UNIT-III

Preprocessor: Commonly used Preprocessor commands like include, define, undef, if, ifdef, ifndef**.**

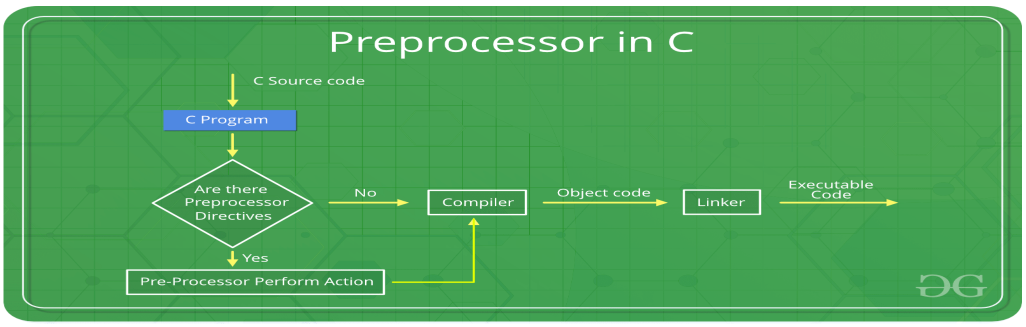
Files: Text and Binary files, Creating and Reading and writing text and binary files, Appending data to existing files, Writing and reading structures using binary files, Random access using fseek, ftell and rewind functions.

1. Briefly explain the pre-processor directives in detail.

Ans:

C Preprocessors:

As the name suggests Preprocessors are programs that process our source code before compilation. There are a number of steps involved between writing a program and executing a program in C .



You can see the intermediate steps in the above diagram. The source code written by programmers is stored in the file **program.c**. This file is then processed by preprocessors and an expanded source code file is generated named program. This expanded file is compiled by the compiler and an object code file is generated named **program .obj**. Finally, the linker links this object code file to the object code of the library functions to generate the executable file **program.exe**.

- Preprocessor programs provide preprocessors directives which tell the compiler to preprocess the source code before compiling.
- All of these preprocessor directives begin with a '#' (hash) symbol.
- This ('#') symbol at the beginning of a statement in a C program indicates that it is a pre-processor directive. We can place these preprocessor directives anywhere in our program

There are 3 main types of preprocessor directives:

1. File Inclusion
2. Macros
3. Conditional Compilation

1. **File Inclusion:** This type of pre-processor directive tells the compiler to include a file in the source code program. There are two types of files which can be included by the user in the program:

a) **Header File or Standard Functions:** These files contains definition of pre-defined functions like printf(), scanf() etc. These files must be included for working with these functions. Different function are declared in different header files. For example standard I/O functions are in 'stdio.h' file whereas functions which perform string operations are in 'string' file.

Syntax:

#include< *file_name* >

where *file_name* is the name of file to be included. The '<' and '>' brackets tells the compiler to look for the file in standard directory.

b) **User defined header files:** When a program becomes very large, it is good practice to divide it into smaller files and include whenever needed. These types of files are user defined files. These files can be included as:

#include"*filename*"

2. **Macros**: Macros are a piece of code in a program which is given some name. Whenever this name is encountered by the compiler the compiler replaces the name with the actual piece of code. The '#define' directive is used to define a macro. Let us now understand the macro definition with the help of a program:

```
#include <stdio.h>
// macro definition
#define LIMIT 5
int main()
{
   for (int i = 0; i < LIMIT; i++) {
      printf("%d \n",i);
   }
   return 0;
 }
```

In the above program, when the compiler executes the word LIMIT it replaces it with 5. The word 'LIMIT' in the macro definition is called a macro template and '5' is macro expansion. Note: There is no semi-colon(';') at the end of macro definition. Macro definitions do not need a semi-colon to end.

Macros with arguments: We can also pass arguments to macros. Macros defined with arguments works similarly as functions. Let us understand this with a program:

```
#include <stdio.h>
  // macro with parameter
#define AREA(l, b) (l * b)
int main()
{
    int l1 = 10, l2 = 5, area;
    area = AREA(l1, l2);
    printf("Area of rectangle is: %d", area);
    return 0;
}
```

We can see from the above program that whenever the compiler finds AREA(l, b) in the program it replaces it with the statement (l*b) . Not only this, the values passed to the macro template AREA(l, b) will also be replaced in the statement (l*b). Therefore AREA(10, 5) will be equal to 10*5.

