Unit-III

Preprocessor:

The C preprocessor is a macro processor that is used automatically by the C compiler to transform your program before actual compilation. It is called a macro processor because it allows you to define macros.
The C preprocessor provides following type:

- **Inclusion of header files** : These are files of declarations that can be substituted into your program.
- **Macro expansion**: You can define macros, which are abbreviations for C code, and then the C preprocessor will replace the macros with their definitions throughout the program.
- **Conditional compilation**: Using special preprocessing directives, you can include or exclude parts of the program according to various conditions.

Preprocessing directives are lines in your program that start with `#'. The `#' is followed by an identifier that is the directive name.

For example, `#define' is the directive that defines a macro. Whitespace is also allowed before and after the `#'.

Sr.No.	Directive & Description
1	**#define** Substitutes a preprocessor macro.
2	**#include** Inserts a particular header from another file.
3	**#undef** Undefines a preprocessor macro.
4	**#ifdef** Returns true if this macro is defined.
5	**#ifndef** Returns true if this macro is not defined.
6	**#if** Tests if a compile time condition is true.
7	**#else** The alternative for #if.
8	**#elif**

	#else and #if in one statement.
9	**#endif** Ends preprocessor conditional.

Header Files: (#include)

A header file is a file containing C declarations and macro definitions. we request the use of a header file in our program with the C preprocessing directive `#include'.

The header files names end with `.h'.

Both user and system header files are included using the preprocessing directive `#include'

- System header files declare the interfaces to parts of the operating system. we include them in our program to supply the definitions and declarations of predefined code.
- User can create their own header file and add in the program if needed.

There are two ways to add header file in program-

#include <file>: It searches for a file named file in a list of directories specified by you, then in a standard list of system directories.

#include "file": It searches for a file named file first in the current directory, then in the same directories used for system header files.

Inclusion of header files:

User can create their own header file. Following program explain how to create

Example: Creating an user header file

File name: fact.h

```
  #include<stdio.h>
   int fact(int n)
    {
        int i=1,f=1;
        while(i<=n)
              {
                    f=f*i;
                    i++;
              }
         Return f;
    }
```

Example: use of an user header file

File name: factprogram.c

```
  #include<stdio.h>
  #include "fact.h"
   int main()
     {
         int i,n=5;
         i=fact(n);
         printf("factorial is %d\n",i);
     }
```

Output: factorial is 120

Macro expansion: (#define)

Simple Macros
Syntax:

```
#define Template_name  value
```

Note: Template_name replace with its value in program.

Example: #define upper 20

when we add above syntax in the program, wherever upper come it replace with 20

```
#include<stdio.h>
  # define upper 20
   int main()
     {
          int i,n=5;
          i=upper+n;
          printf("value of  i  is %d\n",i);
   }
```

Output: value of i is 25.

Macros with Arguments:
Here user pass an argument with the macro template .

Syntax:

```
#define Template_name(Arguments,...)  (Expression)
```

Example:

```
#include<stdio.h>
# define rectanglearea(a,b)   (a*b)
   int main()
     {
      int a,b,area;
     printf("enter a & b value\n");
     scanf("%d%d",&a,&b);
       area=rectanglearea(a,b);
      printf("area of rectangle   is %d\n",area);
 }
```

Output:

```
     Enter a & b value

     2     5

     area of rectangle   is 10
```

Miscellaneous Directives: (#undef)

the #undef directive tells the preprocessor to remove the definitions for the specified macro. A macro can be redefined after it has been removed by the #undef directive.

