ;
ft = fopen ( "TEMP.DAT", "wb" ) ;
rewind ( fp ) ;
while ( fread ( &e, recsize, 1, fp ) == 1 )
{
if ( strcmp ( e.name, empname ) != 0 )
fwrite ( &e, recsize, 1, ft ) ;
}
fclose ( fp ) ;
fclose ( ft ) ;
remove ( "EMP.DAT" ) ;
rename ( "TEMP.DAT", "EMP.DAT" ) ;
fp = fopen ( "EMP.DAT", "rb+" ) ;
printf ( "Delete another Record (Y/N) " ) ;
fflush ( stdin ) ;
another = getche( ) ;
}
break ;
```

```
case '0' :
fclose ( fp ) ;
exit( ) ;
} } }
```

ftell( ) function:

ftell( ) function is used to locate the current position of the pointer.
This has the following form:

n = ftell(fp);

where, fp refers to file pointer
n refers to the byte number or the current positions.

Example: p = ftell(sptr);
m = ftell(tfile);

Note that p will be assigned to the current position. Assume 'p' may be assigned a value 126 which represents the 126th byte in the file.

remove( ) function:

remove( ) function is used to erase or remove a file from the disk.
It has following form: remove("filename");

Example: remove("STUDENT.TXT");
remove("STUDENT.DAT");

rename( ) function:

rename( ) function is used to rename an existing file.
It has the following form:

rename("old filename", "new filename");

Example: rename("TEMP", "STUDENT.DAT");
rename("SAMPLE.TXT", "JP.DOC");

NOTE:- Make sure that the files are closed before remove( ) and rename( ) functions will be used.
Use fclose( ) to close the files before using these functions.

rewind( ) function

This function resets the file pointer at the beginning of the file
rewind(FILE *fp);
Description
The C library function void rewind(FILE *stream) sets the file position to the beginning of the file of the given stream.
Declaration

void rewind(FILE *stream)

Parameters

stream -- This is the pointer to a FILE object that identifies the stream.

Return Value

This function does not return any value.

Example

The following example shows the usage of rewind() function.

```
#include <stdio.h>
int main()
{
  FILE *fp;
  int ch;
  fp = fopen("file.txt", "r");
  if( fp != NULL )
  {
    while( !feof(fp) )
    {
      ch = fgetc(fp);
      printf("%c", ch);
    }
    rewind(fp);
    while( !feof(fp) )
    {
      ch = fgetc(fp);
      printf("%c", ch);
    }
    fclose(fp);
  }

  return(0);
}
```

Assuming we have a text file file.txt having the following content:

```
This is siddhartha
```

Now let us compile and run the above program, this will produce the following result:

```
This is siddhartha
This is siddhartha
```

feof( ) function

The C library function int feof(FILE *stream) tests the end-of-file indicator for the given stream.

Declaration

Following is the declaration for feof() function.

```
int feof(FILE *stream)
```

Parameters

stream -- This is the pointer to a FILE object that identifies the stream.

Return Value

This function returns a non-zero value when End-of-File indicator associated with the stream is set, else zero is returned.

Example

The following example shows the usage of feof() function.

```
#include <stdio.h>

int main ()
{
  FILE *fp;
  int c;
```

```
  int n = 0;

  fp = fopen("file.txt","r");
  if(fp == NULL)
  {
    perror("Error in opening file");
    return(-1);
  }
  while(!feof(fp))
  {
    c = fgetc(fp);
    printf("%c", c);
  }

  fclose(fp);

  return(0);
}
```

Assuming we have a text file file.txt, which has the following content. This file will be used as an input for our example program:
This is Siddhartha
Let us compile and run the above program, this will produce the following result:
This is Siddhartha

Que. - Write a program to read data from the keyboard , write it to a file called INPUT again read the same data from the INPUT file and display it on the screen.

Solution:

```
#include<stdio.h>
void main( )
{
        FILE *f1;
        char c;
        printf("Data Input \n \n");
        /* open the file input */
        f1 = fopen("INPUT", "w");
        /* write a character from keyboard */
        while((c = getchar( ))! = EOF)
        /* write a character to input */
            putc(c, f1);
        /* close the file input */
        fclose(f1);
        printf("\n Data Output \n \n");
        /* reopen the file input */
        f1 = fopen("INPUT", "r");
        /* Read a character from Input */
        while((c = getc(f1))! = EOF)
        /* Display a character on screen */
        printf("%c", c);
```

