Project 4

Due November 20, 2019 at 11:59 PM
This project specification is subject to change at any time for clarification. For this project you will be working with a partner. You will be developing C++ wrappers for two C libraries. The wrappers will allow for reading and writing of both CSV and XML files. You have been provided headers for the CSV and XML wrapper types. You must keep the public interface but may develop the private implementation however you wish. The two libraries that you will be wrapping are libcsv (for CSV) and libexpat (for XML). Fortunately, libexpat is installed on the CSIF, but libcsv is not. The source code for libcsv has been provided for you and you will need to build it in order to include it in your project. After you have created your wrapper classes, you will be developing a program that will convert files between CSV and XML. The files that your program will need to convert are course grade files described later in this project. Make sure to add the temporary files created for your projects `(*.o, *.lo, *.la`, etc.) to your `.gitignore` file.

Create a Makefile to configure, make, and install libcsv. Open source libraries for Linux typically follow the same pattern for being built. The typical workflow is to configure, make, and then install (you will be “installing” into your project directory). Configuration is done by changing into the library source directory and executing the command:
```
./configure --prefix=INSTALLPATH
```
The `INSTALLPATH` is the prefix of where you would like to install the headers and the libraries. On a typical system this is in either `/usr`, `/usr/local`, `/opt`, or `/opt/local`; however, you will be just “installing” into your local project which should be the directory containing `libcsv-3.0.3` directory. You can get the absolute path of the Makefile with the command `$(shell pwd)`. There should be a lot of output where the configuration is figured out. The building of the project should be done next with executing the command:
```
make
```
This should also output a lot of lines but should complete successfully. Next the library should be installed by executing the command:
```
make install
```
This should install the libraries into the `lib` directory in the `INSTALLPATH`, and move the header file into the `include` directory of the `INSTALLPATH`. After you build the library correctly there should be a `libcsv.a` file (this is the static library you will link against) in your `lib` directory, and a `csv.h` in in your `include` directory.

Once the library is built, add to your Makefile to build your CSVReader and CSVWriter object files and to build and run your google test for the CSV. The `CCSVReader` and `CCSVWriter` classes have simple interfaces.

```
// Constructor for CSV reader, CCSVReader should have a
// std::istream & as part of it
CCSVReader(std::istream &in);

// Destructor for CSV reader
```

```
~CCSVReader();

// Returns true if all rows have been read from the CSV
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 CSV writer, CCSVWriter should have a
// std::ostream & as part of it
CCSVWriter(std::ostream &ou);

// Destructor for CSV writer
~CCSVWriter();

// 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);
```

Once your CSV wrappers are complete, add to your Makefile to build your XMLReader and XMLWriter object files and to build and run your google test for the XML. The `CXMLReader` and `CXMLWriter` classes have simple interfaces. The `SXMLEntity` struct is used by both classes and has been provided. You will want to build programs with your wrappers by linking in libexpat with `-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, CXMLReader should have a
// std::istream & as part of it
CXMLReader(std::istream &is);
```

```
// 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, CXMLWriter should have a
// std::ostream & as part of it
CXMLWriter(std::ostream &ou);

// 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);
```

Once you have completed writing the XML wrapper classes, you will be ready to begin developing the baby name analyzer. Your program will take a command line argument of a manifest file in XML format. The manifest will have a single `NAMEFILES` element that will one or more `NAMEFILE` subelements. Each `NAMEFILE` element will have the following attributes:
`FILENAME` – The relative path to the CSV data file, if only the final filename is provided, the CSV file is assumed to be in the same directory as the manifest XML file
`COUNTRY` – The country with which the CSV data file is associated
`YEAR` – The year with which the CSV data file is associated

The CSV files will have a header specifying the columns. The CSV files will have the following headings:
`NAME` – The given name of the baby
`SEX` – The binary sex assigned at birth for the birth certificate (either M or F)
`COUNT` – The number of babies of the assigned sex with the associated name born that year in that country

The name analyzer will need to load the data specified by the command line manifest argument. Once the data has been loaded, it must prompt the user for a name. If the name is present in any of the data for a particular country it must output the most likely sex assigned at birth, and provide a probability based upon the ratio of M to F. The analyzer will also output a year that is the middle year of the highest average occurrences of the name.

An example program can be found in `/home/cjnitta/ecs34/babyname`, and data files for analysis can be found in `/home/cjnitta/ecs34/proj4data/`.

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 proj4given.tgz
```

You **must** submit the source file(s), the Makefile, .git files, 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. Make sure to include your `.git` directory.

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.
- You may find the following line helpful for debugging your code:
  `std::cout<<"In "<<__FILE__<<" @ "<<__LINE__<<std::endl;`
  It will output the line string `"In F @ line: X"` where F is the name of the file 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 be helpful in understanding 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.
- 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.
- Use `std::stringstream` to test your reader and writer wrappers for CSV and XML.
- You may find some useful CSV tests in the `test_csv.c` file that you can implement in your google tests.

- Look at the `test_csv.c` for examples of using the libcsv library functions.
- 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 each library.