Programming For Problem Solving

Unit-3

Preprocessors and File handling in 'c'

Preprocessor:- Commonly used preprocess-ors 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.

C preprocessor Directives:-

The 'C' preprocessor is micro preprocessor that is used by compiler to transform your code before compilation. It is called micro preprocessor because it allows us to add macros.

- All preprocessor directives starts with hash (#) symbol.

Note:- preprocessor directives are executed before compilation.

- There are different types of pre-processor directives which are supported by the 'C' programming language They are:- (i) macro
  (ii) hederfile inclution
  (iii) conditional compilaton
  (iv) other directives.

(i) **Macro:-** Macro define a constant value and can be any one of the basic data type.

- This can be declered with the symbol #defined comment

Syntax:-

```
#defined name_constant value
```

Ex:- #define pi_3.14.

- Let's see a list of pre processor directives

  → #include
  → #define
  → #undef
  → #Ifdef
  → #ifndef
  → #if
  → #else
  → #elif
  → #endif
  → #error
  → #pragma

#include:- The #include pre-processor directive is used to paste the predefined code of a given file into the current file.

- #include is used to define system defined heder files.
- If include file is not found then the compiler erasers error.
- It consists of the definations of pre-defined functions.

There are two variants to use #include directive

1. #include <file name> Eg:- #include <stdio.h>
2. #include "file name" Eg:- #include "stdio.h"

c #undef:- The #undef preprocessor directive is used to undefine the constant or macro defined by #define.

Syntax:-

```
#undef token
```

Example program: Let's see a simple example to define and undefine a constant

```
#include <stdio.h>
#define PI 3.14
#undef PI
main() {
   printf ("%f", PI);
}
```

C #ifdef:- The #ifdef preprocessor directive checks if macro is defined by #define. If yes, it executes the code otherwise #else code is executed, if present.

Syntax:-

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

#include directive Example:-

Let us see an simple example of #include directive. In this program we are including directive (i.e) stdio.h file because printf() function is defined in this program.

Example program:-

```
#include<stdio.h>
int main() {
 printf("Hello C");
return 0;
}
```

C#define:- The #define pre processor direction is used to define constant or micro substitution. It can use any basic data type.

Syntax:-

```
#define token Value
```

Eg:- #define pi 3.14.

Let us see an example of #define to difine a constant.

```
#include<stdio.h>
#define Pi 3.14
main() {
  printf("%f", PI);
}
```

output:- 3.140000

Syntax With #else:-

```
#if def MACRO
//successful code
#else
//else code
#end if.
```

Example program:- let's see a simple example to use #ifdef pre processor directive

```
#include<stdio.h>
#include<conio.h>
#define NO INPUT
void main()
{
 int a=0;
#ifdef NO INPUT
a=2;
#else
printf ("Enter a:");
scanf ("%d",&a);
#endif
printf ("value of a: %d\n", a);
getch();
}
```

Output:-

Value of a:-2.

C #ifndef:- The #ifndef preprocessor directive checks if macro is not defined by #define. If yes, it execute the code otherwise #else code is executed, if present.

Syntax:-

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

syntax with #else:-

```
#ifndef MACRO
//successful code
#else
//else code
#endif
```

Example program:- Let us see a simple example to usee #ifndef preprocessor directive.

```
#include<stdio.h>
#include<conio.h>
#define INPUT
 void main(){
int a=0;
#ifndef INPUT
 a=2;
 #else
printf("Enter a:");
scanf("%d",&a);
#endif
printf("Value of a: %d\n",a);
```

```
getch();
}
```

Output:-

```
Enter a:5
value of a:5
```

C#if:- The #if preprocessor directive evaluates the expression or condition. If condition is true, it execute the code otherwise #elseif or #else or #endif code is executed.

Syntax:-

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

Syntax with #else:-

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

Syntax with #elif & #else:-

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

Example program:- let us see a simple example to use #if preprocessor directive.

