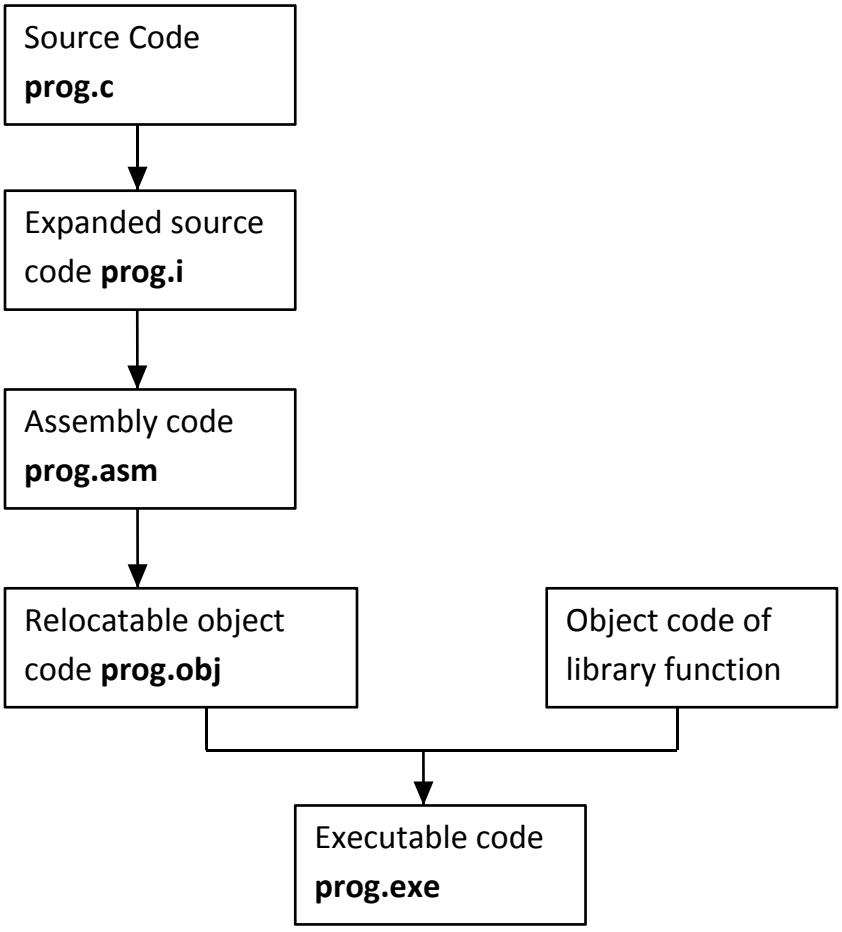
C PREPROCESSOR

- Preprocessor is exactly what its name implies.
- It is a program that process the source program before it is passed to the compiler.
- Preprocessor contains certain commands in it, to process the source code. These commands are known as **"DIRECTIVE"**.
- All these commands together, can be considered as a language with in c- language.
- Without having knowledge about preprocessor , all c- programmers depend on it. **This feature does not exist in other higher level languages.**



Each of the preprocessor directive begins with a # symbol.

There directives can be placed any where in the program, but generally these are written in the beginning of the program, before main() or before a particular function.

Some of the preprocessor directives are:

1. #include directive (file inclusion)
2. #define directive (macro substitution)
3. #undef directive
4. Conditional compilation directives

i) #define directive:

```
#define RANGE 10
main()
{
int k;
for(k=1;k<=RANGE; k++)
{
printf("k=%d\n",k);
}
}
```

Value is constant throughout the program.

In the above program, instead of writing 10, in for loop we are writing as RANGE

RANGE is also defined before main() using statement.

#define RANGE 10
The above statement is called as **MACRO DEFINATION**
During preprocessing, preprocessor replaces, each occurrence of word RANGE with 10.

```
#define A 25
main()
{
int i,j;
printf("enter i value");
scanf("%d",&i);
j=A*i;
printf("product=%d",j);
}
```

Here ; Range & A are called as “Macro templates”.
10 & 25 are called as “Macro Expansions”.

When we compile the program, before source code passes to compiler, it is evaluated by preprocessor.

- Preprocessors checks for the Macro definitions.
- According to the Macro definition given, preprocessor searches the entire for “Macro templates”.
- Whenever, it finds these “Macro templates”, preprocessor replaces them with “Macro expansions”.

- After completion of this procedure, program will be sent to compiler.

```
#define VALUE 3+5
main()
{
int i;
clrscr();
i=VALUE * VALUE;
printf("%d",i);
}
```

i=3+5*3+5
i=3+15+5
i=23

Reason: * has priority over +
First '*' is performed, and then '+' is performed.

NOTE:

(i) Generally, capital letters are used for writing Macro templates. (Small letters can also be used)

(ii) Macro template & Macro expansion are separated by space.

```
#define   RANGE   10
            ↓       ↓
          M.T     M.E
```

MACROS WITH ARGUMENTS:

Macros can have arguments, just like functions

```
#define AREA(x) (3.14*x*x)
main()
{
float i1=2.2,i2=3.4,j;
clrscr();
j=AREA(i1);
printf("Area=%f",j);
j=AREA(i2);
printf("Area=%f",j);
}
```

Preprocessors will replace AREA(x) with 3.14*x*x
Also i1&i2 will be substituted, based on function calls
J=AREA(i1) -> when this function call is executed, i1 is assigned to x.

