



Lab 2: Calculator

Commit ID Form: <https://goo.gl/forms/Cdw0z8gbkyJZLMIE2>

12 Points

Introduction

In this lab you will be writing your first program from scratch. This program will read user input, perform mathematical calculations, and then print the results out to the user. This will be done using the serial port just as you did in part 3 of the last lab, and so will require use of a serial terminal program. See the Serial Communications document for more information on serial communications.

It will rely on knowledge from outside of class such as data types and `printf()` and `scanf()`; pay close attention to the reading list below. This lab will also introduce declaring, defining, and implementing functions. A brief overview of these concepts is available in this document and there is additional information in K&R.

Reading

- K&R All of Chapter 1, 7.4, and Appendix B1.2
- Serial Communications handout
- Software design handout
- Code style handout

Concepts

- `printf()` and `scanf()`
- Functions and Prototypes
- C standard library
- Iterative Code Design

Provided Files

- `lab2.c` - This file contains the program template where you will implement a simple calculator. Calculator functionality will go within the comments in `main()`, function prototypes that will be implemented in the calculator will be

declared right before `main()` with their definitions following `main()` as demonstrated with the `round()` function stub. These are called out in the comments in the `lab2.c` file.

Assignment requirements

- **Program Requirements**

This assignment has the following requirements:

- Welcome the user to your calculator program with a nice greeting
- Prompt the user for first a mathematical operation to perform in the form of a single character, so the user only needs to type a single character to pass this prompt. This prompt should also display all operations that are available to the user. These include the four basic math operations (multiplication, division, addition, subtraction), five additional operations (Absolute Value, Celsius, Fahrenheit, Average, Tangent), and—for extra credit—the Round function. All of these operations must handle negative values correctly.
 - '*' - Performs multiplication on two operands.
 - '/' - Performs division by dividing the first operand by the second. The second operand is checked to make sure a division-by-zero error doesn't occur. If one does, just print 0 out as the result.
 - '+' - Performs addition on two operands.
 - '-' - Performs subtraction on two operands, subtracting the second operand from the first.
 - 'v' – The Average function returns the average of its two arguments.
 - 'a' - Absolute value calculates the absolute value of its argument.
 - 'c' - The Celsius function treats its argument as a value in degrees Fahrenheit and converts it to degrees Celsius.
 - 'f' - The Fahrenheit function treats its argument as a value in degrees Celsius and converts it to degrees Fahrenheit.
 - 't' - The Tangent function takes in a value in degrees and calculates the tangent value and returns the result.
 - **Extra Credit:** 'r'. The Round function rounds a number down if its fractional part is less than 0.5, and rounds up otherwise.
- You must use your new knowledge of functions (after reading the rest of the lab manual) to implement all of the non-basic operations of your calculator: Absolute Value, Fahrenheit to Celsius, Celsius to Fahrenheit, Tangent in degrees, Average (and Round if implemented) using the function names given in the [Operations with Functions](#) section. These functions should not use `scanf()`, `printf()`, or any other functions from `stdio.h`, as that logic should be implemented in `main()`. The characters given above should be used in your calculator to identify the corresponding operator.

- Each of these operations must be implemented as outlined in the Functions section of this lab manual (declaration, implementation, and usage).
- All required calculations and user input & output should be done with values of type double. (This means that all of your functions should use double as both their input and output data types).
- All function-based calculations should return the result using a return statement.
- If the operator is a binary operator (relies on two operands), then your program should prompt the user further for two operands on which to perform the operation, one at a time. If the operator is a unary operator (relies on only one operand), then your program should prompt the user further for only one operand on which to perform the operation. After each operand is written the user should be able to press enter and the prompt will finish.
- Finally print out the result of the mathematical operation along with what operation was performed. The following example will suffice for basic operations: "Result of (3.25 * 4): 13". Notice that you will need two different printf() formats for results that are calculated from a unary operator versus those from a binary operator. Unary operators will require a format that looks more like "Result of |-5.3|: 5.3" (that is an example of an absolute value calculation).

