Blackjack

50 points - Due Friday, March 30, 11:59 PM

Blackjack Rules:

The rules of Blackjack are simple. The player and dealer compete, both trying to get a set of cards whose sum comes as close as possible to 21 (a “blackjack”) without exceeding it. The dealer shuffles the deck and then deals out two cards to the player.

The player then can repeatedly decide to either “hit” (take another card) or “stand” (end her turn by choosing to not take another card). Every time the player hits, the value of her new card is added to her total; if the total goes over 21, then the player has “busted”. If this happens, the hand is over and the dealer wins. Otherwise, if the player stands with a total card value that is less than 21, then it is the dealer’s turn.

The dealer has no choice when to hit or stand: she must follow a set of rules. If her cards total 16 or less, then the dealer must hit. If her cards total 17 or more, then the dealer must stand. If the dealer busts (goes over 21) then the player wins; otherwise, when the dealer stands, the dealer’s card total is compared with the player’s card total. The person with the higher card total wins; if the player and dealer tie, the dealer wins.

The values of the cards are as follows:

- Face cards (Jack, Queen, King) each count as **10 points**.
- An Ace counts as either **1 point** or **11 points**, whichever is better for the owner of the hand. For example: with a hand of Q, A the A would be 11, allowing for a total of 21; with a hand of 7, 8, A the Ace would be 1 for a total of 16, so the player doesn’t bust.
- All other cards are worth their numerical face value.

For more information on blackjack rules, Google ‘em. There’s about a million sites with way more detail than you probably wanted. Note that the rules above are *all* you have to implement: blackjack lets you double down, split, etc. but you don’t have to implement any of that.

Your Assignment:

For this project, you will implement a simple blackjack game - the player will be the user of your program, and the program will function as the dealer. When someone runs your program, the deck will be shuffled, he/she will be dealt two cards, and then will be repeatedly offered the chance to hit or stand until he/she has either busted or decided to stand. Each time a card is drawn, the player should be told what kind card it is, its value, and the new total value of the players cards.

If the player stands with less than 21, the dealer will do the same (according to the dealer rules outlined above) - the cards the dealer draws should be printed out for the player to see. Once the dealer is done, the results should be printed out: the dealer and player’s card totals, and who won the hand. The player should be offered a chance to either quit or play another hand.

Specific Notes:

- How you organize and implement this project is largely up to you. Seeing as how this is a C++ class and all, you should use some of the C++ features we've learned. You should use some simple C++ classes: I'd suggest a **Deck** class, for example, with a **shuffle** method, and maybe a **Card** class. However you choose to organize things, though, keep the good coding practices in mind: encapsulation, access controls, const, friend classes where needed, few or no globals, etc - all the good stuff we've learned. Part of your grade will be based on how well you utilize the tools that C++ gives you for this sort of thing.

- You'll need to generate random numbers for this project. Use the **rand()** function (defined in **<stdlib.h>**), which gives you a large random integer which you can mod to get the range you need. Random number generators need to be "seeded" (given a starting value) to prevent them from generating the same sequence of numbers over and over again: you can do this with the **srand** and **time** functions. A simple example:

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
using namespace std;

int main()
{
      // initialize random number generator... you only have to
      // do this *once*, at the beginning of your program!
      srand( time(NULL) );

      // print out a few random numbers between 0 and 99
      for( int i = 0; i < 10; ++i )
            cout << rand() % 99 << endl;

      return 0;
}
```

What to turn in: a directory containing...

- Your code, of course! Again, you get to choose how to organize this. If you can cram everything into a single source file and still have it all make sense, go for it. Otherwise, separate it out into multiple files using the techniques we discussed in class.

- A README file containing any relevant info, such as your name and compilation instructions (include the exact command you used to compile your code). Again, please include in the README the approximate amount of time you spent on this assignment.

Non-specific Notes

- Start early; ask questions.
- Refer back to the syllabus for hints about writing good C++ programs (or programs in any language): only work on small pieces of code at a time, and test them well before moving on. Compile lots, and fix the warnings and errors. Write lots of comments.
- **Be sure your code compiles on the CS department's Linux machines before turning it in**. Even if you write it elsewhere (on a different OS, a different IDE, etc), give it a quick compile and test on a lab machine before you turn it in. Remember, if your code does not compile, it will get (at most) 50% credit.
- You must submit your code using the department's electronic submit procedure. A link to instructions for doing this is on the class website. Our course number (for the purposes of the submit program) is **c109**.
- Just a reminder: the CLAS academic (dis)honesty policies will be enforced. You're allowed to discuss the assignment with others, but not to collaborate or share code. Nor are you allowed to find or use code off the Internet.