CS 215 - Fundamentals of Programming II
Spring 2020 - Programming Project 2
20 points

Out: February 17, 2021
Due: February 26, 2021 (Friday)

Reminder: Programming Projects (as opposed to Homework problems) are to be your own work. See syllabus for definitions of acceptable assistance from others.

Problem Statement
In a standard deck of playing cards, there are 52 cards, each having a unique suit and value combination. In Figure 1 below, the cards are arranged with each row containing the cards of one suit in the order of Clubs, Spades, Hearts, and Diamonds. The values in each row are named Ace, Two, Three, Four, Five, Six, Seven, Eight, Nine, Ten, Jack, Queen, and King. Individual cards are referred to as "value" of "suit", e.g. Ace of Spades.

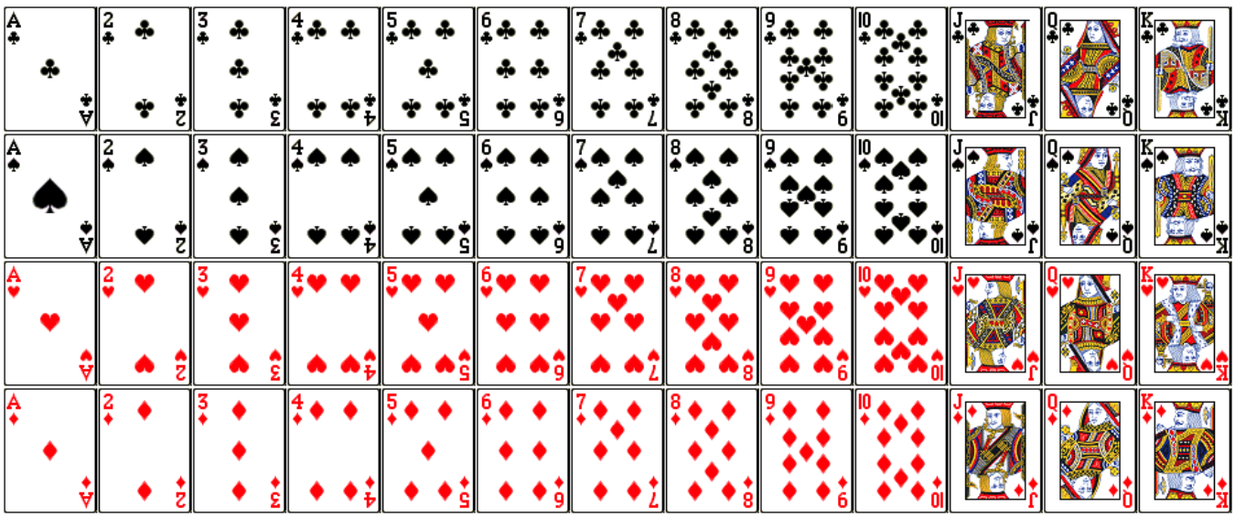

Figure 1: Standard deck of playing cards

The numeric value of each card is as indicated for values Two through Ten (i.e., 2 through 10), Jack is 11, Queen is 12, King is 13. For the purposes of this project, an Ace will be considered the lowest valued card, so it has numeric value of 1. For some card games, the cards are ordered using numeric value first, then suit as a "tiebreaker". If two cards have the same numeric value, then the suit is considered in the following ascending order: Clubs, Diamonds, Hearts, Spades. For example, any Two is always less than any Jack, and the Jack of Clubs is less than the Jack of Spades.

Specification for Card Class
Consider the following specification for a Card class that models a standard playing card. The analysis of each operation is given formally and the function parameters must be declared in the order specified, but the design of each operation is given informally in English.

Data Attributes:

Objects	Type	Name
value character	char	value_char
suit character	char	suit_char

The card values will be represented by single characters (to make input easier). Ace will be indicated by character 'A'; Two through Nine are represented as expected using characters: '2', '3', '4', '5', '6', '7', '8', and '9'; and Ten through King will be represented by: 'T', 'J', 'Q', and 'K', respectively. The card suits are represented by the characters: 'C', 'D', 'H', and 'S' for Clubs, Diamonds, Hearts, and Spades, respectively.

The class invariant is that these attributes must contain a valid value as indicated above. I.e., it should not be possible to have a Card object this not a valid playing card.

Operations

- Default/Explicit-value constructor - initialize attributes from passed values
  Must have default argument values of 'A' and 'C' for initial_value and initial_suit (i.e., Ace of Clubs) - **there are no other constructors for this class**.
  Precondition: Initial values must meet the class invariant. If they do not, the default values should be stored and an error message should be displayed saying so.
  Postcondition: The constructed object is a valid card.

  Analysis

Objects	Default	Type	Movement	Name
initial value	'A'	char	received	initial_value
initial suit	'C'	char	received	initial_suit

- value - Returns the value character of this card

  Analysis

Objects	Type	Movement	Name
value character of this card	char	returned	value_char

- suit - Returns the suit character of this card

  Analysis

Objects	Type	Movement	Name
suit character of this card	char	returned	suit_char

- numeric_value - Returns the numeric value of this card as defined above.