```
        fclose(f1)
}
```

Output:

Data input

This is a program to test the file handling features on this system.

Data output

This is a program to test the file handling features on this system.

Que. - A file named DATA contains a series of integer numbers. Code a program to read these numbers and then write all 'odd' numbers to a file to be called ODD and all 'even' numbers to a file to be called EVEN.

Solution:

```
#include<stdio.h>
void main( )
{
        FILE *f1, *f2, *f3;
        int number,i;
        printf("contents of DATA file \n \n");
        f1 = fopen("DATA", "w");
        /* create data file */
        for(i=1; i < = 30; i++)
        {
                scanf("%d", &number);
                if(!number)
                        break;
                putw(number, f1);
        }
        fclose(f1);
        f1 = fopen("DATA", "r");
        f2 = fopen("ODD", "w");
        f3 = fopen("EVEN","w");
        /* Read from data file */
        while((number = getw(f1)) ! = EOF)
        {
                if(number %2 = = 0)
                        putw(number, f3);
                        /* write to even file */
                else
                        putw(number, f2);
                        /* write to odd file */
        }
        fclose(f1);
        fclose(f2);
        fclose(f3);
```

```
        f2 = fopen("ODD", "r");
        f3 = fopen("EVEN", "r");
        printf("\n \n contents of ODD file \n \n");
        while((number = getw(f2)) ! = EOF)
            printf("%4d", number);
        printf("\n \n contents of EVEN file \n \n");
        while(()number = getw(f3)) ! = EOF)
            printf("%4d", number);
        fclose(f2);
        fclose(f3);
}
```

Output:

```
Contents of DATA file
111  222  333  444  555  666  777  888  999  000  121  232  343  454
565  -1
Contents of ODD file
112  333  555  777  999  121  343  565
Contents of EVEN file
222  444  666  888  0    232  454
```

Que. - Write a program to open a file named INVENTORY and store in it the following data:

Item name	Number	Price	Quantity
AAA-1	111	17.50	115
BBB-2	125	36.00	75
C-3	247	31.75	104

Extend the program to read this data from the file INVENTORY table with the value of each item.

Solution:

```
#include<stdio.h>
void main( )
{
        FILE *fp;
        int number, quantity, i;
        float price, value;
        char item[10], filename[10];
        printf("Input filename \n");
        scanf("%s", filename);
        fp = fopen(filename, "w");
        printf("Input inventory data \n \n");
        printf("Item Name Number Price Quantity \n");
        for(i=1; i<=3; i++)
        {
            fscanf(stdin, "%s %d %f %d", item, &number, &price, &quantity);
            fprintf(fp, "%s %d %.2f %d", item, number, price, quantity);
        }
        fclose(fp);
        fp = fopen(filename, "r");
```

```
        printf("Item name Number Price Quantity Value \n");
        for(i=1; i<=3; i++)
        {
            fscanf(fp, "%s %d %f %d", item, &number, &price, &quantity);
            value = price*quantity;
            fprintf(stdout, "%8s %7d %8.2f %8d %11.2f \n", item, number, price, quantity, value);
        }
        fclose(fp);
}
```

Output:

Input file name
INVENTORY
Input inventory data

Item name	Number	Price	Quantity
AAA-1	111	17.50	115
BBB-2	125	36.00	75
C-3	247	31.75	104

Item name	Number	Price	Quantity	value
AAA-1	111	17.50	115	2012.50
BBB-2	125	36.00	75	2700.00
C-3	247	31.75	104	3302.00

Error handling during I\O operations -

It is possible that an error may occur during I\O operations on a file.
Typical error situations include:

1. Trying to read beyond the end of file mark.
2. Device overflow.
3. Trying to use a file that has not been opened.
4. Trying to perform an operation on a file, when the file is opened for another type of operation.
5. Opening a file with an invalid filename.
6. Attempting to write to a write-protected file.

Que. - Write a program to illustrate error handling in file operations.

Solution:

```
#include<stdio.h>
void main( )
{
        char *filename;
        FILE *fp1, *fp2;
        int i, number;
        fp1 = fopen("TEST", "w");
        for(i=10; i<=100; i+=10)
            putw(i, fp1);
        fclose(fp1);
        printf("\n Input filename \n");
open_file : scanf("%s", filename);
        if((fp2 = fopen(filename, "r")) = = NULL)
```

```
        {
                printf("can not open file \n");
                printf("Type filename again \n \n");
                goto open_file;
        }
        else
                for(i=1; i<=20; i++)
                {
                        number = getw(fp2);
                        if(feof(fp2))
                        {
                                printf("\n Ran out of data \n");
                                break;
                        }
                        else
                                printf("%d \n", number);
                }
        fclose(fp1);
}
```

Output:

```
Input filename
Tets
Can not open the file
Type filename again
TEST
10
20
30
40
50
60
70
80
90
100
Ran out of data.
```

Command line arguments

- It is a parameter supplied to a program when the program is invoked. This parameter may represent a filename the program should process.
- To run a program the name of the program is entered at the OS's prompt. In addition to entering the name of the program, we can also enter other items such as the name of the file the program may be using.

Example: You can start turbo.C's IDE in the following ways:-

1. By just typing the TC at DOS prompt as
   C: \TC>TC ←

2. In this case, the operating screen wire appears and you can load any file into the environment by using file option.
3. By typing TC and name of the program file to work on, as

   C:\TC>Tc Test ←

   In this case, if program file test.c already exists, it will be loaded into the environment and the editing session will start. If program file test.c does not exist, then the environment will start the edit session to create a new file. In later case, the string test is used as a command line arguments - an argument that appears on the command line following the OS prompt. We give multiple arguments on the command line.

Example: The copy command needs the two arguments, where 1st argument is the name of the destination file.

```
/* program to demonstrate the use of command line arguments */
#include<stdio.h>
void main( int agrc , char *argv [ ] )
{
        int i:
        printf("\n command = %s \n", argv[0]);
        printf("\n \n Number of argument = %d",argc);
        printf("\n \n There are \n \n");
        for("i=0; i<argc; i++")
        {
                printf("Argument Number %d = %s \n", i, argv[i]);
        }
}
```

Test Run:

```
C:\TC> cmdline first second third fourth
C:\TC> cmdline.exe
Number of arguments = 4
There are
Arguments number
1 = first, 2 = second, 3 = third, 4 = fourth
```

The two arguments used in the definition of function main( ) are:-

1. The first argument is **argc** that is of type integer and holds the total no. of entries on the command line, including the program name.
2. The second argument is ***argv[ ]**, an array of pointer to strings which holds the individual arguments, the argument argv[0] holds the name of the program argv[1] holds the first arguments, argv[2] holds the second argument and so on.

   These argument can also be accessed using pointer notation as –

   *(argv+0) , *(argv+1) , *(argv+2) and so on.

Que. - Write a program that will receive a filename and a line of text or command line arguments and write the text to the file.

Solution:

```
#include<stdio.h>
void main(int argc, char *argv[ ])
{
        FILE *fp;
        int i;
        char word[15];
        fp = fopen(argv[1],  ”w”);
        printf(“\n No. of arguments in command line = %d \n \n”, argc)
        for(i=2; i<argc; i++)
                fprintf(fp, ”%s”, argv[i]);
        fclose(fp);
        /* writing content of the file to screen */
        printf(“contents of %s file \n \n” argv[1]);
        fp = fopen(argv[1], “r”);
        for(i=2; i<argc; i++)
        {
                fscanf(fp, “%s”, word);
                printf(“%s”, word);
        }
        fclose(fp);
        printf(“\n \n”);
        /* writing the arguments from memory */
        for(i=0; i<argc; i++)
        printf(“%c ”,word[i]);
}
```

Output:

```
C /tc> f12_7 TEXT AA BB CC DD EE FF GG
No. of arguments in command line = 9
Contents of TEXT file
C:/tc|>f12_7.EXE
TEXT
        AA
                BB
                        CC
                                DD
                                        EE
                                                FF
                                                        GG
```

Que. - Write a C program to read a data file containing integers, find the largest and smallest integers and display them.]

Do it ur self

Difference Between Text File and Binary File

Three main areas where text and binary mode files are different. These are:
Handling of newlines
Representation of end of file
Storage of numbers

Text versus Binary Mode: Newlines

We have already seen that, in text mode, a newline character is converted into the carriage return-linefeed combination before being written to the disk. Likewise, the carriage return-linefeed combination on the disk is converted back into a newline when the file is read by a C program. However, if a file is opened in binary mode, as opposed to text mode, these conversions will not take place.

Text versus Binary Mode: End of File

The second difference between text and binary modes is in the way the end-of-file is detected. In text mode, a special character, whose ASCII value is 26, is inserted after the last character in the file to mark the end of file. If this character is detected at any point in the file, the read function would return the EOF signal to the program.
As against this, there is no such special character present in the binary mode files 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.
There is a moral to be derived from the end of file marker of text mode files. If a file stores numbers in binary mode, it is important that binary mode only be used for reading the numbers back, since one of the numbers we store might well be the number 26 (hexadecimal 1A). If this number is detected while we are reading the file by opening it in text mode, reading would be terminated prematurely at that point.
Thus the two modes are not compatible. See to it that the file that has been written in text mode is read back only in text mode. Similarly, the file that has been written in binary mode must be read back only in binary mode.

Text versus Binary Mode: Storage of Numbers

The only function that is available for storing numbers in a disk file is the **fprintf( )** function. It is important to understand how numerical data is stored on the disk by **fprintf( ).** Text and characters are stored one character per byte, as we would expect. Are numbers stored as they are in memory, two bytes for an integer, four bytes for a float, and so on? No. Numbers are stored as strings of characters. Thus, 1234, even though it occupies two bytes in memory, when transferred to the disk using **fprintf( )**, would occupy four bytes, one byte per character. Similarly, the floating-point number 1234.56 would occupy 7 bytes on disk. Thus, numbers with more digits would require more disk space.
Hence 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 and use those functions (**fread( )** and **fwrite( )**) which store the numbers in binary format. It means each number would occupy same number of bytes on disk as it occupies in memory.

Introduction:

A file is a place on disk where a group of related data is stored.
It is advantageous to use files in the following circumstances.

1. When large volume of data are handled by the program and
2. When the data need to be stored permanently without getting destroyed when program is terminated.

There are two kinds of files:

1) Text files or sequential access file 2) Binary files or Random access files

1) Text files or sequential access files: The text file contains the characters in sequence. The computer process the text files sequentially and in forward direction. One can perform file reading; writing and appending operations. A stream of characters is processed in this kind of files. At one time, one character is processed and that too sequentially. Either read a character or write a character operation is performed in these files.

v mode specifies the purpose of opening a **file**

Different modes of opening a text file are

a) "r" : It opens the file for reading if it already exists otherwise returns error.

b) "w" : It the file already exists then it over write the content otherwise a file is created for writing.

c) "a" : If the file exists, the file pointer points to end of file so that data can be appended to file. If the file doesn't exist then creates a new file and file pointer points to beginning of file.

d) "r+" : The existing file is opened to the beginning for both reading and writing

e) "w+" : It is same as w except that it is opened for both reading and writing.

f) "a+" : It is same as a except that it is opened for both reading and writing.

NOTE: In a sequential access system, records are stored in a serial order and retrieved in the same order. If a read operation is to be performed on the last record, then all the records before the last record need to be read because of which a lot of time is wasted. It is achieved using functions like fprintf(), fscanf(), getc(), putc(), etc.

2) Binary files or Random access files: Text mode is inefficient for storing large amount of numerical data because it occupies large space. Only solution to this inefficient memory use is to open a file in binary mode, which takes lesser space than the text mode. These files contain the set of bytes. Byte and character are equivalent in c language hence a text file is no different than the binary file. In this case the operations on the file can be sequence or random.

Different modes of opening a binary file are

rb The existing binary file is opened to the beginning for reading.

wb The existing binary file is opened to for writing.

ab The existing binary file is opened to the end for apending..

rb+ The existing binary file is opened to the beginning for both reading and writing.

wb+ Same as **w** except both for reading and writing.

ab+ Same as **a** except both for reading and writing.

NOTE: In random access file a particular part of a file can be accessed irrespective of other parts. It is

Achieved using the functions like, fseek(), ftell(), and rewind() which are available in io library.

Reading data from file:

Basic operations on file:

1. Naming a file,2. opening a file, 3.Reading data from file, 4. Writing data into file, 5.Closing a File

Naming and opening a file:

A name is given to the file used to store data. The file name is a string of characters that make up a valid file name for operating system. It contains two parts. An optional name and an optional period with extension.

Examples: Student.dat, file1.txt, marks.doc, palin.c.

The general formal of declaring and opening a file is

