UNIT-V

Pre Processor and File Handling

5. Preprocessor:

The Preprocessor is not a Part of the Compiler but it is Separate Step in the Compilation Process. In Simple terms a C Preprocessor is just a text Submission tool and its instructs the Compiler to be required Pre Processing before the actual Compilations

All Preprocessor Commands begin with a hash Symbol (#).

It must be the first non blank characters and for readability a Preprocessor directive Should begin in the first Column.

5.1.1 Commonly used Preprocessor like

Include

The Preprocessor to be used Commonly for directives

#define

It Substitute a Preprocessor Macro.

2) #include

It inserts a particular header from another file

3) #undef

It undefines a preprocessor macro

4) #ifdef

It returns true if this macro is defined

5) #ifndef

It returns true if this macro is not defined

6) #if

It tests if the compile time conditions is true.

7) #else

8) # elif

9) #endif

It end preprocessor conditional statement

10) # Error

It Prints Error Message on Stder

11) # Pragma

It issues Special Commands to the Compiler using a Standardized Methods

The PreProcessor understood various directives

```
#define MAX_ARRAY_LENGTH 20
```

This directive tells the CPP to replace instances of MAX_ARRAY_LENGTH with 20 use #define for Constants to increase readability

```
#include <stdio.h>
#include "myheader.h"
```

These directive tell CPP to get stdio.h from System Libraries and add the text to the Current Source file. The next line tells CPP to get myheader.h from the local directory and the Content of to Current

```
#undef FILE_SIZE
#define FILE_SIZE 42
```

It tells the CPP to undefine existing FILE_SIZE and define is as 42

5.1.2 if ifdef ifndef

1) #ifdef

This directive is the Simplest Conditional directive. This block is called a Conditional directive. The Controlled text will get include in the Preprocessor out. If the Macroname is defined. The Controlled text inside Conditional will embrace PreProcessing directives. They are executed only if the Conditional Succeed. you can nest these in Multiple layers, but they must be Completed.

"#endif" always matched the nearest #ifdef.

Syntax: #ifdef MACRO

Controlled text

#endif /* Macro name */

2) #ifndef

We know that the #ifdef directive if the Macro name is defined. then the block of statements following the #ifdef directive will execute normally but if it is not defined, the Compiler will simply skip this block of statements. The #ifndef directive is simply opposite to that of the #ifdef directive. In case of #ifdef, the block of statement between #ifndef and #endif will be Macro.

```
ifdef Macro_name
    Statement_1
    Statement_2
    Statement_3
    ------
    Statement_N
endif
```

3) #if #else and #elif

These directives works together and Control Compilation of portions of the Program using Some Conditions.

If the Conditions with #if directive evaluates to nonzero value then the group of line immediately after the #if directive will be executed. Otherwise if the Conditions with the #elif directive evaluates to a non zero value then the group of line immediately after the #elif directive will be executed else the lines after #else directive will be executed #define directive then only.

Syntax:

```
#if macro-Condition
   Statements
#elif macro Condition
   Statements
#else
   Statements
#endif
```

5.1.3 C Macros

A Macro is a segment of code which is replaced by the value of Macro.

Macro is defined by #define directive

There are 2 types of Macros

1) object-like Macros
2) function-like Macros

Object Like Macros:

The object-like Macro is an identifier that is replace by value It is widely used to Present numeric Constants

Eg: #define Pi 3.14

Here Pi is the Macro name which will be replaced by the value 3.14

function-like Macros:

The function_like Macro looks like function Call

Eg: #define MIN (a,b) ((a)<(b)? (a) : (b))

there MIN is the Macro name.

ANSI C defines Many Predefined Macros that Can be used in C Programs

NO	Macro	Description
1	-DATE-	Represent Current date MMDDYYYY
2	-TIME-	represent Current time in HH:MM:SS
3	-FILE-	represent Current filename
4	-LINE-	represent Current line number
5	-STDC-	It defined as when Compiler Compiles with the ANSI

5.2 FILES:

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

5.2.1 Text Files

The file has supported as textfiles and the binary files.

The textfile Contains textual information in the form of alphabets, digits and special characters or Symbol.