J=AREA(i2) ->i2 will be assigned to x.

Difference between Macro and Functions:

In a Macro cal, preprocessor replaces, Macro template with its Macro expansion. Macro won't return any value to main().

In a function call, the control is passed to a function, with certain arguments and some value will be returned back to the calling function.

Macros makes programs to run faster, but increases the size of program, whereas, functions decreases the size of program.

```
#define AREA(x) (3.4*x*x)
main()
{
float i=2.2,t;
clrscr();
t=AREA(i);
printf("Area=%f",t);
}
```

1. Preprocessor will replace AREA(x) with 3.14*x*x
2. When function call AREA(i) is made, then i is assigned to x and result is calculated as 3.14*i*i

ii) File Inclusion:

- **It is a process of inserting external files, containing functions into the c-program**
- **This avoids re-writing those functions in the program.**

Suppose we have to use same function in 5 diff programs, then file inclusion technique can be used.

An external file can be loaded into the c- program using #include directive.

General form of file inclusion is:

```
#include filename
```

#include → preprocessor directive for file inclusion

Filename → name of the file containing the required function define to be included in a c-program.

If filename is declared as: #include<filename>

then, the specified file is searched only in standard directories.

If filename is declared as: #inlcude "filename"

then, the specified file is first searched in the current directories and then in the standard directories.

iii) #undef Directive

On some occasions it may be desirable to cause a defined name to become 'undefined'. This can be accomplished by means of the #undef directive. In order to undefined a macro which has been earlier #defined, the directive,

```
#undef macro-template
```

can be used.

EXAMPLE: #define PI 3.1414

```
Void main()
{
float a,r=3;
clrscr();
 a=PI*3*3;
 printf("AREA = %d",a);
 #undef PI
 a=PI*3*3;  /* Gives error undefined PI*/
  printf("Area =%d",a);
  getch();
}
```

For this program it gives error because the macro PI is undefined. After undefining macro we are using that macro in the program therefore it gives error.

iv) Conditional Compilation:

It is a process of selecting a particular segment of code for compilation based on condition.

```
main()
{
#define one
#ifdef 0one
printf("hi");
printf("hellow");
#else
printf("c-lab");
#endif
printf("welcome");
}
```

Output:
i) hi hellow welcome
ii) c-lab welcome (if #define is commented)

- c- preprocessor provides a compilation directive which is used to select alternate segment of code depending upon the condition.
- If there are 2 different versions of a program, it will take more memory to store the program. This problem can be solved by including both version in a single program.
- Then a particular version can be selected based on the requirement.
- In large programs, generally programmers writes comments for each and every statement. If we use comment, instead of conditional compilation directives, we would end up with nested comments. Sometimes, it is not possible to write nested comments in C.

Function Returning Non-integers:

C function returns a value of type ‘int’ as a default value, when no return type is specified.

Two steps must be performed ,to enables a calling function, to receive a non-integer value from the called function:

1. Return-type of the function should be specified in the function header.
2. The called function must be declared at the start of body, in the calling function, like any other variable. This is to tell the calling function, the type of data, that the function is actually returing,.

```
float m(float x, float y);
main()
{
float a,b, mul();
a=12.34;
b=7.32;
printf("%f", mul(a,b));
}
float mul(float x, float y)
{
return x*y;
}
```

Files in C

Generally, a file is used to store user data in a computer. In other words, computer stores the data using files. we can define a file as follows...

File is a collection of data that stored on secondary memory like haddisk of a computer.

C programming language supports two types of files and they are as follows...

- **Text Files (or) ASCII Files**
- **Binary Files**

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

Binary File - The file that contains data in the form of bytes (0's and 1's) is called as binary file. Generally, the binary files are compiled version of text files.

File Operations in C

The following are the operations performed on files in c programming langauge...

- **Creating (or) Opening a file**
- **Reading data from a file**
- **Writing data into a file**
- **Closing a file**

All the above operations are performed using file handling functions available in C.

File Handling Functions in C

File is a collection of data that stored on secondary memory like hard disk of a computer.

The following are the operations performed on files in the c programming language...

- **Creating (or) Opening a file**
- **Reading data from a file**
- **Writing data into a file**
- **Closing a file**

All the above operations are performed using file handling functions available in C.

Creating (or) Opening a file

To create a new file or open an existing file, we need to create a file pointer of FILE type.

Following is the sample code for creating file pointer.

```
File *f_ptr ;
```

We use the pre-defined method **fopen()** to create a new file or to open an existing file. There are different modes in which a file can be opened. Consider the following code...

```
File *f_ptr ;

*f_ptr = fopen("abc.txt", "w") ;
```

The above example code creates a new file called **abc.txt** if it does not exists otherwise it is opened in writing mode.

In C programming language, there different modes are available to open a file and they are shown in the following table.

S. No.	Mode	Description
1	r	Opens a text file in **reading** mode.
2	w	Opens a text file in **wirting** mode.
3	a	Opens a text file in **append** mode.
4	r+	Opens a text file in both **reading and writing** mode.
5	w+	Opens a text file in both **reading and writing** mode. It set the cursor position to the begining of the file if it exists.
6	a+	Opens a text file in both **reading and writing** mode. The reading operation is performed from begining and writing operation is performed at the end of the file.

Note - The above modes are used with text files only. If we want to work with binary files we use

rb, wb, ab, rb+, wb+ and ab+.