Syntax:

```
#undef Template_name(Arguments,...)  (Expression)
```

Example:

```
# define rectanglearea(a,b)   (a*b)   //macro definition

# undef rectanglearea(a,b)   (a*b)    //macro undefined

# define rectanglearea(a,b)   (a*b)    //macro redefined
```

Example 1:

```
#include<stdio.h>
# define area(a,b) (a*b)
# undef area(a,b) (a*b)
  int main()
    {
    int a=2,b=11;
  printf("area  of  rectangle   is  %d", area(a,b));
    return 0;
         }
```

Output:

```
Error   // area is not defined macro
```

Example 2:

```
#include<stdio.h>
# define area(a,b) (a*b)
# undef area(a,b) (a*b)
# define area(a,b) (a*b)

  int main()
    {
    int a=2,b=11;
  printf("area  of  rectangle   is  %d", area(a,b));
    return 0;
         }
```

Output:

```
area of rectangle  is 22
```

Conditional compilation : (#ifdef, #ifndef, #endif)

Using conditional compilation, if user want, compiler to skip over part of code by using preprocessing commands #ifdef and #endif

Syntax:

```
 #ifdef macroname
       Statement 1;
       Statement 1;
       .
       .
#endif
```

If macronme has been #defind, the block of code will be processed as usual; otherwise not.

Example 1:

```
#include<stdio.h>
# define rectanglearea(a,b) (a*b)
# define upper 2
  int main()
    {
    int a,b,area;
     #ifdef upper
    printf("enter a & b value");
    scanf("%d%d",&a,&b);
     area=rectanglearea(a,b);
    printf("area of rectangle  is %d",area);
    #endif
     Printf("\n end of the program"):
    return 0;
          }
```

Output:

```
enter a & b value
5 7
area of rectangle  is 35
end of the program
```

Example 2:

```
#include<stdio.h>
# define rectanglearea(a,b) (a*b)
# define upper 2
  int main()
    {
    int a,b,area;
     #ifdef u
    printf("enter a & b value");
    scanf("%d%d",&a,&b);
     area=rectanglearea(a,b);
     printf("area    of    rectangle      is
%d",area);
     #endif
     Printf("\n end of the program"):
    return 0;
          }
```

Output:

```
end of the program
```

- Sometimes we can use **#ifndef,** instead **#ifdef**.
- #ifndef works exactly opposite to #ifdef.

Example:

```
#include<stdio.h>
# define rectanglearea(a,b) (a*b)
# define upper 2
  int main()
    {
    int a,b,area;

     #ifndef upper
    printf("enter a & b value");
    scanf("%d%d",&a,&b);
     area=rectanglearea(a,b);
     printf("area of rectangle  is %d",area);
     #endif

     Printf("\n end of the program"):

    return 0;
          }
```

Output:

end of the program

#if, #else, #elif, #endif Directives:

The #if directive is used to test whether an expression evaluate to a nonzero value or not. If result is non zero(ie true) the subsequent line of code execute upto #else, #elif or #endif comes; otherwise they are skipped**.**

Example: check number is +ve, -ve or zero.

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

     printf("enter number");
     scanf("%d",&num);
      #if num>0
      printf("number is +ve\n");
      #elif num<0

      printf("number is -ve\n");

     #else

      printf("number is zero\n");

    #endif

     return 0;
            }
```

Output:
```
enter number

25

number is +ve
```

File:

In C programming, file is a place on computer disk where information is stored permanently. Which represents a sequence of bytes ending with an end-of-file marker(EOF) .

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.

Types of Files

1. Text files
2. Binary files

1. Text files

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.

2. Binary files

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.

File Operations

In C, you can perform four major operations on the file, either text or binary:

1. Creating a new file
2. Opening an existing file
3. Closing a file

4. Reading from and writing information to a file

Declaration for file Pointer:

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

Syntax:

```
FILE  *file_pointer_name;
```

Example: FILE *fp;

Common File Functions

unction	**description**
fopen()	create a new file or open a existing file
fclose()	closes a file
getc()	reads a character from a file
putc()	writes a character to a file
fscanf()	reads a set of data from a file
fprintf()	writes a set of data to a file
getw()	reads a integer from a file
putw()	writes a integer to a file
fseek()	set the position to desire point
ftell()	gives current position in the file
rewind()	set the position to the begining point

Opening a File or Creating a File:

The fopen() function is used to create a new file or to open an existing file. this function available in stdio.h file

Syntax:

fp= fopen(“filename”, “mode”);

filename is the name of the file to be opened and **mode** specifies the purpose of opening the file. Mode can be of following types,

Example:

