Lab 5 - Using the STL

Due Date: 5:00 p.m., March 16, 2015
All function interfaces are suggested naming and parameter guidelines. If you feel there is a better way, you are free to alter names, functions interfaces, etc, as long as you follow the lab and style guidelines. Output should match exactly unless otherwise stated.

The goal of Lab 5 is to get you using STL classes in your code.

(Parts A must be completed in lab)

Part A: Creating a Card Class

- The following classes must be developed:
    - Card with the following attributes and behaviors:
        - The attributes of a card are value and suit. Both must be private attributes.
            - You should only have a value constructor. No default constructor.
            - You should have a method to get the value of the Card that returns the numerical value of the card which ranges from 1-14
                - Ace is considered high or low (1 or 14). For now, you can pick one, but I would recommend designing your Card class with this in mind for the future.
            - You should have another method that returns the suit of the card which ranges from 1- 4
            - Your should create a map<int, string> to hold the int to Suit name conversions, for returning a string when printing out the card suit.
                - This is not a requirement (yet), but the map should be static since all Cards will share the same map<int, string>
                - To insert values into the map you should use insert, which requires a STL pair<> parameter, not emplace(). emplace() does not work on all machines.
            - Other operations may be added to this class as determined by your design

Show your TA your code, and that you can compile your code with a makefile.
--END OF IN LAB REQUIRED WORK--

Part B: Creating a Deck of Cards

- Once your Card class is working, you are going to create a Deck for the cards. The constructor for the Deck class should create 52 cards in a loop, and store them in the STL deque internally.
    - http://www.cplusplus.com/reference/deque/deque/
- Your Deck class should have the following methods
    - shuffle - shuffles the deck of cards.
        - You can use the STL shuffle algorithm to make things easy inside of your shuffle method
        - http://www.cplusplus.com/reference/algorithm/random_shuffle/
    - draw
        - Returns and removes the Card object from the Deck

Part C : Driver Code

- Write driver code that:
    - Creates a Deck
    - Removes and prints all cards in the Deck
    - Creates a second Deck, then shuffles it.
    - Removes and prints all cards in the Deck.
- Run your code with valgrind to ensure you do not have any memory leaks.

Part D : Code Organization and Submission

- Required code organization:
    - lab5.cpp
    - Deck.cpp/.h
    - Card.cpp/.h
    - makefile
        - executable should be called: lab5
            - *do not add a .exe extension*
    - readme
- Create a readme file following the format provided, and add it to your project folder
- While inside your lab 5 folder, create a zip archive with the following command
    - zip -r lab5 *
        - This creates an archive of all file and folders in the current directory called lab5.zip
        - **Do not zip the folder itself, only the files required for the lab**
- Upload the archive to Mimir under ‘Lab 5’

Expected Interface and Test Output

- **Test Commands**
  - ./lab5
    - *Expected Test Output*
      - *Does not have to match exactly because the shuffle will be random*
      - output.txt

Grading Guidelines

- **Part A (3 points)**
  - Card class works as expected and uses a map<int, string> to hash the suits
- **Part B (4 points)**
  - Deck class works as expected and uses an stl deque to hold the cards
- **Part C (2 points)**
  - Driver code prints all cards in order and shuffled
- **Part C (1 point)**
  - Follows formatting guidelines, requested project structure and naming conventions, contains written Readme, and submission does not include .o files or binary

Formatting Guidelines

- Stores all values in a named variable.
  - No Magic Numbers.
- Uses indentation to identify code blocks.
  - Every Code block should be indented from it's parent block to identify scope.
- No single letter or non-descriptive variable names
  - The only exception to this rule is 'i' in a for loop
- Separates code blocks and logical sections with whitespace
  - Optimize your code for the reader, not the writer
- Output is formatted with an explanation of the output values
  - Format your output so that someone who does not know what the program is supposed to do would know what the output meant
- Each method is preceded by a comment explaining what the method does
- Each significant code block is preceded by a comment explaining what the code block does.
  - A significant code block is more than 3 lines performing a single logical operation

- CONSTANTS are in all caps
- Only data types start with a capital letter
    - Classes, Enums, Structs, etc.
- Do not use the ‘using namespace’ declaration in a header (.h) file
- In general we will follow the Google C++ style guidelines. If you want more info, you can view them here: https://google.github.io/styleguide/cppguide.html