Reading from a file

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

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

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

Example Program to illustrate getc() in C.

```
#include<stdio.h>
#include<conio.h>
int main(){
   FILE *fp;
   char ch;
   clrscr();
   fp = fopen("MySample.txt","r");
   printf("Reading character from the file: %c\n",getc(fp));
   ch = getc(fp);
   printf("ch = %c", ch);
   fclose(fp);
   getch();
   return 0;
}
```

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

```
#include<stdio.h>
#include<conio.h>
int main(){
   FILE *fp;
   int i,j;
```

```
   clrscr();
   fp = fopen("MySample.txt","w");
   putw(65,fp);  // inserts A
   putw(97,fp);  // inserts a
   fclose(fp);
   fp = fopen("MySample.txt","r");
   i = getw(fp);   // reads 65 - ASCII value of A
   j = getw(fp);   // reads 97 - ASCII value of a
   printf("SUM of the integer values stored in file = %d", i+j);  // 65 + 97 = 162
   fclose(fp);
   getch();
   return 0;
}
```

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

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

   return
}
```

- ***fgets(variableName, numberOfCharacters, *file_pointer) -*** This method is used for reading a set of characters from a file which is opened in reading mode starting from the current cursor position. The fgets() function reading terminates with reading NULL character.

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

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

```
#include<stdio.h>
#include<conio.h>
int main(){
   FILE *fp;
   char *str;
   clrscr();
   fp = fopen ("file.txt", "r");
   fread(str,sizeof(char),5,fp);
   str[strlen(str)+1] = 0;
   printf("str = %s", str);
   fclose(fp);
   getch();
   return 0;
}
```

Writing into a file

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

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

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

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

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

```
#include<stdio.h>

#include<conio.h>

int main(){

   FILE *fp;
   int i;
   clrscr();
   fp = fopen("MySample.txt","w");
   putw(66,fp);
   i = 100;
   putw(i,fp);
   fclose(fp);
   getch();
   return 0;
}
```

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

```
#include<stdio.h>
#include<conio.h>
int main(){
   FILE *fp;
   char *text = "\nthis is example text";
   int i = 10;
   clrscr();
   fp = fopen("MySample.txt","w");
   fprintf(fp,"This is line1\nThis is line2\n%d", i);
   fprintf(fp,text);
   fclose(fp);
   getch();
   return 0;

}
```

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

```
#include<stdio.h>
#include<conio.h>
int main(){
   FILE *fp;
   char *text = "\nthis is example text";
   clrscr();
   fp = fopen("MySample.txt","w");
   fputs("Hi!\nHow are you?",fp);
   fclose(fp);
   getch();
   return 0;

}
```

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

```
#include<stdio.h>
#include<conio.h>
int main(){
   FILE *fp;
   char *text = "Welcome to C Language";
   clrscr();
   fp = fopen("MySample.txt","wb");
   fwrite(text,sizeof(char),5,fp);
   fclose(fp);
   getch();
   return 0;
}
```

Closing a file

Closing a file is performed using a pre-defined method fclose().

```
fclose( *f_ptr )
```

The method fclose() returns '0'on success of file close otherwise it returns EOF (End Of File).

Cursor Positioning Functions in Files

C programming language provides various pre-defined methods to set the cursor position in files. The following are the methods available in c, to position cursor in a file.

1. **ftell()**
2. **rewind()**
3. **fseek()**

- ***ftell(*file_pointer) -*** This function returns the current position of the cursor in the file.

Example Program to illustrate ftell() in C.

```
#include<stdio.h>
#include<conio.h>
int main(){
   FILE *fp;
   int position;
   clrscr();
   fp = fopen ("file.txt", "r");
   position = ftell(fp);
   printf("Cursor position = %d\n",position);
   fseek(fp,5,0);
   position = ftell(fp);
   printf("Cursor position = %d", position);
   fclose(fp);
   getch();
   return 0;
}
```

rewind(*file_pointer) - This function is used reset the cursor position to the beginning of the file.

Example Program to illustrate rewind() in C.

```
#include<stdio.h>
#include<conio.h>
int main(){
   FILE *fp;
   int position;
   clrscr();
   fp = fopen ("file.txt", "r");
   position = ftell(fp);
   printf("Cursor position = %d\n",position);
   fseek(fp,5,0);
   position = ftell(fp);
```

```
   printf("Cursor position = %d\n", position);
   rewind(fp);
   position = ftell(fp);
   printf("Cursor position = %d", position);
   fclose(fp);
   getch();
   return 0;
}
```

fseek(*file_pointer, numberOfCharacters, fromPosition) - This function is used to set the cursor position to the specific position. Using this function we can set the cursor position from three different position they are as follows.

- from beginning of the file (indicated with 0)
- from current cursor position (indicated with 1)
- from ending of the file (indicated with 2)

Example Program to illustrate fseek() in C.

