Lab 1: Calculator

Introduction

In this lab you will be writing your first program from scratch. This program will read user input, perform mathematical calculations, and then return the result to the user. It will rely on knowledge from outside of class such as data types and `printf()` & `scanf()`, and so 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
- Software design handout
- Code style handout

Concepts

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

Provided Files

- lab1.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.

Assignment requirements

- Program Requirements

  This assignment has the following requirements:
o Welcome the user to your calculator program with a nice greeting

o 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 optionally for extra credit the Round function. All of these operations must handle negative values correctly.

- ‘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.
- ‘v’ – The Average function returns the average of its two arguments.
- ‘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 towards 0 if its fractional part is less than 0.5, and rounds away otherwise.

o 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. The characters given with each operation should be used in your calculator to identify the corresponding operator.

o Each of these operations must be implemented with the described implementation of functions outlined in the Functions section of this lab manual (declaration, implementation, and usage).

o All required calculations and user input & output should be done with values of type double. This also means that all implemented functions should use double as their input & output data types.

o All function-based calculations should return the result with a return statement.

o 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:

```
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 save 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.
  - Add the following to the top of every file you submit as comments:
    - Your name
    - The names of colleagues who you have collaborated with
- **Submission:**
  - Submit lab1.c.

**Grading**

This assignment consists of 10 points:

- **1 Point** - Calculator displays a greeting message only once at the start of execution.
- **1 Point** - Correct implementation of the four basic mathematical operations (+, -, *, /).
- **1 Point** - Calculator prompts for input appropriately.
- **4 Points** - Correct implementation of the four function-based calculations (1 point each, with the two temperature conversions treated as a single point).
- **1 Point** - Calculator behaves correctly for unary operators versus binary operators.
- **1 Point** - Calculator displays an acceptable result message and loops continuously asking for another operator after each calculation.
- **1 Point** - Your program follows the provided style guidelines and contains less than 5 errors.
- **1 Point (extra credit)** - Correct implementation of a round function based on the implementation described below.

You will lose points for the following:

- -2: Didn’t submit exactly lab1.c
- -2: Source files don’t include your name and collaborators as a comment as the first line of the lab1.c file
- -2: Warnings displayed on compilation (excluding those issues about "double" datatypes when using scanf())
- No credit if lab1.c doesn’t compile

**`printf()` and `scanf()`**

You will be using both of these functions in your program to interact with the user. This is done with what is called standard input and output. Both of these functions are included within the C standard library that is available to all C programs. 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:

```c
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:
  ```c
double SumOf(double a, double b);
```
  
  This prototype states that the function SumOf() takes in two values of type double and returns a double as well.

- Implementing Functions — The definition of a function actually defines what a function does.

  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 the 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 its arguments.

Operations with functions

- Absolute Value - 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. Function should be named AbsoluteValue() and take in a double and return a double.

- Fahrenheit to Celsius & Celsius to Fahrenheit – Combined with your knowledge from Lab 0 you should be able to implement both of these calculations. Functions should be named FahrenheitToCelsius() and CelsiusToFahrenheit() and both take in a double and return a double.

- Tangent (in Degrees) - This function can rely on the tangent function from the standard math library which uses radians (search/browse the C30 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 defined for you at the top of lab1.c (just utilize the constant name like you would use any number). Function should be named Tangent() and take in a double and return a double.

- Average - This function returns the average of its two inputs. Function should be named Average() and take in a double and return a double.
• **For Extra Credit:** Round – This function must round a number down if the
decimal value is below 0.5 and up 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 has already been created for you in lab1.c,
so 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 an invalid operator message

**Program output**

Example output for one calculation is given below.

<table>
<thead>
<tr>
<th>Virtual Terminal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Welcome to Bryant’s calculator program!</td>
</tr>
<tr>
<td>Enter a mathematical operation to perform (+,/,*,-,v,a,c,f,t,r): v</td>
</tr>
<tr>
<td>Enter the first operand: 1.3</td>
</tr>
<tr>
<td>Enter the second operand: 7.7</td>
</tr>
<tr>
<td>Result of (1.30000 v 7.70000): 4.15000</td>
</tr>
<tr>
<td>Enter a mathematical operation to perform (+,/,*,-,v,a,c,f,t,r):</td>
</tr>
</tbody>
</table>

**Doing this lab**

The Iterative Software Design handout describes a very powerful way to approach any programming project. Below you will see we have given you an example method of completing this lab following the practices described in the Iterative Code design handout. Remember, it is important to stop and test your code for correct functionality before moving onto the next step.

**Step 1**
- Display a greeting message.

**Step 2**
- Prompt the user to input a character.
  - Testing would involve printing that character back out to confirm it works as expected.

**Step 3**
- Continuously prompt the user for a character within an infinite loop
  - Testing should echo this character out every loop

**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").

Step 5
- Continuously prompt the user for an operator and two operands.

Step 6
- Continuously:
  - Prompt the user for an operator and two operands.
  - If the user enters a ‘+’, calculate the result and print it.

Step 7
- Expand code to work for all 4 basic operators: +, -, /, *
  - Note: this will require you do update your valid operator checker as well.

Step 8
- Display the result 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 code. Use test cases: -3, -8.63, 0, and 13.67
  - printf("%f\n", AbsoluteValue(-3));
  - printf("%f\n", AbsoluteValue(-8.63));
  - printf("%f\n", AbsoluteValue(0));
  - printf("%f\n", AbsoluteValue(13.67));
  - Your output should be 3, 8.63, 0, and 13.67 correspondingly.

Step 11
- Implement the absolute value operator in your calculator by updating your operator checkers, and calling the function in the appropriate place.
- Now you will also 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 code. Use Test cases: (55.5, 0), (0.00, -10), (-36.49, 36.49)
  - Your output should be 27.75, -5, and 0.0 correspondingly.

Step 13
- Take out any tests for your Average function.
- Implement the Average operator in your calculator.
Step 14
- Define a Celsius conversion function.
- Test to see that it works with code. Use test cases: 32, -27, 0
  - Your output should be 89.599995, -16.599998, and 32 correspondingly.

Step 15
- Take out any test for your Celsius conversion function.
- Implement the Celsius conversion function in your calculator.

Step 16
- Define a Fahrenheit conversion function.
- Test to see that it works with code. Use test cases 98, -12, 0
  - Your output should be 36.666668, -24.444445, and -17.777779 correspondingly.

Step 17
- Take out any tests for your Fahrenheit conversion function.
- Implement the Fahrenheit conversion function in your calculator.

Step 18
- Define a Tangent in Degrees function.
- Test to see it works with code. Use test cases: 57, 1.5, -33, 0
  - Your output should be 1.5399, 0.0262, -0.6494, and 0 correspondingly.

Step 19
- Take out any tests for your Tangent function.
- Implement the Tangent function in your calculator.

Step 20 (OPTIONAL)
- Take out any tests for your Tangent function.
- Implement the Round function in your calculator.

Step 21
- Remove any dead or left over test code.
- 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 lab1.c 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.