When you need to create the text files in I/o C program

fopen – opens a text files

fclose – closes a textfiles

feof – detects end_of_file Marker in a file

fprintf – Prints formatted output to a file

fScanf – reads formatted input from a file

fputs – Prints a string to a file

fgets – reads a String from a file,

fputc – Prints a Character to a file

fgetc – reads a Character from a file

5.2.2 Binary files:

A binary files is a file that uses an 8 bits of a byte for storing the information It is the form which can be interpreted and understood by the Computer.

The only difference between the text files and binary files can be recognized by the word processor while binary file data cannot be recognized by a word processor.

1) wb (write)

This opens a binary file in write mode.

Syntax:

```
fp = fopen ("data.dat", "wb");
```

2) rb (read)

This opens a binary file in read mode

Syntax:

```
fp = fopen ("data.dat", "rb");
```

3) ab (append)

This opens a binary files in a append mode i.e data can be added at the end of file

Syntax:

```
fp = fopen ("data.dat", "ab");
```

4) r+b (read + write)

This mode opens preexisting file in read and write mode.

Syntax:

```
fp = fopen("data.dat", "r+b");
```

5) w+b (Write + read)

This mode creates a new file for reading and writing in binary mode

Syntax:

```
fp = fopen("data.dat", "w+b");
```

6) a+b (append + write)

This mode opens a file in append mode data can be written at the end of the file

Syntax:

```
fp = fopen("data.dat", "a+b");
```

5.3 File Structure

A file is a type of structure typedef as file. It is considered as opaque data data type as its implementation is hidden.

Eg:

```
#include <stdio.h>
#include <stdlib.h>
    int main()
    {
        char data[10];
        FILE *fp1, *fp2;
if(fp1 = fopen("read.txt", "r") == NULL)
 {
 Printf("Error! opening file");
  exit(1);
 }
if(fp2 = fopen("write.txt", "w") == NULL)
 {
  Printf("Error! opening file");
  exit(1);
 }
while((fgets(data, 100, fp1) != NULL)
 fputs(data, fp2);
 fclose(fp1);
 fclose(fp2);
 return 0;
}
```

5.4 Creating Reading and Writing text and Binary files.

5.4.1 Creating text files:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    FILE *fp;
    char fname[20];
Printf("Enter filename to Create");
Scanf("%s", fname);
fp=fopen(fname, "w");
if(fp==NULL)
{
Printf("File does not Created");
  exit(0);
}
Printf("file Created Successfully");
  return 0;
}
```

Output:

Enter filename to Create: cc.txt

File Created Successfully.

5.4.2 Reading text file

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  char c[1000];
  FILE *fptr;
if((fptr=fopen("p.txt","r"))==NULL)
{
Printf("Error! File cannot be opened");
  exit(1);
}
fScanf(fptr, "%[^|n]", c);
Printf("Data from the file\n%s", c);
  fclose(fptr);
  return 0;
}
```

Output: Data from the file.

C Programming

5.4.3 Writing textfile

To write a file in C using fprintf()

```
#include <stdio.h>
#include<stdlib.h>
int main()
{
 Char Sentence[1000];
 FILE *fptr;
```

5.4.2 Reading text file

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  char c[1000];
  FILE *fptr;
if((fptr=fopen("P.txt","r"))==NULL)
{
Printf("Error! File cannot be opened");
  exit(1);
}
fScanf(fptr,"%[^|n]",c);
Printf("Data from the file\n%s",c);
  fclose(fptr);
  return 0;
}
```

Output: Data from the file.

C Programming

5.4.3 Writing text file

To write a file in C using fprintf()

```
#include <stdio.h>
#include<stdlib.h>
int main()
{
 Char Sentence[1000];
 FILE *fptr;
```

```
fptr = fopen("p1.txt", "w");
if(fptr == NULL)
{
  printf("Error");
  exit(1);
}
printf("Enter a Sentence: \n");
fgets(Sentence, sizeof(Sentence), stdin);
fprintf(fptr);
fprintf(fptr, "%s", sentence);
fclose(fptr);
  return 0;
}
```

Output:

Enter a Sentence : C Programming

Here a file named P1.txt is created

5.5 Appending data to existing files.

Appending a file refers to process that involves adding new data elements to an existing database

An example of a common file append would be the enhancement of a company customer files. Companies often collection basic information on their clients such as phone numbers etc.

Program:

```
#include<stdio.h>
int main()
{
  FILE *fp;
  char ch;
Char *filename = "f.txt";
Char * Content = "This text is appended
                  Data to the file"
fp=fopen (filename, "r");
Printf ("In Contents of %s", filename);
while ((ch == fgetc(fp)) != EOF)
  {
    Printf ("%c", ch);
  }
fclose (fp);
fp =fopen (filename, "a");
fPrintf (fp, "%s\n", Content);
fclose (fp);
fp= fopen(filename, "r");
Printf("In Contents %s", filename);
while ((ch=fgetc(fp))!=EOF)
  {
    Printf("%c", ch);
  }
  fclose (fp);
  return 0;
}
```

Output: This text is already there in the file.

5.6 Writing and Reading structures using binary files.

The Writing into the binary file code Segment data into file in binary format

```
Struct emp
{
  Char ename[20];
  int age;
  float basicSalary;
};
Struct emp e;
fp = fopen ("Emp.dat", "wb");
  if (fp == NULL)
  {
   Puts ("Cannot to open file");
  }
Printf ("Enter name, age, Salary");
Scanf ("%s %d %f", e.name &e.age, &e.basicSalary);
fwrite (&e, sizeof (e), 1, fp);
fclose (fp);
}
```

Reading from the binary file

The following code segment illustrates how to read data from file binary format

```
fp = fopen ("emp.dat", "rb");
while (fread (&e, sizeof (e), 1, fp)
{
printf ("%-5%d %-f\n", e.name, e.age,
                        e.basicSalary);
}
fclose (fp);
```

5-7 File Status functions

C provides three functions to handle file Status

1) test end of file (feof)
2) test Error (ferror)
3) clear Error (clearerr)

1) Test end of file (feof)

This function is used to check if the end of the file has been reached.

If the file is at the end that if it all data have been read the function returns non zero (true)

If the end of file has not been reached zero file (false) is returned

```
int or feof (FILE * stream);
```

2) Test Error (ferror):

Test Error is used to check Error Status of the file.

Errors can be created of many reasons like bad physical media (disk or tape) Illegal operations such as reading in writing mode. The ferror returns true (nonZero) if ferror has occured otherwise false (zero)

3) Clear Error (clearError):

When an Error occurs the Subsequent Calls to ferror returns nonZero, until the Error status of the file is latest The function clear is used for this Purpose.

```
void clearall (FILE * stream);
```

5.8 File positioning function

1) Rewind:

The rewind function simply sets the file position indicator to the beginning of the file

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

2) ftell:

The ftell function reports the current position of the file marker in the file.

```
long int ftell(FILE *stream)
```

3) fseek:

fseek function positions the file location to specified byte position in a file. Seek moves the disk to a position in the file for reading or writing.

```
int fseek(FILE *stream, long offset,
          int wherefrom);
```

File Input/output operations

There are different operations that can be carried out on a file.

They are:

1) Creation of a new file.
2) opening an existing file.
3) Reading from a file
4) writing to a file
5) Moving to a specific location in a file
6) Closing a file

1) Creation of a new file:

A new file is created if it is doesnot exist

2) opening an existing file:

An existing file is opened using fopen()

```
fopen("filename", "mode");
```

3) Reading from a file

It reads a specified number of bytes from a binary file places them into memory at the specified location

```
int fread (void *PIN, Area)
        int elementSize;
        int Count, FILE *SP);
```

PIN Area → pointers to the input are in Memory

4) Writing into a file:

It writes a specified number of items of a binary file.

```
int fwrite (void *pout area,
        int element Size, int Count,
        FILE *SP);
```

5) Moving to a specific location in a file

The Contents can be moved from one file to another file using fcopy

6) closing a file:

A file is closed using fclose() function

```
FILE *SPTemP;
fclose(SPTemP);
```