```
#include<stdio.h>
#include<conio.h>
int main(){
   FILE *fp;
   int position;
   clrscr();
   fp = fopen ("file.txt", "r");
   position = ftell(fp);
   printf("Cursor position = %d\n",position);
   fseek(fp,5,0);
   position = ftell(fp);
   printf("Cursor position = %d\n", position);
   fseek(fp, -5, 2);
   position = ftell(fp);
   printf("Cursor position = %d", position);
   fclose(fp);
   getch();

   return 0;
}
```

FILES

Generally, a file is used to store user data in a computer. In other words, computer stores the data using files. we can define a file as follows...

File is a collection of data that stored on secondary memory like haddisk of a computer.

C programming language supports two types of files and they are as follows...

1)Text Files (or) ASCII Files
2)Binary Files

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

2)Binary File - The file that contains data in the form of bytes (0's and 1's) is called as binary file. Generally, the binary files are compiled version of text files.

File Operations in C

The following are the operations performed on files in c programming langauge...

Creating (or) Opening a file
Reading data from a file
Writing data into a file
Closing a file

1. Introduction to Files in C:

A File is a collection of data stored in the secondary memory. So far data was entered into the programs through the keyboard. So Files are used for storing information that can be processed by the programs. Files are not only used for storing the data, programs are also stored in files. In order to use files, we have to learn file input and output operations. That is, how data is read and how to write into a file.

A Stream is the important concept in C. The Stream is a common, logical interface to the various devices that comprise the computer. So a Stream is a logical interface to a file. There are two types of Streams, Text Stream and Binary Stream. A Text File can be thought of as a stream of characters that can be processed sequentially. It can only be processed in the forward direction.

2. Using Files in C:

To use a file four essential actions should be carried out. These are,

a. Declare a file pointer variable.
b. Open a file using the fopen() function.
c. Process the file using suitable functions.
d. Close the file using the fclose() and fflush() functions.

Declaration of file pointer:

A pointer variable is used to points a structure FILE. The members of the FILE structure are used by the program in various file access operation, but programmers do not need to concerned about them.

FILE *file_pointer_name;

Eg : FILE *fp;

Opening a file

To open a file using the fopen() function. Its syntax is,

FILE *fopen(const char *fname,const char* mode);

const char *fname represents the file name. The file name is like “D:\\501\example.txt”.

Here D: is a drive, 501 is the directory, example.txt is a file name.

const char *mode represents the mode of the file. It has the following values.

Mode	Description
r	Opens a text file in reading mode
w	Opens or create a text file in writing mode
a	Opens a text file in append mode
r+	Opens a text file in both reading and writing mode
w+	Opens a text file in both reading and writing mode
a+	Opens a binary file in reading mode
rb	Opens a binary file in reading mode
wb	Opens or create a binary file in writing mode
Ab	Opens a binary file in append mode
rb+	Opens a binary file in both reading and writing mode
wb+	Opens a binary file in both reading and writing mode
ab+	Opens a binary file in both reading and writing mode

Difference between write mode and append mode is,

Write (w) mode and Append (a) mode, while opening a file are almost the same. Both are used to write in a file. In both the modes, new file is created if it doesn't exists already.

The only difference they have is, when you open a file in the write mode, the file is reset, resulting in deletion of any data already present in the file. While in append mode this will not happen. Append mode is used to append or add data to the existing data of file(if any). Hence, when you open a file in Append(a) mode, the cursor is positioned at the end of the present data in the file.

Closing and Flushing Files

The file must be closed using the fclose() function. Its prototype is ,

*int fclose(FILE *fp);*

The argument fp is the FILE pinter associated with the stream. Fclose() returns 0 on success and -1 on error.

fflush() used when a file's buffer is to be written to disk while still using the file. Use of fflushall() to flush the buffers of all open streams. The prototype of these two functions are,

int flushall(void);

*int fflush(FILE *fp);*

Examples:

/* Program to check whether the file exist or not */

```
void main()
{
        FILE *fp;
        char *x;
        clrscr();
        printf("\n enter the file name : ");
         gets(x);
        fp=fopen(x,"r");
        if(fp==NULL)
        {
                        printf("The file ---%s--- is not found in the present directory",x);
        }
        else
         {
                printf("The file ---%s--- is found in the present directory",x);
         }
```

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

3. Character input / output:

Two functions are used to read the character from the file and write it into another file. These functions are getc() & putc() functions.

Its prototype is,

*int getc(FILE *file_pointer);*

*putc(char const_char, FILE *file_pointer);*

3.1Detecting the end of a file:

When reading from a text-mode file character by character, one can look for the end-of-file character. The symbolic constant EOF is defined in stdio.h as -1, a value never used by a real character.

Eg: while((c=getc(fp)!=EOF). This is the general representation of the End Of the File.

Examples:

/* Read a string into a text file */

```
void main()
{
    FILE *fp;
    char text[80];
    int i;
    clrscr();
    fp=fopen("Sample.txt","w");
    printf("\n Enter the text : ");
    gets(text);
    for(i=0;i<strlen(text);i++)       // Read a stream of charcters
    putc(text[i],fp);
    if(fp!=NULL)
        printf("\n The string copied into a text file ");
    fclose(fp);
    getch();
}
```

4. Working with Text files :

C provides various functions to working with text files. Four functions can be used to read text files and four functions that can be used to write text files into a disk.

These are,

- fscanf() & fprintf()
- fgets() & fputs()
- fgetc() & fputc()
- fread() & fwrite()

The prototypes of the above functions are,

1. int fscanf(FILE *stream, const char *format,list);
2. int getc(FILE *stream);
3. char *fgets(char *str,int n,FILE *fp);
4. int fread(void *str,size_t size,size_t num, FILE *stream);
5. int fprintf(FILE *stream, const char *format,list);
6. int fputc(int c, FILE *stream);
7. int fputs(const char *str,FILE *stream);
8. int fwrite(const void *str,size_t size, size_t count,FILE *stream);

Example programs on text files:

/* Read a string into a text file using fputs() function */

