Project 3

Due March 3, 2025 at 11:59 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
- Exposure to Open Street Map file formats
- Use of git repository
- Exposure to documenting code in Markdown
- 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 developing classes that can parse bus route information and Open Street Map (OSM) files utilizing the DSV and XML classes developed in project 2. The OSM files provided will be in the OSM XML format (See OSM XML for more information). For the OSM files you will only need to worry about the `node` and `way` tags and their children. The bus route information will be provided in two Comma Separated Value (CSV) files. One will provide a list of stop IDs associated with node IDs (node IDs refer to OSM nodes). The other file will provide the routes with a list of stop IDs.

The `COpenStreetMap` class you will be developing will implement the `CStreetMap` abstract interface. There are two struct abstract interfaces that are part of the `CStreetMap`, that will need to be implemented as well (`SNode` and `SWay`). The street map is a collection of nodes, and ways. Each way is a set of nodes that specifies a street path. The nodes and ways may have attributes associated with them that appear in tag elements. For example, a way might specify if bikes are allowed on the way or not.

```
// COpenStreetMap member functions
// Constructor for the Open Street Map
COpenStreetMap(std::shared_ptr<CXMLReader> src);

// Destructor for the Open Street Map
~COpenStreetMap();

// Returns the number of nodes in the map
std::size_t NodeCount() const noexcept override;

// Returns the number of ways in the map
std::size_t WayCount() const noexcept override;
```

```
// Returns the SNode associated with index, returns nullptr if index is
// larger than or equal to NodeCount()
std::shared_ptr<SNode> NodeByIndex(std::size_t index) const noexcept override;

// Returns the SNode with the id of id, returns nullptr if doesn't exist
std::shared_ptr<SNode> NodeByID(TNodeID id) const noexcept override;

// Returns the SWay associated with index, returns nullptr if index is
// larger than or equal to WayCount()
std::shared_ptr<SWay> WayByIndex(std::size_t index) const noexcept override;

// Returns the SWay with the id of id, returns nullptr if doesn't exist
std::shared_ptr<SWay> WayByID(TWayID id) const noexcept override;

// Street Map Node member functions
// Returns the id of the SNode
TNodeID ID() const noexcept override;

// Returns the lat/lon location of the SNode
TLocation Location() const noexcept override;

// Returns the number of attributes attached to the SNode
std::size_t AttributeCount() const noexcept override;

// Returns the key of the attribute at index, returns empty string if index
// is greater than or equal to AttributeCount()
std::string GetAttributeKey(std::size_t index) const noexcept override;

// Returns if the attribute is attached to the SNode
bool HasAttribute(const std::string &key) const noexcept override;

// Returns the value of the attribute specified by key, returns empty string
// if key hasn't been attached to SNode
std::string GetAttribute(const std::string &key) const noexcept override;

// Street Map Way
// Returns the id of the SWay
TWayID ID() const noexcept override;

// Returns the number of nodes in the way
std::size_t NodeCount() const noexcept override;

// Returns the node id of the node at index, returns InvalidNodeID if index
// is greater than or equal to NodeCount()
TNodeID GetNodeID(std::size_t index) const noexcept override;

// Returns the number of attributes attached to the SWay
std::size_t AttributeCount() const noexcept override;

// Returns the key of the attribute at index, returns empty string if index
// is greater than or equal to AttributeCount()
std::string GetAttributeKey(std::size_t index) const noexcept override;

// Returns if the attribute is attached to the SWay
bool HasAttribute(const std::string &key) const noexcept override;
```

```
// Returns the value of the attribute specified by key, returns empty string
// if key hasn't been attached to SWay
std::string GetAttribute(const std::string &key) const noexcept override;
```

