Programming Assignment #3

Due: 10/30/2022 23:59:59

Topic: simulating a simple but complete SOHA game

Objective: learn more about classes and objects

Description:

In this assignment, we will use C++ classes to write a complete Poker game. This assignment is based on the classes that we have implemented in the previous assignments. You will have to write a playable game that involves a dealer and a player. Your program must be able to determine the result of a game. Each part is given 20 points to start with, and the bet for each run of a game is determined according to the number of cards that have been distributed to a player. For example, if the player gives up when he/she has four cards, 4 points will be subtracted from his credit. The program ends with comments if each side runs out of credits. (See the sample executable.)

There are several ways to play a Poker (SOHA) game. The rules we adopt are described as follows.

- For each run of the game, we will use a deck of only 24 cards consisting of 9, 10, J, Q, K, A of four suits.
- The way to determine the pattern for a hand of cards is the same as in the previous assignment except for that A is counted as 14 instead of 1.
- In the beginning of a game, the dealer and the player are all given two cards from a deck of cards dealt by the dealer. The dealer's first card faces down so that the player can only see the value of the second card when asking for more cards.
- The player can ask for more cards for both the player and the dealer from the dealer as long as he/she does not give up.
- When the number of cards for a player reaches 5, no more cards can be added and the result of the game is then determined by the dealer.
- If the patterns of the player and the dealer are the same, the result is determined according to the their total numbers of pips. (The larger one wins.)

You can also find a more detail description of the rules in the following web page: https://cyc27.cycgame.com/cyc/cgi-bin/manual.php?i=manG&game=ShowHand#anchor06

You are asked to implement an interactive program that provides a simple text menu bar for a user to choose desired actions. In addition to the menu bar, cards for both sides (player and dealer) are displayed on the screen. They are distinguished by the name string printed vertically on the first column for each row of cards.

Your program will use the classes you wrote in the previous assignments. On top of that, there are three additional classes in this assignment. You have to write two of them: `SHDealer` and `SHGame`. The third one, called `ConsoleMenu`, is provided to you. This class prints a menu to the standard output and obtains queries from a user through standard input. You will use this class inside the `SHGame` class.

The `SHDealer` class encapsulates what a dealer does in a Poker game. It is instantiated with a deck of 24 cards. The dealer is in charge of distributing cards to players upon requests. In other words, the dealer has to manage the given decks of cards as in a real scenario. In addition to managing cards, the dealer itself is also a player; thus, the `SHDealer` class may contain a `SHPlayer` object as its data member. In addition, it can also determine who win the game according to their pattern.

Another class that you need to implement is the `SHGame` class. The only public function is

