Due Date: Feb 16, 2022 @ 11:59:59

Basics of C++ – MathLib

This assignment will focus on the basic operations of C++. You will create a math library from basic principles using the basic C++ operators (+,-,/,%,type casting, etc). That means you should not use any pre-built functions or libraries except where it is explicitly allowed. You can, of course, use the functions in your library in the other functions of the library.

The library will be a c-style library, meaning a collection of functions (no classes).

The principal goal of this assignment is to (re)acquaint yourself with developing code in the *nix environment. Your will submit your work to Gradescope and it will be both autograded and human graded. The autograder will compile your code with g++-7.5.0.

Be sure to watch Piazza for any clarifications or updates to the assignment.

Don't forget the Assignment 1 Canvas Quiz! Your assignment submission is not complete without the quiz.

MathLib.h

Your math library must consist of 2 files: `MathLib.h` and `MathLib.cpp`. The first being the header file and the second being the implementation of the methods.

For `MathLib.h`:

- Be sure to have include guards.
- Don't import anything that is unnecessary.
- Include file header comments that have your name, student id, a brief description of the library, and be sure to cite any resource (site or person) that you used in implementing the library.
- Each function should have appropriate function header comments.
  - You can use javadoc style (see doxygen) or another style, but be consistent.
  - Be complete: good description of the function, parameters, and return values.
- For this assignment, no classes and no inline methods.

For `MathLib.cpp`:

- All the function bodies should be here.
- Don't clutter your code with useless inline comments (it is a code smell [Inline Comments in the Code is a Smell, but Document the Why], unless it is the why).
- Don't use any pre-built functions.
- Follow normal programming practices for spacing, naming, etc: Be reasonable and consistent.
- Be sure to avoid redundant code.

Functions to implement: [1]

`E absVal(E)` This function should take a number and return the absolute value of that number, where `E` is the data type. Your library should have an `int`, `long`, and `double` version.

`long ceiling(double)` This function returns the ceiling of the double parameter.

`long floor(double)` This function returns the floor of the double parameter.

`long round(double, RoundingRule)` This function will require you to define an enum datatype called `RoundingRule`. The values in the enum are: `ROUND_DOWN, ROUND_UP, ROUND_TO_ZERO, ROUND_AWAY_ZERO, ROUND_HALF_UP, ROUND_HALF_DOWN, ROUND_HALF_TO_ZERO, ROUND_HALF_AWAY_ZERO, ROUND_HALF_TO_EVEN, ROUND_HALF_TO_ODD`. The default `RoundingRule` should be `ROUND_HALF_UP`.

This function will round the double parameter based on the `RoundingRule` as follows:

`ROUND_DOWN` **(resp.** `UP`**)** The floor (resp. ceiling) of the double.

`ROUND_TO_ZERO` **(resp.** `AWAY_ZERO`**)** Round to the integer closer to (resp. further from) zero.

`ROUND_HALF_*` Rounds to the nearest integer, e.g. $5.x$ becomes 5 when $x < 0.5$ and 6 when $x > 0.5$, but requires a tie-breaker when $x = 0.5$. The tie-breaking being:

`DOWN` **(resp.** `UP`**)** Floor (resp. ceiling).

`TO_ZERO` **(resp.** `AWAY_ZERO`**)** Closer to (further from) zero.

`TO_EVEN` **(resp.** `TO_ODD`**)** Round to the nearest even (resp. odd) integer.

`double pow(double, long)` Raises the double value to the long exponent.

`std::string toString(int num, unsigned int radix, bool)` This function builds a `std::string` representation of the int (first) parameter using the unsigned int (second) parameter as the radix (i.e. numerical base). The string representation of the `int` will use the digits from 0 - 9 followed by a - z as needed. Negative numbers are prefixed with `'-'`.

If the radix is greater than 36, return a string containing `"Radix too large."`. The radix should have a default value of `10`.

The bool parameter indicates if the returned number contains lower case (`true`) or upper case (`false`) letters. This only applies when the radix is less than 37. It should have a default value of `true`.

Remember that this should be done without prebuilt functions like printf or streams.