  Analysis

Objects	Type	Movement	Name
numeric value of this card	int	returned	-----

- to_string - Returns the output equivalent (see operator<< below) of this card as a string, e.g. "AC" for the Ace of Clubs. Hint: start with an empty string and concatenate the characters to it.

  Analysis

Objects	Type	Movement	Name
output equivalent of this card	string	returned	-----

- full_name - Returns the full name of this card as a string, e.g. "Ace of Clubs". Hint: determine what the value and suit strings are first, then concatenate them together.

  Analysis

Objects	Type	Movement	Name
full string name of this card	string	returned	-----

- operator== - **friend** overloaded operator, returns true if the two Card objects represent the same playing card. Two Card objects are the same if they have the same value and suit characters.

  Analysis

Objects	**Type**	**Movement**	**Name**
a card	Card	received	lhs
another card	Card	received	rhs
result of comparison	bool	returned	-----

- operator< - **friend** overloaded operator, returns true if the left Card object operand comes before the Card object operand using the ordering rules given above in the problem statement.

  Analysis

Objects	**Type**	**Movement**	**Name**
a card	Card	received	lhs
another card	Card	received	rhs
result of comparison	bool	returned	-----

- operator <=, operator >, operator>=, operator != **- friend overloaded operators that receive two Card objects and return true or false as appropriate. Implement these functions using operator== and operator<.**

- operator>> - **friend** overloaded operator function that reads card values from an input stream without prompting in format **`VS`** (value character, suit character), stores them in the Card attributes, and returns the input stream. The card value should be read in as a string, and checked that it has exactly two characters that meet the class invariant. If not, the Card object is unchanged and the input stream should be put in the failed state and returned. Also, if the input stream fails for any reason, it should just be return as is. **See the design notes below.**

  Analysis

Objects	**Type**	**Movement**	**Name**
input stream	istream	received, passed back, and returned	in_stream
card object	Card	passed back	a_card

- operator<< - **friend** overloaded operator function that prints the Card attributes to an output stream in format **`VS`** (value character, suit character).

  Analysis

Objects	**Type**	**Movement**	**Name**
output stream	ostream	received, passed back, and returned	out_stream
card object	Card	received	a_card

Assignment

This assignment will be distributed by the GitKeeper submission system. Follow the directions in the assignment email. Be sure to commit your changes early and often. The first commit for this assignment is expected to be by Friday, February 19.

The Card class definition and friend operator function prototypes must be put in header file **`card.h`** with suitable compilation guards. The implementations of the Card class and friend operator functions must be put in source file **`card.cpp`**. Note that all public names (the class name, the operation names, and the file names) must be as specified above including capitalization.

Write a main program in file **carddriver.cpp**. This program should demonstrate that your Card class meets all of the specifications given above. Part of your grade will depend on how well you test your class. In addition, the submission system will run a specific driver program to test your Card class.

You must submit a makefile named **Makefile** for your project that has a first target for an executable file named **carddriver** created from linking objects files and additional targets for the object files. It should conform to the examples demonstrated in class and in the *Very Basic make* handout.

REMINDER: Your project must compile for it to be graded. Submissions that do not compile will be returned for resubmission and assessed a late penalty. Submissions that do not substantially work also will be returned for resubmission and assessed a late penalty.

Follow the guidelines in the *C++ Programming Style Guideline* handout. As stated in the syllabus, part of the grade on a programming project depends on how well you adhere to the guidelines. The grader will look at your code listing and grade it according to the guidelines.

Design Notes

1. Since there are multiple places where the input values are checked for meeting the class invariant, it might be worthwhile writing a function that receives the value and suit characters and returns true if they represent a valid card and false otherwise. **Such utility functions are free functions and should have their prototypes placed at the beginning of the source file in which it is used and their function definitions placed at the end of the source file.** Please note that C++ has a boolean type **bool** with literal values **true** and **false**. These should be used in boolean contexts instead of **int** with 1 and 0.

2. By convention, when an operator>> encounters invalid input data, it changes the input stream state to a failed state. **For this project, the input data should be read into a local string variable first.** If the stream has already failed (tested using **(!in_stream)**, which returns true when the stream is in a failed state), the function should return **in_stream** immediately. After this test, the input data should be checked that it meets the class invariant. If it is not valid, the input stream is put into a failed state as shown below. In other words, operator>> will have the following structure:

```
// code for reading in data values into local variables goes here

if (!in_stream)  // stream failed or eof
   return in_stream;

// code for checking class invariant goes here
if (<input is not valid>)
{
   in_stream.setstate (ios_base::failbit);  // fail the stream
   return in_stream;
}

// code for setting data members goes here

return in_stream;
```

How to submit

Reminder: all public names (the class name, the operation names, and file names) must be as specified including capitalization, and all parameters must be implemented in the order listed. Internal names (private data member, private member functions, function parameters, local variables) can be whatever you wish. The grading system also requires that the format of any created strings or output be exactly as expected including whitespace, capitalization, and spelling. Submit projects by first adding and committing changes, then pushing the project. See handout *Using GitKeeper for Submitting Assignments* for additional information.