```
FILE *fp;                    //declaration
fp=fopen ("filename","mode");    //ststement to open file.
```

- Here FILE is data structures defined for files.
- fp is a pointer to data type file. It acts as link between systems and program.
- Filename is the name of the file.
- Mode tells in which mode the file should be opened.

For reading data from file, Input functions used are
(Input and output operations on files)

a) getc(); It is used to read characters from file that has been opened for read operation.
Syntax: C=gets (file pointer)
This statement reads a character from file pointed to by file pointer and assign to c.
It returns an end-of-file marker EOF, when end of file as been reached
b) fscanf(); This function is similar to that of scanf function except that it works on files.
Syntax: fscanf (fp, "control string", list);
Example fscanf(f1,"%s%d",str,&num);
The above statement reads string type data and integer types data from file.
c) getw(); This function reads an integer from file.
Syntax: getw (file pointer);
d) fgets(): This function reads a string from a file pointed by a file pointer. It also copies the string to a memory location referred by an array.
Syntax: fgets(string,no of bytes,filepointer);
e) fread(): This function is used for reading an entire structure block from a given file.
Syntax: fread(&struct_name,sizeof(struct_name),1,filepointer);

Writing data to a file:

To write into a file, following C functions are used

a. putc(): This function writer a character to a file that has been opened in write mode.

Syntax: This statement writes the character contained in character variable c into a file whose pointer is fp.

b. fprintf(): This function performs function, similar to that of printf.

Syntax: fprintf(f1,"%s,%d",str,num);

C. putw(): It writes an integer to a file.

Syntax: putw (variable, fp);

d) fputs(): This function writes a string into a file pointed by a file pointer.

Synatax: fputs(string,filepointer);

e) fwrite(): This function is used for writing an entire block structure to a given file.

Syntax: fwrite(&struct_name,sizeof(struct_name),1,filepointer);

Closing a file:

A file is closed as soon as all operations on it have been completed.

Library function for closing a file is

fclose(file pointer);

Example: fclose(fp); This statement closes a file pointed by file pointer fp.

Or fcloseall(), to close all opened files at a time.

Error handling in files:

It is possible that an error may occur during I/O operations on a file. Typical error situations include:

1. Trying to read beyond the end of file mark.
2. Device overflow
3. Trying to use a file that has not been opened
4. Trying to perform an operation on a file, when the file is opened for another type of operations
5. Opening a file with an invalid filename
6. Attempting to write a write protected file

If we fail to check such read and write errors, a program may behave abnormally when an error occurs. An unchecked error may result in a premature termination of the program or incorrect output.

a) **feof():** the feof() function can be used to test for an end of file condition. It takes a FILE pointer as its only argument and returns a nonzero integer value if all of the data from the specified file has been read, and returns zero otherwise. If fp is a pointer to file that has just opened for reading, then the statement

```
if(feof(fp))
  printf("End of data");
```

would display the message "End of data". On reaching the end of file condition.

b) **ferror():** The ferror() function reports the status of the file indicated. It also takes a FILE pointer as its argument and returns a nonzero integer if an error has been deteched upto that point, during processing. It returns zero otherwise. The statement

```
if(ferror(fp)!=0)
 printf("an error has occurred\n");
```

c) we know that whenever a file is opened using fopen function, a file pointer is returned. If the file cannot be opened for some reson, then the function returns a null pointer. This facility can be used to test whether a file has been opened or not. Example

```
if(fp==NULL)
 printf("File could not be opened.\n");
```

d) **perror():** it is a standard library function which prints the error messages specified by the compiler

example:

```
if(ferror(fp))
   perror(filename);
```

Random access functions:

a) **fseek() function:** it is a file function. It positions file pointer on the stream. We can pass three arguments through this function.

1) file pointer
2) negative or positive long integer number to reposition the file pointer towards the background or forward direction.
3) The current position of the file pointer. Given bellow

Integer value constant		location in the file
0	SEEK_SET	Beginning of the file
1	SEEK_CUR	current position of the file pointer
2	SEEK_END	End of the file

Example: fseek(fp,10,0) OR fseek(fp,10,SEEK_SET)
The file pointer is repositioned in the forward direction 10 bytes
The statement fseek(fp,-10,SEEK_END) to read from backward direction from the end of file.

b) **ftell():** it returns the current position of the file ponter. it returns the pointer from the beginning of file. Syntax: ftell(filepointer);

c) **rewind():** this function resets the file pointer at the beginning of the file.
Syntax: rewind(filepointer);

2Q) Explain the command line arguments. What are the syntactic constructs followed in 'C'.

Command line argument is the parameter supplied to a program when the program is invoked.This parameter may represent a file name the program should process. For example, if we want to execute a program to copy the contents of a file named X_FILE to another one name Y_FILE then we may use a command line like
c>program X_FILE Y_FILE
Program is the file name where the executable code of the program is stored. This eliminates the need for the program to request the user to enter the file names during execution.
The 'main' function can take two arguments called argc, argv and information contained in the command line is passed on to the program to these arguments, when 'main' is called up by the system. The variable argv is an argument vector and represents an array of characters pointers that point to the command line arguments. argc is an argument counter that counts

the number of arguments on the command line. The size of this array is equal to the value of argc. In order to access the command line arguments, we must declare the
'main' function and its parameters as follows:

```
main(argc,argv)
int argc;
char *arg[ ];
{
…….
…….
}
```

Generally arg[0] represents the program name.

Example:1. write a program to copy the contents of one file into another? Or demonistrate command line arguments

```
Void main(int argc,char* argv[]) /* command line arguments */
 {
   FILE *ft,*fs; /* file pointers declaration*/
   Int c=0;
   If(argc!=3)
      Printf("\n insufficient arguments");
/* opening files*?
   Fs=fopen(argv[1],"r");
   Ft=fopen(argv[2],"w");
/*error handling*/
 If (fs==NULL)
   {
     Printf("\n source file opening error");
     Exit(1) ;
   }
If (ft==NULL)
   {
     Printf("\n target file opening error");
```

```
    Exit(1) ;
  }
/*Reading file and writing into second file*/
 While(!feof(fs))
  {
        Fputc(fgetc(fs),ft);
        C++;
    }
Printf("\n bytes copied from %s file to %s file =%d",argv[1].argv[2],c);
C=Fcloseall(); /*closing all files*/
Printf("files closed=%d",c);
}
```

2.write a program to open a pre existing file and add information at the end of the file. Display the contents of the file before and after appending.

```
Void main( )
 {
  FILE *ft; /* file pointers declaration*/
  char c;
printf("displaying the contents of the file before appending");
   Ft=fopen("ram.txt","r");
If (ft==NULL)
  {
    Printf("\n target file opening error");
    Exit(1) ;
  }
While(!feof(fs))
 {
    C=fgetc(ft);
    Printf("%c",c);
    }
fclose(ft);
```

```
/*opening the file for appending data*/
    Ft=fopen(“ram.txt”,”a”);
/*appending data*/
While(c!=’.’)
 {
 C=getche();
 Fputc(c,ft);
}
Fclose(ft);
Printf(“ displaying the contents of the file after appending */
    Ft=fopen(“ram.txt”,”r”);
While(!feof(fs))
   {
     C=fgetc(ft);
     Printf(“%c”,c);
     }
fclose(ft);
}
```

3. Write a C program to read data from the keyboard, write it to a file called INPUT, again read the same data from the INPUT file, and display it on the screen. OR write a program to display the contents of a given file file1 on to a standard output.

