Concordia University
Department of Computer Science
and Software Engineering

Advanced program design with C++
COMP 345 --- Fall 2019

Team project assignment #1

Deadline:	October 12th, 2019
Evaluation:	8% of final mark
Late submission:	not accepted
Teams:	this is a team assignment

Problem statement

This is a team assignment. It is divided into five distinct parts. Each part is about the development of a part of the topic presented as the team project. Even though it is about the development of a part of your team project, each assignment is to be developed/presented/tested separately. The description of each part describes what are the features that the part should implement, and what you should demonstrate. Note that the following descriptions describe the baseline of the assignment, and are related to the project description. See the course web page for a full description of the team project, as well as links to the details of the game rules to be implemented.

Design Requirements

1. All data members of all classes must be of pointer type.
2. All file names and the content of the files must be according to what is given in the description below.

Part 1: Map

Implement a group of C++ classes that implement the structure and operation of a map for the game of Risk. The map must be implemented as a connected graph, where each node represents a country. Edges between nodes represent adjacency between countries. Each country can have any number of adjacent countries. Continents must also be connected subgraphs, where each country belongs to one and only one continent. Each country is owned by a player and contain a number of armies. The map class can be used to represent any map configuration (i.e. not only the "Risk" map). All the classes/functions that you implement for this component must all reside in a single **`.cpp/.h`** file duo named **`Map.cpp/Map.h`**. You must deliver a file named **`MapDriver.cpp`** file that contains a main function that creates a map and demonstrates that the map class implements the following verifications: 1) the map is a connected graph, 2) continents are connected subgraphs and 3) each country belongs to one and only one continent. The driver must provide test cases for various valid/invalid maps.

Map implemented as a connected graph. Node represents a country. Edges between nodes represent adjacency between countries.
Continents are connected subgraphs. Each country belongs to one and only one continent.
Country is owned by a player and contain a number of armies.
Map class can be used to represent any map configuration.
Driver creates a map and demonstrates that the map class implements the following verifications: 1) the map is a connected graph, 2) continents are connected subgraphs and 3) each country belongs to one and only one continent. The driver must provide test cases for various valid/invalid maps.

Part 2: Map loader

Implement a group of C++ classes that reads and loads a map file in the `.map` text file format as found in the "Domination" game source files, available at: http://domination.sourceforge.net/getmaps.shtml, The map loader must be able to read any such map. The map loader should store the map as a graph data structure (see Part 1). The map loader should be able to read any text file (even ones that do not constitute a valid map). All the classes/functions that you implement for this component must all reside in a single **`.cpp/.h`** file duo named **`MapLoader.cpp/MapLoader.h`**. You must deliver a file named **`MapLoaderDriver.cpp`** file that contains a main function that reads various files and successfully creates a map object for valid map files, and rejects invalid map files of different kinds.

Map loader can read any Conquest map file.
Map loader creates a map object as a graph data structure (see Part 1).
Map loader should be able to read any text file (even invalid ones).
Driver reads many different map files, creates a graph object for the valid ones and reject the invalid ones.

Part 3: Dice

Implement a group of C++ classes that implements a dice rolling facility to be eventually used during the attack phase. The dice rolling facility should enable the player object to decide how many 6-sided dice are being rolled (from 1 to 3 dice), and then return the values in a sorted container. Each player will have his own dice rolling object, and each dice rolling object must keep track of the percentage of each value that has been rolled up to now for this player in the game. All the classes/functions that you implement for this component must all reside in a single **`.cpp/.h`** file duo named **`Dice.cpp/Dice.h`**. You must deliver a file named **`DiceDriver.cpp`** file that creates at least 2 dice rolling facility objects, tests that 1) one can request from 1 to 3 dice being rolled, 2) that the container returns the right number of values, 3) that no value outside of the 1-6 range is ever returned, 4) that there is an equal share of 1-6 values returned and 5) that every dice rolling facility object maintains a percentage of all individual values rolled up to now.

Enable the player object to decide how many 6-sided dice are being rolled (from 1 to 3 dice).
Put the rolled values in a sorted container.
Each player has a separate dice rolling object.
Each dice rolling object keeps track of each value that has been rolled up to now for this player in the game.
Driver that test: 1) one can request from 1 to 3 dice being rolled, 2) that the container returns the right number of values in a sorted container, 3) that no value outside of the 1-6 range is ever returned, 4) that there is an equal share of 1-6 values returned and 5) that every dice rolling facility object maintains a percentage of all individual values rolled up to now.