3. Conditional Compilation: Conditional Compilation directives are type of directives which helps to compile a specific portion of the program or to skip compilation of some specific part of the program based on some conditions.

a) #undef

To undefine a macro means to cancel its definition. This is done with the #undef directive.

Syntax:

```
#undef token
```

define and undefine example

```
#include <stdio.h>
#define PI 3.1415
#undef PI
main()
{
   printf("%f",PI);
}
Output:  Compile Time Error: 'PI' undeclared
```

b). #ifdef

The #ifdef preprocessor directive checks if macro is defined by #define. If yes, it executes the code.

Syntax:

```
#ifdef MACRO
//code
#endif
```

c) #ifndef

The #ifndef preprocessor directive checks if macro is not defined by #define. If yes, it executes the code.

Syntax:

```
#ifndef MACRO
//code
#endif
```

d). #if

The #if preprocessor directive evaluates the expression or condition. If condition is true, it executes the code.

Syntax:

```
#if expression
//code
#endif
```

e) #else

The #else preprocessor directive evaluates the expression or condition if condition of #if is false. It can be used with #if, #elif, #ifdef and #ifndef directives.

Syntax:

```
#if expression
//if code
#else
//else code
#endif
```

Syntax with #elif

```
#if expression
//if code
#elif expression
//elif code
#else
//else code
#endif
```

Example

```
#include <stdio.h>
#include <conio.h>
#define NUMBER 1
void main() {
#if NUMBER==0
printf("Value of Number is: %d",NUMBER);
#else
print("Value of Number is non-zero");
#endif
getch();
}
```

Output: Value of Number is non-zero

2. Define file and explain about the types of files with examples.

Ans :

- ❖ A file is an external collection of related data treated as a unit. A file is a place on a disk where a group of related data is stored and retrieved whenever necessary without destroying data.
- ❖ The primary purpose of a file is to keep record of data. Record is a group of related fields. Field is a group of characters which convey meaning.
- ❖ Files are stored in auxiliary or secondary storage devices. The two common forms of secondary storage are disks (hard disk, CD and DVD) and tapes.
- ❖ Each file ends with an end of file (EOF) at a specified byte number, recorded in file structure.
- ❖ A file must first be opened properly before it can be accessed for reading or writing. When a file is opened an object (buffer) is created and a stream is associated with the object.

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 depending upon the format in which data is stored:

1) Text files 2) Binary files

1) Text files:

A text file stores textual information like alphabets, numbers, special symbols, etc. actually the ASCII code of textual characters its stored in text files. Examples of some text files include c, java, c++ source code files and files with .txt extensions . 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. These operations are performed with the help of inbuilt functions of c.

2) 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.

3. Explain different modes of opening files with syntax and example?

Ans:

To store data in a file three things have to be specified for operating system.

They include

1. FILENAME :

It is a string of characters that make up a valid file name which may contain two parts,a primary name and an optional period with the extension.

Prog1.c

Myfirst.java

Data.txt

store

2. DATA STRUCTURE:

It is defined as FILE in the library of standard I/O function definitions. Therefore all files are declared as type FILE.FILE is a define data type.

FILE *fp;

3. PURPOSE:

It defines the reason for which a file is opened and the mode does this job.

fp=fopen("filename","mode");

The different modes of opening files are :

"r" (read) mode:

The read mode (r) opens an existing file for reading. When a file is opened in this mode, the file marker or pointer is positioned at the beginning of the file (first character).The file must already exist: if it does not exist a NULL is returned as an error. If we try to write a file opened in read mode, an error occurs.

Syntax: **fp=fopen ("filename","r");**

"w" (write) mode

The write mode (w) opens a file for writing. If the file doesn"t exist, it is created. If it already exists, it is opened and all its data is erased; the file pointer is positioned at the beginning of the file It gives an error if we try to read from a file opened in write mode.

Syntax: **fp=fopen ("filename","w");**

“a” (append) mode

The append mode (a) opens an existing file for writing instead of creating a new file. However, the writing starts after the last character in the existing file ,that is new data is added, or appended, at the end of the file. If the file doesn‟t exist, new file is created and opened. In this case, the writing will start at the beginning of the file.

Syntax: **fp=fopen (“filename”,”a”);**

“r+” (read and write) mode

In this mode file is opened for both reading and writing the data. If a file does not exist then NULL, is returned.

Syntax: **fp=fopen (“filename”,”r+”);**

“w+” (read and write) mode

In this mode file is opened for both writing and reading the data. If a file already exists its contents are erased and if a file does not exist then a new file is created.