```
void main()
{
        FILE *fp;
        char text[80];
        int i;
        clrscr();
        fp=fopen("Sample.txt","w");
        printf("\n Enter the text : ");
        gets(text);
        fputs(text,fp);                    //write a string into a file
        if(fp!=NULL)
                printf("\n The string copied into a text file ");
        fclose(fp);
        getch();
}
```

/* Program to copy from one file to another file */

```
void main()
{
        FILE *fp,*fp1;
        int ch;
        clrscr();
        fp=fopen("ex_file4.c","r");
        fp1=fopen("ex_file2.c","w");
        if(fp==NULL)
                printf(“File is not available”);
        ch=getc(fp);
        while(ch!=EOF)
        {
                putc(ch,fp1);
                ch=getc(fp);
        }
        fclose(fp);
        fclose(fp1);
        getch();
}
```

/* Program to display the content of the file */

```
void main()
{
        FILE *fp;
        int ch;
        clrscr();
        fp=fopen("ex_file4.c","r");
        if(fp==NULL)
                printf(“File is not available”);
        ch=getc(fp);
        while(ch!=EOF)
        {
                printf("%c",ch);
                ch=getc(fp);
        }
        fclose(fp);
        getch();
}
```

/* read and write the content of the file using fprintf() and fscanf() functions */

```
void main()
{
    FILE *fptr;
    char name[20];
    int age;
    float salary;
    clrscr();
    /* open for writing */
 fptr = fopen("abc.txt", "w");
    if (fptr == NULL)
    {
            printf("File does not exists \n");
            return;
    }
    printf("Enter the name \n");
    scanf("%s", name);
    fprintf(fptr, "Name = %s\n", name);
    printf("Enter the age\n");
    scanf("%d", &age);
    fprintf(fptr, "Age = %d\n", age);
    printf("Enter the salary\n");
    scanf("%f", &salary);
    fprintf(fptr, "Salary = %f\n", salary);
    getch();
    fclose(fptr);
}
```

5. Working with binary files

A Binary file is similar to the text file, but it contains only large numerical data. The Opening modes are mentioned in the table for opening modes above.

Following steps are sued for copying a binary file into another are as follows,

1. open the source file for reading in binary mode.
2. Open the destination file for writing in binary mode.
3. Read a character from the source file. Remember, when a file is first opened, the pointer is at the start of the file, so there is no need to position the file pointer explicitly.
4. If the function feof() indicates that the end of thesource file has been reached, then close both files and return to the calling program.
5. If end-of- file has not been reached, write the character to the destination file, and then go to step 3.

Example:

```
/* program to copy the binary file from another file      */
void main()
{
        FILE *fp,*fp1;
        char *s,*s1;
        int ch;
        clrscr();
        printf("\n enter the source file : ");
        gets(s);
        printf("\n enter the destination file : ");
        gets(s1);
        fp=fopen(s,"rb");
        fp1=fopen(s1,"wb");
        ch=fgetc(fp);
        while(!feof(fp))
        {
                fputc(ch,fp1);
                ch=fgetc(fp);
        }
        fclose(fp);
        fclose(fp1);
        getch();
}
```

Sequential versus Random File Access

Every open file has an associated file position indicator, which describes where read and write operations take place in the file. The position is always specified in bytes from the beginning of the file. When a new file is opened, the position indicator is always at the beginning of the file, i.e., at position 0. Because the file is new and has a length of 0, there is no other location to indicate. When an existing file is opened, the position indicator is at the end of the file if the file was opened in the append mode, or at the beginning of the file if the file was opened in any other mode.

Sequential files are generally used in cases where the program processes the data in a sequential fashion – i.e. counting words in a text file – although in some cases, random access can be feigned by moving backwards and forwards over a sequential file.

True random access file handling, however, only accesses the file at the point at which the data should be read or written, rather than having to process it sequentially. A hybrid approach is also possible whereby a part of the file is used for sequential access to locate something in the random access portion of the file, in much the same way that a File Allocation Table (FAT) works.

6. Random Access to files of records

For random access to files of record, the following functions are used.

1. fseek() -- by using fseek(), one can set the position indicator anywhere in the file. It's prototype is,

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

It has the origins as, SEEK_SET, SEEK_CUR,SEEK_END

2. ftell() --- To determine the value of a file's position indicator, use ftell(). The prototype is,

```
long ftell(FILE *fp);
```

3. rewind() ------ To set the position indicator to the beginning of the file, use the function rewind(). Its prototype is,

```
void rewind(FILE *fp);
```

Example:

/* program to copy the content from one file to another file using fseek() function. */

```
void main()
{
   /*
   File_1.txt is the file with content and,
   File_2.txt is the file in which content of File_1
   will be copied.
   */
   FILE *fp1, *fp2;
   char ch;

   int pos;
   clrscr();
   if ((fp1 = fopen("File_1.txt","r")) == NULL)
   {
         printf("\nFile cannot be opened");
         return;
   }
   else
   {
         printf("\nFile opened for copy...\n ");
   }
```

