ITP 365: Managing Data in C++

Homework 5: Vector and Poker Hands Due: Friday, 11/3/2017 @ 11:59PM

Goal

In this homework, you will implement a simplified version of the vector collection. You will then use this version of vector to generate a random hand of five cards and determine the poker hand.

Lab Setup

- For this homework, you will be editing **main.cpp**, **ITPVector.hpp**, and **Cards.h/cpp**.
- At the top of the above files, make sure the comments have the following (using your name, email, and specify whether you did your homework on Windows or Mac):

```
// ITP 365 Fall 2017
// HW5 – Vector and Poker Hands
// Name: Tommy Trojan
// Email: ttrojan@usc.edu
// Platform: Windows
```

- **For this homework, do not modify the function names, return types, or parameters of any of the functions. We will be testing your code using automated testing programs, so changing these will result in an automatic deduction.**

Part 1: Main Menu

- In **main.cpp**, create a text-based menu that looks like this:

```
Select an option...
1. Test ITPVector
2. Poker Hands
>
```

- Each option should invoke a separate function in **main.cpp**. These functions will demonstrate the code you write in the later parts.
- You can assume that the only user input provided will be `1` or `2`. We will not provide invalid input to your menu.

Part 2: ITPVector class

- In this part, you will implement the templated `ITPVector` class in **ITPVector.hpp**. You will find lecture notes extremely helpful for this part.
- **Note that although the book (and many other online references) list implementations for a vector class, do not just directly copy such code, or there will be a severe penalty up to and including being reported to SJACS for academic dishonesty.**
- As you implement the functions in `ITPVector`, at the same time you should add code to test your implementations in **main.cpp**. This way, you can make sure that your functions work as you expect before moving onto the next function. This testing code should run if the user selections option 1 in the main menu.
- If you look at the existing code in **ITPVector.hpp**, you will notice that several of the functions currently return values, but these are incorrect values as these values were added simply to make sure the stubbed functions still compile. The return values that need to be fixed are commented.
- In any event, it is recommended you implement `ITPVector` in the following sequence:
    1. Add three member variables – two unsigned integers to track the capacity and size, respectively, and a pointer to a `T` that you will use to keep track of the underlying array data.
    2. Implement the capacity function that returns the capacity of the `ITPVector`
    3. Implement the size function that returns the size of the `ITPVector`
    4. Implement the `ITPVector` constructor. This constructor should initialize the capacity to `INITIAL_CAPACITY` and the size to `0`. It should then proceed to *dynamically allocate* the underlying array of `T`s to have capacity elements in it.
    5. Implement the ITPVector destructor. The destructor should delete the underlying array and set the capacity and size to 0.
    6. Implement the `insert_back` function. If the current size is less than the current capacity, this is relatively straightforward. However, if the current size is equal to the capacity (meaning the insertion will go over the capacity), you will need to create new underlying array data. Consult the lecture notes for a discussion of this.
    7. Implement the two subscript operators. Note that both implementations should check for an out-of-bounds index, in which case the `error` function should be called with an appropriate error message.
    8. Implement the << operator. This should loop through the vector from the start to end indices, and output a comma separated list of the elements in the vector.
    9. Implement the `get_back` function. This should return the last element in the `ITPVector`. Note that if the size of the vector is `0`, this function should call the `error` function with an appropriate message.
    10. Implement the `remove_back` function. This removes the last element in the `ITPVector`. Note that the size of the vector is `0`, this function should call the `error` function with an appropriate message.
- The output for your testing code can be whatever you want, so long as it demonstrates that your functions work. **Most importantly, you must demonstrate that your insert_back function works properly when the size increases beyond the current capacity**. The sample below is one approach to proving this (user input in **red**):

```
***Constructing an ITPVector...
Contents: {}
Capacity: 10, Size: 0
Enter a number of elements to add:5
Contents: {0,5,10,15,20}
Capacity: 10, Size: 5
The back element is: 20
Called remove_back
Contents: {0,5,10,15}
Capacity: 10, Size: 4
***Test Complete!
***Constructing an ITPVector...
Contents: {}
Capacity: 10, Size: 0
Enter a number of elements to add:15
Contents: {0,5,10,15,20,25,30,35,40,45,50,55,60,65,70}
Capacity: 20, Size: 15
The back element is: 70
Called remove_back
Contents: {0,5,10,15,20,25,30,35,40,45,50,55,60,65}
Capacity: 20, Size: 14
***Test Complete!
***Constructing an ITPVector...
Contents: {}
Capacity: 10, Size: 0
Enter a number of elements to add:21
Contents: {0,5,10,15,20,25,30,35,40,45,50,55,60,65,70,75,80,85,90,95,100}
Capacity: 40, Size: 21
The back element is: 100
Called remove_back
Contents: {0,5,10,15,20,25,30,35,40,45,50,55,60,65,70,75,80,85,90,95}
Capacity: 40, Size: 20
***Test Complete!
```