Example output:

- "Result of (4.5 deg->F): 40.099998"
- "Result of (57 deg->C): 13.888889"
- "Result of tan(3.7): 0.0647"
- "Result of round(5.8): 6.000000"
- "Result of (3 * 3): 9.000000"

This is what it looks like in the terminal:

```
Virtual Terminal
Welcome to Bryant's calculator program!
Enter a mathematical operation to perform (*,/,+,-,v,a,c,f,t,r): f
Enter the first operand: 4.5
Result of 4.500000 deg->F: 40.099998
Enter a mathematical operation to perform (*,/,+,-,v,a,c,f,t,r): c
Enter the first operand: 57
Result of 57.000000 deg->C: 13.888889
Enter a mathematical operation to perform (*,/,+,-,v,a,c,f,t,r): t
Enter the first operand: 3.7
Result of tan(3.700000): 3.705152
Enter a mathematical operation to perform (*,/,+,-,v,a,c,f,t,r): r
Enter the first operand: 5.8
Result of round(5.800000): 6.000000
Enter a mathematical operation to perform (*,/,+,-,v,a,c,f,t,r): *
Enter the first operand: 3.5
Enter the second operand: -37
Result of (3.500000 * -37.000000): -129.500000
```

- Return to prompting the user for another mathematical operation to perform. This should result in an infinite loop of prompting the user for another calculation after displaying the results of the prior calculation.
- Your program must use functions to implement the non-arithmetic operations: absolute value, Celsius to Fahrenheit, Fahrenheit to Celsius, average, and tangent. NOTE: you may not use any functions from the standard math library (except `tan()` for the tangent function).
- **Extra Credit:** implementation of a round function along with it being usable by the user of your calculator program.
- **Code style:** Follow the standard style formatting procedures for syntax, variable names, and comments.
- **README.txt:** Create a README.txt file like you did for the first lab. In it specify
 - Your name
 - The names of other students who you collaborated with
 - How long this lab took you
 - Any parts that were difficult or unclear
- **Required Files:**
 - lab2.c
 - README.txt

Grading

This assignment consists of 12 points:

- **Input and Output (3.25 points)** - User input is properly handled and results are output correctly.
- **Correct calculations (5.75 points)** - All operators are implemented correctly, helper functions for advanced calculations are also correct and named properly.
- **Code style (1 point)** - Your code follows the style guidelines and contains less than 10 errors total.
- **README.txt (1 point)** - A README.txt file was provided with the necessary contents.
- **GIT Hygiene (1 point)** – You should have a minimum of 10 commits with significant differences and good commit messages on this lab
- **Extra credit (1 point)** - Correct implementation of a round function based on the implementation described below.

You will lose points for the following:

- -2: Warnings displayed on compilation excluding "warning: format '%f' expects type 'float *', but argument 2 has type 'double *'" when using scanf()
- No credit at all if lab2.c doesn't compile. There is no excuse to turn in code that does not compile.

printf() and scanf()

You will be using both of these functions in your program to interact with the user. This is done through standard input and output. Both of these functions are included within the C standard library (part of the C language). They are declared in the header file `stdio.h`. You will need to add an include statement to include the `stdio.h` standard library header below the comment stating "**** Include libraries here ****".

Example usage of these functions follows:

```
char g;
printf("Type in any character:");
scanf("%c", &g);
printf("You input '%c'", g);
```

Please note the ampersands (&) in front of the variables passed as arguments to `scanf()`. These are very important! For the code that you're writing you will need one before all of the variable arguments to `scanf()`. You don't need to know the details of

this right now and it will be covered later when we get to Array and Pointers. For more information about these topics refer to chapter 5 of K&R.

Note that `scanf()` is a little finicky about how it handles input. If you use `scanf("%f", &x)` to read in a double and type a number and press Return, not all of the characters will be processed. All of the numbers will end up parsed and placed into the `x` variable, but the newline character will not have been processed and can be captured by future calls to `scanf()`. To solve this use `scanf("%f%c", &x, &c)` (where `x` is of type `double` and `c` is of type `char`) so that the Return character is placed into the `c` variable and will not end up being processed by following calls to `scanf()`.