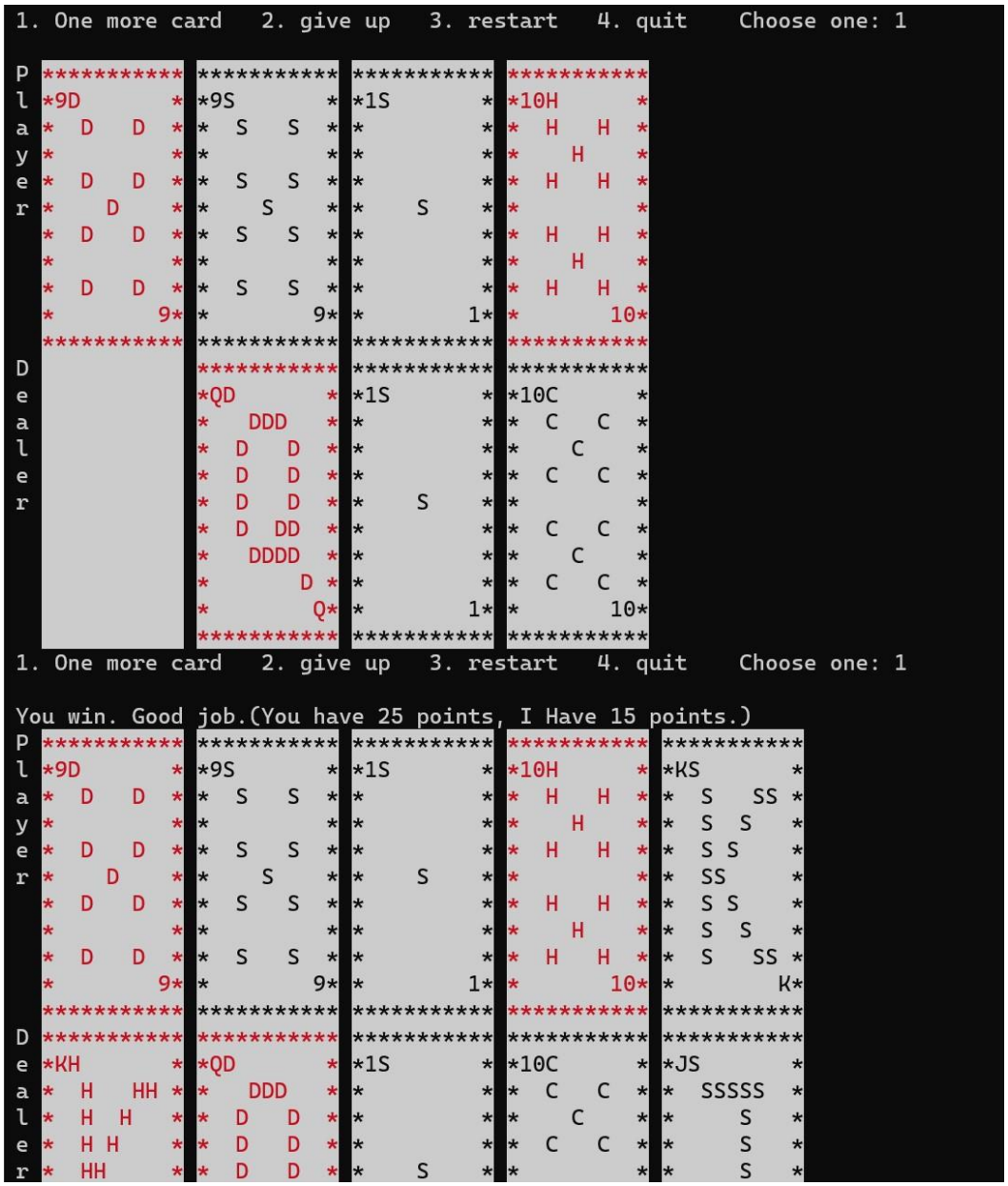
`oneRun()`. The user of this class will keep on calling this function as long as it returns `TRUE` in the previous run. When initializing the class, you are given two objects: a Poker player and a Poker dealer. It also contains a `ConsoleMenu` object that is used as an interface to query a user's action. This class also needs to handle inappropriate inputs from the user and make sure that the game will not enter an incorrect state. For example, a user cannot give up a game that has not been started yet. One can not restart a game if it is not over yet. One cannot continue to play if it has run out of credits.

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 on these functions.

Assignment Directory: `/usr/local/class/oop/assign/assign3`

Sample Output:

A sample executable can be found in the assignment directory. The program takes one optional command-line arguments. The argument is a seed for the random number generator. A sample output of the program is shown below.



Files in the Assignment Directory:

- **`Makefile`**: a sample makefile.
- **`SHTest.cc`**: a testing module. You need not modify this file except for providing your own ID in calling the PrintMyID function. (We provide.)
- **`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 same as in the previous assignment. (You have it already.)
- **`SHPlayer.h`** and **`SHPlayer.cc`**: the class for simulating a Poker player. The same as in the previous assignment. (You have it already.)
- **`ConsoleMenu.h`** and **`ConsoleMenu.cc`**: the class for printing a simple menu. You have to study how to use this class in your code. (We provide.)
- **`SHDealer.h`** and **`SHDealer.cc`**: the class for simulating a Poker dealer. It distributes cards to the user and himself/herself according to the rules in a Poker game. The specification is given in SHDealer.h but the details need to be worked out in SHDealer.cc by you. **(We define and you write.)**
- **`SHGame.h`** and **`SHGame.cc`**: the class that encapsulates the overall Poker game. The class needs to be initialized with a player and a dealer. The only public member function is oneRun() which will be called in a loop until the user quits the game. In this function, you have to ensure correctness of game status. **(We define and you write.)**

What to Hand in:

You need to hand in a complete version of your source code electronically.

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 specifications. You are fully responsible for maintaining the consistency of the data in your class. To make grading easier, please use the same menu items as shown in the sample output and 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 the results to a file.