
CSC200

Fall 2005

Project 02

BlackJack

Due: November 7, 2005

1 Purpose

There are plenty of computer games out there. How do these work? Computer games are one of the largest and the fastest growing industries involving computer science. While most modern games have fancy graphics and user interfaces, it wasn't always this way. The first computer games created were text-based games. The purpose of this project is to create a simple two player (you vs. dealer) Blackjack game utilizing the standard input we have seen so far. Like the Cryptography project, this style of game is not often used much in practice anymore, but it should serve to give you an idea of the processes involved in modeling a game with computers.

2 Background

Blackjack is a card game involving at least one standard 52 card deck. The basic premise of the game is that you want to have a hand value that is closer to 21 than that of the dealer, without going over 21.

In our simplified two person game, the dealer shuffles the deck, then begins to deal. The deal consists of (in this order) a card face down to the player, a card face down to the dealer, then a card face up to the player, and finally a card face up to the dealer. The player can then **Hit** to draw another card, or **Stay** to keep the current hand and let gameplay continue. If the player decides to **Hit** then the player can keep on hitting until the the player decides to **Stay** or the total value of the hand is **Bust** or over 21. If the player goes **Bust** then the dealer automatically wins. Otherwise, after the player has decided to **Stay** the dealer then plays out his hand. (Dealer play is completely determined by rules.) The dealer shall always play by the rule that he must **Hit** until he has 17.

In blackjack, the cards are valued as follows: An Ace can count as either 1 or 11, as demonstrated below. The cards from 2 through 9 are valued as indicated. The 10, Jack, Queen, and King are all valued at 10.

The suits of the cards do not have any meaning in the game. The value of a hand is simply the sum of the point counts of each card in the hand. For example, a hand containing (5,7,9) has the value of 21. The Ace can be counted as either 1 or 11. You need not specify which value the Ace has. It's assumed to always have the value that makes the best hand. An example will illustrate: Suppose that you have the beginning hand (Ace, 6). This hand can be either 7 or 17. If you stop there, it will be 17. Let's assume that you draw another card to the hand and now have (Ace, 6, 3). Your total hand is now 20, counting the Ace as 11. Let's backtrack and assume that you had instead drawn a third card which was an 8. The hand is now (Ace, 6, 8) which totals 15. Notice that now the Ace must be counted as only 1 to avoid going over 21.

So with a little background in Blackjack, how can we think about implementing this in C++? To

start, an array of size 52 could be used to represent the deck of cards. Also, the C++ library with header file `<cstdlib>` contains *pseudo-random* number generator function named `rand`. This function has no arguments. When your program invokes `rand`, the function returns an integer in the range 0 to `RAND_MAX`, inclusive. `RAND_MAX` is a defined integer constant that is also defined in the the library with header file `<cstdlib>`. The exact value is system-dependent but will always be at least 32767. For our case, when we shuffle we want an integer between 0 and 51 (our valid index values for the deck array). To ensure that the value is within this range, we can use

```
rand() % 52;
```

What is meant by saying that invocations of `rand` return *pseudo-random* numbers? Well, there is no such thing as true randomness within computers. A computer can only do what it is told to do via an algorithm. Generating a truly *random* set of numbers would necessarily involve ambiguity, so *rand* only generates seemingly random numbers (hence *pseudo-random*) based on a *seed* value. One can set the seed by a call to a function which takes an integer named `srand`. If a program is run with the same seed again and again, then each time it will produce the same (random looking) sequence of numbers. For example the following will output two identical sequences of ten pseudo-random numbers:

```
int i;
srand(99);
for(i=0; i<10; i++)
cout << (rand() % 11) << endl;
srand(99);
for(i=0; i<10; i++)
cout << (rand() % 11) << endl;
```

The closest that we can get to true random number generation can then be to make sure that the program is run with a different seed value every time. There is a C++ library (with header file `<ctime>`) that allows access to the current time. Using a function from this library a very standard way to initialize a random seed value is to use the `time()` function as follows:

```
srand(time(0)); // Initialoize random number generator
```

If we use this at the beginning of the program, then a different seed will be used each time the program is run. The call to `time(0)` returns the current integer number of seconds from the system clock. So now we know how to use a random number generator to help us shuffle the deck of cards. (Hint: Use a fixed seed while working on the project to try to work out bugs, then add the `srand(time(0))` call at the end). We will talk a little bit more in class about what needs to be done to represent a deck of cards using an array and what needs to be done to properly shuffle the deck.

3 Requirements

For this project you will need to implement a program in C++ which implements a simple blackjack game as described above. The following must be present in your project:

- One source code file named `blackjack.cpp`
- Use `cin` (or `getline`) and `cout` to create a simple user interface to show the cards visible to the player, and to allow the player to choose *Hit* or *Stay* when possible.

- Game is dealt, played, and won as stated in Background section above. The dealer must play his hand by the rule stated above,
- At least one function:
 - `int card_value(int card)`
- Program uses an array of size 52 to represent the deck.
- Program makes a call to `srand(time(0))` to initialize the seed value.
- Program shuffles the deck in some manner using calls to `rand()`.
- Program uses a while loop to allow multiple games to be played (reshuffling each time).

The program output should look something like:

```
Dealer: 10H
Player: 9S 8D
Hit or Stay?(H or S)S

Dealer: 10H 8C
Dealer has 18, Player has 17,
Dealer Wins!!

Play Again?(Y or N)Y

Dealer: 2D
Player: 7S 4C
Hit or Stay?(H or S)H
Dealer: 2D
Player: 7S 4C 8H
Hit or Stay?(H or S)S

Dealer 2D 6S JD
Dealer has 18, Player has 19,
Player Wins!!!

Play Again?(Y or N)N
...
```

4 Grading

The following table shows the scoring that will be used for this project:

Area	Max Points
Compiles in Dev-C++	50
Report and Source Code Documentation	25
Correctness	25

There are three chances for extra credit on this project:

- Instead of one deck use six(6) decks shuffled together. (10 Points).
- Implement one of the following functions and use them correctly in the game:

- `void printCard(int card)` which takes a card (0-51) and prints the string value of the card (what you print). (5 Points)
- `string card_to_str(int card)` which takes a card (0-51) and returns the string value of the card (what you print). (Hint: one way to do this is to use a stringstream.) (10 points)
- `int winning_hand(int[] dealer, int[] player)` This one is tricky because we haven't gone over it, but two arrays (representing the hands of the dealer and the player) are passed in, then an integer is returned indicating which hand is the winning hand (0 - draw, 1 - dealer, 2 - player). Think of the game play and the order of cards here. Also, passing arrays to functions is a little different than passing regular values. (10 Points)