Lecture 8

Pre-Processor,Command Line Argument

The Pre-processor

A unique feature of c language is the pre-processor. A program can use the tools provided by pre-processor to make his program easy to read, modify, portable and more efficient.

Pre-processor is a program that processes the code before it passes through the compiler. It operates under the control of pre-processor command lines and directives.

Preprocesses directives

Preprocessor directives follow the special syntax rules and begin with the symbol #and do not require any semicolon at the end. A set of commonly used preprocessor directives

Directives	Function
#define	Defines a macro substitution
#undef	Undefined a macro
#include	Specifies the files to be included
#ifdef	Tests for a macro definition
#endif	Specifies the end of #if
#ifndef	Tests whether a macro is not defined
#if	Tests a compile-time condition
#else	Specifies alternatives when #if test fails

Preprocesses directives

The preprocessor directives can be divided into three categories

1. Macro substitution division
2. File inclusion(Including) division
3. Compiler control division

The Preprocessor

MACRO SUBSTITUTION

This is a process of replacing an identifier of a C program by a constant or a symbolic constant. This can be accomplished by the directive #define. The #definition is a macro definition statement.

Syntax:

#define identifier CSCE

Where,

#define — A hash define directive

Identifier — A valid C identifier, conventionally written in upper case

CSCE — May be constant, symbolic constant or expression.

The Preprocessor

Example:

1. **#define MARKS 100**
2. **#define AVERAGE 72.20**
3. **#define PI 3.142**

Example Program

```
#include<stdio.h>
#define PI 3.142
main()
{
        float rad,area;
        printf("enter the radius");
        scanf("%f",&rad);
        area= PI*rad*rad;
        printf("area of a circle=%f",area);
}
```

In this program, the macro PI is defined as 3.142. hence, whenever PI occurs in a C program, it is replaced by the value 3.142.

The Preprocessor

FILE INCLUSION

This is a process of inserting external files containing functions or macro definitions into the C program. An external file containing function can be include in a C program using the #include directive.

Syntax:

```
# include filename
```

Where,

include → Preprocessor directive for file inclusion

Identifier → Name of the file containing the required function definition to be include in a C program

The Preprocessor

FILE INCLUSION

There is no space between #symbol and include. But,there must be at least one blank space between #include and filename. Filename can be written within a pair of a angle brackets or double quotation marks.

<filename>

Example:

1. #include <stdio.h>
2. #include <math.h>
3. #include “Grade.c”

When the filename is written within a pair of angle ,the specified file is searched only in the standard directories .But, when the filename is written in a pair of double quotes ,the specified file is first searched in the current directory and then in the standard directories.

The Preprocessor

CONDITIONAL DIRECTIVES

C processor provides a conditional compilation directive which is used to select alternate segments of code in a C program depending upon the condition.

Suppose there are two different version of a program, and they are more alike than are different. It will be redudant if you maintain both versions .

We can overcome this problem by simply including both version in a single program. then, it would be possible to select a particular version depending on the requirement.

The Preprocessor

CONDITIONAL DIRECTIVES

Example:

```
Main()
{
……..
#ifdef VERSION1
{
……..
……..                    Code of VERSION 1
……
}
#else
{
…….
…….                     Code of VERSION 2
……

…….
}
#indif
……..
}
```

If we want to execute VERSION1 of a program then the following statement must be included in this program.

#define VERSION1

The main( ) function

- So far, we have been defining the main() function to receive no arguments.
- Actually, the main( ) function can receive two arguments.
- To do that, the main( ) function should be defined as below.
- This is how arguments can be passed in at the command line.

```
int main(int argc, char *argv[ ])
{   …

}
```

Command Line Arguments

C language programs also creates an executable file(.exe).this executable file has the same name as the program file name and stores the executable code of the program. The creation of executable files saves the user to compile a program again and again.

Other advantage with executable file is their execution on DOS prompt. And they does not required C-editor .

We can execute any of the C program from DOS prompt .But the use of Command Line Arguments .

Command Line Arguments is much useful for filehandling program execution because the file name can be supplied at command prompt.

Command Line Arguments

Now ,how main() function would recognize /accept the file name from the DOS prompt to operate?

So, file name supplied at command prompt are used as **parameter by the main()function** and these parameters are called as **“Command Line Arguments”.**

These arguments are captured by the main () funcation of the program and supplied to the programming statements for future processing.

Command Line Arguments

For example:

A programmer creates a program file “Readf” to read the contents of a file. Then the following command line may be used to invoke this program.

C:\>Readf SFILE.TXT

The “Readf” file contains executable code of the program(Readf.exe) and the filename “SFILE.TXT” will be accepted by main() function as the command line argument.

For this purpose ,main() function can take two arguments called **argc and argv**

Command Line Arguments

- The argument variable argc is an integer variable and acts like "argument counter" which counts the number of arguments on the command line

For example:

According to last example ,the variable argc contains numeric value 2.

- The argument variable argv is a character array of pointers so called "argument vector". Each subscript of this array points to each command line argument respectively.

For example:

According to last example ,the variable argv is the array of two variable (equals to the value of argc) and point the command line arguments as follows:

Argv[0]- points to "Readf"
Argv[1]- points to "SFILE.TXT"

Command Line Arguments

```
int main(int argc, char* argv[])
{
……;
……;
……;
}
```

argc

Number of arguments (including program name)

argv

Array of char*s (that is, an array of ‘c’ strings)

argv[0]: = program name

- **argv[1]**: = first argument
- **…**

argv[argc-1]: last argument