EECS 10: Assignment 4

Prepared by: Prof. Rainer Dömer

July 17, 2017

Due Monday July 24, 2017 at 11:00pm

1 A menu-driven calculator [50 points]

The goal of this assignment is to practice using functions in C programming by performing arithmetic operations on double precision floating point numbers.

1.1 The Option Menu

Your program should be a menu driven program which shows the current value (the previous result) at the top and then offers a menu of operations to the user. Specifically, it should look like this:

```
Welcome to my calculator!
-----------------------------------------
Current result: 0
-----------------------------------------
 0. Enter a new current value
 1. Add a number to the current result
 2. Subtract a number from the current result
 3. Multiply the current result by a number
 4. Divide the current result by a number
 5. Calculate the absolute value of the current result
 6. Calculate the sine of the current result
 7. Calculate the cosine of the current result
 8. Calculate the tangent of the current result
 9. Calculate the square root of the current result
10. Calculate the Nth root of the current result
11. Quit.
Please enter your selection:
```

As indicated above, your program should let the user select an option, numbered 0 through 11. If the user selects 11, your program simply exits. For other selections, your program should perform the corresponding floating point operation on the current value. At the beginning, we will assume that the current value is zero, as shown above.

If the user selects an operation that requires two arguments, prompt the user to input a suitable floating point number as the operand. Once the user inputs the second operand, your program should perform the requested operation and display the new result and the option menu again.
Let's assume the user selects option 1, the add operation. The program then prompts the user for the summand. If the user inputs the value 4.2, the screen should look like this:

```
Please enter your selection: 1
Please enter the summand: 4.2
-----------------------------------------
Current result: 4.2
-----------------------------------------
 0. Enter a new current value
 1. Add a number to the current result
 2. Subtract a number from the current result
 3. Multiply the current result by a number
 4. Divide the current result by a number
 5. Calculate the absolute value of the current result
 6. Calculate the sine of the current result
 7. Calculate the cosine of the current result
 8. Calculate the tangent of the current result
 9. Calculate the square root of the current result
10. Calculate the Nth root of the current result
11. Quit.
Please enter your selection:
```

The current value always shows the result of the previous operation (or zero at the beginning). In order to display the current result in easily readable fashion, let your program switch automatically to scientific notation, if the value becomes too large or too small.

To start a new calculation independent of the previous result, the user can enter a new current value by use of option 0.

```
Please enter your selection: 0
Please enter the new value: 1000000000
-----------------------------------------
Current result: 1e+09
-----------------------------------------
 0. Enter a new current value
 1. Add a number to the current result
 2. Subtract a number from the current result
 3. Multiply the current result by a number
 4. Divide the current result by a number
 5. Calculate the absolute value of the current result
 6. Calculate the sine of the current result
 7. Calculate the cosine of the current result
 8. Calculate the tangent of the current result
 9. Calculate the square root of the current result
```

```
10. Calculate the Nth root of the current result
11. Quit.
Please enter your selection:
```

1.2 Implementation

To implement the program, you need to define a few variables and a set of functions. Each function performs the calculation for a specific option, such as (1) addition, (2) subtraction, (3) multiplication, and so on.

1.2.1 Function Declarations

As a starting point, use the following function declarations:

```
double Add(double op1, double op2);
double Subtract(double op1, double op2);
double Multiply(double op1, double op2);
double Divide(double op1, double op2);
double Abs(double op1);
/* plus any other functions you may need... */
```

Be sure to pass in the correct values as arguments to the parameters and store the result of the operation in a suitable variable.

1.2.2 Error handling

A robust program should be able to handle many situations and report any problem or invalid input to the user. To get full credit, your program should be able to handle the following user errors:

If the user enters 0 as divisor, print `ERROR: Division by zero!` and skip the division.

If the user uses a value outside of the range -1.3 through 1.3 for the tangent operation, print `ERROR: Input out of range!` and skip the operation.

If the user performs a root calculation on a negative value, print `ERROR: Root of a negative number!` and skip the approximation.

If the user performs the n-th root calculation where $n \leq 0$, then print `ERROR: Invalid integer N!` and skip the operation.

1.2.3 Math library usage

You may use the standard math library for the following three menu options:

```
 6. Calculate the sine of the current result
 7. Calculate the cosine of the current result
 8. Calculate the tangent of the current result
```

To utilize the math library, be sure to include the correct header files and pass the correct linker flags to the compiler to build your program.