```
#include<stdio.h>
main()
 {
   Char ch;
   FILE *fp;
    clrscr();
   fp=fopen[(“Input”,‘‘w”);
 If(fp==NULL)
       {
```

```
        printf("file is not opened");
       exit(0);
       }
   while((ch=getchar())!=EOF)
          {
          putc(ch,fp);
          }
 fclose(fp);
 fp=fopen("Input",''r");
 while((ch=getc(fp)!=EOF)
 printf("%c",ch)
fclose(fp);
}
```

4) write a program for indexed sequential file for the employee database for the following operation: a) Add record b) Delete Record c) Search Record based on the department

```
Typedef struct
 {
   Unsigned int empno;
   Char name[50];
   Int salary;
 }st_rec;
Void main()
 {
Int addrecord();
Int removerecord();
Iint seachrecord();
 Int ch;
St_rec rec;
FILE *fp;
Fp=fopen("employee.rec","wb");
```

```
Printf("\n enter the empno,name and salary");
/* read from key board write to a file, press ctrl+z to stop");
While(scanf("%u%c%d",rec.empno,rec.name,&rec.salary)!=EOF)
 Fwrite(&rec,sizeof(rec),1,fp);
Fclose(fp);
Printf("\n");
/* read from file and write to a screen*/
Fp=fopen("employee.rec","rb");
While(fread(&rec,sizeof(rec),1,fp)   /* while loop terminates when fread returns 0 */
 Printf("%5u%-10c%3d\n",rec.empno,rec.name,rec.salary);
Printf( "a) Add record b) Delete Record c) Search Record based on the department ");
Printf("enter ur choice");
Scanf("%d",&ch);
Switch(ch)
 {
 Case 'a': int addrecord()
                    {
                      Printf("\n enter the empno,name and salary");
                         /* read from key board write to a file, press ctrl+z to stop");
                       While(scanf("%u%c%d",rec.empno,rec.name,&rec.salary)!=EOF)
                     Fwrite(&rec,sizeof(rec),1,fp);
                    }
                    Fclose(fp);
                  Break;
 Case 'b': int removerecord(const char * employeerecord.index);
              Break;
 Case 'c': int searchrecord()
               /* read from file and write to a screen*/
              Fp=fopen("employee.rec","rb");
                While(fread(&rec,sizeof(rec),1,fp)   /* while loop terminates when fread returns 0
*/
                Printf("%5u%-10c%3d\n",rec.empno,rec.name,rec.salary);
               Fclose(fp);
```

```
            }
 }
Fclose(fp);
}
```

5) write a program to read the following data, to find the value of each item and display the contents of the file.

Item code price quantity

Pen 101 RS.20 5

Pencil 103 Rs.3 100

```
void main()
 {
 FILE *fp;
 int code,price,quantity,I;
 char item[10],filename[10];
 float value;
 printf("\n enter file name");
 scanf("%s",filename);
 fp=fopen(filename, "w");
 printf(" enter data into file");
 printf("item
   code
price
            quantity");
 for(i=1;i<=2;i++)
    {
     fscanf(stdin,"%s %d %d %d",item,&code,&price,&quantity");
     fprintf(fp,"%s %d %2d %d ", item,code,price,quantity);
    }
 fclose(fp);
 fprintf(stdout,"\n\n");
 fp=fopen(filename,"r");
```

```
 printf(item number price quantity value\n");
 for(i=1;i<=2;i++)
   {
     fscanf(fp,"%s %d %d %d",item,&code,&price,&quantity");
      value=price*quantity;
     fprintf(stdout,"%s %d %2d %d ", item,code,price,quantity);
    }
Fclose(fp);
}
```

6) write a program to read last n characters from a file?

```
Void main()
 {
    FILE *fp;
    int n,ch;
    fp=fopen("ram.txt","r");
    printf("\n contents of file");
   while(ch=fgetc(fp)!=EOF)
     printf("\n contents of file");
printf("how many characters u want to skip");
scanf("%d",&n);
fseek(fp,n,SEEK_END);
printf("last %d characters of a file are",n);
while(ch=fgetc(fp)!=EOF)
     printf("\n contents of file");
fclose(fp);
}
```

6) write a program to count no of words, characters, lines in a file.

Experiment 59

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

```
 FILE *fs,*ft;
 int c=0,w=1,s=0;
 char ch;
 clrscr();
 fs=fopen("a.txt","r");
 ft=fopen("b.txt","w");
 while(!feof(fs))
 {
        ch=fgetc(fs);
        if(ch>64&&ch<92||ch>97&&ch<123)
        c++;
        else if(ch==32||ch>46&&ch<57)
        w++;
        else if(ch==46||ch==44||ch==63)
        s++;

 }
 fprintf(ft,"The File a.txt contains %d Characters\nIt contains %d words.And %d
Sentences",c,w,s);
 fcloseall();

 getch();
 return 0
}
```