Note that there is a compiler bug where a warning will be generated if the token `"%f"` is used with variable of type `double` with `scanf()`. You can safely ignore this warning for this lab and you will not lose credit for it.

Functions

- Declaring functions – before a function can be used, it must first be declared (just like a variable). These declarations are also referred to as *function prototypes*. They are used to describe everything about the function EXCEPT what it actually does (the part in between the curly-braces). These declarations need to occur in the source code BEFORE the function is first referenced. This means if you call a function in `main()`, but the function is implemented after `main()`, you'll need to put a function prototype before `main()`.

An example of a function prototype is as follows:

```
double SumOf(double op1, double op2);
```

This prototype states that the function `SumOf()` takes in two values of type `double` and returns a `double` as well. Note that the names you give the arguments can be anything you want, and are used internally in the function.

- Implementing Functions — the definition of a function actually defines what a function does. This includes the same information as the function prototype (without the `;`) and then has the actual code between two `{}`. A function ends when it hits a return statement or the closing `}`. It can have temporary variables declared after the opening `{`.

An example of a function definition is as follows:

```
double SumOf(double op1, double op2)
{
    return op1 + op2;
}
```

This creates a summation function that is called with two variables of type `double`. The function returns a value of type `double` that is the sum of the values passed in as its arguments.

- Using Functions — Functions can be used for various things, but in this lab all that is necessary to know is how to store the return value of a function into a variable. This is done just like storing any value into a variable. On the left-hand side of the assignment operator (`=`) is the variable that will hold the value and on the right-hand side is the function call.

An example, of how this is done is as follows:

```
double result, operand1 = 1, operand2 = 2;
result = SumOf(operand1, operand2);
```

With `result` being the variable holding the return value of the function, `SumOf()` being the function itself, and `operand1` & `operand2` are the arguments passed to the function `SumOf()`.

Operations with functions

The functions you will need to implement for this lab are listed below, along with the exact names for each of them. You will need to insert the prototype in the file above `main` (see the comments for where) and the definition below `main` where it is called out in the comments. Again, none of these functions perform input or output using `scanf()` or `printf()`, that functionality belongs in `main()`.

- `AbsoluteValue()` — This function is unary and takes in a `double` and returns a `double`. It returns the absolute value of the argument. This can be done with testing whether or not the value is positive and if it is not then return the positive value. Note: You CANNOT utilize the absolute value function from `math.h`.
- `FahrenheitToCelsius()` and `CelsiusToFahrenheit()` — These functions are both unary and take in a `double` and return a `double`. They convert from $^{\circ}\text{F}$ to $^{\circ}\text{C}$ and from $^{\circ}\text{C}$ to $^{\circ}\text{F}$. Combined with your knowledge from Lab 0 you should be able to implement both of these calculations easily.
- `Tangent()` — This function is unary and takes in a `double` and returns a `double`. It implements the tangent function but the argument is in degrees. You can rely on the tangent function from the standard math library which uses radians (search/browse the XC32 standard library help to find this function and the header that declares it). You must perform the necessary conversions to receive input in degrees. For this function

you must use the constant `M_PI` that is defined in the standard library, a quick search through the help will reveal what header file it's in).

- Average() – This function is binary and takes two doubles as inputs and returns a double. The function returns the average of its two inputs.
- **For Extra Credit:** Round() – This function is unary and takes in a double and returns a double. This function rounds the input towards zero if the decimal value is within 0.5 and away otherwise. You may not use any of the functions within the standard library or Microchip's peripheral library to implement this! You will have to think a little bit about how this can be done. One method utilizes type casting (described in section 2.7 of K&R, page 42). Another would be to use a while loop that counts down to find the fractional part. This function stub and prototype has already been created for you in `lab2.c`; just replace the body of that function.

