Hao Huynh

Professor Mark Lehr

CSC 17A – 48983

14 October 2015

Project 1: Poker Game.

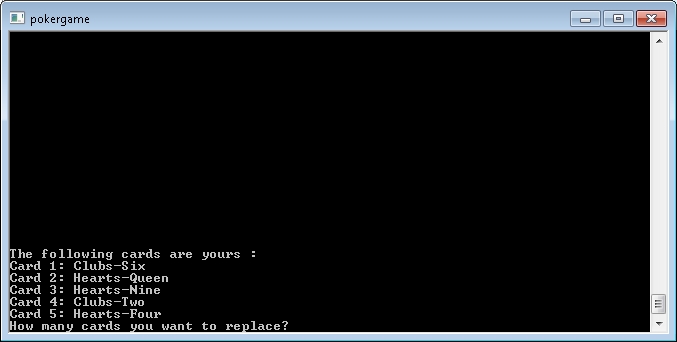
1. Introduction.

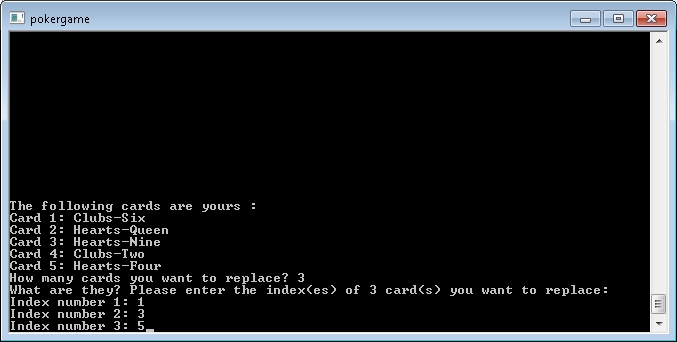
This Poker Game is a simple form of the standard gambling Poker card games. The purpose of this game is to find a winning hand, and there is only one player. The player is given five random cards at the beginning. Then, that player can choose to exchange any number of cards from the current given hand. The player wins the game if the final five cards can make:

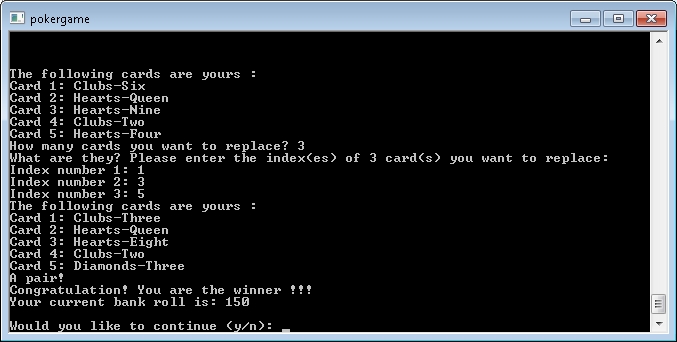
* A Royal Flush (A Straight Flush with an ACE)
* A Straight Flush (A combination of Flush and Straight)
* A Flush (All five cards are in the same suit)
* A Straight (All five cards increase continuously by rank)
* A Four of the Same Kind (by rank)
* A Full House (There are three cards in a rank and the others in another rank)
* A Three of the Same Kind (by rank)
* Two Pairs (by ranks)
* A Pair (by rank)

I decided to implement this game because the algorithms behind the screen was not so complicated, so I could complete it within two weeks. Besides, I could demonstrate techniques such as memory allocation/deallocation, data structure, functions and arrays of structures, pointer arrays, character arrays, string object, binary file IO, and the OOP with classes and instances.