Part 4: Player

Implement a group of C++ classes that implement a Risk player using the following design: A player owns a collection of countries (see Part 1). A player owns a hand of Risk cards (see Part 5). A player has his own dice rolling facility object (see Part 3). A player must implement the following methods, which are eventually going to get called by the game driver: `reinforce()`, `attack()`, `fortify()`. All the classes/functions that you implement for this component must all reside in a single **`.cpp/.h`** file duo named **`Player.cpp/Player.h`**. You must deliver a file named **`PlayerDriver.cpp`** file that creates player objects and demonstrates that the player objects indeed have the above-mentioned features.

Player owns a collection of countries (see Part 1)
Player owns a hand of Risk cards (see Part 5)
Player has his own dice rolling facility object (see Part 3)
Player must implement `reinforce()`, `attack()`, `fortify()`.
Driver creates players and demonstrates that the above features are available.

Part 5: Cards deck/hand

Implement a group of C++ classes that implements a deck and hand of Risk cards. Note that the cards should not be read from the cards file from the Domination game implementation. The deck object is composed of as many cards as there are countries in the map. Each card has a type from: infantry, artillery, cavalry. The deck must have a `draw()` method that allows a player to draw a card at random from the cards remaining in the deck and place it in their hand of cards. The hand object is a collection of cards that has an `exchange()` method that allows the player to exchange cards in return for a certain number of armies (see the Risk game rules for details). All the classes/functions that you implement for this component must all reside in a single **`.cpp/.h`** file duo named **`Cards.cpp/Cards.h`**. You must deliver a file named **`CardsDriver.cpp`** file that creates a deck of Risk cards and demonstrates that it is composed of an equal share of infantry, artillery and cavalry cards by drawing all the cards and counting each sort drawn. The driver must also create a hand object that is filled by drawing cards from the deck, and that return the right number of armies when the `exchange()` method is called, depending on what cards are in the hand.

Deck object is composed of as many cards as there are countries in the map.
Each card has a type from: infantry, artillery, cavalry.
Deck has a `draw()` method that allows a player to draw a card at random from the cards remaining in the deck and place it in his hand.
Hand object is a collection of cards that has an `exchange()` method that allows the player to exchange cards in return for armies.
Driver creates a deck of cards with an equal share of infantry, artillery and cavalry. Creates a hand object that is filled by drawing cards from the deck, and that return the right number of armies when the `exchange()` method is called, depending on what cards are in the hand.

Assignment submission requirements and procedure

You are expected to submit a group of C++ files implementing a solution to all the problems stated above (Part 1, 2, 3, 4, 5). Your code must include a *driver* (i.e. a `main` function) for each part that allows the marker to observe the execution of each part during the lab demonstration. Each driver should simply create the components described above and demonstrate that they behave as mentioned above.

You have to submit your assignment before midnight on the due date using the ENCS Electronic Assignment Submission system under the category "programming assignment 1". Late assignments are not accepted. The file submitted must be a .zip file containing all your C++ code. Do not submit other files such as the project file from your IDE. You are allowed to use any C++ programming environment as long as you can demonstrate your assignment in the labs.

Evaluation Criteria

Knowledge/correctness of game rules: *Mark deductions: during the presentation or code review it is found that the implementation does not follow the rules of the game of Risk.*	**2 pts (indicator 4.1)**
Compliance of solution with stated problem (see description above): *Mark deductions: during the presentation or code review, it is found that the code does not do some of which is asked in the above description.*	**12 pts (indicator 4.4)**
Modularity/simplicity/clarity of the solution: *Mark deductions: some of the data members are not of pointer type, or the above indications are not followed regarding the files needed for each part.*	**2 pts (indicator 4.3)**
Mastery of language/tools/libraries: *Mark deductions: the program crashes during the presentation and the presenter is not able to right away correctly explain why.*	**2 pts (indicator 5.1)**
Code readability: naming conventions, clarity of code, use of comments: *Mark deductions: some names are meaningless, code is hard to understand, comments are absent, presence of commented-out code.*	**2 pts (indicator 7.3)**
Total	**20 pts (indicator 6.4)**