# WEEK 15 Programming For Problem Solving

## Discuss Conditional compilation with examples.

In C programming, you can instruct preprocessor whether to include a block of code or not. To do so, conditional directives can be used.

It's similar to a if statement with one major difference.

The if statement is tested during the execution time to check whether a block of code should be executed or not whereas, the conditionals are used to include (or skip) a block of code in your program before execution.

### Uses of Conditional

* use different code depending on the machine, operating system
* compile same source file in two different programs
* to exclude certain code from the program but to keep it as reference for future purpose

### How to use conditional?

To use conditional, #ifdef, #if, #defined, #else and #elif directives are used.

### #ifdef Directive

#ifdef MACRO

// conditional codes

#endif

Here, the conditional codes are included in the program only if MACRO is defined.

### #if, #elif and #else Directive

#if expression

// conditional codes

#endif

Here, expression is an expression of integer type (can be integers, characters, arithmetic expression, macros and so on).

The conditional codes are included in the program only if the expression is evaluated to a non-zero value.

The optional #else directive can be used with #if directive.

#if expression

conditional codes if expression is non-zero

#else

conditional if expression is 0

#endif

You can also add nested conditional to your #if...#else using #elif

#if expression

// conditional codes if expression is non-zero

#elif expression1

// conditional codes if expression is non-zero

#elif expression2

// conditional codes if expression is non-zero

#else

// conditional if all expressions are 0

#endif

### #defined

The special operator #defined is used to test whether a certain macro is defined or not. It's often used with #if directive.

#if defined BUFFER\_SIZE && BUFFER\_SIZE >= 2048

// codes

1. Write a program in C that illustrates the use of macros to find the area of a rectangle.

#include <stdio.h>

// Macro to calculate the area of a rectangle

#define RECTANGLE\_AREA(length, width) ((length) \* (width))

int main() {

// Variables to store the length and width of the rectangle

float length, width;

// Get user input for the length and width

printf("Enter the length of the rectangle: ");

scanf("%f", &length);

printf("Enter the width of the rectangle: ");

scanf("%f", &width);

// Calculate the area using the macro

float area = RECTANGLE\_AREA(length, width);

// Display the result

printf("The area of the rectangle with length %.2f and width %.2f is: %.2f\n", length, width, area);

return 0;

}

In this program, the **RECTANGLE\_AREA** macro is defined to take two parameters (**length** and **width**) and calculates the area of the rectangle using the formula **length \* width**. The macro is then used in the **main** function to calculate and display the area based on user input for the length and width of the rectangle.

1. List a few predefined macros.

| **Macro** | **Value** |
| --- | --- |
| \_\_DATE\_\_ | A string containing the current date |
| \_\_FILE\_\_ | A string containing the file name |
| \_\_LINE\_\_ | An integer representing the current line number |
| \_\_STDC\_\_ | If follows ANSI standard C, then the value is a nonzero integer |
| \_\_TIME\_\_ | A string containing the current date. |

1. What's pragma?

The ‘#pragma’ directive is the method specified by the C standard for providing additional information to the compiler, beyond what is conveyed in the language itself. The forms of this directive (commonly known as *pragmas*) specified by C standard are prefixed with STDC. A C compiler is free to attach any meaning it likes to other pragmas. Most GNU-defined, supported pragmas have been given a GCC prefix.