University of Newcastle
Discipline of Computing and Information Technology
Semester 2, 2020 - SENG1120

Assignment 2

Due using the Blackboard Assignment submission facility:

11:59PM – Wednesday, 21st October 2020

NOTE: *The important information about submission and code specifics at the end of this assignment specification.*

INTRODUCTION

In lectures we have discussed the use of `template`s to provide the compiler with blueprints for functions and classes. These are used by the compiler to produce code that implements functions and/or classes that are suitable for the type(s) to which you apply them in your program code.

ASSIGNMENT TASK

Your task in this assignment is to re-use some classes from Assignment 1 and implement a simplified version of a **Blackjack** game. Here are the basic steps in the implementation of Assignment 2:

1. Implement a class `Card`, which will represent instances of playing cards. `Card` will have three private member variables: `string face`, `int value` and `bool faceUp`. `face` will store the string, e.g. "10-H" (for 10 of hearts); `value` will store the value of the card in points, e.g. 10; and `faceUp` will indicate if the card is placed facing up or down (this is important for the Blackjack game itself. These variables will need getters and setters. `Card` will also need an overloaded `cout <<` operator that outputs the value of the string.

2. In addition, you will produce a class template for each of:
   - `Node`, which stores `Card`s and is used as a component in the construction of
   - `LinkedList`

3. Implement the class template `Queue`, which uses LinkedList.

4. Implement the class `DeckOfCards`, which is a common class (i.e. not a template) that uses a `Queue` instance as member variable. The front of the queue will be the top of the deck. The back of the queue will be the bottom of the deck. `DeckOfCards` will store all cards of the game, initially. In the constructor of `DeckOfCards`, you need to use `enqueue()` to populate the underlying queue first. `DeckOfCards` will also have a function `shuffle()`, implemented as follows:
   - Separate the 52 cards in the deck into 4 groups of 12. The order does not matter, and you can use 4 instances of `DecksOfCards` for that.

- Then, choose one deck at random, pick the top card, and place it at the bottom of one of the other decks. Repeat 1000 times and merge the 4 decks back to form the main one.
- To use the C++ random number generator, please have a look at:
  - http://www.cplusplus.com/reference/cstdlib/rand/

5. Implement the class `HandOfCards`, which is also a common class, and uses a `Queue` as the underlying data structure. `HandOfCards` will have the following functionality:

   - Constructor, which creates an empty `HandOfCards`.
   - `int count()` counts the value of the hand stored in the list (but only the cards facing up). Each card 2-10 have their face value. J, Q and K are worth 10 points each. A is worth 11 points (*this is a simplification of the traditional Blackjack rules*).
   - `int countAll()` counts the value of the hand stored in the list (all cards facing both up and down).
   - `string value()` returns a string displaying the sequence of cards stored in `HandOfCards`. Face-down cards must be displayed with "?-?" symbol.
   - `void faceUp()` makes all cards in the hand face-up.
   - Overloaded output operator (i.e. '`<<`') outputs the content of the `HandOfCards` in a form suitable for printing.
   - `void add(Card, boolean)` takes a single `Card` and a `boolean` arguments and adds the card in `HandOfCards`, either facing-up (`true`) or down (`false`).

As usual, we provide the demo file, which will:

1. Create a new instance of `DeckOfCards` storing a full deck of cards (to be implemented in the constructor). Then it will shuffle its contents.
2. Create two instances of `HandOfCards`, named `player` and `dealer`.
3. Loop through the player and the dealer, giving them one card each, facing up. Cards are taken from the top of the deck of cards, using the function `draw()`.
4. Gives a second card to the player, facing up, and a second card to the dealer, facing down.
5. Display each hand's content - for player and the dealer – one hand per line, followed by the number of points.
6. Check if someone has won the game.
7. Ask if the player wants to hit or stand. If "hit", the player receives another card from the deck, face-up, and goes back to step (5). If the answer is stand, go to step (8).
8. At this point, the player has "stood" in the last loop. Now it is the time to do the dealer's play. If the count of the dealer's hand is <=16, give it a new card, face-up. The dealer must take cards until the hand is 17 or more.
9. Make all cards in the dealer's hand face-up and display the contents of both hands along with the count for each hand.
10. Check who has the highest count among the two hands (but not over 21) and state the winner.

For more information about the Blackjack game itself, please have a look at:

https://en.wikipedia.org/wiki/Blackjack

A typical output of the game is presented below:

```
>>./BlackjackDemo.exe
Player: 4-C 6-D (10 points)
Dealer: 8-C ?-? (8 points)

Player, do you want to Hit (1) or Stand (2)?
1

Player: 4-C 6-D 5-C (15 points)
Dealer: 8-C ?-? (8 points)

Player, do you want to Hit (1) or Stand (2)?
1

Player: 4-C 6-D 5-C 5-S (20 points)
Dealer: 8-C ?-? (8 points)

Player, do you want to Hit (1) or Stand (2)?
2

Player: 4-C 6-D 5-C 5-S (20 points)
Dealer: 8-C 2-H 4-D 5-H (19 points)

THE PLAYER WON!
```

Now the dealer's strategy is executed and the final score is 19.

SUBMISSION

Make sure your code works with the files supplied, and DO NOT change them. For marking, we will add the `BlackJackDemo` file and compile everything using the `makefile`, together with your own files. If it does not compile or run, your mark will be **zero**.

Your submission should be made using the Assignments section of the course Blackboard site. **Incorrectly submitted assignments will not be marked. Assignments that do not use the specified class names will not be further marked.** You should provide all your `.h`/`.cpp`, and `.hpp` files. Also, if necessary, provide a `readme.txt` file containing any instructions for the marker. Each program file should have a proper header section including your name and student number, and your code should be properly documented.

Remember that your code should compile and run correctly using Cygwin. There should be no segmentation faults or memory leaks during or after the execution of the program.

Compress all your files into a *single .zip file*, using your student number as the filename, for example, if your student number is **c9876543** you would name your submission:

```
c9876543.zip
```

Submit by clicking in the link that will be created in the Assignments section on Blackboard.

Late submissions are subject to the rules specified in the Course Outline. Finally, a completed Assignment Cover Sheet should accompany your submission.

This assignment is worth 10 marks of your final result for the course.

Alex and Dan – v1.01 (2020-09-22)