**Name:**

**Advanced Programming in C++**

**Lab Exercise 3/11/2020 Stardate: 73191.26**

In this exercise you will learn about working with multiple classes in a project significantly larger in scope. I will provide you with the following classes and a drive program for you to learn the basics of how this works. In the Data Files/C++ folder on the server you are to download these files and put them into a project named Poker. When you are asked if you would overwrite poker.cpp in the project with poker.cpp that I am providing you, answer “Yes”. You should have the following files:

card.cpp card.h deck.cpp deck.h hand.cpp hand.h

randgen.cpp randgen.h poker.cpp

poker.cpp contains the main function. The .h files contain the class definition and the other .cpp files contain the class implementations.

1. Once you have your project working and printing out the deck of cards prior to and after shuffle and dealing one hand, you are to modify it to deal four hands of cards and print out the four hands.
2. Modify your project to deal a hand of seven card stud instead of a standard poker hand.

**//poker.cpp**

#include <iostream>

#include <vector>

using namespace std;

#include "deck.h"

#include "card.h"

#include "hand.h"

void Print(const Deck& d);

void Print(const vector<Card> &);

int main()

{

const int HAND\_SIZE = 5;

Deck d;

int k;

Print(d);

cout << "\n---after shuffling---\n" << endl;

d.Shuffle();

Print(d);

vector<Card> player(HAND\_SIZE);

for(k=0; k < HAND\_SIZE; k++)

{

player[k] = d.GetCard();

}

Print(player);

return 0;

}

void Print(const Deck& d)

// post: print all cards in the Deck d

// (cards still in d when done)

{

Deck copy(d); // make a copy to get cards from

int count = d.Size();

int k;

for(k=0; k < count; k++)

{

cout << copy.GetCard() << endl;

}

}

void Print(const vector<Card> & hand)

{

int k;

cout << "------------" << endl;

for (k=0; k< hand.size(); k++)

{

cout << hand[k] << endl;

}

cout << "------------" << endl;

}

**//card.h**

#ifndef \_CARD\_H

#define \_CARD\_H

// This class represents a playing card, i.e., "ace of spades"

// a Card is constructed from a rank (int in range 1..12)

// and a suit (Card::spades, Card::hearts, Card::diamonds,

// Card::clubs)

//

// Cards should be created by a Deck (see deck.h), a Deck returns

// good cards, but conceivably (if called improperly) might return

// the "zero of doughnuts" which should be thought of as a joker

//

// Jokers are different from every card, including other jokers

//

// The function toString() converts a card to a string, e.g., to print

//

// Accessor functions include

//

// int GetRank() -- returns 1, 2, ..., 13 for ace, two, ..., king

//

// bool SameSuitAs(c) -- returns true if same suit as Card c

//

// string suitString() -- returns "spades", "hearts", "diamonds" or "clubs"

//

// bool IsJoker() -- returns true if card is a joker, the zero of doughnuts

#include <iostream>

#include <string>

using namespace std;

class Card

{

public:

enum Suit {spades, hearts, diamonds, clubs};

private:

string rankString(int r) const; // return "ace", "two", ...

int myRank;

Suit mySuit;

public:

//enum Suit {spades, hearts, diamonds, clubs};

Card(); // default, ace of spades

Card(int rank, Suit s);

string toString() const; // return string version

bool SameSuitAs(const Card& c) const; // true if suit same as c

int GetRank() const; // return rank, 1..13

string suitString(Suit s) const; // return "spades", "hearts",...

bool IsJoker() const; // true if card is valid

};

ostream& operator << (ostream& out, const Card& c);

bool operator == (const Card& lhs, const Card& rhs);

#endif

**//card.cpp**

#include "card.h"

Card::Card()

{

myRank = 1;

mySuit = spades;

}

Card::Card(int rank, Suit s)

{

myRank = rank;

mySuit = s;

}

string Card::suitString(Suit s) const

{

if (s == spades) return "spades";

else if (s == hearts) return "hearts";

else if (s == diamonds) return "diamonds";

else return "clubs";

}

string Card::rankString(int r) const

// pre: 1 <= r <= 13

{

switch (r)

{

case 1: return "ace";

case 2: return "two";

case 3: return "three";

case 4: return "four";

case 5: return "five";

case 6: return "six";

case 7: return "seven";

case 8: return "eight";

case 9: return "nine";

case 10: return "ten";

case 11: return "jack";

case 12: return "queen";

case 13: return "king";

default: return "joker";

}

}

string Card::toString() const

{

if (1 <= myRank && myRank <= 13)

{

return rankString(myRank) + " of " + suitString(mySuit);

}

return "zero of doughnuts";

}

bool Card::SameSuitAs(const Card& c) const

// post: return true iff mySuit == c.mySuit

{

return mySuit == c.mySuit;

}

int Card::GetRank() const

{

return myRank;

