UNIT-3

Preprocessor and File handling in C.

PREPROCESSOR:

Commonly used preprocessor commands like include, define, undef, if, ifdef, ifndef.

Preprocessor:

Preprocessor is a program that processes the source code before it passes through the compiler. It operates under the control of preprocessor directive which is placed before the main().

Before the source code is passed to compiler, appropriate actions are taken on the preprocessor directives.

preprocessor directives are lines but not statements for the preprocessor.

preprocessive directives are preceded by a hash sign (#) directive.

preprocessive directives are executed before the actual compilation of code. ie., preprocessor directives are executed before the main() program.

To extend a preprocessor directive to multiple lines place a backslash character (\)

preprocessor directive is placed before the main().

Advantages of using preprocessor directives in a C Program

- Program is readable and easy to understand.
- Program is easily modified or updated.
- Program becomes portable because easily compile in different execution environments.
- Program becomes more efficient.

Types of Preprocessor Directives

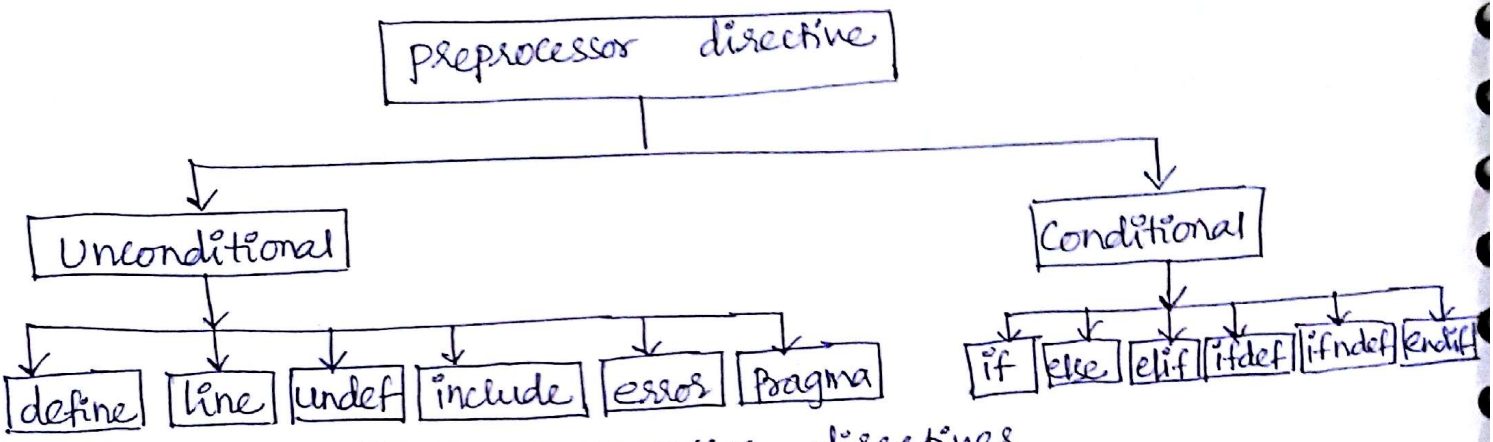

Figure: preprocessor directives

Preprocessor directives are categorized into two groups.

They are,

1) Unconditional preprocessor directives
2) Conditional preprocessor directives.

Unconditional preprocessor directives

#DEFINE

#define is a macro definition.

There are two types of #define macros. They are,

1) object like macro
2) Function like macro.

Object like macro

Eg: #define PI 3.14

Syntax: #define identifier string.

Here, PI is replaced by 3.14.

Function like macro

Eg(1) #define MUL(a, b) (a*b)

```
int a=2, b=3, c;
c = MUL(a,b);        // c=a*b;
```

Eg(2): #define SQUARE(x) (x*x)

Invoke the macro as

```
int a=2, b=3, c;
c = SQUARE(a);       // c = 2*2;
```

Here, above statement return the value 4.

Nesting of Macros

Using a macro in another macro is Nesting of Macros.

```
#define  SQUARE(x)     ((x) * (x))
#define  CUBE(x)       (SQUARE(x) * (x))
```

It can be nested to 31 levels.