1. Summary.
   1. The program statistics:
      1. The number lines of code (LOC) are about 1000 lines which are greater than 250 lines in the requirement.
      2. There are 8 main variables.
      3. There are about 38 constants.
      4. There is 1 constructor and 1 destructor.
      5. There are about 27 methods.
   2. Miscellaneous: This program took me approximately 24 hours for coding and fixing errors. Moreover, I had learned about the logical database mapping, the insertion sort for an empty vector, and collections such as vector and set.
2. Description.
   1. The program problems:
      1. Manage 52 cards.
      2. Deal any 5 of them to the player at the first time, and then replace any number of those 5 cards without duplicating the cards.
      3. Apply the Poker game rules to check whether the player wins or not.
   2. The program solutions:
      1. Create a logical Poker Card Table.
      2. Use a vector to store all cards that had been dealt for duplication testing.
      3. Sort all cards in the current hand by rank and by suit before checking for a Poker game result.
   3. Sample Input/Output:
      1. The Poker game menu: (The player choose an option by its index)
      2. The player enters a bet roll before playing the game:
      3. The player gets the first five cards:



* + 1. The player chooses any number of the first five ones for replacing:
    2. The final result:



* 1. Flowchart and Pseudocode:
  2. Variables:

|  |  |  |
| --- | --- | --- |
| Type | Name | Location |
| vector<Card\*> | crCards  (The player’s current cards) | class PokerCardTable |
| vector<Card\*> | crSRnks  (The player’s current cards sorted by rank) |
| vector<Card\*> | crSSuits  (The player’s current cards sorted by suit) |
| set<int> | rIndexes  (The on-screen replacing indexes) | method PokerCardTable::replaceCards() |

* 1. Concepts:

|  |  |
| --- | --- |
| Concept | Location |
| 1. Memory allocation | * main function. * PokerCardTable::populateCardBy() |
| 1. Memory deallocation | * PokerCardTable::deleteCardsBy() * PokerCardTable::deleteCards() |
| 1. Data structure | * Card in “PokerCardTable.h” |
| 1. Functions with structures | * Most functions in class PokerCardTable |
| 1. Arrays with structures |  |
| 1. Pointer arrays |  |
| 1. Character arrays | * main function. |
| 1. String object | * main function. * class PokerCardTable * class PokerHelper |
| 1. Object Oriented Programming (OOP) | * class PokerCardTable * class PokerHelper |

1. Program (Header Files).
   1. PokerCardTable.h

*/\**

*\* File: PokerCardTable.h*

*\* Author: HaoHuynh*

*\**

*\* Created on October 10, 2015, 10:49 AM*

*\*/*

**#ifndef POKERCARDTABLE\_H**

**#define POKERCARDTABLE\_H**

**#include <vector>**

**#include <string>**

**#include <set>**

using namespace std;

*/\**

*\* There are thirteen ranks per suit: from Ace to King.*

*\*/*

**enum** CARD\_RANKS {

ACE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN, JACK, QUEEN, KING

};

*/\**

*\* There are four suits in fifty two desk cards.*

*\*/*

**enum** CARD\_SUITS {

HEARTS, DIAMONDS, CLUBS, SPADES

};

*/\**

*\* A card is defined by a combination of rank and suit.*

*\* The id is the index of a card in fifty two desk cards*

*\*/*

**typedef** **struct** CARD {

**int** id;

CARD\_SUITS suit;

CARD\_RANKS rank;

} Card;

*/\**

*\* This class is used for managing user's cards base on the below logical mapping table*

*\**

*\* 0 1 2 3 4 5 6 7 8 9 10 11 12*

*\* Suits/Ranks Ace 2 3 4 5 6 7 8 9 10 Jack Queen King*

*\* 0 Hearts 0 1 2 3 4 5 6 7 8 9 10 11 12*

*\* 1 Diamonds 13 14 15 16 17 18 19 20 21 22 23 24 25*

*\* 2 Clubs 26 27 28 29 30 31 32 33 34 35 36 37 38*

*\* 3 Spades 39 40 41 42 43 44 45 46 47 48 49 50 51*

*\*/*

class PokerCardTable {

private:

*/\**

*\* The total of cards are 52*

*\*/*

**static** **const** **int** CARDS\_TOTAL **=** 52;

*/\**

*\* There are thirteen ranks per suit.*

*\*/*

**static** **const** **int** RANKS\_PER\_SUIT **=** 13;

*/\**

*\* Maximum poker cards for each player.*

*\*/*

**static** **const** **int** FIVE\_POKER\_CARDS **=** 5;

*/\**

*\* There are 4 suits in 52 desk cards.*

*\*/*

**static** **const** **int** SUIT\_MAX **=** 4;

*/\**

*\* The first poker card index*

*\*/*

**static** **const** **int** INDEX\_0 **=** 0;

*/\**

*\* The second poker card index*

*\*/*

**static** **const** **int** INDEX\_1 **=** 1;

*/\**

*\* The third poker card index*

*\*/*

**static** **const** **int** INDEX\_2 **=** 2;

*/\**

*\* The fourth poker card index*

*\*/*

**static** **const** **int** INDEX\_3 **=** 3;

*/\**

*\* The fifth poker card index*

*\*/*

**static** **const** **int** INDEX\_4 **=** 4;

*/\**

*\* Labels for displaying a card suit.*

*\*/*

**static** **const** string CARD\_SUIT\_LABELS[SUIT\_MAX];

*/\**

*\* Labels for displaying a card rank.*

*\*/*

**static** **const** string CARD\_RANK\_LABELS[RANKS\_PER\_SUIT];

*/\**

*\* Vector contains all the user's current cards*

*\*/*

vector**<**Card**\*>** crCards;

*/\**

*\* Vector contains all the user's current cards sorted by rank*

*\*/*

vector**<**Card**\*>** crSRnks;

*/\**

*\* Vector contains all the user's current cards sorted by suit*

*\*/*

vector**<**Card**\*>** crSSuits;

public:

*/\*\**

*\* The default constructor of PokerCardTable*

*\*/*

PokerCardTable();

*/\*\**

*\* The default destructor of PokerCardTable*

*\*/*

**~**PokerCardTable();

*/\*\**

*\* This function interacts with players to drive them through a game*

*\*/*

**bool** **populateConsole**();

private:

*/\*\**

*\* This function create a Card by mapping the id to a Card structure through*

*\* the logical Card Table*

*\* @param id*

*\* @return Card*

*\*/*

Card**\*** populateCardBy(**int** id);

*/\*\**

*\* This function uses the id to check if a card has been dealt*

*\* @param id : the Card Table index*

*\* @return true/false : A card is existed or not*

*\*/*

**bool** **isCardExistedBy**(**int** id);

*/\*\**

*\* This function tries to dealt a new card to the current hand without*

*\* duplicating any existed one*

*\* @param id : the Card Table index*

*\* @return true/false : Insertion success or not*

*\*/*

**bool** **isCardInsertedBy**(**int** id);

*/\*\**

*\* This function deals first five poker cards for the player*

*\*/*

**void** **dealsCards**();

*/\*\**

*\* This function sends all current hand cards to the console*

*\*/*

**void** **displayPlayerCards**();

*/\*\**

*\* This function generates a new card for the replacing process.*

*\* The new one should not be duplicated with any card in the current hand*

*\* @return a new Card structure*

*\*/*

Card**\*** **dealsNewCard**();

*/\*\**

*\* This function generates a new one that has not been dealt before*

*\* for the replacement process*

*\* @return a new Card structure*

*\*/*

Card**\*** **getNewCardForReplacement**();

*/\*\**

*\* This functions collects all the cards user want to replace and*

*\* replaces them with new ones*

*\*/*

**void** **replaceCards**();

*/\*\**

*\* This functions creates a list of current hand cards sorted by rank.*

*\*/*

**void** **sortCardsByRank**();

*/\*\**

*\* This functions creates a list of current hand cards sorted by suit.*

*\*/*

**void** **sortCardsBySuit**();

*/\*\**

*\* This functions check if there is an ACE, and all five poker cards are flush and straight*

*\* @return true/false*

*\*/*

**bool** **isRoyalFlush**();

*/\*\**

*\* This functions check if all five poker card are flush and straight*

*\* @return true/false*

*\*/*

**bool** **isStraightFlush**();

*/\*\**

*\* This function checks if all five poker cards have the same suit.*

*\* After sorting by suit, if the lowest card has the same suit as the highest one,*

