Project 2

Due October 19, 2022 at 9:00 PM

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

Desired Outcomes

- Exposure to GoogleTest
- Exposure to Expat XML library
- Use of git repository
- An understanding of how to develop Makefiles that build and execute unit tests
- An understanding of delimiter-separated-value files
- An understanding of XML files
- An understanding of how to integrate a third-party C library in C++

Project Description

You will be implementing several C++ classes to generate and parse both delimiter-separated-value (DSV) and XML files. To guide your development and to provide exposure to Test Driven Development, you will be developing GoogleTest tests to test your classes. 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. DSV is a way of exchanging information, you can read more here. Some of the most common forms are comma-separated-value (CSV) and tab-separated-value (TSV) files. You will be implementing two classes `CDSVReader` and `CDSVWriter` that have simple interfaces for parsing and generating DSV respectively.

```
// Constructor for DSV reader, src specifies the data source and delimiter
// specifies the delimiting character
CDSVReader(std::shared_ptr< CDataSource > src, char delimiter);

// Destructor for DSV reader
~CDSVReader();

// Returns true if all rows have been read from the DSV
bool End() const;

// Returns true if the row is successfully read, one string will be put in
// the row per column
bool ReadRow(std::vector< std::string > &row);

// Constructor for DSV writer, sink specifies the data destination, delimiter
// specifies the delimiting character, and quoteall specifies if all values
// should be quoted or only those that contain the delimiter, a double quote,
// or a newline
CDSVWriter(std::shared_ptr< CDataSink > sink, char delimiter,
    bool quoteall = false);
```

```
// Destructor for DSV writer
~CDSVWriter();

// Returns true if the row is successfully written, one string per column
// should be put in the row vector
bool WriteRow(const std::vector< std::string > &row);
```

Some nuances to keep in mind when developing the functions.

- Values that have either the delimiter, double quote character '"', or newline must be quoted with double quotes.
- Double quote character in the cell must be replace with two double quotes.
- An empty line is a valid row where there are not values

You will be implementing two classes `CXMLReader` and `CXMLWriter` that have simple interfaces for parsing and generating XML respectively. The `SXMLEntity` struct is used by both classes and has been provided. Complete XML parsing is complicated and beyond the scope of a single assignment; therefore, you will be utilizing the Expat library, a commonly used library for parsing XML. To link the Expat library (libexpat) with your project, use the `-lexpat` as a linker option.

```
SXMLEntity{
    EType DType; // Type of entity start/end/complete element
    std::string DNameData;
    std::vector< TAttribute > DAttributes;
}

// returns true if the attribute name is in the DAttributes
bool AttributeExists(const std::string &name) const;

// returns the value of the attribute, or empty string if
// doesn't exist
std::string AttributeValue(const std::string &name) const;

// Sets the attribute name to the value
bool SetAttribute(const std::string &name, const std::string &value);

// Constructor for XML reader, src specifies the data source
CXMLReader(std::shared_ptr< CDataSource > src);

// Destructor for XML reader
~CXMLReader();

// Returns true if all entities have been read from the XML
bool End() const;

// Returns true if the entity is successfully read if skipcdata
// is true only element type entities will be returned
bool ReadEntity(SXMLEntity &entity, bool skipcdata = false);

// Constructor for XML writer, sink specifies the data destination
CXMLWriter(std::shared_ptr< CDataSink > sink);
```

```
// Destructor for XML writer
~CXMLWriter();

// Outputs all end elements for those that have been started
bool Flush();

// Writes out the entity to the output stream
bool WriteEntity(const SXMLEntity &entity);
```

To abstract the data input/output for the DSV and XML classes, two simple abstract classes `CDataSource` and `CDataSink` have been provided for input and output respectively. String implementations of `CDataSource` and `CDataSink`, `CStringDataSource` and `CStringDataSink` respectively have been provided with associated GoogleTest tests.

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 cpp files using C++17
- Must link string utils and string utils tests object files to make `teststrutils` executable
- Must link `StringDataSource` and `StringDataSourceTest` object files to make `teststrdatasource` executable
- Must link `StringDataSink` and `StringDataSinkTest` object files to make `teststrdatasink` executable
- Must link DSV reader/writer and DSV tests object files to make `testdsv` executable
- Must link XML reader/writer and XML tests object files to make `testxml` executable
- Must execute the `teststrutils`, `teststrdatasource`, `teststrdatasink`, `testdsv`, and `testxml` executables
- 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 proj2given.tgz
```

You **must** submit the source file(s), your Makefile, README file, and `.git` directory 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 your project 1 and provided files.
2. Update your project 1 Makefile to meet the specified requirements. The order of the tests to be run should be `teststrutils`, `teststrdatasource`, `teststrdatasink`, `testdsv`, and then `testxml`
3. Verify that your string utils, string data source, and string data sink tests all compile, run and pass.
4. Create the files and skeleton functions for DSVReader.cpp, DSVWriter.cpp, XMLReader.cpp, XMLWriter.cpp, DSVTest.cpp, and XMLTest.cpp.
5. Write tests for the DSV and XML classes. Each test you write should fail, make sure to have sufficient coverage of the possible input parameters.
6. Once tests have been written that fail with the initial skeleton functions, begin writing your DSV functions. You may want to start with the writer, this may allow the use of the writer in testing the reader.
7. Once the DSV classes are complete, begin writing your XML functions. Like the DSV functions, you may want to start with the writer, this may allow the use of the writer in testing the reader.

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
5	Makefile meets specified requirements
5	Has DSV google tests that fail with initial skeleton functions
5	Has XML google tests that fail with initial skeleton functions
10	Student DSV google tests have reasonable coverage
10	Student XML google tests have reasonable coverage
10	Google tests detect errors in instructor buggy code
10	DSV functions pass all student tests
10	XML functions pass all student tests
5	DSV functions pass instructor tests
5	XML 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.
- Make sure to use a tab and not spaces with the Makefile commands for the target
- `make` will not warn about undefined variables by default, you may find the `--warn-undefined-variables` option very helpful
- The debug option for make can clarify which targets need to be built, and which are not. The basic debugging can be turned on with the `--debug=b` option. All debugging can be turned on with the `--debug=a` option.
- Make sure to use a `.gitignore` file to ignore your object files, and output binaries.
- Do not wait to the end to merge with you partner. You should merge your work together on a somewhat regular basis (or better yet pair program).
- Use `CStringDataSource` and `CStringDataSink` to test your reader and writer classes for DSV and XML.
- You will probably want to use static functions in your classes for the callbacks to the library calls that require callbacks. The call data (`void *`) parameter that the functions take and the callbacks pass back as a parameter, should be `this` from your object.
- You may find https://www.xml.com/pub/1999/09/expat/index.html helpful for describing the libexpat functions. You are not going to need to use every function in the Expat library.