```
fp= fopen(“sample.txt”, “w”);

fp= fopen(“sample.txt”, “r”);
```

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 text file in both reading and writing 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

If fopen( ) cannot open "test.dat " it will a return a NULL pointer which should always be tested for as follows.

Example:

```
FILE *fp ;
```

```
if ( ( fp = fopen( "test.dat", "r" ) ) == NULL )
        {
        puts( "Cannot open file") ;
        exit( 1) ;
        }
```

This will cause the program to be exited immediately if the file cannot be opened.

Closing a File:

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

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

Syntax:

fclose(file_pointer_name);

Example:

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

Reading and writing to a text file:

For reading and writing to a text file, we use the functions fprintf() and fscanf().

They are just the file versions of printf() and scanf(). The only difference is that, fprint and fscanf expects a pointer to the structure FILE.

Writing to a text file:

Example 1: Write to a text file using fprintf()

```
#include <stdio.h>
int main()
{
   int num;
   FILE *fptr;
   fptr = fopen("program.txt","w");

   if(fptr == NULL)
   {
      printf("Error!");
      exit(1);
   }

   printf("Enter num: ");
   scanf("%d",&num);

   fprintf(fptr,"%d",num);
   fclose(fptr);

   return 0;
}
```

This program takes a number from user and stores in the file program.txt.

Reading from a text file

Example 2: Read from a text file using fscanf()

```
#include <stdio.h>
int main()
{
   int num;
   FILE *fptr;

   if ((fptr = fopen("C:\\program.txt","r")) == NULL){
       printf("Error! opening file");

       // Program exits if the file pointer returns NULL.
       exit(1);
   }

   fscanf(fptr,"%d", &num);

   printf("Value of n=%d", num);
   fclose(fptr);

   return 0;
}
```

This program reads the integer present in the program.txt file and prints it onto the screen.

If you succesfully created the file from **Example 1**, running this program will get you the integer you entered.

Note: Other functions like fgetchar(), fgetc(),fputc(),fgets(),fputs() etc. can be used in similar way.

Reading and writing to a binary file:

Functions fread() and fwrite() are used 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);
```

Example 3: Writing to a binary file using fwrite()

```
#include <stdio.h>

struct threeNum
{
   int n1, n2, n3;
};

int main()
{
   int n;
   struct threeNum num;
   FILE *fptr;

   if ((fptr = fopen("C:\\program.bin","wb")) == NULL){
       printf("Error! opening file");

       // Program exits if the file pointer returns NULL.
       exit(1);

   }
   for(n = 1; n < 5; ++n)

   {

      num.n1 = n;

      num.n2 = 5n;
      num.n3 = 5n + 1;
      fwrite(&num, sizeof(struct threeNum), 1, fptr);
   }
   fclose(fptr);
     return 0;
}
```

In this program, you create a new file program.bin in the C drive.

We declare a structure threeNum with three numbers - n1, n2 and n3, and define it in the main function as num.

Now, inside the for loop, we store the value into the file using fwrite.

The first parameter takes the address of num and the second parameter takes the size of the structure threeNum.

Since, we're only inserting one instance of num, the third parameter is 1. And, the last parameter *fptr points to the file we're storing the data.

Finally, we close the file.

Reading from a binary file

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

```
fread(address_data,size_data,numbers_data,pointer_to_file);
```

Example 4: Reading from a binary file using fread()

```
#include <stdio.h>

struct threeNum
{
   int n1, n2, n3;
};

int main()
{
   int n;
   struct threeNum num;
   FILE *fptr;
```

```
  if ((fptr = fopen("C:\\program.bin","rb")) == NULL){
      printf("Error! opening file");

      // Program exits if the file pointer returns NULL.
      exit(1);
  }

  for(n = 1; n < 5; ++n)
  {
     fread(&num, sizeof(struct threeNum), 1, fptr);
     printf("n1: %d\tn2: %d\tn3: %d", num.n1, num.n2, num.n3);
  }
  fclose(fptr);

  return 0;
}
```

In this program, you read the same file program.bin and loop through the records one by one.