*\* then all five poker cards will have the same suit.*

*\* @return true/false*

*\*/*

**bool** **isFlush**();

*/\*\**

*\* This function checks if all five poker cards make a straight:*

*\* Case 1: There is an ACE and the other fours are {TEN, JACK, QUEEN, KING}*

*\* or {TWO, THREE, FOUR, FIVE}*

*\* Case 2: The ranks of all cards are increasing continuously*

*\* @return true/false*

*\*/*

**bool** **isStraight**();

*/\*\**

*\* This function checks for four of the same card in a rank*

*\* Two cases: 4 + 1 or 1 + 4*

*\* @return true/false*

*\*/*

**bool** **isFourOfAKind**();

*/\*\**

*\* This function checks for 3 of the same card in a rank*

*\* and 2 of the same card in another rank*

*\* Two cases: 3 + 2 or 2 + 3*

*\* @return true/false*

*\*/*

**bool** **isFullHouse**();

*/\*\**

*\* This function checks for 3 of the same card after checking four of a kind and full house.*

*\* Three cases: 3 + 1 + 1 or*

*\* 1 + 3 + 1 or*

*\* 1 + 1 + 3.*

*\* @return true/false*

*\*/*

**bool** **isThreeOfAKind**();

*/\*\**

*\* This function checks for 2 different pairs after checking four of a kind, full house,*

*\* and three of a kind.*

*\* Three cases: 2 + 2 + 1 or*

*\* 2 + 1 + 2 or*

*\* 1 + 2 + 2.*

*\* @return true/false*

*\*/*

**bool** **isTwoPairs**();

*/\*\**

*\* This function checks for two of the same card after checking four of a kind, full house,*

*\* three of a kind and two pairs*

*\* Four cases: 2 + 1 + 1 + 1 or*

*\* 1 + 2 + 1 + 1 or*

*\* 1 + 1 + 2 + 1 or*

*\* 1 + 1 + 1 + 2 or*

*\* @return true/false*

*\*/*

**bool** **isPair**();

*/\*\**

*\* This function checks if the current hand win or lose*

*\* @return true/false: win/lose*

*\*/*

**bool** **isPlayerWin**();

*/\*\**

*\* This function deallocates the memories of a set of elements in cards and clears the cards*

*\* @param cards : a vector of Card Structure pointers*

*\* @param poss : a set of positions in cards*

*\*/*

**void** **deleteCardsBy**(vector**<**Card**\*>&** cards, **const** set**<int>&** poss);

*/\*\**

*\* This function deallocates the memories of all elements in cards and clears the cards*

*\* @param cards : a vector of Card Structure pointers*

*\*/*

**void** **deleteCards**(vector**<**Card**\*>&** cards);

*/\*\**

*\* This function reset all the current cards for a new game*

*\*/*

**void** **clean**();

};

**#endif** */\* POKERCARDTABLE\_H \*/*

* 1. PokerHelper.h

*/\**

*\* File: PokerHelper.h*

*\* Author: HaoHuynh*

*\**

*\* Created on October 10, 2015, 10:35 AM*

*\*/*

**#ifndef POKERHELPER\_H**

**#define POKERHELPER\_H**

**#include <string>**

**#include <fstream>**

class PokerHelper {

private:

*/\**

*\* Relative maximum lines of a monitor*

*\*/*

**static** **const** **int** MAX\_MONITOR\_LINES **=** 200;

*/\*\**

*\* The default file that stores the current bank roll*

*\*/*

**static** **const** std**::**string DEFAULT\_FILE\_NAME;

public:

*/\*\**

*\* Clear the screen after a certain event.*

*\*/*

**static** **void** clearMonitor();

*/\*\**

*\* Validate input of an integer number from lowerLimit to upperLimit*

*\* @param number*

*\* @param lowerLimit*

*\* @param upperLimit*

*\*/*

**static** **void** **validateValueOf**(**int&** number, **int** lowerLimit, **int** upperLimit);

*/\*\**

*\* This function saves the current bank roll to a file*

*\* @param crBkRoll : the current bank roll*

*\* @param fName : a file name*

*\*/*

**static** **void** **save**(**int** crBkRoll);

*/\*\**

*\* This function loads the current bank roll from a file*

*\* @param crBkRoll : the current bank roll*

*\* @param fName : a file name*

*\*/*

**static** **void** **load**(**int&** crBkRoll);

};

**#endif** */\* POKERHELPER\_H \*/*