Programming Assignment #2

Due: 10/19/2022 23:59:59

Topic: Poker game player simulation

Objective: basic class constructs (basic constructor, accessor, mutator, access control, and encapsulation)

Description:

In this assignment, we will use C++ classes to simulate player in a poker (Soha) game. Many functions need to be implemented in order to simulate a complete poker game. However, in this assignment we ONLY simulate the player part of the game.

There is more than one way to play a poker game but the basic rules for counting points for a hand of cards are the same. Each user is given a set of 5 cards, called a hand. The face values (pips) and patterns of the five cards determine the order of a hand of cards. For example, a pattern consisting of two cards of the same pip is called a pair. A pattern of cards with five consecutive pips is called a straight.

You have two classes to implement in this simulation: **Card** and **SHPlayer**. The Card class encapsulates what a regular card has in a poker game: pip and suit (or a unique ID number for a combination of pip and suit). A user of the Card class calls its member functions to query and set card ID's or other attributes. The SHPlayer class encapsulates what a poker player needs to have. For example, a player needs to keep track of a hand of 5 cards given one by one. You also need to check for overflows on array indices.

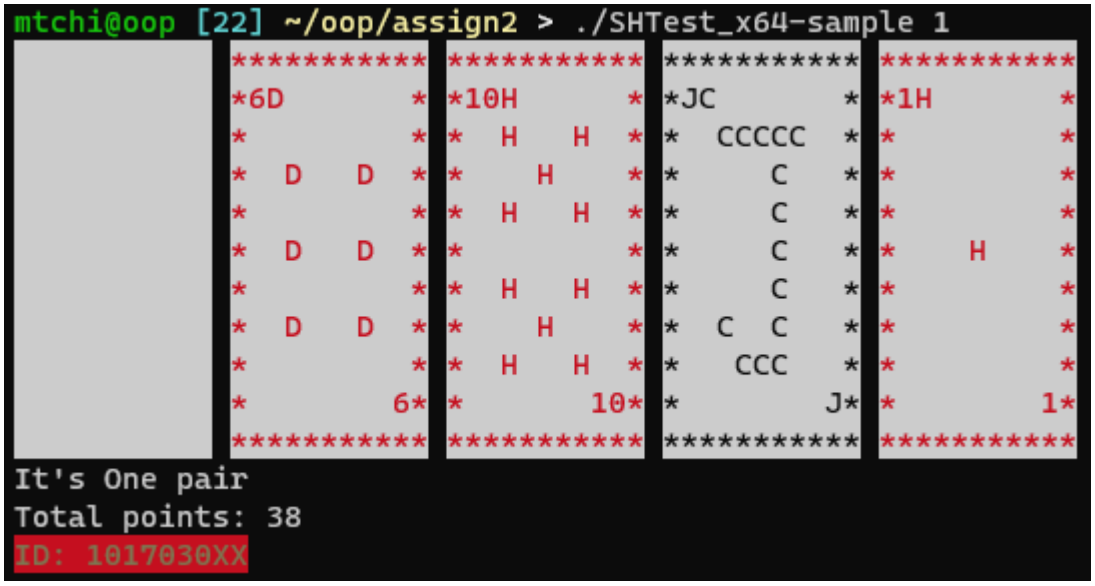
The specifications of the public interfaces for these two classes are given to you in the header files. You are asked to add necessary data and implement member functions for the given public interface. Read the comments in the class declaration for details of these functions. Not every member functions will be used in this assignment but you are still asked to implement them for completeness of the classes.

To test these two classes, your program should draw 5 cards and print them to the screen in the way we did in the first assignment except for that the first card may not be printed with its face up according to command-line arguments. In addition, print the sum of the card values to the screen as well. (Ace is counted as one.) ~~One minor change to the card printing routine from the last assignment is that you should print the name of the player on the leftmost column of the cards.~~

Assignment Directory: `/usr/local/class/oop/assign/assign2`

Sample Output:

A sample output from testing the classes is given as follows. A sample executable can be found in the assignment directory. The program takes two optional command-line arguments, which are a seed for the random number generator and a flag indicating whether to print the first card facing up or down.



Files in the Assignment Directory:

- **`Makefile`**: a sample makefile.
- **SHTest.cc**: a testing program. You need to modify the PrintMyID function. You also need to add testing code such that the program produces the same result as shown in the sample output. (We provide partial code.)
- **`AnsiPrint.cc`** and **`AnsiPrint.h`**: the `AnsiPrint` functions from assignment #1. (We provide.)
- **`CardPat.h`**: an array of skeletons for card patterns. This array is the same as in the previous assignment. (We provide.)
- **`Card.h`** and **`Card.cc`**: the class for a poker card. The specification is given in the `Card.h` file. You need to define the private part of the class and implement all necessary member functions in the `Card.cc` file. (We define and you write.)
- **SHPlayer.h** and **SHPlayer.cc**: class for a poker player. The specification is given in the header file. You need to define the private part of the class and implement all necessary member functions in the SHPlayer.cc file. Specifically, you are asked to write a method that can sort the cards. To reduce your load, all the implementation for determining card patterns except for the flush straight pattern is given. (We define and you write partially.)

What to Hand in:

You need to hand in a complete version of your source code electronically as in the previous assignments.

Implementation Notes:

You can feel free to design your own data structure or internal functions as long as the interface/public functions adhere to the specification. You are fully responsible for maintaining the consistency of the data in your class. For example, you are responsible for checking if an array index is out of bound and all data members are appropriately initialized. In addition, to make grading easier, please do not add extra I/O's to your program. For example, please DO NOT add code to query options from the user or to output your results to a file.