Syntax: **fp=fopen (“filename”,”w+”);**

“a+” (append and read) mode

In this mode file is opened for reading the data as well as data can be added at the end.

Syntax: **fp=fopen (“filename”, “a+”);**

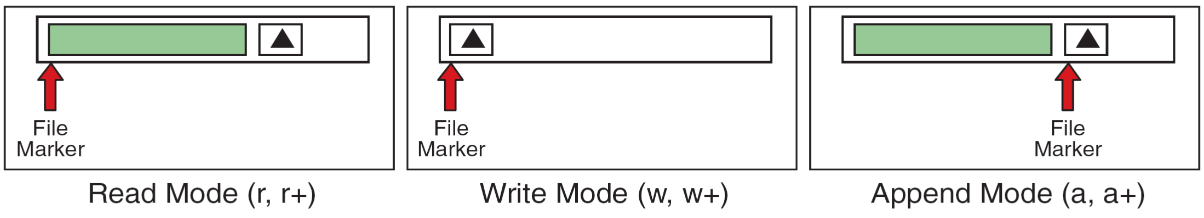


Figure : File Opening Modes

NOTE: To perform operations on binary files the following modes are applicable with an extension b like rb,wb,ab,r+b,w+b,a+b,which has the same menaing but allows to perform operations on binary files.

4. Write about the basic operations on files?

Ans:

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

In order to perform the basic file operations C supports a number of functions .Some of the important file handling functions available in the C library are as follows :

FUNCTION NAME	OPERATION
fopen()	Creates a new file for use if file not exists Opens an existing file for use
fclose()	Closes a file which has been opened for use
fcloseall()	Closes all files which are opened
getc()/fgetc()	Reads a character from a file
putc()/fputc()	Writes a character to a file
fprintf()	Writes a set of data values to files
fscanf()	Reads a set of data values from files
getw()	Reads an integer from file
putw()	Writes an integer to a file
gets()	Reads a string from a file
puts()	Writes a string to a file
fseek()	Sets the position to a desired point in afile
ftell()	Gives the current position in the file
rewind()	Sets the position to the beginning of the 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. A primary name and an optional period with the extension.

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

The general format of declaring and opening a file is

```
FILE *fp;                //declaration
```

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

Here FILE is a data structure defined for files. fp is a pointer to data type FILE. filename is the name of the file. mode tells the purpose of opening this file.

Reading data from file :

Input functions used are (Input operations on files)

a) **getc();** It is used to read characters from file that has been opened for read operation.

Syntax: c=getc (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 has 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 type 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 writes a character to a file that has been opened in write mode.

Syntax: putc(c,fp);

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.Closing a file ensures that all outstanding information associated with the file is flushed out from the buffers and all links to the file are broken. Another instance where we have to close a file is to reopen the same file in a different mode. Library function for closing a file is

Syntax: fclose(file pointer);

Example: fclose(fp);

Where fp 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.

5. What are the file I/O functions in C. Give a brief note about the task performed by each function?

Ans:

In order to perform the file operations in C we must use the high level I/O functions which are in C standard I/O library. They are

i) getc() and putc() functions:

getc()/fgetc() : It is used to read a character from a file that has been opened in a read mode. It reads a character from the file whose file pointer is fp. The file pointer moves by one character for every operation of getc(). The getc() will return an end-of –marker EOF, when an end of file has been reached.

Syntax: getc(fp);

Ex: char ch;

ch=getc(fp);

putc()/fputc() -: It is used to write a character contained in the character variable to the file associated with the FILE pointer fp. fputc() also returns an end-of –marker EOF, when an end of file has been reached**.**

Syntax: **putc(c,fp);**

Example: char c;

putc(c,fp);

Program using fgetc() and fputc():

```
#include<stdio.h>
void main()
{
FILE *fp;
char ch;
fp=fopen("input1.txt","w");
printf("\n enter some text hereand press cntrl D or Z to stop :\n");
while((ch=getchar())!=EOF)

fputc(ch,fp);
fclose(fp);
fp=fopen("input1.txt","r");
printf("\n The entered text is : \n");
while((ch=fgetc(fp))!=EOF)
putchar(ch);
fclose(fp);
}
```

ii) fprintf() and fscanf():

In order to handle a group of mixed data simultaneously there are two functions that are fprintf() and fscanf().These two functions are identical to printf and scanf functions,except that they work on files. The first argument of these functions is a file pointer which specifies the file to be used.

fprintf(): The general form of fprintf() is

Syntax: fprintf(fp,”control string”,list);

where fp is a file pointer associated with a file that has been opened for writing . The control string contains output specifications for the items in the list. .

