Lesson 6 The C Preprocessor

Introduction

The C Preprocessor is a program that is executed just before the source code is presented to the compiler. One can think of preprocessor directives as a language within the C language. Normal program statements are instructions to the microprocessor, preprocessor directives are instructions to the actual compiler. It's actions consists of

- replacement of defined identifiers by pieces of text
- conditional selection of parts of source code
- inclusion of other files
- renumbering and renaming of source code lines and files.

Using the #define Preprocessor Directive

Every instruction to the preprocessor is known as a directive. The instruction must occupy a line of its own and must also start with the `#` character. A directive may appear at any place in a source file.

Consider the following code to calculate the area of a circle

```
#define PI 3.14159
#include<stdio.h>

int main(void)
    {
     float radius;
     printf(" type in the radius\n");
     scanf("%f",&radius);
     printf("area = %f \n",PI*radius*radius);
     return(0);
    }
```

It should be clear to you that every time the phrase PI is meet, it is replaced with the value 3.1459. PI is called the identifier and 3.1459 is called the text. You could have written similar code using a variable for PI instead of using #define, however this should not be encouraged. The reasons are as follows, firstly, PI is a constant throughout the code and so should be treated as one. Secondly, by declaring PI as a variable this leads to the possibility of its value been changed somewhere in the code. Finally, the compiler can generate faster or better optimised code using constants rather than when using variables.

Using Macros

The #define can also be used in a similar manner to a function call. It has the ability to take arguments. This is achieved as follows.

#define identifier(arg1,arg2.....) text

We can modify the above program without using a function, instead we use a macro with an argument.

```
#define PI 3.14159
#define AREA(x) (PI*x*x)
#include<stdio.h>
int main(void)
    {
        float radius;
        printf(" type in the radius\n");
        scanf("%f",&radius);
        printf("area = %f \n",AREA(radius));
        return(0);
    }
```

It is advisable to use parentheses when dealing with macros. Here is an example of what might go wrong if you don't.

Write a macro called ADD that adds two numbers. Include this in a program which contains the line *ans=2*ADD(3,8)*. Test and see the output when, firstly parentheses are used and secondly no parentheses are used around the text.

Macros or Functions ?

The answer to this question depends on your program. However, the following should be kept in mind. Macros generate more code than functions, but they can execute faster than functions. Marcos will not be typed checked by the compiler but functions will.

Using #include preprocessor directive

The #include preprocessor directive is used to include one source file into another source file. This can be useful when for example you have written code which you may wish to use in another program or perhaps created a library of functions.

Suppose you have a file with many #define statements for various constants and formulae. We will call this file say "file.h" (where h stands for header). All of the #define statements can be accessible in your current file by including *#include "file.h"* at the beginning of your current file.

Its also acceptable to use *#include <file.h>*. The difference between the two is that the former will search for the file in the same directory as the new file whereas the latter will search for the file in the standard header directory.

Conditional Compilation of Source Code #ifdef #else #endif

The preprocessor allows the compilation of selected pieces of source code. An example may arise if your source code is written for say two different machines. For example a Windows PC environment and a Linux machine environemnt. The following code shows how this can be achieved

```
#define PC
 /* by defining only one of PC or LINUX we can determine
what section of code gets executed */

#define LINUX

#ifdef PC
          /* lines of code  */
#else
#ifdef LINUX
          /* lines of code  */
#endif
```

Using The #undef Preprocessor Directive

The #undef directive cancels the previous #define directive. Thus if you had used ***#define ON*** then, if later you wish not to use it, the following line will do this ***#undef ON*** . Think of an example where this may be useful ?

Programming Exercises

1. Write a program that will count from 20 to -15 by counting backwards. Use #define statements to define the upper and lower limits.

2. Explain briefly in your own words how the C Preprocessor works.
3. Using the #define directive, write a short piece of code to calculate the circumference of a circle.
4. You are asked to write a program that uses the #define directive which accepts two arguments used to find the area of a rectangle.
5. Explain how the #include works. Think of an example where it may be used in conjunction with your own source code.
6. Write a program which uses the #define and #undef directives as a switch to print out either your first name only or your full name.