Operators Related to macros

```
#include <stdio.h>
#define PRINT(num)  printf(#num "= %d", num)
main()
 {
   PRINT(20);
 }
```

Here, macro call expands to

```
printf("num = %d", num);
```

Eg: (2):

```
#define  MAC(x)     #x

MAC("10")     // string literal "10"
MAC("HI")     // string literal "HI".
```

Merge operator (##)

It concatenate / Append / Add two tokens into a third token.

```
#include<stdio.h>
#define  JOIN(A,B)     A##B
main()
{
   int i;
   for(i=1; i<=5; i++)
   printf("\n HI  JOIN(USER, i):");
}
```

output:

```
HI  USER1
HI  USER2
HI  USER3
HI  USER4
HI  USER5
```

#UNDEF

#undef directive undefines or removes a macro name previously created with #define.

Eg:
```
#define MAX 10
#undef MAX
```

#INCLUDE

#include directive includes the specified standard functions of definitions into our program. It can be included to our program in two ways. They are,

```
#include<stdio.h>        (or)
#include "stdio.h"
```

#include directive insert contents of specified file into the source code of a program

Syntax:
```
#include <file name>    (or)
#include "file name"
```

Eg:
```
#include "c:\students\my_header.h"
```

Nesting of include files can be 10 levels.

PRAGMA Directive

It is used to control the actions of Compiles.

#pragma directive is an instruction to the Compiles.

Syntax: #pragma string

Some pragma directives

COPYRIGHT
COPYRIGHT_DATE

pragma Copyright

#pragma COPYRIGHT "string"

Here, string Specifies set of characters included in Copyright message in the object file.

Eg: #pragma COPYRIGHT "JRT Software Ltd"

pragma Copyright_date

#pragma COPYRIGHT_DATE "2018-2019"

#LINE Control directive

It is used to give compiler messages of line numbers and file name. It tells the preprocessor to change compiler internally stored line number and file name.

#line directive is used to make the compiler provide error messages.

For example,

```
#include<stdio.h>
main()
{
   #line  10  "Error.c"
   int  a=10:
   #line 20
} printf("%d, a);
```

≡

```
#include<stdio.h>
main()
{
   int  a=10:
   printf("%d, a);
}
```

This code generates error as, Error in file "Error.c, line 10 and 20.

Syntax:

```
#line  line_number  "file name"
```

#ERROR Directive

It is used to produce compiles-time error messages.

Syntax: #error string

It is used to detect programming inconsistencies and violation of constraints during preprocessing.

When #error directive is encountered, the Compilation process and messages specified in string is printed to stderr.

For example, below code process the errors during preprocessing.

```
#ifndef SQUARE
#error MACRO not defined.
#endif
```

#error directive causes the preprocessor to report a fatal error.

It is used to determine whether a given line is compiled or not.

It ensures a particular macro has defined or not.

CONDITIONAL DIRECTIVES

A conditional directive is used to ~~check~~ instruct the preprocessor to select or not the code to the compiler.

#ifdef

It is a conditional preprocessor directive, it is used to check for existence of macro definitions.

Syntax:

```
#ifdef MACRO
    controlled text
#endif
```

If the MACRO is defined then preprocessor included output will be displayed.

Ex:
```
#ifdef  MAX
    int  STACK[MAX];
#endif
```

#else Directive

It is used with #ifdef and #ifndef

```
#ifdef  MACRO
        controlled text1;
#else
        controlled text2;
#endif
```

```
#if Condition
    Controlled text 1
#else
    Controlled text 2
#endif
```

#ifndef

It is opposite of #ifdef directive. It checks whether the MACRO has not been defined or if the definition is removed by #undef.

#ifndef is successful and returns a non-zero value, if the MACRO is not defined, otherwise #ifndef returns false (0).

#ifndef is executed when a MACRO is not defined.

Syntax:
```
#ifndef  MACRO
        Controlled text
#endif
```

#endif

It is used to mask the end of the #ifdef and #ifndef.

Syntax:
```
#ifdef  MACRO
        text
#endif
```

#if directive

- It is used to control compilation of portion of a source file

If the condition is true, it return non zero value then execute the #if directive block of statements.