Part 3: Poker Hands

- Now that you have a working vector class, let's use it to draw poker hands!
- If you are unfamiliar on poker hands, consult the Wikipedia article on the topic.
- If you take a look at **Cards.h/cpp**, you will see that some of the code has already been provided to you, including a `Suit` `enum` and a `Card` `struct` (that has both rank and suit for the card).
- Of the provided implementations, the most important is `createDeck`. This function will take an empty `ITPVector` of `Cards`, fill it with a standard 52 card deck, and then shuffle the deck.
- You will need to implement the following member functions of `PokerHand` (in **Cards.cpp**):
    1. The `PokerHand` constructor. This will take in a deck of cards, by reference, draw 5 cards and store them in the `mHand` member variable. To do this, you will use `get_back`, `insert_back`, and `remove_back`. The constructor also needs to sort the hand – in order to do that, simply uncomment the `std::sort` line that is provided for you.
    2. `hasStraight` – Returns `true` if the hand has a straight. Since they are sorted, you can check simply by making sure that the rank at index 0 is equal to one less than the rank at index 1, the rank at index 1 is one less than the rank at index 2, and so on.
    3. `hasFlush` – Returns `true` if the hand has a flush. To check if it's a flush, make sure the suit of every card is identical.
    4. `hasFourOfAKind` – Returns `true` if the hand has a four of a kind. Since it's sorted, there are only two possibilities: either the first four cards have the same rank, or the last four cards have the same rank
    5. `hasFullHouse` – Returns `true` if the hand has a full house. As with four of a kind, there are only two possibilities: x x x y y *or* x x y y y
    6. `hasThreeOfAKind` – Returns `true` if the hand has a three of a kind. There are three possibilities: x x x y z *or* x y y y z *or* x y z z z
    7. `hasTwoPairs` – Returns true if the hand has two pairs. There are three possibilities: x x y y z or x y y z z or x x y z z
    8. `hasPair` – Returns true if the hand has a pair. There's a pair if the rank of any card is equal to the rank of its neighbor
    9. `getBestPokerHand` – This returns a string describing which poker hand you have . Start by checking for the best hand (straight flush) using the appropriate member functions and continue down from there
- In **main.cpp**, add code that uses this functionality. First, it should create a deck, then create a `PokerHand`, and output both which cards you have and what poker hand it forms. Keep repeating this process until the user says to stop
- **Hint:** You can check your code for the better hands by creating a "stacked" deck of only the cards you want to draw.
- Output for this part should look something like this (user input in red):

```
You drew: {6 of Diamonds,10 of Clubs,10 of Hearts,10 of Spades,Ace of Diamonds}
You have a three of a kind
Try again (y/n):y
You drew: {3 of Hearts,6 of Clubs,10 of Spades,Jack of Diamonds,King of Clubs}
You have a high card
Try again (y/n):y
You drew: {6 of Diamonds,6 of Hearts,7 of Hearts,8 of Hearts,King of Diamonds}
You have a pair
```

Submission

1. You must submit a ZIP file to Blackboard called **HW5.zip**, using the designated Homework 5 submission link in the "Assignments" section
2. Make sure to double-check that your zip file was created properly prior to submission.

Grading

Item	Points
Part 1: Main Menu	5
Part 2: ITP Vector	
ITPVector::size/ITPVector::capacity	5
ITPVector constructor	5
ITPVector destructor	5
ITPVector::insert_back	15
ITPVector subscript operators	10
ITPVector::get_back/ITPVector::remove_back	10
ITPVector << operator	5
Part 3: Poker Hands	
PokerHand constructor	10
PokerHand::has* functions (3 points each)	21
PokerHand::getBestPokerHand	4
Comments, style, and correct use of separate files	5
Total	**100**