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

Output:-

Value of Number is : 0.

C#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 code
#elif expression
#elif code
#else
//else code
#endif.
```

Example program:- Let us see a simple program to use #else proprocessor directive.

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

Output:-

value of Number is non-zero.

C# error:- The #error preprocessor directive indicates error. The compiler gives fatal error if #error directive is found and skips further compilation process.

c#error example:-

let us see a simple example to use #error pre-processor directive.

```
#include<stdio.h>
#ifndef _MATH_H
#error First include then compile
#else
void main(){
 float a;
 a= sqrt(7);
 printf("%f", a);
}
#endif
```

Output:-

compile Time Error: First include then compile

C# pragma:- The #pragma preprocessor directive is used to provide additional information to the compiler. The #pragma directive is used by the compiler to offer machine or operating-system feature

Syntax:-

```
#pragma token
```

Example program:- let us see a simple example to use # pragma preprocessor directive.

```
#include <stdio.h>
#include <conio.h>
void func();
#pragma startup func
#pragma exit func
void main(){
printf("\nI am in func");
getch();
}
```

Output:-

```
I am in func
I am in main
I am in func
```

File handling in 'C':-

→ In programming, we may requires some specific input data to be generated several names number of times.

→ sometimes, it is not enough to only display the data on the console.

→ The data to be displayed may be very large, and only a limited amount of data can be displayed on the console, and since the memory is volatile, it is impossibl to recover the programmatically generated data again

and again. However, if we need to do so, we may store it onto the local file system which is volatile and can be accessed every time. Here, comes the need of file handling in C.

The following operations are performed on a file.

- Creating of new file
- opening an existing file
- Reading from the file
- Writing to the file
- Deleting the file.

Functions for file handling:-

There are many functions in the C library to open, read, write, search and close the file. A list of file functions are given below:

SNO	Function	Description
1.	fopen()	Opens new or existing file
2.	fprintf()	Write data into the file
3.	fscanf()	reads data from the file
4.	fputc()	writes a character into the file
5.	fgetc()	reads a character from file
6.	fclose()	closes the file
7.	fseek()	sets the file pointer to given position
8.	fputw()	Writes an integer to file
9.	ftell()	returns current position
10.	rewind()	sets the file pointer to beginning of the file

Implimenting file operations by using 'c' programming

To impliment file operations we need to learn the following syntax's for each and every file operations.

- **Creating of new file:-**

The file is created or declered to store the standard input and output data.

The following is the syntax for decleration of file pointer variable

```
FILE *file_pointer1;
```

- **Opening of existing file:-**

If any file is already created then we want to open that existing file by using the pre-defined function is called fopen() function.

syntax:-

```
type fopen ("file name", "mode of access");
```

- **Reading a file:-**

After opening a file to read the data from a file we can use a pre defined function called fgetc

syntax:-

```
fgetc (file_pointer);
```

- **Writing a file:-** If the file is created newly then we can insert the data into the file (or) if the file is already existed then the modify the existing data we can use the file writing operations by using a predefined function called as fputc.

syntax:-

```
fputc (character, file_pointer);
```

- **close operation:-** After performing all the required operations we must close the given file by using a predefined function is called fclose.

syntax:-

```
fclose (file_pointer);
```

Example program:- Write a 'c' program to write and read data from a file.

```
#include<stdio.h>
voidmain(){
FILE *fp;
char ch;
fp=fopen("data txt","w");
if (fp==NULL){
 printf("cannot open file");
exit(0);
}
printf("type text (to stop press '.'): ");
while (ch!='.'){
 ch=getche();
 fputc(ch,fp);
}
 fclose(fp);
 printf("In constants read:");
 fp=fopen("data.text","r");
while(!feof(fp))
```