Syntax:

```
#if  condition
     control text
#endif
```

#elif Directive

It appears between #if and #endif directives.

It is used when there are more two alternatives.

Syntax:

```
#if  condition
     controlled text1
#elif  new_condition
     controlled text2
#else
     controlled text3
#endif
```

Eg:

```
#define MAX 10
#if OPTION == 1
    int STACK[MAX];
#elif OPTION == 2
    float STACK[MAX];
#elif OPTION == 3
    char Stack[MAX];
#else
    printf("\n INVALID OPTION");
#endif
```

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.

FILE:

A File is a collection of related data.

- It is a permanent named unit.
- It is a permanent stored memory.
- File is stored as the data in CD's, Tapes, DVD's and Hard Disks.
- A Buffer is a temporary storage area in main memory.
- Buffer is used to transfer data from main memory to Secondary memory.

'C' provides a data structure called FILE, it is declared in file of stdio.h to access stream of characters for the disk files.

'C' provides a number of Library functions for I/O operations to perform basic file operations suchas opening a file, read/~~write~~ data from a file and write data to a file etc.,

- Every file fulfil 3 important points. They are,

1) Naming of a File:

- File name is a string of characters.

- Extension of a file depends on usage of a file, such as,

i) For the text file, we use .txt extension.

ii) For the 'c' program, .c extension will be used.

iii) For a Heades file, .h extension.

iv) For a c++, .cpp and for Java, .java.

2) Data structure of a File:

FILE is the data type defined for a file.

Syntax for declaring a file

FILE pointer type;

Eg: FILE *fp;

Here, fp is a pointer to the FILE structure. This acts as a link between operating system and a program.

3) purpose of a File:

It should be defined whether the file is to be read, written or append by defining the modes such as, 'r', 'w', 'a' respectively.

Text and Binary Files

Files are categorized into 2 types. They are, 1) Text File 2) Binary File.

Text File

A Text file stores data in the form of alphabets, digits and other special symbols by storing their ASCII values and they are in a human readable format.

For example, any file with a .txt, .c, etc., extensions.

- Data stored as series of bits
- Bits in text files represents characters
- Data is stored as line of characters with each line terminated by \n, which is translated into carriage return + line feed.
- Translation of new line character is carriage return + line feed.

Data can be read easily using any of the text editor.

Easily edit, create and delete using Notepad/any text editors.

Binary File

A Binary file contains a sequence or a collection of bytes which are not in a human readable format. Binary data is stored in the file cannot be read using any of the text editors.

For example: .bin file.

- Data stored as series of bits.
- Bits in binary files represent custom data.
- Data is stored on the disk in the same way as it is represented in the computer memory.
- No translation of any type.

Data can be read only by specific programs written for them. Data cannot read easily.

Data cannot easily edit but provides better security than text files.

Numbers of characters written/read in text file not same as on disk of external device.

~~Numbers of cha~~

WR

Number of characters written/read in memory same on external device of disk.

— Not one-to-one relationship of characters written/read in file and disk.

— There is one-to-one relationship between the characters written/read in a file and disk.

— Data 2345 requires 4 bytes. Here, each character occupies 1 byte.

— Data 2345 requires 2 bytes.

Creating, Reading and Writing Text and Binary Files.

Streams and Standard Input/out Functions

Stream:

Stream is a collection of characters for reading / writing data from/to the file.

- If the flow from Input device then Stream is called 'Input stream'.
- If the flow is to the output device then stream is called an Output stream.

- Streams are used to access the files efficiently.

File operations on Text/Binary files:
1) creating a new file
2) opening an existing file
3) Read/write to a file
4) close a file

1) Opening a file:

To use a file, it is to be opened by fopen() function. with 2 parameters in the function. They are,

i) file name ii) file open mode.

Syntax: fopen(file_name , file_open_mode);

Ex: fopen("myfile.txt" , "r");

fopen() function is defined in "stdio.h" header file.

- A File can be opened in "read" mode, "write" mode or "append mode" or combination of any two modes.