Example: fprintf(fp,”%s%d”,name,age);

fscanf() : It is used to read a number of values from a file.

Syntax: fscanf(fp,”control string”,list);

Example: fscanf(fp2,”%s %d”,item,&quantity);

like scanf , fscanf also returns the number of items that are successfully read. when the end of file is reached it returns the value EOF.

Program using fscanf() and fprintf():

```
#include<stdio.h>
void main()
{
int a=22,b;
char s1[20]="Welocme_to_c",s2[20];
float c=12.34,d;
```

```
        FILE *f3;
        f3=fopen("mynew3","w");
        fprintf(f3,"%d %s %f",a,s1,c);
        fclose(f3);

        f3=fopen("mynew3","r");
        fscanf(f3,"%d %s %f",&b,s2,&d);
        printf("\n a=%d \t s1=%s \t c=%f \n b=%d \t s2=%s \t d=%f",a,s1,c,b,s2,d);
        fclose(f3);
        }
```

iii) getw() and putw():

The getw() and putw()are integer oriented functions .They are similar to the getc() and putc() functions and are used to read and write integer values . These functions would be useful when we deal with only integer data. The general form of getw() and putw() are

Syntax: putw(integer,fp);

Syntax: getw(fp);

Program using getw() and putw():

```
/*Printing odd numbers in odd file and even numbers in even file*/
#include<stdio.h>
void main()
{
int x,i;
FILE *f1,*fo,*fe;//creating a file pointer
f1=fopen("anil.txt","w"); //opening a file
printf("\n enter some numbers into file or -1 to stop \n");
for(i=0;i<20;i++)
{
scanf("%d",&x);
if(x== -1)break;
putw(x,f1); //writing read number into anil.txt file one at a time
}
fclose(f1);    //closing a file opened for writing input
printf("\n OUTPUT DATA\n");
f1=fopen("anil.txt","r");//open anil in read mode to read data
fo=fopen("odd3f","w");
fe=fopen("even3f","w");
while((x=getw(f1))!=EOF)
{
printf("%d\t",x);
if(x%2==0)
putw(x,fe);
else
putw(x,fo);
}
fcloseall();
fo=fopen("odd3f","r");
printf("\n contents of odd file are :\n");
while((x=getw(fo) )!= EOF)
printf(" %d\t",x);
fe=fopen("even3f","r");
printf("\n contents of even file are :\n");
while((x=getw(fe)) != EOF)
printf(" %d\t",x);
fcloseall();
}
```

iv) fputs() and fgets():

fgets(): It is used to read a string from a file pointed by file pointer. It copies the string to a memory location referred by an array.

Syntax:fgets(string,length,filepointer);

Example: fgets(text,50,fp1);

fputs(): It is used to write a string to an opened file pointed by file pointer.

Syntax: fputs(string,filepointer);

Example: fputs(text,fp);

Program using fgets() and fputs():

```
#include<stdio.h>
void main()
{
FILE *fp;
char str[50];
fp=fopen("fputget.txt","r");
printf("\n the read string is :\n");
fgets(str,50,fp);
puts(str);
fclose(fp);
fp=fopen("fputget.txt","a+");
printf("\n Enter string : \n");
gets(str);
fputs(str,fp);
puts(str);
fclose(fp);
}
```

Block or structures read and write:

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

v) fread() and fwrite():

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);**

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);**

Program using fread() and fwrite():

```
#include<stdio.h>
struct player
{
char pname[30];
int age;
int runs;
};
void main()
{
struct player p1,p2;
FILE *f3;
f3=fopen("player.txt","w");
```

```
        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.txt","r");
        fread(&p2,sizeof(p2),1,f3);
        fflush(stdout);
        printf("\nPLAYERNAME:=%s\tAGE:=%d\tRUNS:=%d ",p2.pname,p2.age,p2.runs);
        fclose(f3);
        }
```

6. Explain about random access to files with example?

Ans:

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():-

ftell takes a file pointer and returns a number of type **long,** that corresponds to the current position. This function is useful in saving the current position of the file, which can later be used in the program.

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

n would give the Relative offset (In bytes) of the current position. This means that already **n** bytes have a been read or written

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.

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: **fseek(fp,10,0) ;**

fseek(fp,10,SEEK_SET);// file pointer is repositioned in the forward direction 10 bytes.

fseek(fp,-10,SEEK_END); // reads from backward direction from the end of file.

