CS 2413 001: Data Structures, Summer 2000
Programming Project #2: Dealing Cards
Due in class Friday 7 July 2000

Note: because of the July 4 holiday, projects submitted Monday 10 July 2000 will be penalized 10%, not 25%.

`http://www.cs.ou.edu/~cs2413/`

This project will give you experience using arrays, strings, vectors and linked lists. You will use the same development process as in Programming Projects #0 and #1.

A standard deck of playing cards has 52 cards, consisting of four *suits* — clubs (♣), diamonds (♢), hearts (♡) and spades (♠) — each of which has cards of 13 *ranks* (2, 3, 4, 5, 6, 7, 8, 9, 10, jack, queen, king, ace). For this project, assume that a deck of cards has no jokers.

A *shuffle* is a random reordering of the cards in a deck. To shuffle in real life, you perform several *riffles* (cut the deck into two halves, then combine the halves together). Computerized shuffling is described below.

A *player* is a person playing the card game, who therefore is dealt a certain number of cards. For purposes of this programming project, you may assume that all players will be dealt the same number of cards. The collection of the player's cards is called their *hand*.

A *game* is a group of players, each of whom is dealt a hand of cards.

For this project, write a program that deals a game of cards. (Note that you are not required to implement any actual play.) Implement the following classes:

1. `ArrayClass`, `String`, `Vector` and `LinkedList`.
   You are welcome to use the classes in the textbook, or to implement your own, in which case you should make sure to implement the same set of methods as the version in the textbook.
2. `Card`: a single playing card.
   The value of the playing card actually consists of two components: a suit and a rank. You may implement the card values any way you like (e.g., as an `int` valued 0-51, where 0 represents the two of clubs and 51 represents the ace of spades). For this class, you will need the following public methods:
   (a) a default constructor;
   (b) a constructor with an argument representing the card's value;
   (c) a copy constructor;
   (d) a destructor;
   (e) a mutator to set the value of the card;
   (f) an overloaded assignment operator;
   (g) overloaded relational operators (`==, !=, <, <=, >, >=`) — note that, for our purposes, the Ace has the greatest rank (as above), the suits' ascending order is Club, Diamond, Heart, Spade (as above), and every card of a greater suit is greater than any card of a lower suit (e.g., two of diamonds is greater than ace of clubs);
   (h) a friend output stream operator that outputs the value of the card as a two-character code: rank output as 2/3/4/5/6/7/8/9/T/J/Q/K/A (note that rank 10 is represented by T), followed by suit output as C/D/H/S. For example, five of diamonds is represented as 5D, and ten of spades is represented TS.

3. `Deck`: a subclass of `Vector<Card>`.
   For this class, you will need the following public methods:

   (a) a default constructor;

   (b) a copy constructor;

   (c) a destructor;

   (d) a mutator to shuffle the cards, which has a single argument representing the number of riffles, and which performs the shuffle as follows:

   ```
   For each riffle
       For each position in the deck
           Randomly choose another position in the deck.
           Swap the values in the current and random positions.
   ```

   (For a description of how to choose something at random, see below.)

   (e) a mutator that deals a single card (i.e., removes it from the deck and returns its value): it should deal the **last** card in the deck, which in this programming project will be considered the top of the deck;

   (f) a friend output stream operator that outputs the list of cards in order from the beginning to the end of the deck vector.

4. `Player`: a subclass of `LinkedList<Card>`, with an additional protected field representing the player's name. For this class, you will need the following public methods:

   (a) a default constructor;

   (b) a constructor with an argument representing the name of the player (a `String`);

   (c) a copy constructor;

   (d) a destructor;

   (e) an accessor that returns the player's name;

   (f) an accessor that returns the number of cards that the player has;

   (g) a mutator that sets the name of the player;

   (h) a mutator that draws a single card from a deck and places it into the player's hand **in ascending sorted order** (see below);

   (i) a friend output stream operator that outputs the player's name, followed by the list of the player's cards in ascending order; e.g.,
   `Henry:  5C 7C TD JD AD 2H 4H 6H 8H QH JS QS KS`

   Note: it is recommended to implement the player's name as a **pointer** to a `String`, because that will simplify construction.