To open a file it should be existed in the secondary storage device then file is to be loaded into memory and it returns a file pointer of type FILE.

If file is not existed in secondary storage device, a macro in the header file "stdio.h" is called then NULL is returned.

2) Reading / Writing a File:

When the file is opened (ie., file is loaded into the main memory from secondary storage device) the associated pointer points to the starting character of the file content.

Reading data from a file:-

A File can be read using various functions provided by 'c' standard library. For example, fgetc().

Ex: ch= fgetc(fp);

fgetc() reads character pointed by fp, and the pointer position gets incremented.

Here, 'ch' holds the character (After read from the file).

Writing data into a file:-

TO write data in a File, it has to be opened then use fputc() function.

Ex: fputc(ch, fp);

Here, ch is the character to be written into the file.

3) closing file :-

After reading, writing or appending operations on a file, it has to be closed.

Syntax: fclose(file_pointer);

Ex: felose(fp);

Here, fp is a parameter.

To close morethan one file then fcloseall() function is used. This function is not require any parameter.

Syntax: fcloseall();

Ex: fcloseall();

Different Modes of operations:

1) "r" (Read) Mode:

This mode opens the existed file for reading only. If file does not exist then error occurs.

Syntax: fp= fopen("Read.txt", "r");

2) "w" (write) Mode:

This mode is used to write data into a file. Before, write data, a file has to be opened.

Syntax: fp= fopen("write.txt", "w");

3) "a" (Append) Mode:

To append (add) data to the existing file "a" mode is declared in the fopen() function.

The file pointer points to the end of the file of existed file then data can be appended.

Syntax: fp= fopen("Append.txt", "a");

4) "r+" (Read and Write) Mode:

This mode is declared for both reading and writing the data in the file.

If a file does not exist then NULL is returned to the file pointer.

Syntax: fp=fopen("Event.txt", "r+");

5) "w+" (Write and Read) Mode:

In this mode, writing and Reading operations can be done on the file. If a file is already existed then the contents of the file is eliminated. If the file does not exist then a new file is created.

Syntax: fp=fopen("Good.txt", "w+");

6) "a+" (Append and Read) Mode:

Data can be added and read the file by the use of "a+" mode.

Syntax: fp=fopen("Data.txt", "a+");

7) "wb" (Write Binary) Mode:

A binary file is opened in write mode.

Syntax: fp=fopen("Input.txt", "wb");

8) "rb" (Read Binary) Mode:

It is used to read a binary file.

Syntax: fp=fopen("output.txt", "rb");

9) "ab" (Append Binary) Mode:

It is used to add data at the end of Binary file.

Syntax: fp=fopen("Add.txt", "ab");

File Formatted Input/output Functions:

1) fputc() and fgetc() functions:

These Input/output functions are used to read/write a single character from/to stream.

fgetc() function:

It is a Input function, reads one character from input stream refered by fp.

Syntax: fgetc(fp);

fputc() function:

It is a output function, writes a character to a stream.

Syntax: fputc(char, fp);

Here, first argument is the character, second argument is the file pointer.

2) fgets() and fputs() functions:

fgets() function:

It is used to read a string.

Syntax: fgets(string, length, fp);

fputs() function

It is used to write a string.

Syntax: fputs(string, fp);

fputs() writes a string to the stream pointed by fp.

fgets() function requires 3 arguments. First argument is address of string in which read data is to be placed. Second argument is the length of the string to be read and the Last argument is the file pointer. fgets() function reads a string until either a new line character or number of characters given in the second argument.

3) fscanf() and fprintf() functions

File Formatted functions of fscanf() and fprintf() functions work on Disk files.

Syntax: fscanf(fp, "format string", argumentlist);

fprintf(fp, "format string", argumentlist);

These functions read/write from/to the Specified Stream pointed by fp. The argument list includes variables, Constants and strings.

fscanf() is used to read data from the disk file and fprintf() is to write data in disk file.

4) fread() and fwrite() functions:

fread() and fwrite() functions perform I/O operations on Binary files.

These functions read data from a file into a structure variable or to write data from a structure variable to a file.