`int gcd(int, int)` Return the greatest common divisor of the two parameters.

demo.cpp

You must create a program (place it in `demo.cpp`) that demos all the functions in the library. There is no fixed output required, but you demo.cpp requires:

- A `main` function.
- It must produce output (use the `std::cout`) that demos all the functions.
- There must at least 1 call to each of the functions described above for `MathLib.h`.

[1] Many of the functions involve rounding in some way: Wiki:Rounding may be a useful resource.

Makefile

Your submission must include a `Makefile`. The minimal requirements are:

- Compile using the demo executable with the `make` command.
- Use appropriate variables for the compiler and compiler options.
- It must contain a clean rule.
- The demo executable must be named something meaningful (not a.out).

README.md

Your submission must include a `README.md` file (see https://guides.github.com/features/mastering-markdown/). The file should give a brief description of the program, the organization of the code, how to compile the code, and how to run the program.

Submission

You will upload all the required files to Gradescope. The detail on how to do that will become available to you once you have completed the Assignment 1 Canvas quiz.

There are no limits on the number of submissions. See the syllabus for the details about late submissions.

Grading

For this assignment, half of the marks will come from the autograder. For this assignment, none of the test details have been hidden, but there may be hidden tests in future assignments.

The other half of the marks will come from human-grading of the submission. Your files will be checked for style and to ensure that they meet the requirements described above. In addition, all submitted files should contain a header (comments or otherwise) that, at a minimum, give a brief description of the file, your name, and wisc id.

Rubric

- Autograder Tests - 40
- DEMO - 10
  - After completing step 4 (see below), come out to office hours and demo your working environment to Marc or Ziyi. This should be done in the first two weeks!
- README File - 5
  - Name / student id
  - Typeset/organized with markdown
  - Program description
  - Compilation instructions
  - Run instructions
  - Code organization description

- Makefile - 5
  - File header comments including name/id
  - Demo build rule with appropriate named executable
  - MathLib.o build rule
  - Clean build rule
  - Correct file name
- demo.cpp - 4
  - File header comments including name/id
  - All library functions tested
- MathLib.h - 6
  - File header comments including name/id
  - Include guards
  - All includes are required for header file
  - Function header comments with function description, return value and parameters
  - Comment detailing other elements like enums
  - Function prototypes include default values as per the specification
  - Only inline function bodies are defined in header
- MathLib.cpp - 10
  - File header comments including name/id
  - No unused/unneeded includes
  - Consistent style (braces, spacing, etc)
  - Avoid inline comments
  - Avoid duplicate code
  - Avoid c-style casting
  - Avoid gotos
  - Well-designed functions

Tackling Assignment 1

1. Figure out your coding environment. Since the autograder environment uses `g++`, it is highly recommended to establish a similar environment.
2. Create a directory for your assignment 1 project, and create the required files: `MathLib.cpp`, `MathLib.h`,`demo.cpp`, `README.md`, and `Makefile`.
3. Start small and make sure you have the basics down: implement the `abs` functions. Put the function header details in `MathLib.h`, the function details in `MathLib.cpp`, some test calls in `demo.cpp`, and set up your makefile.

4. Test your `abs` functions in the autograder: For the autograder to compile, all the functions must be defined. So, add the other function prototypes and enum definition to `MathLib.h`, and add empty function bodies to `MathLib.cpp`. Then, submit to gradescope. (Pro tip: When adding you function prototypes, also do the function header comment).
5. DEMO YOUR ENVIRONMENT: Come out to Marc or Ziyi office hours to demo your coding environment, and your autograder submission.
6. Implement and test the `ceiling` and `floor` functions. Do they call each other?
7. Implement and test the `round` function. Did you define the enum in `MathLib.h`? Did you avoid duplicate code? Did you use calls to `ceiling` and `floor`?
8. Implement and test `pow`.
9. Implement and test `toString`.
10. Implement and test `gcd`.
11. Don't forget to do the `README.md` file.
12. Final submission: make sure you have file comments, and name on all files. Double check the rubric, and do your final submission to Gradescope.

HAPPY CODING!