5. `Game`: a subclass of `ArrayClass<Player>`. For this class, you will need the following public methods:

   (a) a default constructor;

   (b) a constructor with arguments representing the name of the game (a `String`) and the names of the players (`ArrayClass<String>`, which implicitly carries the number of players);

   (c) a copy constructor;

   (d) a destructor;

   (e) an accessor that returns the game's name;

(f) an accessor that returns the number of players;

(g) an accessor that returns the number of cards that each player is supposed to be dealt;

(h) a mutator that sets the name of a given player;

(i) a mutator that sets the number of cards that each player is supposed to be dealt;

(j) a mutator that deals all of the cards that all players require, in round-robin fashion (e.g., every player should have four cards before any player gets a fifth card);

(k) a friend output stream operator that outputs the name of the game and then all of the players' information.

For all of these classes, you should also declare appropriate exception classes, and you should handle exceptions as gracefully as possible.

The main program should behave like this:

1. Prompt the user and input the name of the game.
2. Prompt the user and input the number of players.
3. For each player, prompt the user and input that player's name.
4. Prompt the user and input the number of cards per player.
5. Instantiate the game using the name of the game and an array of players' names.
6. Set the number of cards per player.
7. Deal.
8. Output the game (i.e., the name of the game and the list of players and their hands) using the appropriate output stream operator.

Again, note that you are not required to implement any actual play.

Run your program using the following inputs:

1. - Game: Bridge
   - 4 players: Jenny, Ed, Alisa, Henry
   - 13 cards per player
2. - Game: Fish
   - 5 players: Alpha, Beta, Gamma, Delta, Epsilon
   - 5 cards per player
3. - Game: Trumps
   - 7 players: Charles, Ada, Alan, John, Grace, Don, Marsha
   - 7 cards per player

Random Numbers

The C++ standard function library contains a function named `random` that takes no arguments and returns a `long` value that is randomly chosen. So, a random number between zero and a specified maximum can be chosen this way:

```
    int random_0_51 = (int)random() % 52;
```

Note that use of the `random` requires that the standard library header file `stdlib.h` be included at the top of any file that contains a call to `random`.

Inserting Into a Linked List in Ascending Sorted Order

This project description specifies that a player's cards should be maintained in ascending sorted order, and that therefore each newly dealt card should be inserted in ascending sorted order. To accomplish this, add a method to the class `LinkedList` that inserts an element in ascending sorted order. This method will look very similar to the existing method `insertAt`.

Templated Classes and Conditional Compilation

In C++, classes that are not parameterized (templated) can be compiled independently and then brought together during the link stage of compilation.

In the case of parameterized classes, however, the code for method implementation (the `.cpp` file) does not represent actual code, but rather is a template for what the code should look like when a particular version of the class is instantiated. For example, `ArrayClass<Object>` is simply a generic description of what a particular `ArrayClass`, such as `ArrayClass<char>`, would look like.

Therefore, it's not enough to include the header (`.h`) file for a parameterized class. The implementation (`.cpp`) file must be included as well.

Put this latter `#include` inside the header file, like so:

```
#ifndef MyClass_h
#define MyClass_h
...
class MyClass {
...
} ; // class MyClass

#include "MyClass.cpp"

#endif // #ifndef MyClass_h
```

This way, you can still include just the header file in other `.cpp` files.

Programming Style

The style for this programming project is the same as described for Programming Project #1.

What To Turn In

You should turn in the same items, in the same format, as for Programming Project #1.

You may turn your project in early if you choose; otherwise, turn it in during class on Friday 7 July. If you turn it in after the close of class (3:20pm), it will be considered late, at which point you will lose 10% of the maximum value but can turn it in any time through the close of class on Monday 10 July. Submissions after 3:20pm Monday 10 July will receive no credit.

References

S. Radhakrishnan, L. Wise & C. N. Sekharan, *Object-Oriented Data Structures Featuring C++*, 1999.
`http://www.pagat.com/quartet/gofish.html`
`http://www.pagat.com/whist/kowhist.html`