```
    printf ("%d", fgetc(fp));
    fclose (fp);
}
```

Types of files:- There are four types of files which is supported by the 'c' programming language. They are:-

1. Text files
2. Binary files
3. Sequential files
4. Random Access file

1. Text file :-

- Text file is consists a stream of characteristics that can be processed sequentially in a forward direction
- Text file is also as -Flat file-.
- Text file formate is Basic file formate in 'c' programm -ing for giving as an input.
- Text file is simple sequence of ASCII characters.
- Each line is characterised by EOL character

[EOL -> End of line]

Example:- Notepad, Wordpad, excelseed etc.

- The text file have .txt Extension the compiler can take as input to a 'c' program by seeing the .txt extension and it understand the given input file is text file.
- The text files contains mostly english characters

Advantages:-

- Text file is very user friendly and it is easy to understand by the user.

Drawbacks:-

- Text files are moving only forward direction but its not possible to move random Access.
- Text file consists only sequence of characters but its not possible to insert images.
- To access text file it will take more time because the content in the file is there in characters instead of binary values (0's and 1's).

Modes of text file:- In 'c' programming we can open a file in different modes such as reading mode, writing mode and apending mode depending on the requirement of the user for handling a file

Following are the different modes of file:-

Opening Mode	purpose	Previous data
Reading	File will be opened just for reading purpose	Retained
Writing	File will be opened just writing purpose	Flushed
Appending	File will be opened for appending some thing in file	Retained

The mode of access on text file:-

To perform any operations on text file first we must create textfile (or) open a text file if it is already exists by using the following syntax:-

syntax:- fopen("file name", "mode of access");

The following are different modes of text file:

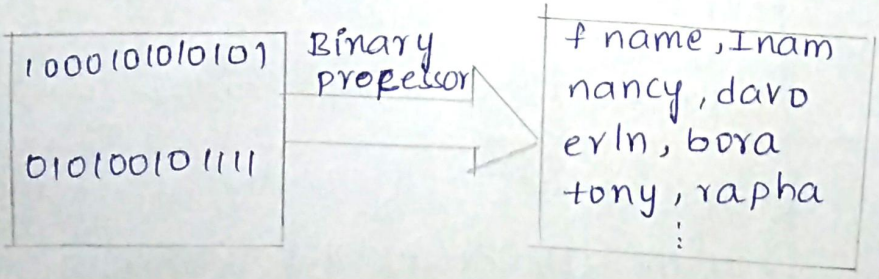
SNO	Mode of access	Purpose.
1.	"r"	→ Open a file in read mode → If file exist the marker is at position at beginning. → If file doesnot exist, error returned.
2.	"w"	→ Open a file in write mode → If file exists, all its data is erased. → If file doesnot exist, it is created.
3.	"a"	→ open a file in append mode → If file exist the marker is position at end. → If doesnot exist, it is created.
4.	"r+"	→ open a file in read and write mode → If the file exist, the marker is positioned at beginning. → If file doesnot exist., NULL returned.
5.	"w+"	→ Open a file in read and write mode. → If file exists, all its data is erased. → If file doesnot exists, it is created.

6.	"a+"	open a file in read and append mode file exist, the marker is positioned at end. If file doesnot exist, it is created.

2. Binary file:-

- It is a collection of bytes like images.
- Binary files contains the information mostly in binary formate (0s and 1s)
- Binary files are difficulty to read for user and it's difficulty to understand by the user because the binary files consists complex formate.
- The user can read the binary files only after processing
- The binary files can be specified with the .Bin extension

Ex:-



Explaination:- As shown in figure. Binary file is stored in Binary formate (in 0s and 1s). The Binary file is difficult to read for user. so Generally binary file is given as input to the Binary file preprocessor. processor will convert binary file into equivalent readable file.

Ex:- Executable files
Database files

Mode of access on binary files:-

The following mode of access are used to create or opening a binary file.

rb, wb, ab, rb+, wb+, ab+ are the modes to operator a file as binary file.

Ex:- fopen ("Bin", "wb");

Here the binary file can open right mode of access by using the symbol wb.

Text file vs Binary file:-

Text file	Binary file.
Data are human readable characters because text file data consist english language	Data is in the form of sequence of bytes (0's & 1's) so that it is very difficulty to read by the user.
Each line ends with new line character (EOL). EOL → End of the line	There are no lines or new line character.
Ctrl+z (or) ctrl+D is end of file character	An EOF marker is used to terminate binary file.
Data is read in forward direction only	Data may read in any direction
Data is converted in the internal formate (0's & 1's) before it is stored in memory.	Data stored in a file are in same formate that they are stored in memory.

3. Sequential File:-

- Data stored in sequentially, to read the last record of the file we need to traverse all the previous records before reaching the last record.

Ex:- Files on Magnetic tapes (CD & DVD's)

- Text files are also comes under the category of the sequential files.
- sequential files can move only forward direction
- sequential files can take more time to access records or data.

4. Random Access file :-

- Data can be accessed or modified randomly in any direction
- We can read any record directly in a random access file.

Ex:- Files on disks (hard disks in computer)

- The Random access files can takes very less time to access any record of a file when compare with Sequential Access file.

Operations On Random Access file:-

The following operations which are performed in generally on random access file and also on text files, binary files and sequential files they are:-

i) Creating a file (Random Access file)
ii) Opening Random Access file if exists (fopen)
iii) Reading a Random Access file (fgetc)
iv) Writing random Access file (fputc)
v) closing random Access file. (fclose).

The following are some of the special operations which we performed on only random Access file. They are

1) fseek() function
2) ftell() function
3) Rewind() function

1) fseek() function:

→ The fseek() function is used to set the file pointer to the specified offset (to the specified location)
→ It is used to write data into file at desired location

Syntax:- The following is syntax for fseek function