In simple terms, you read one threeNum record of threeNum size from the file pointed by *fptr into the structure num.

Reading & Writing Characters:

Once a file pointer has been linked to a file we can write characters to it using the fputc() function.

fputc(ch, fp) ;

If successful the function returns the character written otherwise EOF. Characters may be read from a file using the fgetc() standard library function.

ch = fgetc(fp) ;

When EOF is reached in the file fgetc( ) returns the EOF character which informs us to stop reading as there is nothing more left in the file.

Example :- Program to copy a file byte by byte

```
#include  <stdio.h>
void main()
{
FILE *fin, *fout ;
char dest[30], source[30], ch ;

puts( "Enter source file name" );
gets( source );
puts( "Enter destination file name" );
gets( dest ) ;
```

```
        if ( ( fin = fopen( source, "rb" ) ) == NULL )
                {
                puts( "Cannot open input file ") ;
                puts( source ) ;
                exit( 1 ) ;
                }
// open as binary as we don't  know what is in file

        if ( ( fout = fopen( dest, "wb" ) == NULL ) )
                {
                puts( "Cannot open output file ") ;
                puts( dest ) ;
                exit( 1 ) ;
                }

        while ( ( ch = fgetc( fin ) )  !=  EOF  )
                fputc( ch , fout ) ;

        fclose( fin ) ;
        fclose( fout ) ;
        }
```

Note : When any stream I/O function such as fgetc() is called the current position of the file pointer is automatically moved on by the appropriate amount, 1 character/ byte in the case of fgetc() ;

Appending Data To Existing File

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**.

In general, to **append** is to join or add on to the end of something. For example, an appendix is a section appended (added to the end) of a document. In computer **programming**, **append** is the name of a procedure for concatenating (linked) lists or arrays in some high-level **programming** languages.

```
FILE *pFile;
FILE *pFile2;
char buffer[256];

pFile=fopen("myfile.txt", "r");
pFile2=fopen("myfile2.txt", "a");
if(pFile==NULL) {
   perror("Error opening file.");
}
else {
   while(fgets(buffer, sizeof(buffer), pFile)) {
      fprintf(pFile2, "%s", buffer);
   }
}
fclose(pFile);
fclose(pFile2);
```

program

WRITING AND READING STRUCTURES USING BINARY FILES

Read, **write** and seek operations can be performed on **binary files with** the help of fread(), fwrite() and fseek() functions, respectively. After **reading** or **writing** a **structure**, the **file** pointer is moved to the next **structure**. The fseek() function can move the pointer to the position as requested

Reading and Writing from Binary Files

Binary files are very similar to arrays except for the fact that arrays are temporary storage in the memory but binary files are permanent storage in the disks. The most important difference between binary files and a text file is that in a binary file, you can *seek*, *write*, or *read* from any position inside the file and insert structures directly into the files. You must be wondering - why do we need binary files when we already know how to handle plaintexts and text files? Here are the **reasons why binary files are necessary**:

1. I/O operations are much faster with binary data.

Usually, large text files contain millions of numbers. It takes a lot of time to convert 32-bit integers to readable characters. This conversion is not required in the case of binary files as data can be directly stored in the form of bits.

2. Binary files are much smaller in size than text files.

For data that is in the form of images, audio or video, this is very important. Small size means less storage space and faster transmission. For example, a storage device can store a large amount of binary data as compared to data in character format.

3. Some data cannot be converted to character formats.

For example, Java compilers generate bytecodes after compilation.

Having said that, let's move on to handling I/O operations in a binary file in C. The **basic parameters that the read and write functions of binary files accept** are:

- the memory address of the value to be written or read
- the number of bytes to read per block
- the total number of blocks to read
- the file pointer

program

For writing in file, it is easy to write string or int to file using **fprintf** and **putc**, but you might have faced difficulty when writing contents of struct. **fwrite** and **fread** make task easier when you want to write and read blocks of data.