Program for Read/Write operations on a file for character.

```
#
#
Void main()
{
   FILE  *f1;
   char c;
   clrscr();
   Printf("Data Input\n");
   f1=fopen("input.txt", "w");     //open the file input.txt
   while((c=getchar())!=EOF)   //Get a character from keyboard.
      putc(c, f1);
   fclose(f1);                 //close the file input.txt
   Printf("\n Data output");
   f1=fopen("input.txt", "r");  /*Reopen file input.txt */
   while((c=getc(f1))!=EOF)  /*Read a character from
                               file input.txt */
      printf("%c", c); //Display a character on screen
   fclose(f1);
   getch();
}
```

Program for Write a file and display the contents in a File.

```
Void main()
{
 Struct stu{ int rno, m1, m2, tot;
             float av;
           } st;
 FILE  *fp;
 clrscr();
 fp=fopen("stu.dat", "w");
 printf("Enter Roll Number\n");
 scanf("%d", &st.rno);
 Printf("Enter Marks\n");
 scanf("%d%d", &st.m1, &st.m2);
 st.tot = st.m1 + st.m2;
 st.av= st.tot/2.0;
 fprintf(fp, "%d\n%d\n%d\n%d\n%f", st.rno, st.m1, st.m2, st.tot,
                                                    st.av);
 getch();
}
```

output:

```
Enter Roll Number
   401
Enter Marks
   60
   70
^Z
```

ctrl	Z

```
C:\>type stu.dat

401
60
70
130
65.00
```

Writing to a text file

Program for write to a text file using fprintf()

```
#
int main()
{
    int num;
    FILE *fptr;
    fptr = fopen("C:\\program.txt", "w");
    if(ptr==NULL)
    {
        printf("Error!");
        exit(1);
    }
    printf("Enter num:");
    scanf("%d", &num);
    fprintf(fptr, "%d", num);
    fclose(fptr);
    getch();
    return 0;
}
```

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

After compile and run this program, a text file of program.txt created in C drive of your Computer. When you open the file, you can see the integer you entered.

Reading from a text file

Program for Reading from a text file using fscanf().

```
#
void main()
{
    int num;
    FILE *fptr;
    clrscr();
    if ((fptr= fopen("C:\\program.txt","r"))==NULL)
    {
        printf("Error! in opening file");
        exit(1); //program exits if file pointer returns NULL.
    }
    fscanf(fptr, "%d", &num);
    printf("value of n=%d", num);
    fclose(fptr);
    getch();
}
```

This program reads integer from program.txt file and prints it on the screen.

Writing to a Binary File using fwrite()

To write into a binary file, fwrite() is used.

This function takes 4 arguments.

Syntax:

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

4 arguments are,

Address of data to be written in disk.

Size of data to be written in disk.

Number of type of data

pointer to the file where you want to write.

Program for writing to a binary file using fwrite()

```
#
struct threeNum
{
 int n1, n2, n3;
};
void main()
{
 int n;
 struct threeNum num;
 FILE *fptr;
 clrscr();
 if ((fptr = fopen("C:\\program.bin", "wb"))==NULL)
 {
    printf("Error in opening file");
    exit(1); //program exits if file pointer returns NULL
 }
 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);
 getch();
}
```

Reading from a Binary file using fread()

```
#
void main()
{
struct threeNum
{
  int n1, n2, n3;
};
void main()
{ int n;
  struct threeNum num;
  FILE *fptr;
  clrscr();
  if ((fptr = fopen("c:\\program.obin", "rb"))==NULL)
  {
    printf("Error in opening file");
    exit(1);
  }
  for(n=1; n<5; ++n)
  {
    fread(&num, sizeof(struct threeNum), 1, fptr);
    printf("n1: %d\t n2: %d\t n3: %d", num.n1,
                                       num.n2, num.n3);
  }
  fclose(fptr);
  getch();
}
```

Program for copy source file into destination file by using Command line arguments

```
#
void main(int argc, char *argv[])
{
   FILE *f,*P;
   char a;
   clrscr();
   if(argc != 3)
    {
      printf("Invalid Numbes of Arguments\n");
      exit(0);
    }
   f=fopen(argv[1], "r");
   p= fopen(argv[2], "w");
   while( a=getc(f) != EOF)
     {
       fputc(a, P);
     }
   fclose(f);
   fclose(P);
   getch();
}
```

Output: After Compile & Run goto DOS SHELL.

c:\tc\bin> file name aa.txt bb.txt

Result output: The content of aa.txt is copied into bb.txt.

