CHAPTER 13

C Language - The Preprocessor

C Language - The Preprocessor

In this tutorial you will learn about C Language - The Preprocessor, Preprocessor directives, Macros, #define identifier string, Simple macro substitution, Macros as arguments, Nesting of macros, Undefining a macro and File inclusion.

The Preprocessor

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

Preprocessor is a program that processes the code before it passes through the compiler. It operates under the control of preprocessor command lines and directives. Preprocessor directives are placed in the source program before the main line before the source code passes through the compiler it is examined by the preprocessor for any preprocessor directives. If there is any appropriate actions are taken then the source program is handed over to the compiler.

Preprocessor directives follow the special syntax rules and begin with the symbol #bin column1 and do not require any semicolon at the end. A set of commonly used preprocessor directives

Preprocessor directives:

Directive	Function
#define	Defines a macro substitution
#undef	Undefines a macro
#include	Specifies a file to be included
#ifdef	Tests for macro definition
#endif	Specifies the end of #if
#ifndef	Tests whether the macro is not def
#if	Tests a compile time condition
#else	Specifies alternatives when # if test fails

The preprocessor directives can be divided into three categories

1. Macro substitution division
2. File inclusion division
3. Compiler control division

Macros:

Macro substitution is a process where an identifier in a program is replaced by a pre defined string composed of one or more tokens we can use the #define statement for the task.

It has the following form

#define identifier string

The preprocessor replaces every occurrence of the identifier int the source code by a string. The definition should start with the keyword #define and should follow on identifier and a string with at least one blank space between them. The string may be any text and identifier must be a valid c name.

There are different forms of macro substitution. The most common form is

1. Simple macro substitution

2. Argument macro substitution
3. Nested macro substitution

Simple macro substitution:

Simple string replacement is commonly used to define constants example:

```
#define pi 3.1415926
```

Writing macro definition in capitals is a convention not a rule a macro definition can include more than a simple constant value it can include expressions as well. Following are valid examples:

```
#define AREA 12.36
```

Macros as arguments:

The preprocessor permits us to define more complex and more useful form of replacements it takes the following form.

```
# define identifier(f1,f2,f3…..fn) string.
```

Notice that there is no space between identifier and left parentheses and the identifier f1,f2,f3 …. Fn is analogous to formal arguments in a function definition.

There is a basic difference between simple replacement discussed above and replacement of macro arguments is known as a macro call

A simple example of a macro with arguments is

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

If the following statements appears later in the program,

```
volume=CUBE(side);
```

The preprocessor would expand the statement to

```
volume =(side*side*side)
```

Nesting of macros:

We can also use one macro in the definition of another macro. That is macro definitions may be nested. Consider the following macro definitions

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

Undefining a macro:

A defined macro can be undefined using the statement

```
# undef identifier.
```

This is useful when we want to restrict the definition only to a particular part of the program.

File inclusion:

The preprocessor directive "`#include file name`" can be used to include any file in to your program if the function s or macro definitions are present in an external file they can be included in your file

In the directive the filename is the name of the file containing the required definitions or functions alternatively the this directive can take the form

```
#include< filename >
```

Without double quotation marks. In this format the file will be searched in only standard directories.

The c preprocessor also supports a more general form of test condition #if directive. This takes the following form

```
#if constant expression

{
statement-1;
statemet2’
….
….
}
#endif
```

the constant expression can be a logical expression such as test < = 3 etc