1. **fwrite :** Following is the declaration of fwrite function
2. **`size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream)`**

```
3. ptr - This is pointer to array of elements to be written
4. size -  This is the size in bytes of each element to be written
5. nmemb - This is the number of elements, each one with a size of size
   bytes
   stream - This is the pointer to a FILE object that specifies an
   output stream
```

fread : Following is the declaration of fread function

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream)
ptr - This is the pointer to a block of memory with a minimum size of
size*nmemb bytes.
size - This is the size in bytes of each element to be read.
nmemb - This is the number of elements, each one with a size of size
bytes.
stream - This is the pointer to a FILE object that specifies an input
stream.
```

Random Access to the file:

There is no need to read each record sequentially, if we want to access a particular record. C supports these functions for random access file processing.

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

fseek():
This function is used for seeking the pointer position in the file at the specified byte.
Syntax: fseek(file_pointer, displacement, pointer position);

OR

int fseek (FILE *fp, long num_bytes, int origin) ;

Where
file_pointer ---- It is the pointer which points to the file.

displacement ---- It is positive or negative.

This is the number of bytes which are skipped backward (if negative) or forward(if positive) from the current position.

This is attached with L because this is a long integer.

Pointer position:

This sets the pointer position in the file.

Value	**pointer position**
0	Beginning of file.
1	Current position
2	End of file

Example:

1) fseek(p,10L,0)

0 means pointer position is on beginning of the file, from this statement pointer position is skipped 10 bytes from the beginning of the file.

2) fseek(p,5L,1)

1 means current position of the pointer position.From this statement pointer position is skipped 5 bytes forward from the current position.

3)fseek(p,-5L,1)

From this statement pointer position is skipped 5 bytes backward from the current position.

ftell():

This function returns the value of the current pointer position in the file. The value is count from the beginning of the file.

Syntax: ftell(fptr);

Where fptr is a file pointer.

rewind():

This function is used to move the file pointer to the beginning of the given file.
Syntax: rewind(fptr);

Where fptr is a file pointer.

Example: Write a program to read last ‘n’ characters of the file using appropriate file functions(Here we need fseek() and fgetc()).