When the operation is successful fseek returns 0 and when we attempt to move a file beyond boundaries fseek returns -1.

Some of the Operations of fseek function are as follows:

STATEMENT	**MEANING**
fseek(fp,0L,0);	Go to beginning similar to rewind()
fseek(fp,0L,1);	Stay at current position
fseek(fp,0L,2);	Go to the end of file, past the last character of the file.
fseek(fp,m,0);	Move to (m+1)th byte in the file.
fseek(fp,m,1);	Go forward by m bytes
fseek(fp,-m,1);	Go backwards by m bytes from the current position
fseek(fp,-m,2);	Go backwards by m bytes from the end.(positions the file to the mth character from the end)

Program on random access to files:

```
#include<stdio.h>
void main()
{
FILE *fp;
char ch;
fp=fopen("my1.txt","r");
fseek(fp,21,SEEK_SET);
ch=fgetc(fp);
while(!feof(fp))
{
printf("%c\t",ch);
printf("%d\n",ftell(fp));
ch=fgetc(fp);
}
rewind(fp);
printf("%d\n",ftell(fp));
fclose(fp);
}
```

7. 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>

void main()
{
FILE *fp;
char ch;
fp=fopen("my1.txt","r");
if(fp==NULL)
printf("\n file cannot be opened");
while(!feof(fp))
{
ch=getc(fp);
if(ferror(fp))
perror("problem in the file");
else

putchar(ch);
}
fclose(fp);
}
```

8. Write a c program to read and display the contents of a file?

Ans: Program:

```
#include<stdio.h>
void main()
{
FILE *f1;
char ch;
f1=fopen("data.txt","w");
printf("\n enter some text here and press cntrl D or Z to stop :\n");
while((ch=getchar())!=EOF)
fputc(ch,f1);
fclose(f1);
printf("\n the contents of file are \n:");
f1 = fopen("data.txt","r");
while( ( ch = fgetc(f1) ) != EOF )
putchar(ch);
fclose(f1);
}
```

9. Write a c program to copy the contents of one file to another?

Ans: Program:

```
#include<stdio.h>
void main()
{
FILE *f1,*f2;
char ch;
f1=fopen("mynew2.txt","w");
printf("\n enter some text here and press cntrl D or Z to stop :\n");
while((ch=getchar())!=EOF)
fputc(ch,f1);
fclose(f1);
f1=fopen("mynew2.txt","r");
f2=fopen("dupmynew2.txt","w");
while((ch=getc(f1))!=EOF)
```

```
   putc(ch,f2);
   fcloseall();
   printf("\n the copied file contents are :");
   f2 = fopen("dupmynew2.txt","r");
   while( ( ch = fgetc(f2) ) != EOF )
   putchar(ch);
   fclose(f2);
  }
```

10. Write a c program to merge two files into a third file? (Or)

Write a c program for the following .there are two input files named "first.dat" and "second.dat" .The files are to be merged. That is,copy the contents of first.dat and then second.dat to a new file named result.dat?

Ans: Program:

```
#include  <stdio.h>
#include <stdlib.h>
int main()
{
FILE *fs1, *fs2, *ft;
char ch, file1[20], file2[20], file3[20];
printf("Enter name of first file\n");
gets(file1);
printf("Enter name of second file\n");
gets(file2);
printf("Enter name of file which will store contents of two files\n");
gets(file3);
fs1=fopen(file1,"w");
printf("\n enter some text into file1 here and press cntrl D or Z to stop :\n");
while((ch=getchar())!=EOF)
fputc(ch,fs1);
fclose(fs1);
fs2=fopen(file2,"w");
printf("\n enter some text into file2 here and press cntrl D or Z to stop :\n");
while((ch=getchar())!=EOF)
fputc(ch,fs2);
fclose(fs2);
```

```
        fs1 = fopen(file1,"r");
        fs2 = fopen(file2,"r");
        if( fs1 == NULL || fs2 == NULL )
        {
        perror("Error ");
        exit(1);
        }
        ft = fopen(file3,"w");
        while( ( ch = fgetc(fs1) ) != EOF )
        fputc(ch,ft);
        while( ( ch = fgetc(fs2) ) != EOF )
        fputc(ch,ft);
        printf("Two files were merged into %s file successfully.\n",file3);
        fcloseall();
        ft = fopen(file3,"r");
        printf("\n the merged file contents are :");
        while( ( ch = fgetc(fs1) ) != EOF )
        putchar(ch);
        fclose(ft);
        return 0;
        }
```

11. Write a c program to append the contents of a file ?

Ans: Program:

```
#include<stdio.h>
void main()
{
FILE *fp1;
char ch;
fp1=fopen("sunday.txt","w");
printf("\n Enter some Text into file :\n");
while((ch=getchar())!='.')
fputc(ch,fp1);
fclose(fp1);
fp1=fopen("sunday.txt","a+"); //to append
printf("\n Enter some MORE Text into file :\n");
while((ch=getchar())!='.')
fputc(ch,fp1);
rewind(fp1);
printf("\n The complete Text in file is :\n");
while((ch=fgetc(fp1))!=EOF)
putchar(ch);
fclose(fp1);
}
```

12. Write a c program to display the reverse the contents of a file?

Ans:

Program:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
FILE *fp1;
char ch;
int x;
fp1=fopen("any1.txt","w");
printf("\n Enter some text into file :\n");
while((ch=getchar())!=EOF)
fputc(ch,fp1);
fclose(fp1);
fp1=fopen("any1.txt","r");
if(fp1==NULL)
{
printf("\n cannot open file:");
exit(1);
}
fseek(fp1,-1L,2);
x=ftell(fp1);
printf("\n Th text in the file in reverse order is : \n");
while(x>=0)
{
ch=fgetc(fp1);
printf("%c",ch);
x--;
fseek(fp1,x,0);
```

```
 }
fclose(fp1);
  }
```

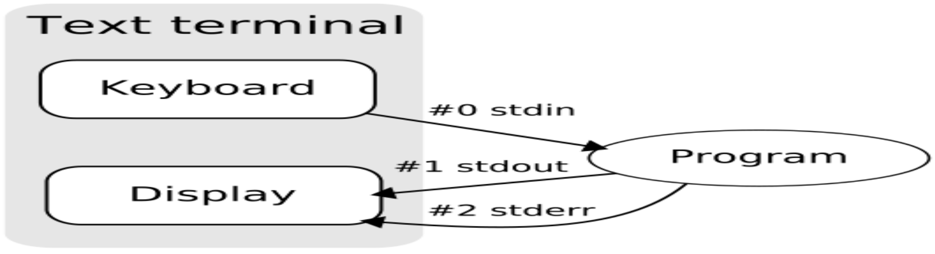
13. Explain the concept of streams and their significance in I/O operations.

Ans:

File I/O Streams in C:

1. In C all **input and output** is done with streams
2. Stream is nothing but the **sequence of bytes of data**
3. A sequence of bytes flowing into program is called **input stream**
4. A sequence of bytes flowing out of the program is called **output stream**
5. Use of Stream make I/O machine independent.

Types of Streams in C :



• Standard input stream is called "stdin" and is normally connected to the keyboard.

- **stdin** stands for (**St**an**d**ard **In**put)
- Keyboard is **standard input device** .
- Standard input is data **(Often Text) going into a program**.
- The program requests data transfers by use of the read operation.

• Standard output stream is called "stdout" and is normally connected to the display screen.

- **stdout** stands for (**St**an**d**ard **Out**put)
- Screen(Monitor) is **standard output device** .
- Standard output is data **(Often Text) going out from a program**.
- The program sends data to output device by using write operation.

• Standard error stream is called "stderr" and is also normally connected to the screen.

14. Explain the command line arguments. What are the syntactic constructs followed in "C"?

Ans :

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.

The **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 *argv[ ];
{
……….
}
```

Generally argv[0] represents the program name.

__Example:-__ A program to copy the contents of one file into another using command line arguments.

```
#include<stdio.h>
#include<stdlib.h>
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");
        fs=fopen(argv[1],”r”);
        ft=fopen(argv[2],”w”);
        if (fs==NULL)
        {
        printf(“\n source file opening error”);
        exit(1) ;
        }
        if (ft==NULL)
        {
        printf(“\n target file opening error”);
        exit(1) ;
        }
         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);  }
```

15. Write a c program to add two numbers using command line arguments?

Ans:

Program:

```
#include<stdio.h>
int main(int argc, char *argv[])
{
int x, sum=0;
printf("\n Number of arguments are:%d", argc);
printf("\n The agruments are:");
for(x=0;x<argc; x++)
 {
printf("\n agrv[%d]=%s", x, argv[x]);
if(x<2)
continue;
else
sum=sum+atoi(argv[x]);
  }
printf("\n program name:%s",argv[0]);
printf("\n name is:%s",argv[1]);
printf("\n sum is:%d",sum);
return(0);
 }
```