Program flow

Your program will loop continuously while reading and writing from the terminal. This concept is outlined for you within the ``while (1)`` loop (which will loop forever) in the pseudo code below. The basic outline of your program looks as follows:

Output greeting to the user

```
while (1)
    get operator as a char
    if operator is invalid
        set operator to 0
    if operator is valid (at this point not 0)
        get operand1
        if operator is a binary operator
            get operand2
        if operator is addition
            result <- sum of operands
        else if operator is subtraction
            result <- difference of operands
        else if operator is multiplication
            result <- product of operands
        else if operator is division
            result <- quotient of operands
        else if operator is a 'v'
            result <- average function
        else if operator is an 'a'
            result <- absolute value function
        else if operator is a 'c'
            result <- Fahrenheit to Celsius function
        else if operator is an 'f'
            result <- Celsius to Fahrenheit function
```



```

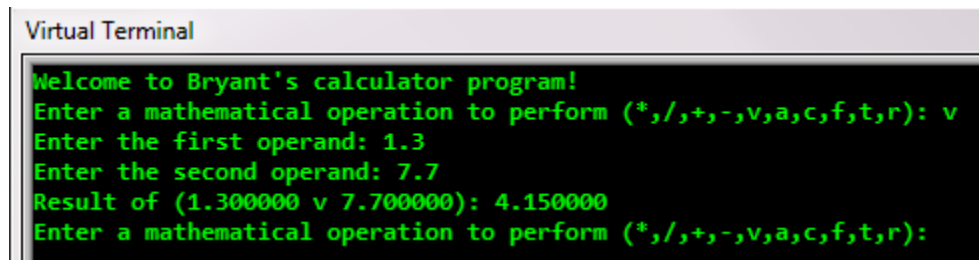
else if operator is a 't'
    result <- tangent function
else if operator is a 'r'
    result <- round function

if operator is a unary operator
    print the result of a unary operation
else
    print the result of a binary operation
else if operator is invalid
    print invalid operator message

```

Program output

Example output for one calculation is given below.



```

Virtual Terminal
Welcome to Bryant's calculator program!
Enter a mathematical operation to perform (*,/+, -, v, a, c, f, t, r): v
Enter the first operand: 1.3
Enter the second operand: 7.7
Result of (1.300000 v 7.700000): 4.150000
Enter a mathematical operation to perform (*,/+, -, v, a, c, f, t, r):

```

Doing this lab

The Iterative Software Design handout describes a very powerful way to approach any programming project. Make sure you read it and understand it—this will be useful to you far beyond this class. Below you will see we have given you an example method of completing this lab loosely following the practices described in the Iterative Code design handout. Remember, it is important to stop and test your code for correct functionality at each step before moving on. Note that at each of these steps you should commit before and after you finish that step at a minimum.

Step 1

- Display a greeting message using `printf()`.

Step 2

- Prompt the user to input a character.
 - There is no need to print this character back out to test this, part of BOARD code automatically echoes back to the user whatever they typed in. So if you see the input you typed in the terminal, that means it was successfully received.
 - Add in an echo back message: "Character received was: 'x'"

Step 3

- Prompt the user for a character within an infinite loop
 - These characters should all be echoed like they were in the above step.

Step 4

- Now add an invalid operator checker
 - Checking for this should set your operator variable to a standard error value (-1) if operator is not one of your valid operator's (at this point you can just use '+').
 - Now print the operator if it is not equal to your standard error value, and print an error message otherwise ("Error, not a valid operator").
 - At a minimum, test your code with all valid operators and a large number of invalid ones to convince yourself that it works (this is called unit testing).

Step 5

- Continuously prompt the user for an operator and two operands.
 - Echo the results ("You input 3.5 + 4.7") to ensure you are capturing the correct values.
 - If the user enters an invalid operator, print the error message
 - Test this extensively

Step 6

- Continuously prompt the user for an operator and two operands.
 - If the user enters a '+', calculate the result and print it
 - If the user enters an invalid operator, print the error message
 - Test and ensure that your addition is correct. Make sure you use both + and - numbers.

