Assignment 1 (Map file save/load)

Course: COMP345 (Sec D, Fall 2015)
Author: Stewart Adam
Student ID: 21710176

1. Build & usage instructions

The project is built using the CMake build system, and can be compiled on a number of platforms. CMake can output the necessary build files for OS X (Xcode project or Makefiles), Windows (VS project or MinGW Makefiles) and Linux (Makefiles).

My assignment follows best build practices and **requires CMake 3.0 or higher**, and uses an out-of-source build:

```
$ cd build/<platform>
$ cmake ../../
```

1.1. Building on Concordia lab computers

I have tested building my assignment using the ENCS lab computers booted with **Scientific Linux 7**. SL7 comes with CMake 2.8 pre-installed, so to install 3.0 or higher you may choose to download & compile CMake from its website here: https://cmake.org/download

However, for your convenience I have made available this pre-compiled binary tarball (link will be valid until 2015-12-01): https://www.dropbox.com/s/1465d4j7wmzdw7r/cmake-3.3.2-Linux-x86_64.tar.gz

Below is an example, from start to finish, of how to download CMake 3.3.2 and build my assignment on lab computers:

```
$ cd /path/to/Risk-a1-21710176
$ wget https://www.dropbox.com/s/1465d4j7wmzdw7r/cmake-3.3.2-Linux-x86_64.tar.gz
$ tar xfz cmake-3.3.2-Linux-x86_64.tar.gz
$ cd build/linux
$ ../../cmake-3.3.2-Linux-x86_64/bin/cmake ../../
```

1.2 Usage instructions

The build will output a binary named `risk-a1-cli` in the `cli` subfolder where the build is started. The program provides minimal instructions if not called correctly, but in short simply call the program with a map file and the driver will read the map file, parse it, create a map, validate the continents and countries (per the assignment spec), and then output a new map file at the same filename with `.new` appended. Since I have included some sample `.map` files from http://windowsgames.co.uk in the `maps` folder, it can easily be tested after building (per above):

```
$ cli/risk-a1-cli ../../maps/Canada/Canada.map
```

1.3. Troubleshooting CMake errors

If you encounter a CMake error, you may need to flush out the cached CMake build files before attempting to reconfigure:

```
$ rm -rf CMakeCache.txt CMakeFiles
```

1.4. Contact information

If you encounter any other issues please feel free to reach out to me at s.adam@diffingo.com, and I will be happy to demo my application or troubleshoot building the assignments from the lab computers.

2. Design Documentation

My assignment involved the creation of a class that would save and load a map. I implemented a Map class with a static `load(std::string path)` method that would read the file specified and return a pointer to a new, fully-populated Map object using the map data from the file. In addition, I implemented a static `save(std::string path, Map*)` method that would serialize the passed Map object's data into a file.

I chose to implement these methods as static because I felt that they were separate from the other operations the Map class would have to do (such as returning a list of all loaded countries or continents). These function were strictly related to input I/O and as such I felt it would be more suiting for them to be static and stateless.

As well, in order to load continents and countries, I needed to implement these classes. They are stub-like classes with only the few properties, getters and setters required to complete the assignment.

2.1. File format

I chose to use the reference file format specified by the professor (and available at http://www.windowsgames.co.uk/conquest_maps.html) as this would give me a large pool of sample files I could test my `load()` implementation against.

2.2. Object allocation

The Map class must keep track of the continents and countries it contains, and to do so it instantiates all of the countries and continents on the heap, and stores their respective pointers in two `std::map` instance variables that map the string names of the continents and countries (respectively) to pointers to the actual instances. This allows for easy lookup of an object without having to redundantly store and maintain multiple copies of its data.

It also has the benefit that because only pointers are used, it allows for cyclical dependencies: a country knows what continent it belongs to (by storing a pointer to the Continent instance), continents know what countries they are composed of and each country knows which countries neighbor it. Because pointers are used for all of these relationships, cyclical dependencies do not cause runaway memory usage or invoke the copy constructor several times. Updating a single object *once* updates it everywhere it is referenced.

This introduced a new problem though: the Map class now needs to be responsible for the destruction of the continents and country objects it allocated on the heap during `load()`. Thus, in the destructor for the Map class I first iterate through the two `std::maps` in order to properly clean out all of the heap-allocated objects.

The caller of `load()` is responsible for deleting the Map object returned to it by `load()`. In doing so, all memory allocated by `load()` will be cleaned up.

2.3 Validation

I did not use a graph ADT to store the country-to-country border relationships, however I can still perform the necessary fully connected graph validations by performing a depth-first-search algorithm:

1. I initialize an `std::map` that maps a country pointer to a boolean
2. I recursively visit each neighboring country:
    a. marking it as visited if it has not yet been visited, or;
    b. immediately returning if I have visited this country before
3. I then iterate over the full set of countries, and verify that it was visited using in my `std::map`.

The process for validating a fully connected continent is similar, except for that:

1. I add verification before step 2a. If the neighboring country is not in the same continent, it immediately returns.
2. The “full set of countries” is instead replaced by the “set of countries in the continent”.

The third validation requirement, that a continent only belong to one continent, is an inherent part of my design: duplicated countries are ignored and not added to the maps during `load()`. If further example of this is required, look at the Country class, whose `continent` instance variable defines the relationship to Continent as 1:1.