

CST8110: Introduction to Computing: Midterm Part 2: Winter 2008

Name:

Solution

This part of the midterm test will be adjusted to represent 55% of the blended midterm test mark (thus, Monday's will represent 45% of the blended mark).

Programming (20 marks)

Overview

Write a C++ program that plays a game of *Card War*. There will be two players, both managed by the program (in other words, not directly controlled by the user of the program).

Your program should behave much as is shown on this screen capture (though it doesn't have to look exactly the same).

Details: The *struct* Specifications

Create a *struct* to specify a *Card*. It will consist of:

- face value: 1 through 13 to correspond to ace¹ through king.
- suit: one of four suits² – used to support display of card information.

Create another *struct* to capture data about the current state of the player. It will consist of:

- *Card* object: stores the face value and suit of the current battling card.
- score: used to track the number of battles won by each player.

Details: Activities in *main()*

Create two player objects.

In this simplified (and not entirely complete) game of *Card War*, each of the two players will be dealt a randomly selected card³. The random card information will be captured in each player's card object. The card values will be displayed for both players.

The dealing process happens again and again until the deck of cards is ultimately exhausted. Of course, this suggests a loop. Assume that you are dealing with a traditional deck of 52 cards.

After the two cards are dealt (one to each player object), determine the winner and increment their score by one. The winner is determined by looking only at the face value⁴. In cases where the face values are the same, neither player wins, and program execution continues to the next dealing process.

After leaving the loop (when the deck of cards is exhausted) display the score for each player.



```
Command Prompt
Player 1 has 7 of Spades Player 2 has 5 of Diamonds
Player 1 wins
Battle Number: 14
Player 1 has 1 of Spades Player 2 has 9 of Hearts
Player 2 wins
Battle Number: 15
Player 1 has 6 of Diamonds Player 2 has 13 of Hearts
Player 2 wins
Battle Number: 16
Player 1 has 3 of Hearts Player 2 has 6 of Hearts
Player 2 wins
Battle Number: 17
Player 1 has 12 of Clubs Player 2 has 4 of Spades
Player 1 wins
Battle Number: 18
Player 1 has 5 of Spades Player 2 has 9 of Hearts
Player 2 wins
Battle Number: 19
Player 1 has 5 of Spades Player 2 has 6 of Clubs
Player 2 wins
Battle Number: 20
Player 1 has 13 of Diamonds Player 2 has 9 of Hearts
Player 1 wins
Battle Number: 21
Player 1 has 11 of Clubs Player 2 has 1 of Clubs
Player 1 wins
Battle Number: 22
Player 1 has 9 of Clubs Player 2 has 6 of Diamonds
Player 1 wins
Battle Number: 23
Player 1 has 5 of Diamonds Player 2 has 13 of Hearts
Player 2 wins
Battle Number: 24
Player 1 has 12 of Hearts Player 2 has 11 of Spades
Player 1 wins
Battle Number: 25
Player 1 has 11 of Diamonds Player 2 has 12 of Hearts
Player 2 wins
Player 1 won 13 hands.
Player 2 won 12 hands.
```

¹ To simplify your work, treat aces as low value cards (that is, as the value 1, rather than a value higher than King).

² A traditional deck of cards has the suits: Spades, Clubs, Hearts and Diamonds.

³ This will be purely random. Thus, it is likely that a particular card (let's say the 3 of Hearts) might be randomly created more than once. Clearly, this isn't how a real *Card War* game would work, but with the limited time during the test, the functionality will be kept as simple as possible.

⁴ You can ignore the suit in determining a winner because of time constraints. You should display the suit and face value during game play.

Possible Solution

There are many correct ways to solve this programming problem. The following code is just one of those approaches.

Other major alternatives involve different methods of handling the suit. Instead of using the enumerated type (as I did below), you could implement suit as a *string* or as an *int*. These alternatives could be equally correct.

```
#include <conio.h>

#include <iostream>
using namespace std;

struct Card {
    enum Suit { Club, Spade, Heart, Diamond };
    int nValue;
    Suit sSuit;
};

struct Player {
    Card cCard;
    int nScore;
};

void main()
{
    system("cls");
    Player Player1;
    Player Player2;
    Player1.nScore = 0;
    Player2.nScore = 0;

    const int nNumOfBattles = 52/2;
    for (int i=0; i<nNumOfBattles; ++i) {
        cout << "\nBattle Number: " << i;
        Player1.cCard.nValue = rand()%13+1;
        Player1.cCard.sSuit = Card::Suit(rand()%4);
        cout << "\nPlayer 1 has " << Player1.cCard.nValue << " of ";
        switch (Player1.cCard.sSuit) {
            case Card::Club:      cout << "Clubs"; break;
            case Card::Spade:     cout << "Spades"; break;
            case Card::Heart:     cout << "Hearts"; break;
            case Card::Diamond:   cout << "Diamonds"; break;
        }

        Player2.cCard.nValue = rand()%13+1;
        Player2.cCard.sSuit = Card::Suit(rand()%4);
        cout << " Player 2 has " << Player2.cCard.nValue << " of ";
        switch (Player2.cCard.sSuit) {
            case Card::Club:      cout << "Clubs"; break;
            case Card::Spade:     cout << "Spades"; break;
            case Card::Heart:     cout << "Hearts"; break;
            case Card::Diamond:   cout << "Diamonds"; break;
        }

        if (Player1.cCard.nValue == Player2.cCard.nValue) {
            cout << "\n Draw . . . same face value.";
            continue;
        }
        else if (Player1.cCard.nValue > Player2.cCard.nValue) {
            cout << "\n Player 1 wins";
            Player1.nScore++;
        }
        else {
            cout << "\n Player 2 wins";
            Player2.nScore++;
        }
        getch();
    }
    cout << "\nPlayer 1 won " << Player1.nScore << " hands.\n";
    cout << "Player 2 won " << Player2.nScore << " hands.\n";
}
```