Step 7

- Expand code to work for all 4 basic operators: +, -, /, *
 - Note: this will require you do update your valid operator checker as well.
 - Ensure that your division traps the divide by zero
 - Test to make sure that all the functions work correctly. Lots of testing here.

Step 8

- Display the results nicely as the requirements describe.

Step 9

- Add an operator for an absolute value calculation 'a'.
- Add checking for one or two operands. This checking should make it so your program only prompts for one operand when given the absolute value operator (don't do the calculation just print something to show it works).

Step 10

- Define an absolute value function.
- Test that it works with hard coded inputs. (I.e.: use test cases: -3, -8.63, 0, and 13.67)
 - `printf("%f\n", AbsoluteValue(-3));` // result should be 3
 - `printf("%f\n", AbsoluteValue(-8.63));` // result should be 8.63
 - `printf("%f\n", AbsoluteValue(0));` // result should be 0

- `printf("%f\n", AbsoluteValue(13.67)); // result should be 13.67`
- Test this extensively with other inputs and make sure you have the correct results. Try to think of inputs that would be problematic and see how it handles them.

Step 11

- Implement the absolute value operator in your calculator by updating your operator checkers, and calling the function in the appropriate place.
- You will now need a new result message with a `printf()` formatted to display a calculation with only one operand (by now you should know how to do this with an operator checker).

Step 12

- Define an Average function.
- Test to see if it works with hard coded inputs. I.e.: (55.5, 0), (0.00, -10), (-36.49, 36.49)
 - Your outputs should be 27.75, -5, and 0.0 correspondingly.

Step 13

- Implement the Average operator in your calculator.
- Test it extensively
- Check that it now works within the full loop with the other operators

Step 14

- Define a Celsius to Fahrenheit conversion function.
- Test to see that it works with hard coded inputs. I.e: inputs 32, -27, 0
 - Your outputs should be 89.599995, -16.599998, and 32 correspondingly.

At this point you should see a pattern. Implement a new function, test it using hard coded inputs (that you know the corresponding outputs). This tests the functionality. Extensive testing at this stage can reveal bugs which are easy to fix. Try to find inputs that break your function—these are known as “corner cases.” Once you have thoroughly tested the function, put it into the rest of the code, and make sure it still works within the new program flow. Test some more, and fix the bugs you find.

Step 15

- Implement the Celsius conversion function in your calculator.
- Test it extensively.

Step 16

- Define a Fahrenheit to Celsius conversion function.
- Test with hard coded inputs (i.e.: 98, -12, 0)
 - Your output should be 36.666668, -24.444445, and -17.777779 correspondingly.

Step 17

- Implement the Fahrenheit conversion function in your calculator.
- Test, test, test. Squash bugs

Step 18

- Define a Tangent in Degrees function.
 - Implement a helper function to convert degrees to radians. Test it.
 - Put together the full function.
 - Test it with hard coded inputs (i.e.: 57, 1.5, -33, 0)
 - Your outputs should be 1.5399, 0.0262, -0.6494, and 0 correspondingly.

Step 19

- Implement the Tangent function in your calculator.
- Test, test, test. Squash bugs, retest.

Step 20 (OPTIONAL)

- Implement the Round function in your calculator.

Step 21

- Remove any dead code and comment out any leftover test code. Keeping your test code in the project is generally a good idea, as you can easily uncomment it and test your code again after making changes. Later in the quarter, you will be shown a more elegant way to do this.
- Double-check that you met all program requirements listed in this document.
- Compare your code to the examples in the Style Guidelines document, fixing any errors you see.
- **Submit your finished lab2.c and README.txt through the online submission tools.**

Frequently Asked Questions:

*I get [warning: format '%f' expects type 'float *', but argument 2 has type 'double *'](#) when compiling.*

This isn't a problem so long as it only pops up when using scanf with the '%f' format specifier and the double datatype.

After testing an operation once, and seeing the result, I can no longer select an operation, as it appears to be chosen automatically, but results in an invalid operator.

This is due to not consuming the extra newline character that is the result of pressing ENTER after typing in your operands. See the [printf\(\) and scanf\(\) section](#).