Project 1

Due October 5, 2022 at 9:00 PM

You will be working alone for this project. This specification is subject to change at any time for additional clarification.

Desired Outcomes

- Exposure to using C++ std::string
- Exposure to GoogleTest
- Use of git repository
- An understanding of how to develop Makefiles that build and execute unit tests
- An understanding of how to calculate edit distance

Project Description

You will be implementing a set of C++ string manipulation utilities that are like those available in python. To guide your development and to provide exposure to Test Driven Development, you will be developing GoogleTest tests to test your functions. You will also be developing a Makefile to compile and run your tests. You must use good coding practice by developing this project in a git repository. The string utility functions that you will have to develop are as follows:

```
// Returns a substring of the string str, allows for negative values as in
// python end == 0 means to include end of string
std::string Slice(const std::string &str, ssize_t start, ssize_t end=0);

// Returns the capitalized string as in python
std::string Capitalize(const std::string &str);

// Returns the upper- or lower-case strings as in python
std::string Upper(const std::string &str);
std::string Lower(const std::string &str);

// Returns the left/right/both stripped strings (white space characters are
// removed from left, right or both)
std::string LStrip(const std::string &str);
std::string RStrip(const std::string &str);
std::string Strip(const std::string &str);

// Returns the center/left/right justified strings
std::string Center(const std::string &str, int width, char fill = ' ');
std::string LJust(const std::string &str, int width, char fill = ' ');
std::string RJust(const std::string &str, int width, char fill = ' ');

// Returns the string str with all instances of old replaced with rep
std::string Replace(const std::string &str, const std::string &old, const
std::string &rep);

// Splits the string up into a vector of strings based on splt parameter, if
// splt parameter is empty string, then split on white space
```

```
std::vector< std::string > Split(const std::string &str, const std::string
&splt = "");

// Joins a vector of strings into a single string
std::string Join(const std::string &str, const std::vector< std::string >
&vect);

// Replaces tabs with spaces aligning at the tabstops
std::string ExpandTabs(const std::string &str, int tabsize = 4);

// Calculates the Levenshtein distance (edit distance) between the two
// strings. See https://en.wikipedia.org/wiki/Levenshtein_distance for
// more information.
int EditDistance(const std::string &left, const std::string &right, bool
ignorecase=false);
```

The Makefile you develop needs to implement the following:

- Must create `obj` directory for object files (if doesn't exist)
- Must create `bin` directory for binary files (if doesn't exist)
- Must compile string utils file and string utils tests using C++17
- Must link string utils and string utils tests object files to make `teststrutils` executable
- Must execute the `teststrutils` executable
- Must provide a `clean` that will remove the `obj` and `bin` directories

You can unzip the given tgz file with utilities on your local machine, or if you upload the file to the CSIF, you can unzip it with the command:

```
tar -xzvf proj1given.tgz
```

You **must** submit the source file(s), your Makefile, and README.txt file, in a tgz archive. Do a `make clean` prior to zipping up your files so the size will be smaller. You can tar gzip a directory with the command:

```
tar -zcvf archive-name.tgz directory-name
```

You should avoid using existing source code as a primer that is currently available on the Internet. You **must** specify in your readme file any sources of code that you have viewed to help you complete this project. All class projects will be submitted to MOSS to determine if students have excessively collaborated. Excessive collaboration, or failure to list external code sources will result in the matter being referred to Student Judicial Affairs.

Recommended Approach

The recommended approach is as follows:

1. Create a git repository and add the provided files.
2. Create a Makefile to meet the specified requirements. Since no tests have been written, all tests should pass.
3. Write tests for each of the functions. Each test you write should fail, make sure to have sufficient coverage of the possible input parameters.

4. Once tests have been written that fail with the initial skeleton functions, begin writing your functions. You may find that you can write some of your functions based upon others you have already developed.

Grading

The point breakdown can be seen in the table below. Make sure your code compiles on the CSIF as that is where it is expected to run. You will make an interactive grading appointment to have your assignment graded. You must have a working webcam for the interactive grading appointment. Project submissions received 24hr prior to the due date/time will received 10% extra credit. The extra credit bonus will drop off at a rate of 0.5% per hour after that, with no additional credit being received for submissions within 4hr of the due date/time.

Points	Description
10	Has git repository with appropriate number of commits
5	Has Makefile and submission compiles
10	Makefile meets specified requirements
15	Has google tests that fail with initial skeleton functions
15	Student google tests have reasonable coverage of string util functions
10	Google tests detect errors in instructor buggy code
15	String util functions pass student google tests
10	String util functions pass instructor tests
10	Student understands all code they have provided

Helpful Hints

- Read through the guides that are provided on Canvas
- See http://www.cplusplus.com/reference/, it is a good reference for C++ built in functions and classes
- Use lenth(), substr(), etc. from the string class whenever possible.
- If the build fails, there will likely be errors, scroll back up to the first error and start from there.
- You may find the following line helpful for debugging your code:
  `std::cout<<__FILE__<<" @ line: "<<__LINE__<<std::endl;`
  It will output the line string "FILE `@ line: X`" where FILE is the source filename and X is the line number the code is on. You can copy and paste it in multiple places and it will output that particular line number when it is on it.