Program 2 - Highcard Tournament

Due Date: 5:00 p.m., April 20, 2016
All public interfaces are minimum requirements. You may add additional public methods if needed as long as you follow the lab and style guidelines.

Welcome to the high stakes world of backroom high card, where the best of the best walk away with it all, and only the richest players get to play. For this assignment, you are to implement a High Card tournament by putting all players into a max-heap sorted by their budgets. The losers of each round are placed back into the heap, and the 4 richest players from the heap are added back to the game. This may be the same players or new players who have deeper pockets.

Part A: Setting up the Tournament

For Part A you will read in a text document containing players and their budget (from 1-200). As you read in the players, you must store them in a binary max-heap that you write. The heap is prioritized by the player's current budget and must be an array based heap. You may not use the STL library for the heap itself, but you can use a vector (or other STL container) for the internal array.

Below is the the class interface for the players and the heap.

- class Player
    - Public Instance Variables
        - Card hand;
    - Public Methods
        - Player(String name, int budget)
        - string getName()
        - int getBudget()
        - bool bet(int amount)
            - Removes a specified amount of money from the player's budget
        - void collectWinnings(int amount)
            - Adds a specified amount of money to the player's budget
- class Heap
    - Public Methods
        - Heap(string filename)
            - Takes a filename and reads the player's name and budget
                - (see file linked below for format)
        - Heap(const Heap &copy)
        - void addPlayer(Player newPlayer)
        - Player getPlayer()

        - Removes the priority player (the player with the highest budget) from the heap.
        - Remember that after removing the player, your heap still needs to be a heap
    - vector<Player> getArray()
    - bool empty()
    - unsigned int size()

*****Must pass through Test #3 in the test driver by next Thursday lab (4/14).*****

Part B: Making Your High Card game

The Game's Idea

In the game of high card the following actions occur.

1. The 52 cards in the deck are shuffled (randomized).
2. Each player ante's (puts in a minimum bet)
3. Then each player is given one card from the top of the deck.
4. Now each player's card is evaluated and the winner is determined.
5. The losers are placed back into the heap, and the next group of players sit at the table.

Below is the class you will need to run the game:

- Class Table
    - Public Methods
        - Table(int num_seats, int ante)
        - bool emptySeat()
        - vector<Player> playRound()
            - Returns a vector of the losers of the round
            - The winner stays for the next round while the losers leave
            - The players in this vector will be put back into the heap
        - void addPlayer(Player p);
        - void printWinner()

Once you have you Players and Table set up and are passing the first 5 tests in the driver code, you should then start on the actual high card game itself. You should use your Deck class and Card class from your previous lab to implement the highcard game.
You may need to change or add additional methods to those classes (verify that shuffle is actually random).

Part C: Testing your code

Once you are passing all of my tests, you must come up with two more tests for your code on your own. State explicitly in a comment preceding the test:

- what you are testing
- what is the expected outcome.

Whether you use assert assert() or some other mechanism, the tests must end your program abnormally if it fails.

Part D : Code Organization and Submission

- Required code organization:
  - program2.cpp
  - Player.h/.cpp
  - Table.h/.cpp
  - Heap.h/.cpp
  - Deck.h/.cpp
  - Card.h/.cpp
  - Makefile
    - **Your makefile must include c++11 extensions**
  - readme
- Create a readme file following the format provided, and add it to your project folder
- While inside your program2 folder, create a zip archive with the following command
  - zip -r program2 *
    - This creates an archive of all file and folders in the current directory called program2.zip
    - **Do not zip the folder itself, only the files required for the lab**
- Upload the archive to Mimir under ‘Program 2’

Tests

- **Test Driver**
  - Program2.cpp
- **Test players files**
  - players.txt
  - players2.txt

Grading Guidelines

- **Part A**
  - Passes tests 1-3
    - 2 points for each test
- **Part B**
  - Passes tests 4-7
    - 3 points for each test
- **Part C:**
  - Passes 2 additional tests written by the student
    - 1 point for each test
- **Part D:**
  - Follows formatting guidelines, requested project structure and naming conventions, and submission does not include .o files or binary (1 point)

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