```
#include<stdio.h>
#include<conio.h>
void main()
{
     FILE *fp;
     char ch;
     clrscr();
     fp=fopen("file1.c", "r");
     if(fp==NULL)
        printf("file cannot be opened");
     else
     {
            printf("Enter value of n  to read last ‘n’ characters");
            scanf("%d",&n);
            fseek(fp,-n,2);
            while((ch=fgetc(fp))!=EOF)
            {
                 printf("%c\t",ch);
            }
      }
    fclose(fp);
    getch();
}
```

OUTPUT: It depends on the content in the file.

Previous year JNTUH question

DEC 2017:

1. What is the use of rewind ()?
2. What is the impact of fclose () on buffered data.
3. Explain the different types of files are used in C

   b. Write a C program to print the records in reverse order. The file must be opened in binary mode.

4. Write a program to append a binary file at the end of another. b. Explain the function calls of fseek ().

SEPT 2017(Adv. Supply)

1. Compare fread and fscanf functions.
2. What is meant by opening a data file? How is this accomplished?
3. Write a program to copy contents of one file to another using file names passed as the command line arguments.
4. What are the modes in which files can be opened?

   b. Write a program to store information (id, name, address, marks) into a file and print the information from the file.

May 2017 (Supply):

1. Write syntax of opening a file. Give example.
2. List the advantages of using files.
3. Discuss in detail about the file positioning functions.

   b. Write a C program to count number of words, white spaces and tab spaces present in a file.

4. Explain the file input and output function with examples Programs.
   b. Distinguish r, r+, and w and w+ modes.

May 2017:

1. Explain about fseek ().
2. Discuss the different modes available for opening a file.

3. Write a C program to count the number of characters in a file. b. Explain various standard library functions for handling files.

Or

4. Write a C program to create a file contains a series of integer, read them and write all odd numbers in odd file and also write all even numbers to a file called even.

DEC 16:

1. What is meant of text file?
2. Discuss about rewind () function?

3. What is meant of state of a file? Write a C program to copy the contents of one text file to another text file.
4. What is meant by binary file? Discuss about file positioning file functions.

May 2017 (REG)

1. Explain about the preprocessor commands.

Objective question:

1. #define PI 3.14 is a kind of .
(a) Compiler directive
(b) Pre compiler directive
(c) Preprocessor directive
(d) Processor directive

2. Which of the following would compute the square of x in 'C'
(a) power(x,2)
(b) pow(x,2)
(c) pow(2,x)
(d) x**2

3. Which of the following is a correct way of defining a symbolic constant pie in C
(a) #define pie= 3.142
(b) # Define pie 22/7
(c) # define pie = 22/7
(d) #define pie 22/7

4. Which of these complier directives access to the print function
(a) #include <stdio.h >
(b) #define print
(c) include stdio.h;
(d) #include conio.h;

5. Which of the following is the symbol for preprocessor
(a) *
(b) #
(c) <>
(d) $

6. How would you declare a constant of 5 called "MYCONST"?
(a) var int MYCONST=5
(b) int myconst = 5;
(c) constant MYCONST = 5;
(d) #define MYCONST 5

7. Preprocessing is typically done in
(a) after loading
(b) either before or at the beginning of compilation
(c) at the time of execution
(d) after compilation but before execution

8. If actual arguments are more than the formal arguments then
(a) extra actual arguments are discarded
(b) a compilation error
(c) extra actual arguments are initialized to garbage values
(d) extra actual arguments are initialized to zero

9.A file is
(a) a region of storage
(b) a variable
(c) a data type
(d) a data type and a region of storage

10. A file pointer is
(a) a stream pointer
(b) a Boolean pointer

(c) an integer pointer
(d) a pointer to FILE datatype

11 The structure FILE is defined in which of the following header files
(a) stdio.h
(b)conio.h
(c) math.h
(d) file.h

12. We declare file pointer as:FILE *fptr; FILE is a …………..data type defined in stdio.h
(a) char
(b) string
(c) pointer
(d) struct

13. To open an existing file named myfile for reading only function is used where fp is a file pointer
(a) fp=fopen("myfile",'w');
(b) fp=fopen("myfile","w");
(c) fp=fopen("myfile","r");
(d) fp=fopen("myfile",'r');

14. The action of disconnecting a file from a program is obtained by the function
(a) fdisconnect()
(b) clear()
(c) fclose()
(d) delete()

15.If a file is opened in "r+" mode , then
(a) reading and writing are possible
(b) only reading possible
(c) a file will be created if it is not existing
(d) none

16. If a file is opened in "w+" mode , then
(a) appending is possible
(b) if the file already exists,it's contentents are not erased
(c) it creates a file and opens it for both reading and writing
(d) none

17. the ftell() function in file management
(a) gives the current position in the file
(b) can be used to find the size of a file
(c) can guarantee to work properly only on binary files
(d) all the above

18. The fseek() function
(a) needs 3 arguments
(b) needs 2 arguments
(c) meant for checking whether a given file exists or not
(d) None

19.The rewind() function
(a) erases the contents of a file
(b) sets the position indicator to the end of the file
(c) sets the position indicator to the begining of the file
(d) reverses the data in a file

20. The fscanf() function reads data from
(a) keyboard
(b) data file
(c) keyboard and data file
(d) none

21. when fopen function fails to open data file, it returns
(a) 1
(b) -1
(c) null
(d) none

22. which one of the following functions is used to detect the end of file
(a)fseek()
(b) ferror()
(c)feof()
(d)fgetc()

23.The value of EOF is
(a) -1
(b)0
(c)1
(d) null

24. The action of disconnecting a file from a program is obtained by the function
(a) fdisconnect()
(b) clear()
(c) fclose()
(d) delete()

25. ……… function is used to move the position indicator to a desired location within the file
a) fseek()
(b) ftell()

(c) rewind()
(d) fputc()

26 ……….. function can be used to find the end of the file maker
(a) fseek()
(b) ftell()
(c) rewind()
(d) none

27. To write strings in files, we use ……….. function.
(a) puts()
(b) gets()
(c) fputs()
(d) fgets()