Faculty of Information Technology, University of Moratuwa
BSc. (Hons) in Information Technology
BSc. (Hons) in Artificial Intelligence
Fundamentals of Programming IN1101

Level 1 – Semester 1 Macros

Introduction

In this lab session, you'll learn Preprocessor and how to write a macro in C.

The C Preprocessor is not part of the compiler but is a separate step in the compilation process. In simplistic terms, a C Preprocessor is just a text substitution tool, and they instruct compilers to do required pre-processing before actual compilation. We'll refer to the C Preprocessor as the CPP. The C preprocessor is a very simple but powerful tool in the C programming language. Every C program is processed by the preprocessor before compiling it. The preprocessor allows to combine files using the #include or to define constants and macros using #define.

When you use the C preprocessor, you will not have to invoke it explicitly; the C compiler will do so automatically.

All preprocessor commands begin with a pound symbol (#). It must be the first nonblank character, and for readability, a preprocessor directive should begin in the first column. The following section lists all important preprocessor directives.

Three kinds of directives

- Macros

– #define

- File inclusion

– #include

- Conditional compilation

– #if, #ifdef, #ifndef, #elif, #else, #endif

Directive	Description
#define	Substitutes a preprocessor macro
#include	Inserts a particular header from another file
#undef	Undefines a preprocessor macro
#ifdef	Returns true if this macro is defined
#ifndef	Returns true if this macro is not defined
#if	Tests if a compile time condition is true
#else	The alternative for #if
#elif	#else an #if in one statement
#endif	Ends preprocessor conditional
#error	Prints error message on stderr
#pragma	Issues special commands to the compiler, using a standardized method

The use of the preprocessor is advantageous since it makes:

- programs easier to develop,
- easier to read,
- easier to modify
- C code is more transportable between different machine architectures.

Difference between macro and function

No	Macro	Function
1	Macro is **Preprocessed**	Function is **Compiled**
2	**No Type Checking**	**Type Checking** is Done
3	**Code** Length **Increases**	**Code** Length remains **Same**
4	Use of macro can lead to **side effect**	No **side Effect**
-		
5	Speed of Execution is **Faster**	Speed of Execution is **Slower**
6	Before Compilation macro name is replaced by macro value	During function call , Transfer of Control takes place
7	Useful where small code appears many time	Useful where large code appears many time
8	Generally Macros do not extend beyond one line	Function can be of any number of lines
9	Macro does not Check **Compile Errors**	Function Checks **Compile Errors**

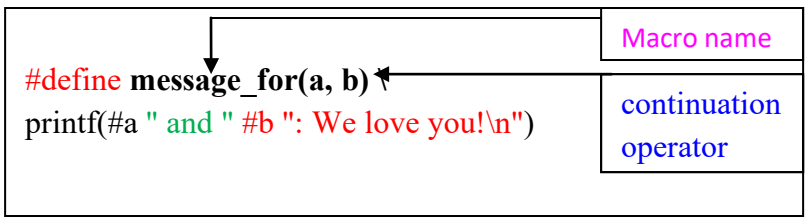
Macro

Preprocessor Operators

The C preprocessor offers the following operators to help you for writing the macros:

Macro Continuation (\)

A macro usually must be contained on a single line. The macro continuation operator continues a macro that is too long for a single line. **For example:**



Stringize (#)

The stringize or number-sign operator ('#'), when used within a macro definition, converts a macro parameter into a string constant. This operator may be used only in a macro that has a specified argument or parameter list.

For example:

```
#include <stdio.h>
#define message_for(a, b) \
printf(#a " and " #b ": Welcome you!\n")

void main()
{
message_for(Kamani, Sasanka);
}
```

OUTPUT

Kamani and Sasanka: Welcome you!

The defined() Operator

The preprocessor defined operator is used inconstant expressions to determine if an identifier is defined using #define. If the specified identifier is defined, the value is true (non-zero). If the symbol is not defined, the value is false (zero). The defined operator is specified as follows:

```
#include <stdio.h>
#if !defined (MESSAGE)
#define MESSAGE "You wish!"
#endif

void main()
{
printf("Here is the message: %s\n", MESSAGE);
}
```

OUTPUT

Here is the message: You wish!

Parameterized Macros

One of the powerful functions of the CPP is the ability to simulate functions using parameterized macros. For example, we might have some code to square a number as follows:

```
int square(int x) {
return x * x;
}
```

We can rewrite above code using a macro as follows:

```
#define square(x) ((x) * (x))
```

Macros with arguments must be defined using the #define directive before they can be used. The argument list is enclosed in parentheses and must immediately follow the macro name. **Spaces are not allowed between macro name and open parenthesis**. **For example:**

```
#include <stdio.h>
#define MAX(x,y) ((x) > (y) ? (x): (y))

void main( )
{
printf("Max between 20 and 10 is %d\n", MAX(10, 20));
}
```

```
OUTPUT

Max between 20 and 10 is 20
```

Exercises

1. Write a C Program to find area of a circle using macro. [Area of circle=πr2]
2. Write a C program to swap two variables using macro expansions.
3. Write a C program to find the Summation and multiplication for two numbers using macro function.
4. Write a C Program to find odd and even for a given number using macro.
5. Write a C Program to convert the temperature in Centigrade to Fahrenheit using macro. (F = (9/5) C + 32)
6. Write a macro that returns TRUE if its parameter is divisible by 10 and FALSE otherwise.
7. Write a macro *is_digit* that returns TRUE if its argument is a decimal digit.
8. Write a second macro *is_hex* that returns true if its argument is a hex digit (0-9, A-F, a-f). The second macro should reference the first.