```
    fp2 = fopen("File_2.txt", "w");
    fseek(fp1, 0L, SEEK_END); // file pointer at end of file
    pos = ftell(fp1);
    fseek(fp1, 0L, SEEK_SET); // file pointer set at start
while (pos--)
    {
        ch = fgetc(fp1); // copying file character by character
        fputc(ch, fp2);
    }
    getch();
    fcloseall();
}
```

Files in C

Computer stores the data using files. File is a collection of data that is stored on secondary memory like Hard disk of a computer or we can define a file as follows...

In C programming, file is a place on your physical disk where information is stored

Why files are needed?

- When a program is terminated, the entire data is lost. Storing in a file will preserve your data even if the program terminates.
- If you have to enter a large number of data, it will take a lot of time to enter them all. However, if you have a file containing all the data, you can easily access the contents of the file using few commands in C.
- You can easily move your data from one computer to another without any changes.

C programming language supports two types of files and they are as follows...

- **Text Files (or) ASCII Files**
- **Binary Files**

Text File (or) ASCII File –

- **Text files are the normal .txt files that you can easily create using Notepad or any simple text editors.**
- **When you open those files, you'll see all the contents within the file as plain text. You can easily edit or delete the contents.**
- **They take minimum effort to maintain, are easily readable, and provide least security and takes bigger storage space.**
- **The file that contains ASCII codes of data like digits, alphabets and symbols is called text file (or) ASCII file.**

Binary File –

- **Binary files are mostly the .bin files in your computer.**
- **Instead of storing data in plain text, they store it in the binary form (0's and 1's).**
- **They can hold higher amount of data, are not readable easily and provides a better security than text files.**
- **Hence a file that contains data in the form of bytes (0's and 1's) is called as binary file. Generally, the binary files are compiled version of text files.**

Working with files

When working with files, you need to declare a pointer of type file. This declaration is needed for communication between the file and program.

```
FILE *fptr;
```

File Operations in C

The following are the operations performed on files in c programming language...

- **Creating (or) Opening a file**
- **Reading data from a file**
- **Writing data into a file**
- **Closing a file**

All the above operations are performed using file handling functions available in C.

File Handling Functions in C

Creating (or) Opening a file

To create a new file or open an existing file, we need to create a file pointer of FILE type.

Following is the sample code for creating file pointer.

FILE *fptr ;

Opening a file is performed using the library function in the "stdio.h" header file: fopen().

The syntax for opening a file in standard I/O is:

```
fptr = fopen("fileopen","mode")
```

For Example:

```
fptr=fopen("D:\\cprograms\\Sample.txt ","w")
```

Note: We use the pre-defined method **fopen()** to create a new file or to open an existing file. There are different modes in which a file can be opened. Consider the following code...

The above example code creates a new file called **Sample.txt** if it does not exists otherwise it is opened in writing mode.

In C programming language, there different modes are available to open a file and they are shown in the following table.

S. No.	Mode	Description
1	r	Opens a text file in **reading** mode. If the file does not exist, fopen() returns NULL
2	w	Opens a text file in **writing** mode. If the file exists, its contents are overwritten. If the file does not exist, it will be created.
3	a	Opens a text file in **append** mode . i.e, Data is added to end of file. If the file does not exists, it will be created.
4	r+	Opens a text file in both **reading and writing** mode. If the file does not exist, fopen() returns NULL
5	w+	Opens a text file in both **reading and writing** mode. It set the cursor position to the beginning of the file if it exists. If the file exists, its contents are overwritten. If the file does not exist, it will be created.
6	a+	Opens a text file in both **reading and writing** mode. The reading operation is performed from beginning and writing operation is performed at the end of the file. If the file does not exist, it will be created.

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

Reading from a file	Writing into a file
The reading from a file operation is performed using the following pre-defined file handling method. 1. fgetc() 2. getw() 3. fscanf() 4. fgets() 5. fread()	The writing into a file operation is performed using the following pre-defined file handling methods. 1. fputc() 2. putw() 3. fprintf() 4. fputs() 5. fwrite() .

- ***getc(*file_pointer) -*** This function is used to read a character from specified file which is opened in reading mode. It reads from the current position of the cursor. After reading the character the cursor will be at next character.
- ***putc(char, *file_pointer) -*** This function is used to write/insert a character to the specified file when the file is opened in writing mode.

Example Program to illustrate getc() in C.

```
#include<stdio.h>
int main()
{
FILE *fp;
char ch;
fp=fopen("one.txt","r");
printf("Reading the character \n");
ch=fgetc(fp);
printf("Printing the character ");
printf("ch=%c",ch);
fclose(fp);
return(0);
}
```

Example Program to illustrate putc() in C.

```
#include<stdio.h>
int main()
{
FILE *fp;
char ch;
fp=fopen("first.txt","w");
fputc('X',fp);
ch='Y';
fputc(ch,fp);
fclose(fp);
return(0);
}
```

- ***getw(*file_pointer) -*** This function is used to read an integer value form the specified file which is opened in reading mode. If the data in file is set of characters then it reads ASCII values of those characters.
- ***putw(int, *file_pointer)*** – This function is used to writes/inserts an integer value to the specified file when the file is opened in writing mode.

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

- ***fscanf(*file_pointer, typeSpecifier, &variableName)*** – This function is used to read multiple datatype values from specified file which is opened in reading mode.
- ***fprintf(*file_pointer, "text")*** – This function is used to writes/inserts multiple lines of text with mixed data types (char, int, float, double) into specified file which is opened in writing mode.

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
  FILE *fp;
  int age;
  char name[20];
  float marks;
  fp=fopen("info.txt","w+");
  if(fp==NULL)
  {
    printf("ERROR!");
    exit(1);
  }
  else
  {
    printf("Enter name age marks \n");
    scanf("%s %d %f \n",name,&age,&marks);
    fprintf(fp,"%s %d %f",name,age,marks);
    fclose(fp);
    fp=fopen("info.txt","r+");
    printf("Entered  data is \n");
    fscanf(fp,"%s %d %f \n",name,&age,&marks);
    printf("\n Name=%s  Age=%d Marks=%f \n",name,age,marks);
    fclose(fp);
  }
}
```

