Project 1

Due October 9, 2019 at 11:59 PM

This project specification is subject to change at any time for clarification. For this project you will be working alone. You will be implementing a set of C++ string manipulation utilities that are similar to those available in python. In addition, you will be implementing a C++ path class that will ease with the manipulation of path names in Linux. In order to guide your development and to provide exposure to Test Driven Development, your implementation will be required to pass a set of GoogleTest tests that have been provided. 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
std::string Slice(const std::string &str, ssize_t start, ssize_t
end=0);

// Returns the capitalize or title capitalization strings as in
// python
std::string Capitalize(const std::string &str);
std::string Title(const std::string &str);

// Returns the left/right/both stripped strings
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
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);
```

In developing the CPath class, you may find chapter 2 of The Linux Command Line useful. You may also want to test the behavior of the equivalent functions in python os.path. See https://docs.python.org/3.7/library/os.path.html for python descriptions. Not all functions have an equivalent in python, and the behavior may slightly differ. The path member functions that you will have to develop are as follows:

```
// Constructors/destructor
CPath();
CPath(const CPath &path);
CPath(const std::string &path);
~CPath();

// Assignment operator overload
CPath &operator=(const CPath &path);

// Path concatenation, returned path should be calling path
// concatenated with the path parameter if path parameter is a
// relative path. If path is an absolute path, path is returned.
CPath operator+(const CPath &path) const;

// Compares paths to be equal or not equal
bool operator==(const CPath &path) const;
bool operator!=(const CPath &path) const;

// Returns the directory component of a path
CPath Directory() const;

// Returns the basename component (the "filename") of the path
std::string BaseName() const;

// Returns the extention of the basename, or empty string if one
// does not exist
std::string Extension() const;
```

```
// Returns true if the path is an absolute path starting with /
bool IsAbsolute() const;

// Converts the path into a string
std::string ToString() const;
operator std::string() const;

// Returns the path as an absolute path
CPath AbsolutePath() const;

// Finds the longest common path between the path and path
// parameter
CPath CommonPath(const CPath &path) const;

// Normalizes the path, removing any unnecessary .., . in the
// middle of the path.
CPath NormalizePath() const;

// Returns a relative path to the path parameter from the
// calling path
CPath RelativePathTo(const CPath &path) const;

// Static absolute path function
static CPath AbsolutePath(const CPath &path);

// Returns the current working path
static CPath CurrentPath();

// Static common path function
static CPath CommonPath(const CPath &path1, const CPath &path2);

// Expands ~ paths into absolute paths using the home
static CPath ExpandUserPath(const CPath &path);

// Returns the home path
static CPath HomePath();

// Static path normalization function
static CPath NormalizePath(const CPath &path);

// Static relative path function
static CPath RelativePath(const CPath &path, const CPath
&startpath = CurrentPath());
```

Two static functions have been provided for you: `GetCWD`, and `GetHome`. These will be helpful in implementing the `CurrentPath`, `ExpandUserPath` and `HomePath` functions. Once all

tests have passed a simple interactive program will be generated that uses the string and path utilities. A working example can be found on the CSIF in /home/cjnitta/ecs34/.

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), the provided unmodified 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
```

Provide your interactive grading timeslot csv file in the tgz. The directions for filling it out have been posted.

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.

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.
- Use your StringUtils functions to implement your CPath class.
- Make sure you are in the project directory with the `Makefile` when you type `make`.
- 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.