1.2.4 Square root approximation [20 points]

For this assignment, you should program the root calculations yourself (not by using the math library).

```
 9. Calculate the square root of the current result
10. Calculate the Nth root of the current result
```

Let's start with option 9 for the square root. We can then reuse the implementation for the more general case of the *n*-th root.

```
double ApproximateRoot2(double op1);
double ApproximateRootN(double op1, unsigned int n);
```

We will use a binary search approximation technique for this assignment. In particular, the program will always keep a range of a left bound L and a right bound R, where the actual square root S lies somewhere between L and R: $L \leq S \leq R$
Consequently, it follows that $L*L \leq N = S*S \leq R*R$. Thus, to find S, we can compare $L*L$ or $R*R$ with N.
The binary approximation then works as follows. First, we compute a value M that lies in the middle between the left bound L and the right bound R: $M = L+(R-L)/2$. Then, if $M*M < N$, the square root obviously lies somewhere in the right half of the current range (i.e. within M to R), otherwise in the left half of the current range (i.e. within L to M). The program then can use the selected half of the range as the new range and repeat the whole process.
With each iteration, the search range is effectively reduced to half of its previous size. Because of this, this technique is called *binary search*. To start the search, we will use the range from 0 to $max(1,N)$ which is guaranteed to contain the square root of N. We will stop the iteration, once we have reached a range that is smaller than 0.000001 so that we reach a precision of 6 digits after the decimal point for our approximation.
The pseudo-code of the *binary search* algorithm can be written as follows.

Start with a range of 0 to N
As long as the range is not accurate enough, repeat the following steps:
  Compute the middle of the range
  Compare the square of the middle value with N
  If the middle value is less than the square root
    Use middle-to-right as the new range
  Else
    Use left-to-middle as the new range
  Output the middle of the latest range as result

For example, to compute the square root of 10, the program will start with 5, which is in the middle between 0 and 10. Since $5*5 = 25$ is larger than 10, the program will try the middle number 2.5 of left bound (0 to 5). Thus, the program compares $2.5*2.5$ with 10. Because the result 6.25 is smaller than 10, it will pick 3.75 (the middle number of 2.5 and 5) as the next guess. By picking the middle number every time and comparing its square with the original number, the program gets closer to the actual square root.
To demonstrate the approximation procedure, your program should print the approximated square root value in each iteration, as follows:

```
Please enter your selection: 0
Please enter the new value: 10
```

```
-----------------------------------------
Current result: 10
-----------------------------------------
 0. Enter a new current value
 1. Add a number to the current result
 2. Subtract a number from the current result
 3. Multiply the current result by a number
 4. Divide the current result by a number
 5. Calculate the absolute value of the current result
 6. Calculate the sine of the current result
 7. Calculate the cosine of the current result
 8. Calculate the tangent of the current result
 9. Calculate the square root of the current result
10. Calculate the Nth root of the current result
11. Quit.
Please enter your selection: 9
Iteration 1: the square root of 10.000000 is approximately 5.000000
Iteration 2: the square root of 10.000000 is approximately 2.500000
Iteration 3: the square root of 10.000000 is approximately 3.750000
Iteration 4: the square root of 10.000000 is approximately 3.125000
[...]
Iteration 25: the square root of 10.000000 is approximately 3.162278
-----------------------------------------
Current result: 3.16228
-----------------------------------------
```

Note that your program should calculate the square root of any natural number.

1.2.5 *n*-th root approximation [5 bonus points]

Extend your program with the 10th option so that it can calculate the *n*-th root of any positive value. For example, your program should look like this for the bonus part:

```
Please enter your selection: 0
Please enter the new value: 64
-----------------------------------------
Current result: 64
-----------------------------------------
 0. Enter a new current value
 1. Add a number to the current result
 2. Subtract a number from the current result
 3. Multiply the current result by a number
 4. Divide the current result by a number
 5. Calculate the absolute value of the current result
 6. Calculate the sine of the current result
 7. Calculate the cosine of the current result
```

```
 8. Calculate the tangent of the current result
 9. Calculate the square root of the current result
10. Calculate the Nth root of the current result
11. Quit.
Please enter your selection: 10
Please input the value of integer N: 6
Iteration 1: the 6th root of 64.000000 is approximately 32.000000
Iteration 2: the 6th root of 64.000000 is approximately 16.000000
Iteration 3: the 6th root of 64.000000 is approximately 8.000000
Iteration 4: the 6th root of 64.000000 is approximately 4.000000
Iteration 5: the 6th root of 64.000000 is approximately 2.000000
-----------------------------------------
Current result: 2
-----------------------------------------
```

Note that your program should run correctly for any natural number n which is greater than 0.

2 Submission

You need to save your program as **`calculator.c`**.
In order to demonstrate that your code works correctly, perform the following calculations in the script file:

1. $5 \times 3 + 27$
2. $\cos(\sin(3.1415927))$
3. $\sqrt{1764}$
4. Only if you implemented the bonus option 11: $\sqrt[10]{1024}$

Name the script file as **`calculator.script`**. Also submit a **`calculator.txt`** file which briefly explains your implementation.
The submission for these files is the same as for the previous assignments. Put all the files for this assignment in directory **`hw4`** and run the `~eecs10/bin/turnin.sh` command in the parent directory of **`hw4`** to submit your homework.