- ***fgets(variableName, numberOfCharacters, *file_pointer) -*** This method is used for reading a set of characters from a file which is opened in reading mode starting from the current cursor position. The fgets() function reading terminates with reading NULL character.
- ***fputs("string", *file_pointer)*** - This method is used to insert string data into specified file which is opened in writing mode.

```
#include<stdio.h>
void main()
{
char name[60];
int i;
FILE *fptr;
fptr=fopen("string.txt","w");
printf("enter the text \n");
gets(name);
fputs(name,fptr);
fclose(fptr);
fptr=fopen("string.txt","r");
if((fgets(name,30,fptr)!=NULL)
   {
       while(name[i]!='\0')
    {
        putchar(name[i]);
        i++;
    }
   }
   fclose(fptr);
}
```

Binary files:

Text mode is inefficient for storing large amount of numerical data because it occupies large space. Only solution to this is to open a file in binary mode, which takes lesser space than the text mode. These files contain the set of bytes which stores the information in binary form. One main drawback of binary files is data is stored in human unreadable form. Examples of binary files are .exe files, video stream files, image files etc. C language supports binary file operations with the help of various inbuilt functions.

Opening Modes in Standard I/O		
File Mode	Meaning of Mode	During Inexistence of file
rb	Open for reading in binary mode.	If the file does not exist, fopen() returns NULL.
wb	Open for writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
ab	Open for append in binary mode. i.e, Data is added to end of file.	If the file does not exists, it will be created.
rb+	Open for both reading and writing in binary mode.	If the file does not exist, fopen() returns NULL.
wb+	Open for both reading and writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
ab+	Open for both reading and appending in binary mode.	If the file does not exists, it will be created.

Working with binary files

A Binary file is similar to the text file, but it contains only large numerical data. The Opening modes are mentioned in the table above.

Reading and writing to a binary file

Large amount of integers or float data require large space on disk in text mode and turns out to be inefficient. For this we prefer binary mode and the Functions used are fread() and fwrite() for reading from and writing to a file on the disk respectively in case of binary files.

Writing to a binary file

To write into a binary file, you need to use the function fwrite(). The functions takes four arguments: Address of data to be written in disk, Size of data to be written in disk, number of such type of data and pointer to the file where you want to write.

fwrite(address_data,size_data,numbers_data,pointer_to_file)- This function is used to insert specified number of characters into a binary file which is opened in writing mode

fwrite(): It is used for writing an entire structure block to a given file in binary mode.

Syntax:

fwrite(&structure variable,sizeof(structure variable),1,filepointer);

Example:

fwrite(&stud,sizeof(stud),1,fp);

Reading from a binary file

Function fread() also take 4 arguments similar to fwrite() function as below.

fread(address_data, size_data,numbers_data, pointer_to_file)- This function is used to read specific number of sequence of characters from the specified file which is opened in reading mode.

fread():It is used for reading an entire structure block from a given file in binary mode.

Syntax: fread(&structure variable, sizeof(structure variable),1,filepointer);

Example: fread(&emp,sizeof(emp),1,fp1);

The general format of declaring and opening a file is

FILE *fp; //declaration

fp=fopen (“filename”,”mode”); //statement to open file.

Note: Here FILE is a data structure defined for files. fp is a pointer to data type FILE.

Filename is the name of the file. Examples: Student.dat, file1.txt, marks.doc, palin.c.

Mode tells the purpose of opening this file.

Writing to a binary file

Example-1

```
// To demonstrate Binary files handling
#include<stdio.h>
#include<stdlib.h>
struct three_nums
  {
     int n1, n2, n3;
  };
int main()
  {
     int n;
     struct three_nums num;
     FILE *fp;
     fp = fopen("binary.bin","wb");
     if(fp==NULL)
     {
        printf("ERROR READING INPUT FILE");
        exit(1);
     }
     for(n=1; n<5; ++n)
     {
        num.n1 = n;
        num.n2 = 5*n;
        num.n3 = 5*n+1;
        fwrite(&num, sizeof(struct three_nums), 1, fp);
     }
     fclose(fp);
     printf("\nReading from File\n");
     fp = fopen("binary.bin", "rb");
     for(n=1; n<5; ++n)
     {
        fread(&num, sizeof(struct three_nums), 1, fp);
        printf("n1 = %d \t n2 = %d \t n3 = %d \n", num.n1, num.n2, num.n3);
     }
     fclose(fp);
     return(1);
  }
```

Example-2

```
// To demonstrate Binary files handling
#include<stdio.h>
struct player
 {
   char pname[30];
    int age;
  int runs;
 };
 void main()
 {
  struct player p1,p2;
  FILE *f3;
  f3=fopen("player.exe","wb");
  printf("\n Enter details of player name ,age and runs scored :\n ");
  //fflush(stdin);
  scanf("%s %d %d",p1.pname,&p1.age,&p1.runs);
  fwrite(&p1,sizeof(p1),1,f3);
  fclose(f3);
  f3=fopen("player.exe","r");
  fread(&p2,sizeof(p2),1,f3);
  fflush(stdout);
  printf("\nPLAYERNAME:=%s\tAGE:=%d\tRUNS:=%d,p2.pname,p2.age,p2.runs);
  fclose(f3);
 }
```

Closing a File

The file (both text and binary) should be closed after reading/writing.