The `CCSVBusSystem` class you will be developing will implement the `CBusSystem` abstract interface. There are two struct abstract interfaces that are part of the `CBusSystem`, that will need to be implemented as well (`SStop` and `SRoute`). The bus system is a collection of stops, and routes. Each route is a set of stops that specifies the locations where the bus stops on the route.

```
// CCSVBusSystem member functions
// Constructor for the CSV Bus System
CCSVBusSystem(std::shared_ptr< CDSVReader > stopsrc, std::shared_ptr<
CDSVReader > routesrc);

// Destructor for the CSV Bus System
~CCSVBusSystem();

// Returns the number of stops in the system
std::size_t StopCount() const noexcept override;

// Returns the number of routes in the system
std::size_t RouteCount() const noexcept override;

// Returns the SStop specified by the index, nullptr is returned if index is
// greater than equal to StopCount()
std::shared_ptr<SStop> StopByIndex(std::size_t index) const noexcept override;

// Returns the SStop specified by the stop id, nullptr is returned if id is
// not in the stops
std::shared_ptr<SStop> StopByID(TStopID id) const noexcept override;

// Returns the SRoute specified by the index, nullptr is returned if index is
// greater than equal to RouteCount()
std::shared_ptr<SRoute> RouteByIndex(std::size_t index) const noexcept
override;

// Returns the SRoute specified by the name, nullptr is returned if name is
// not in the routes
std::shared_ptr<SRoute> RouteByName(const std::string &name) const noexcept
override;

// Bus System Stop member functions
// Returns the stop id of the stop
TStopID ID() const noexcept override;

// Returns the node id of the bus stop
CStreetMap::TNodeID NodeID() const noexcept override;

// Bus System Route member functions
// Returns the name of the route
std::string Name() const noexcept override;

// Returns the number of stops on the route
std::size_t StopCount() const noexcept override;
```

```
// Returns the stop id specified by the index, returns InvalidStopID if index
// is greater than or equal to StopCount()
TStopID GetStopID(std::size_t index) const noexcept override;
```

An example CSV and OSM file set will be provided, the files will be used in project 4. Your tests should be built with them in mind, but you shouldn't load the files as part of the 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 link CSVBusSystem and CSVBusSystem tests object files to make `testcsvbs` executable
- Must link OpenStreetMap and OpenStreetMap tests object files to make `testosm` executable
- Must execute the `teststrutils`, `teststrdatasource`, `teststrdatasink`, `testdsv`, `testxml`, `testcsvbs` and `testosm` executables
- Must provide a `clean` that will remove the `obj` and `bin` directories

You must have a `docs` directory that contains Markdown (.md) files that document the `CBusSystem`, `CCSVBusSystem`, `CStreetMap`, and `COpenStreetMap` classes and their use. The documentation of each class and function should be consistent. Code examples are excellent for documenting the use of the developed classes.

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

```
unzip proj3given.zip
```

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

```
zip -r archive-name.zip 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.md file any sources of code that you have viewed to help you complete this project. You **MUST** properly document **ALL** uses of Generative AI following the guidelines outlined in the Generative AI Restrictions. 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`, `testxml`, `testcsvbs` and then `testosm`
3. Verify that your string utils, string data source, string data sink, DSV reader, DSV writer, XML reader, XML writer tests all compile, run and pass.
4. Create the files and skeleton functions for CSVBusSystem.cpp, OpenStreetMap.cpp, CSVBusSystemTest.cpp, and OpenStreetMapTest.cpp.
5. Write tests for the `CCSVBusSystem` and `COpenStreetMap` classes. Each test you write should fail, make sure to have sufficient coverage of the possible data input.
6. Once tests have been written that fail with the initial skeleton functions, begin writing your `CCSVBusSystem` functions. You likely will want to implement the `SStop` and `SRoute` classes inside the `SImplementation` struct.
7. Once the `CCSVBusSystem` classes are complete, begin writing your OSM functions. Like the `CCSVBusSystem` functions, you likely will want to implement the `SNode` and `SWay` classes inside the `SImplementation` struct.

Grading

Make sure your code compiles on Gradescope and passes all the test cases.

Helpful Hints

- See http://www.cplusplus.com/reference/, it is a good reference for C++ built in functions and classes
- Use length(), 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 until the end to merge with your 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.
- The OSM XML file format is described https://wiki.openstreetmap.org/wiki/OSM_XML. Though not necessarily important for this project, the tag features are described are https://wiki.openstreetmap.org/wiki/Map_features.