[HW] C-String Calculator

Or: How I Learned to Stop Worrying and Love Classes

# Overview

Yes, you’ve built a string calculator. Bjarne is very impressed. But, Bjarne’s friend Dennis is not impressed. “C++ is too easy, too high-level!”, Dennis exclaims, “Do you even understand how it works?”. Dennis challenges you to remove the dependency on the C++ string library entirely. Oh… Oh! [Challenge Accepted](https://meers2review.files.wordpress.com/2015/06/challenge-accepted-meme.jpg). Afterall, you’re learning about classes and dynamic memory right now, so this will be really great practice. All you have to do is recreate the parts of the string library that you used in your String Calculator. [You’ve Got This](https://www.barexambrief.com/wp-content/uploads/2015/07/Fotolia_86635796_Subscription_Monthly_M.jpg)!

# Objectives

1. Practice creating and using user-defined objects in C++
   1. Implement a portion of the C++ string library from scratch
2. Practice problem-solving
   1. Implement standard string processing algorithms
   2. Create or discover novel algorithms
   3. Think before you code
3. Practice test-driven development
   1. Unit testing string functions
4. Read the header file

# Starter Code

The starter code on Mimir contains six files in two groups:

1. Starter code from String Calculator updated to include the new header files and use the new my\_string class instead of std::string and to use pass by constant reference instead of pass by value for the my\_string objects (this will help with runtime efficiency)
   1. main.cpp := the user interface that drives the program.
   2. string\_calculator.h := contains function declarations.
   3. string\_calculator.cpp := contains function definitions
   4. **You should replace the contents of the source files with your existing source files for String Calculator, taking care to s/std::string/my\_string (find+replace “std::string” with “my\_string”) and s/std::istringstream/my\_istringstream.**
2. Starter code for creating the my\_string and my\_istringstream classes
   1. my\_string.h := contains a basic class definition for my\_string and a few suggestions for methods to implement.
   2. my\_sstream.h := contains a basic class definition for my\_istringstream and a few useful methods already declared (here) and implemented (in my\_string.cpp).
   3. my\_string.cpp := contains definitions of methods for both the my\_string and my\_istringstream classes.
   4. **You should identify the string and istringstream methods you used in your String Calculator, declare them in the header files, and define them in the source file.**

**Read the Header and Source Files**

# Requirements

1. The source files to submit are named
   1. main.cpp
   2. my\_sstream.h
   3. my\_string.cpp
   4. my\_string.h
   5. string\_calculator.cpp
   6. string\_calculator.h
2. You are required to implement whatever functions from the string library that you use in (re-)solving the String Calculator problem.
   1. All of the same functional requirements from the original HW remain. [Re-read the original prompt](https://docs.google.com/document/d/1sTOW4hkO8ERdSlW9RqWiTwnXV2v0fei144d15tV9Ebo/edit?usp=sharing) to re-familiarize yourself with those requirements.
      1. Changes:
         1. std::string → my\_string
            1. <string> → “my\_string.h”
         2. my\_string parameters are passed by constant reference, e.g. const my\_string& num
         3. If you used std::istringstream, you should replace it with my\_istringstream.
            1. <sstream> → “my\_sstream.h”
   2. A list of suggested functions to implement is provided in the starter code
      1. Read the header and source files
      2. Not all of the suggested functions are required and some required functions are not suggested
         1. What is required to be implemented depends on what is used in the String Calculator.
         2. If you did not use a certain method, e.g. string::front(), you do not need to implement it.
         3. If you need a certain method, but it is not suggested, e.g. string::back(), then you need to implement it.
            1. Corollary: if you cannot figure out how to implement the method you need, you can think about a way to do the same thing with other methods which you do know how to implement, e.g. if you used a reverse iterator, you could instead use a simple for loop.
   3. Unit tests for the my\_string and my\_istringstream classes may be posted to Mimir later in the week
      1. **You are responsible for testing your own code**
      2. The original test cases for the String Calculator are on Mimir, which you can use to help you test the correctness of your string library implementation
3. You may use the following includes:
   1. iostream
   2. limits
   3. my\_sstream
   4. my\_string
   5. string\_calculator.h

**Use of an unapproved header file will result in a score of 0.**

**Use (or even comment) of std::string anywhere will result in a score of 0.**

1. The program must compile without warnings or errors.

g++ -std=c++17 -Wall -Wextra -pedantic -Weffc++ -g string\_calculator.cpp main.cpp my\_string.cpp

* 1. We also recommend that you use the Sanitizers and/or Valgrind to help you find and fix memory errors.
     1. Sanitizers:

g++ -std=c++17 -Wall -Wextra -pedantic -Weffc++ -g **-fsanitize=address,undefined** string\_calculator.cpp main.cpp my\_string.cpp

./a.out

* + 1. Valgrind:

g++ -std=c++17 -Wall -Wextra -pedantic -Weffc++ -g string\_calculator.cpp main.cpp my\_string.cpp

**valgrind --leak-check=full ./a.out**

1. The program must run without errors or unhandled exceptions.

# Recommendations

1. Read the header and source files.
2. Think before you code.
3. Pick small problems to solve first, write test cases first, and develop incrementally.
4. Use descriptive (long) naming conventions for variables and functions.
5. Add comments to the code to describe anything which is not obvious from the code.
6. Use whitespace (indentation, newlines) to visually organize code.
7. Use functions to reduce code duplication and increase abstraction.

# Getting Started

1. Start now.
2. Download the starter code.
3. Read the header and source files.
4. Compile the program.
   1. It will not compile due to undefined reference errors.
   2. You can comment out the code you don’t need yet
      1. It’s mostly the my\_istringstream methods at the bottom of my\_string.cpp
   3. Or, you can pick those functions to implement first.
5. Submit it to Mimir.
   1. So that you can see the test cases.
6. Pick a string library method to implement
   1. Recommendations: my\_string::size(), my\_string::at(size\_type), operator<<(std::ostream&, const my\_string&) (the ostream insertion operator for my\_string objects)
   2. Think before you code: plan your program on paper (digital or analog) before mashing the keyboard
   3. Write test cases before writing the implementation of the method
7. Recompile and rerun (run == test).
   1. Check for errors.
   2. If no errors, move on
   3. Else, start debugging
8. Submit to Mimir.
   1. So that you can checkpoint your code and see what your new effort to implement new functionality has earned for you.
9. Continue by pickling a new string library method to implement (step 6)
10. When you get stuck, ask for help on Piazza and attend office hours. The TAs and the instructors will help you.
    1. Read the prior posts before posting a new question (which may be a duplicate).

# Submission

1. The source files to submit are named
   1. main.cpp
   2. my\_sstream.h
   3. my\_string.cpp
   4. my\_string.h
   5. string\_calculator.cpp
   6. string\_calculator.h

# Have Fun!

Did you read the header and source files, yet? =)