EECS 10: Assignment 4

Prepared by: Guantao Liu, Prof. Rainer Dömer

July 13, 2015

Due Monday July 20, 2015 at 11:00pm

1 A menu-driven calculator for floating point numbers: Option 1-5 [25 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. At the start of your program, the user is prompted to input a floating point number. For instance, it may look like this:

```
Welcome to my floating point number calculator!
Please input a floating point number: 4.2
```

Then the program should display the current result and a menu. In our example, it looks like this:

```
-------------------------------------------
The current result is: 4.200000
1.  Add a floating point number to the current result;
2.  Subtract a floating point number from the current result;
3.  Multiply the current result by a floating point number;
4.  Divide the current result by a floating point number;
5.  Take the absolute value of the current result;
6.  Get the approximate square root of the current result;
7.  Get the sine of the current result;
8.  Get the cosine of the current result;
9.  Get the tangent of the current result ;
10. Get the approximate Nth root of the current result;
11. Quit.

Please enter a selection:
```

Your program should let the user select an option, 1 through 11. If the user selects 11, your program simply exits. For other selections, your program should perform the corresponding floating point number operations.

When the user selects an operation, first ask the user to input another floating point number as an operand (with the exception of the operations requiring no 2nd operand and quit operations). Once the user inputs the operand, your program should perform the required operation and display the current result and the option menu again.

In our example, the user selects 1, the add operation. So the program should ask the user to input a floating number to be added to 4.2, like this:

```
    Please input a floating point number operand: 1.8
```

Then the program should bring up the current result and menu again. This time the current result should display 6.000000 (4.2 + 1.8 = 6.000000).

```
-----------------------------------------
The current result is: 6.000000
1.  Add a floating point number to the current result;
2.  Subtract a floating point number from the current result;
3.  Multiply the current result by a floating point number;
4.  Divide the current result by a floating point number;
5.  Take the absolute value of the current result;
6.  Get the approximate square root of the current result;
7.  Get the sine of the current result;
8.  Get the cosine of the current result;
9.  Get the tangent of the current result ;
10. Get the approximate Nth root of the current result;
11. Quit.

Please enter a selection:
```

1.2 Printing the Current Value

The current floating point number always holds the result of the previous operation. At the beginning, there is no previous operation, so the current result just shows the floating point number you input at the start. In our example, it is 4.2.

1.3 Implementation

To implement the program, you need to define a set of functions. Each function does a specific job, such as addition, subtraction, and division etc.

1.3.1 Function Declarations

As a starting point, use the following function declarations:

```
/* Calculate the absolute of the input value */
double Abs(double op);

/* Add two floating point numbers and return the value */
double Add(double op1, double op2);

/* Subtract a floating point number from another */
double Subtract(double op1, double op2);

/* Other functions you need to implement for this calculator */
...
```

Function `Add()` requires two floating point numbers as its input parameters, which are the first and second operand for the binary operation. `Abs()` requires one input parameter because it works only on the current result. You have to think out how many input argument(s) is/are required for other functions like Subtract, Multiply or Divide.

1.3.2 Error handling

A robust program is able to handle all situations, and reports any problem to the user if an error occurs. To get full credit, your program should be able to handle the following error:

- Notify the user with **`ERROR: Division by zero!`** if the user selects the division operation and inputs zero as the operand, and reprompt the user until a proper operand is entered.

2 Option 6-10 of the menu-driven calculator for floating point numbers [25+5 points]

Based on the program **`calculator.c`** you implement in the first part, you will be asked to extend the floating-point calculator with several advanced functions:

- **option 6**: Get the approximate square root of the current result;
- **option 7**: Get the sine of current result;
- **option 8**: Get the cosine of current result;
- **option 9**: Get the tangent of the current result;
- **option 10**: Get the approximate Nth root of the current result (Extra credits);

Note: In this assignment, please make "Quit" as **option 11** (no matter you have the bonus or not).

2.1 Option 6: Square root approximation [15 points]

Extend the calculator with the 6*th* function to calculate the approximate square root of the current floating-point value.

2.1.1 Square root approximation algorithm

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 $M*M$ is less than 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 proper 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 0.00001 so that we reach a precision of 5 digits after the decimal point for our approximation.

The pseudo-code of the algorithm can be written as follows.

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 in each iteration, as follows, if the user chooses "6" as the option:

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

```
-----------------------------------------
The current result is: 10.000000
1.  Add a floating point number to the current result;
2.  Subtract a floating point number from the current result;
3.  Multiply the current result by a floating point number;
4.  Divide the current result by a floating point number;
5.  Take the absolute value of the current result;
6.  Get the approximate square root of the current result;
7.  Get the sine of the current result;
8.  Get the cosine of the current result;
9.  Get the tangent of the current result ;
10. Get the approximate Nth root of the current result;
11. Quit.

Please enter a selection: 6
-----------------------------------------
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 20: the square root of 10.000000 is approximately 3.162279
```

Note that your program should run properly for any natural number which is the current result value.