Closing a file is performed using library function fclose().

fclose(fptr); //fptr is the file pointer associated with file to be closed

Where fptr is the file pointer returned by the call to fopen(). fclose() returns 0 on success (or) -1 on error. Once a file is closed, its file pointer can be reused for another file.

Note: fcloseall() can be used to close all the opened files at once.

Cursor Positioning Functions in Files
or
Random access to files

C programming language provides various pre-defined methods to set the cursor position in files. The following are the methods available in c, to position cursor in a file .

1) **ftell()**
2) **rewind()**
3) **fseek()**

At times we needed to access only a particular part of a file rather than accessing all the data sequentially, which can be achieved with the help of functions fseek, ftell and rewind available in IO library.

- ***ftell(*file_pointer) -*** This function returns the current position of the cursor in the file.

Syntax: n**=ftell(fp);** n would give the Relative offset (In bytes) of the current position. This means that already **n** bytes have been read or written

Example Program to illustrate ftell() in C.

```
// To demonstrate fseek()
#include<stdio.h>
#include<stdlib.h>
int main()
  {
      FILE *fp;
      int size;
      fp = fopen("ftell.txt","r");
      if(fp==NULL)
      {
          printf("ERROR READING INPUT FILE");
          exit(1);
      }
      fseek(fp,0,SEEK_END);
      size  = ftell(fp);
      printf("The size of the file is %d bytes\n", size);
```

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

- ***rewind(*file_pointer) -*** This function is used to reset the cursor position to the beginning of the file.

rewind():- It takes a file pointer and resets the position to the start of the file.

Syntax: rewind(fp);
n=ftell(fp);

Would assign 0 to n because the file position has been set to start of the file by rewind(). The first byte in the file is numbered 0, second as 1, so on. This function helps in reading the file more than once, without having to close and open the file. Whenever a file is opened for reading or writing a rewind is done implicitly.

Example Program to illustrate rewind() in C.

```
// To demonstrate rewind()
#include<stdio.h>
#include<stdlib.h>
int main()
  {
      FILE *fp;
      char c;
      fp = fopen("rewind.txt","at+");
      if(fp==NULL)
      {
          printf("ERROR READING INPUT FILE");
          exit(1);
      }
      printf("Enter some data to write into the file rewind.txt\n Enter z to end\n");
      while((c=getchar())!= 'z')
        fputc(c,fp);
        rewind(fp);
        while((c=getc(fp))!= EOF)
        printf("%c", c);
        fclose(fp);
        return(1);
```

```
}
```

- ***Getting data using fseek()***
  - If you have many records inside a file and need to access a record at a specific position, you need to loop through all the records before it, to get the record.
  - This will waste a lot of memory and operation time. An easier way to get to the required data can be achieved using fseek().

As the name suggests, fseek() seeks the cursor to the given record in the file.

Syntax of fseek()

fseek(FILE * stream, long int offset, int whence)- This function is used to set the cursor position to the specific position. Using this function we can set the cursor position from three different position they are as follows.

fseek:- fseek function is used to move the file pointer to a desired location within the file.

Syntax:

fseek(file ptr,offset,position);

file pointer is a pointer to the file concerned, offset is a number or variable of type long and position is an integer number which takes one of the following values. The offset specifies the number of positions(**Bytes**) to be moved from the location specified by the position which can be positive implies moving forward and negative implies moving backwards.

POSITION VALUE	VALUE CONSTANT	MEANING
0	SEEK_SET	BEGINNING OF FILE
1	SEEK_CUR	CURRENT POSITION
2	SEEK_END	END OF FILE

Example Program to illustrate fseek() in C.

```
// To demonstrate fseek()
#include<stdio.h>
int main()
 {
    FILE *fp;
    fp = fopen("fseek.txt","w");
    fputs("This is CSE class\n",fp);
    fseek(fp,8,SEEK_SET);
    fputs("C Programming Language Class", fp);
    fclose(fp);
    return(1);
 }
```

Example 2:

```
#include<stdio.h>
#include<conio.h>
int main(){
   FILE *fp;
   int position;
   clrscr();
   fp = fopen ("file.txt", "r");
   position = ftell(fp);
   printf("Cursor position = %d\n",position);
   fseek(fp,5,0);
   position = ftell(fp);
   printf("Cursor position = %d\n", position);
   rewind(fp);
   position = ftell(fp);
   printf("Cursor position = %d", position);
   fclose(fp);
   getch();
   return 0;
```

}

Explain about error handling in files?

Ans : 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.

In C we have two status - inquiry library functions feof and ferror that can help us detect I/O errors in the files.

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 non zero 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 detected up to that point, during processing. It returns zero otherwise. The statement
if(ferror(fp)!=0)
printf(“an error has occurred\n”);
would print an error message if the reading is not successful

c). fp==null: We know that whenever a file is opened using fopen function, a file pointer is returned. If the file cannot be opened for some reason, 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. For example:

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

Program for error handling in files:

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
FILE *fp;
char ch;
 fp=fopen("random access.c","r");
if(fp==NULL)
{
 perror("Error generated by compiler: ");
 printf("\n File cannot be opened");
 exit(1);
}
while(!feof(fp))
 {
ch=getc(fp);
putchar(ch);
}
fclose(fp);
}
```