Streams

Stream is a logical interface to the devices (Keyboard and Monitor) and a file (program).

- Stream is a flow of collection of characters.

There are 3 streams. They are,

1) stdin (standard input)
2) stdout (standard output)
3) stderr (standard error)

stdin (standard input):

It is a stream from this program receives data by read operation.

Stdout (standard output)

It is a stream from this program writes output by write operation.

~~Standard~~ stderr (standard error)

It is an output stream, used to report error messages.

Random Access of Files using fseek(), ftell(), and rewind() Functions.

fseek(), ftell() and rewind() functions are used to access a part of a file.

fseek():-

This function is used to move the file pointer position to desired location within a file.

Syntax: fseek(file_pointer, offset, position);

It contains 3 arguments.

Here, offset is a long integer and position is an integer. offset is the new position of the file pointer relative to the position specified by the third argument position.

The argument position can take one of the following three values as shown in Table.

Value	Meaning	Defined as in stdio.h
0	Beginning of the file	SEEK_SET
1	current position of pointer position.	SEEK_CUR
2	End of file	SEEK_END

Table: Values of the Argument position.

The offset may be positive or negative.

- A positive value offset moves the pointer towards end of file.
- A Negative offset value moves pointer towards beginning of file.
- Negative offset is used only when third argument of function is 1 (current position).

- When the operation is successful, fseek() returns zero.

- An Error occurs if file pointer moves beyond file boundaries.

- fseek() function returns a non-zero value when performs an operation.

```
fseek(fp, 0L, 0);    // Pointer position Go to the beginning of the file.
fseek(fp, 0L, 2);    /* Go to the end of the file */
fseek(fp, 10L, 1);   // Move forwards 10 bytes from current position.
fseek(fp, -10L, 1);  /* Move backwards 10 bytes from the
                        current position */
fseek(fp, 10L, 2);   /* Move the file pointer 10 bytes from the
                        end of the file */
```

In stdio.h header file, the following constants are defined.

```
EOF = -1
NULL=0
```

EOF is the end of file marker, which is placed at the end of the disk file when the file is created.

EOF indicates end of the file that is nothing further to read.

ftell():-

- ftell() function takes file pointer as an argument and returns a current position in the file.

Ex: p= ftell(fp);

Here, p is offset, in bytes of the current position.

- ftell() returns 0 when file pointer position at the beginning of the file.

- when the current file pointer position is at the end of the file, the value of p is equal to the size of the file in bytes.

rewind():-

This function repositions the file pointer to the beginning of a file.

Eg: rewind(fp);

Rewind is done implicitly whenever a file is opened.

Example program for fseek():

Write a program to read last 'n' characters of the file using appropriate file functions (fseek() and fgetc()).

```
##
##
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\n");
        scanf("%d", &n);
        fseek(fp, -n, 2);
        while((ch=fgetc(fp))!=EOF)
        {
            printf("%c\t", ch);
        }
    }
    fclose(fp);
    getch();
}
```

Output: Last 2 characters of file1.c will be displayed on the screen.

Program for ftell(), rewind() and fseek()

```
#
#
Void main()
{
    FILE *fp;
    long int size; long int n;
    clrscr();
    fp = fopen("myfile.txt", "rb");
    if(fp==NULL)
     {
        printf("Error in opening file");
     }
    else
     {
        fseek(fp, 0, SEEK_END);
        size= ftell(fp);
        printf("size of myfile.txt : %ld bytes\n", size);
        rewind(fp);      /* same as fseek(fp, 0, SEEK_SET); */
     } n = ftell(fp); printf("present pointer position is: %ld", n);
    fclose(fp);
    getch();
}
```

Here, size displays the size of the file in bytes. and

n displays (returns) 0, because file pointer position is at the beginning of the file