2.2 Option 10: Bonus Problem [5 points]

Improve your program with the 10*th* option so that it can calculate the N-th root of any value. The value *N* should be a positive integer input by the user.

For example, your program should look like this for the bonus part:

```
-----------------------------------------
The current result is: 42.000000
1.  Add a floating point number to the current result;
2.  Subtract a floating point number from the current result;
3.  Multiply the current result by a floating point number;
4.  Divide the current result by a floating point number;
5.  Take the absolute value of the current result;
6.  Get the approximate square root of the current result;
7.  Get the sine of the current result;
8.  Get the cosine of the current result;
```

```
9.  Get the tangent of the current result ;
10. Get the approximate Nth root of the current result;
11. Quit.

Please enter a selection: 10
----------------------------------------
Please input the value of integer n (n>0): 5
Iteration 1: the 5th root of 42.000000 is approximately 21.000000
Iteration 2: the 5th root of 42.000000 is approximately 10.500000
...
Iteration 26: the 5th root of 42.000000 is approximately 2.111786
```

Note that your program should run properly for any natural number which is the current result value and any integer *N* which is greater than 0 (not only for the demo, the 5th root of value 42).

2.3 Option 7/8/9: Get the sine/cosine/tangent of current result value [10 points]

Extend the calculator with the 7*th*, 8*th* and 9*th* operation to get the sine, cosine and tangent value of the current result.

Please reuse the **`tan.c`** program from the previous assignment to create these three functions, and use them to get the result value.

2.4 Implementation

To implement the program, you need to define a set of more functions. Each function does a specific job.

2.4.1 Function Declarations

The following function declarations are recommended:

```
/* Get the approximate square root and return the value */
double ApproximateRoot2(double op1);

/* Get the approximate nth root and return the value */
double ApproximateRootN(double op1, unsigned int n);

/* Get the approximate sine value */
double ApproximateSin(double op1);

/* Get the approximate cosine value */
double ApproximateCos(double op1);

/* Get the approximate tangent value */
double ApproximateTan(double op1);
```

Functions `ApproximateRoot2()`, `ApproximateSin()`, `ApproximateCos()`, and `ApproximateTan()` requires one floating point number as its input parameter, which is the current result value. `ApproximateRootN()` requires one floating point number and one integer as its input parameters where the first one is the current result value and the second one is the *n* for Nth root.

2.4.2 Error handling

Note that the input value have to be within the range (-1.3, 1.3) for the tangent calculation. When the input value is out of this range and user wants to calculate the tangent value with it, your program should notify the user with **`ERROR: Input out of the range!`**, and reprompt the user until a proper operand is entered.

Also, if the current result value is negative and the user selects the option to get the approximate square/Nth root, the program should notify the user with **`ERROR: Square root of a negative number!`** / **`ERROR: Nth root of a negative number!`**, and reprompt the user until a proper operand is entered. In addition, for the approximate Nth root operation, if the input integer *n* is less than or equal to 0, your program should print the error message **`ERROR: Invalid integer N!`**, and prompt the user until a positive integer is entered.

3 Submission

You need to save your program as **`calculator.c`**.

The way you demonstrate your code depends on whether or not you completed the extra credit. If you did not complete the extra credit, perform the following steps to generate the script file:

1. Compile and run your program
2. Input the floating point number 4.2
3. Add the current result with 1.8
4. Subtract 2.71 from the current result
5. Multiply the current result by -10
6. Divide the current result by 32.9
7. Take the absolute value
8. Divide the current result by 0 /*error handling*/
9. Divide the current result by 0.333333
10. Multiply the current result by 0
11. Add the current result with 10
12. Get the square root of the current result value
13. Multiply the current result by 0
14. Add the current result with 1.1
15. Get the sine value for the current result
16. Get the cosine value for the current result
17. Get the tangent vaule for the current result
18. Exit your program

If you did complete the extra credit, perform the following steps to generate the script file:

1. Compile and run your program
2. Input the floating point number 4.2

3. Add the current result with 1.8
4. Subtract 2.71 from the current result
5. Multiply the current result by -10
6. Divide the current result by 32.9
7. Take the absolute value
8. Divide the current result by 0 /*error handling*/
9. Divide the current result by 0.333333
10. Multiply the current result by 0
11. Add the current result with 10
12. Get the square root of the current result value
13. Multiply the current result by 0
14. Add the current result with 28
15. Get the 5th root of the current result value
16. Multiply the current result by 0
17. Add the current result with 1.1
18. Get the sine value for the current result
19. Get the cosine value for the current result
20. Get the tangent vaule for the current result
21. Exit your program

Name the script file as **`calculator.script`**. In addition to the above steps, your script file shows that you compile your program.

You also need to create a **`calculator.txt`** file, which briefly explains your implementation.

Submission for these files will be similar to last week's assignment. Put all the files for assignment 4 in directory **`hw4/`** and run the **/ecelib/bin/turnin10** command in the parent directory of **`hw4/`** to submit your homework.