```
int fseek (FILE *stream, long int offset, int whence)
```

Here int is written type of fseek() function
FILE *stream is the parameter of fseek() function to specify the location of Random Access file.
offset is one of the parameter to specify the desire location in a random Access file whence is one of the Parameter of fseek() function which consists 3 constants.
(i) SEEK_SET (ii) SEEK_CUR (iii) SEEK_END.

Example program:-

```
#include<stdio.h>
void main() {
 FILE *fp;
fp= fopen ("myfile.txt", "w+");
fputs ("This is amar", fp);
fseek (fp, 7, SEEK_SET);
fputs ("rajesh", fp);
fclose (fp);
}.
```

2) ftell() function:-

→ The ftell() function returns the current file position of the specified steam.

→ We can use ftell() function to get the total memory size of a file after moving a file pointer at the end of the file.

→ We can use SEEK_END constant to move the file pointer at the end of the file.

syntax:-

The following is the syntax for ftell() function

```
longint ftell (FILE* stream).
```

Example program:-

```
#include<stdio.h>
#include<conio.h>
void main(){
  FILE *fp;
  int length;
  clrscr();
 fp= fopen("file.bin","rb");
 fseek(fp,0,SEEK_END);
  length=ftell(fp);
  fclose(fp);
  printf("size of file :%d bytes",length);
  getch();
}
```

Output:- size of a file is : 21 bytes

3) Rewind() function :-

→ The Rewind() function sets the file pointer at the beginning of a stream or file.

→ It is useful if you have to use stream many times.

syntax:

The following is the syntax for rewind() function.

```
void rewind(FILE *stream)
```

Ex: File: file.txt

Example program:-

```
1. #include<stdio.h>
2. #include<conio.h>
3. Void main(){
4. FILE *fp;
5. char c;
6. clrscr();
7. fp=fopen("File.bin","r");
8.
9. while((c=fgetc(fp))!=EOF){
10. printf("%c",c);
11. }
12.
13. rewind(fp); // moves the file pointer at beginning of file.
14.
15. while((c=fgetc(fp))!=EOF){
16. printf("%c",c);
17. }
18.
19. fclose(fp);
20. getch();
21. }
```

Output:-

This a simple text. this is a simple text.