|  |  |  |
| --- | --- | --- |
| |  | | --- | | Lab 6 – Macros  CPSC 2311- Spring 2021 | |  |

# Introduction

The goal of the lab is to practice writing and using macros, as well as, understanding ternary operators.

**Due:**

Monday, March 1, 2021, midnight.

# Lab Instructions

In class we talked about macros and defining macros for constant variables and functions. For this lab, you will implement several functions using macros.

I will provide you with a driver.c, functions.c and functions.h files. You will add your code to the functions.h, functions.c, and driver.c. You must add the needed #includes to each file as well as header guards to functions.h

Part 1:

Before we talk about the macros you will implement, we need to discuss what a ternary operator is and how to use one. (We discussed his in class last Friday.)

Ternary operators are similar to if-else statements but takes less space and helps to write the if-else statements in the shortest way possible.

The syntax for a ternary operator is:

Variable = Expression1 **?** Expression2 **:** Expression3

This says if Expression1 is true then Variable equals Expression2, otherwise Variable equals Expression3.

This is the same as the traditional if-else statement.

if(Expression1 is true){

Variable = Expression2;

}

else{

Variable = Expression3;

}

Below is another example of a ternary operator. This one is a multi-dimensional ternary operator.

Assume:

Action:

*1 : ==*

*2 : <*

*3 : >*

*/\*This is a ternary operator that has the form of*

*if a then b*

*else if c then d*

*else if e then f*

*else g*

*\*/*

action == 1 ? a==b if action is equal to 1 then **a** equals **b**, otherwise

: action == 2 ? a < b if action is equal to 2 then **a** is less than **b**, otherwise

: action == 3 ? a > b if action is equal to 3 then **a** is greater than **b** otherwise

: -50000 -50000

In functions.h, I gave you a function prototype called printReturn. Using the example code above, implement printReturn in functions.c. Then in driver.c (main) call printReturn four times using actions, 1, 2, 3, and 4(default). Play around with the values of a and b. This exercise is to convince yourself this function does work and you understand what it does.

Part 2:

Now that we understand ternary operators, we will discuss/write several macros.

1. Write a macro called MAX that has two parameters. You must use a ternary operator for this macro. MAX will return the value that is the largest of two parameters used for this macro.

In the driver, test this macro using a print statement that will print the following assuming the two data points passed to the macro(functions) were:

5, 4 Max: 5

10.5, 11.9 Max: 11.9

‘a’, ‘b’ Max: b

2. Write a macro called MIN that has two parameters. (I believe this one was given to you in the notes from Friday.) You must use a ternary operator for this macro. MIN will return the value that is the smallest of the two parameters used for this macro.

In the driver test this macro using a print statement that will print the following assuming the two data points passed in were:

Data Point Output

5, 4 Min: 4

10.5, 11.9 Min: 10.5

‘a’, ‘b’ Min: a

3. Write a macro called ABS that has one parameter. You must use a ternary operator for this macro and you are **not** allowed to use the math library. ABS will give the absolute value of the number passed to the macro.

In driver, test this macro using a print statement that will print the following assuming the number passed in is:

Data Point Output

-3 ABS: 3

3 ABS: 3

0 ABS: 0

Interestingly enough, we can call a macro within the definition of another macro. This is what you are going to do for the next two macros that you will implement.

4. Write a macro called LARGEST that has 3 parameters. You must use a ternary operator as well as one of the macros you have already defined. LARGEST will return the largest value of the three parameters passed to the macro.

In driver, test this macro using a print statement that will print the following assuming the 3 data points passed in are:

Data Points Output

1, 2, 3 LARGEST: 3

‘a’,’b’,’c’ LARGEST: c

‘c’,’b’,’a’ LARGEST: c

5. Write a macro called SMALLEST that has 3 parameters. You must use a ternary operator, as well as, one of the macros you have already defined. SMALLEST will return the smallest value of the three parameters passed to the macro.

In driver, test this macro using a print statement that will print the following assuming the 3 data points are passed in are:

Data Points Output

1,2,3 SMALLEST: 1

‘a’,’b’,’c’ SMALLEST: a

‘c’,’b’,’a’ SMALLEST: a

The next macro that you will write will be helpful in debugging.

6. Write a macro called DEBUG\_FPRINT that has no parameters. It will print, to stderr, the file that the call to DEBUG\_FPRINT is in, and the line number in the code that DEBUG\_FPRINT is being called on. Below is an example of what your output will look like.

Output

In File driver.c line 36

7. Write a macro called ISEVEN\_ODD that has one parameter. As the name indicates this macro will determine if a number is an even or odd number. The following is a sample of the output when ran with an even, then an odd number.

Output

4 is Even

1 is Odd

8. Because I know everyone loves allocating memory, we are going to write a macro called MALLOC. MALLOC has two parameters, one that will represent the ‘type’ of data. The second will determine how many elements of the ‘type’ you want to allocate. Ex: if you want to allocated memory for 5 integers the parameters will be 5, int.

In driver.c using the following steps check to ensure the MALLOC macro works.

Create a pointer variable, called num of type int and initialize it to NULL. Call the MALLOC macro passing in MALLOC(5,int). Then check to make sure the variable is not NULL. If it is not NULL print a statement similar the following.

Output

The address of num is: (the address of the allocated memory would print here)

The above are just a small sampling of small functions you can implement using macros. You should experiment with macros; they can be somewhat helpful.

There are some drawbacks to macros which we discussed in class last Friday.

There is one other macro that I want to discuss, which I am giving you the macro.

#define ADD\_TO\_SUM(sum\_number, value) sum ## sum\_number += value

This macro actually does what the name indicates. It adds a value to a variable. So what is ##? This is the concatenate symbol. It says concatenate the value of sum\_number to the string sum. Hence, sum ## sum\_number. Suppose we call ADD\_TO\_SUM(5, 25) it will concat 5 to the string ‘sum’ making a string ‘sum5’. It then adds 25 to sum5. Basically, this would require you to have a variable in main called sum5 and 25 would be added to it.

You are to add the ADD\_TO\_SUM macro to your .h file. You must test the macro by adding a print statement to main that prints the following:

Assume you called Add\_to\_Sum passing in 1 and 15. Also assume I have a variable in main called sum1 with the value of 1.

Add\_to\_Sum: 16

You must follow all of the directions in this document. If you do not, a substantial point penalty will be applied. There are 8 macros that you will implement each will carry a point value of ~10 points. The remaining points will apply to the following: adding the PrintReturn function I basically gave you, adding the macro Add\_to\_Sum I gave you, the test cases you created in main, and last but not least, following directions.

You must add a header to all files similar to the following:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

\*Your name \*

\*CPSC2310 Spring 2021 \*

\*UserName: \*

\*Instructor: Dr. Yvon Feaster \*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

I**f your files do not compile you will receive a 0 on this lab.**

Submission Instructions

**Add a makefile, tar zip all files naming the tarred file <username>lab6.tar.gz ; submit it to handin.** ([http://handin.cs.clemson.edu)](http://handin.cs.clemson.edu)/)

**Be sure to check your tarred files after submission to handin. If any of your files are corrupt or missing you will receive a 0 on this lab.**