}

bool Card::IsJoker() const

{

return myRank < 1 || 13 < myRank;

}

ostream& operator <<(ostream& out, const Card& c)

{

out << c.toString();

return out;

}

bool operator == (const Card& lhs, const Card& rhs)

{

if (lhs.IsJoker() || rhs.IsJoker())

{

return false;

}

return lhs.GetRank() == rhs.GetRank() && lhs.SameSuitAs(rhs);

}

**//deck.h**

#ifndef \_DECK\_H

#define \_DECK\_H

#include "card.h"

#include <vector>

using namespace std;

// this class respresents a deck of cards

// When a Deck is constructed, it contains 52 cards

// in a "regular" order (aces, twos, threes, ... , kings)

//

// Shuffling a deck makes it consist of 52 cards in a random order

//

// GetCard() returns a card from the deck and decreases the

// number of cards in the deck (returned by Size())

// The idea is that after shuffling, calling GetCard() 52 times

// returns each card in the deck after shuffling.

//

// Calling Shuffle again replenishes the deck with 52 cards.

class Deck

{

private:

vector<Card> myCards;

int myIndex; // current card to deal

public:

Deck(); // pristine, sorted deck

void Shuffle(); // shuffle the deck, all 52 cards present

Card GetCard(); // get a card, after 52 a joker is returned

int Size() const; // # cards left in the deck

};

#endif

**//deck.cpp**

#include "deck.h"

#include "randgen.h" //For random number generation

const int SIZE = 52;

Deck::Deck()

: myCards(SIZE),

myIndex(0) //initializer list

{

int rank;

int suit;

int num = 0;

for(rank=1; rank <= SIZE/4; rank++)

{

for (suit = Card::spades; suit <= Card::clubs; suit++)

{

myCards[num] = Card(rank,Card::Suit(suit));

num++;

}

}

}

void Deck::Shuffle()

{

myIndex = 0;

RandGen gen;

int k;

for(k=0; k < SIZE-1; k++)

{

int swapIndex = gen.RandInt(k,SIZE-1);

Card temp = myCards[swapIndex];

myCards[swapIndex] = myCards[k];

myCards[k] = temp;

}

}

Card Deck::GetCard()

{

Card c;

if (0 <= myIndex && myIndex < SIZE)

{

c = myCards[myIndex];

myIndex++;

}

else

{

c = Card(0,Card::spades); // make a joker

}

return c;

}

int Deck::Size() const

{

return SIZE - myIndex;

}

**//hand.h**

#ifndef \_HAND\_H

#define \_HAND\_H

#include "deck.h"

#include <vector>

using namespace std;

class Hand

{

private:

vector<Card> myCards;

public:

Hand();

void DealFrom(Deck& d);

void Print() const;

};

#endif

**//hand.cpp**

#include "hand.h"

const int SIZE = 5;

Hand::Hand()

: myCards(SIZE) //initializer list to call vector constructor

{

}

void Hand::DealFrom(Deck& d)

{

int k;

for(k=0; k < SIZE; k++)

{

myCards[k] = d.GetCard();

}

}

void Hand::Print() const

{

int k;

for(k=0; k < SIZE; k++)

{

cout << myCards[k] << endl;

}

}

**//randgen.h**

#ifndef \_RANDGEN\_H

#define \_RANDGEN\_H

#include <climits> // for INT\_MAX

using namespace std;

// designed for implementation-independent randomization

// if all system-dependent calls included in this class, then

// other classes can make use of this class in independent manner

// all random numbers are uniformly distributed in given range

//

// RandGen() --- constructor sets seed of random # generator

// once per program, not per class/object

//

// RandInt(int max)

// RandInt(int low,int max) - return random integer in range [0..max)

// when one parameter used, [low..max] when

// two parameters used

//

// examples: rnd.RandInt(6) is random integer [0..5] or [0..6)

// rnd.RandInt(3,10) is random integer [3..10]

// rnd.RandInt() is random integer [0..INT\_MAX)

//

// RandReal() -- returns random double in range [0..1)

// RandReal(double low, double max) -- random double in range [low..max)

class RandGen

{

public:

RandGen(); // set seed for all instances

int RandInt(int max = INT\_MAX); // returns int in [0..max)

int RandInt(int low, int max); // returns int in [low..max]

double RandReal(); // returns double in [0..1)

double RandReal(double low, double max); // range [low..max]

static void SetSeed(int seed); // static (per class) seed set

private:

static int ourInitialized; // for 'per-class' initialization

};

#endif

**//randgen.cpp**

#include <ctime> // for time()

#include <cstdlib> // for rand/srand

#include "randgen.h"

#include <cmath>

using namespace std;

int RandGen::ourInitialized = 0;

void RandGen::SetSeed(int seed)

// postcondition: system srand() used to initialize seed

// once per program (this is a static function)

{

if (0 == ourInitialized)

{ ourInitialized = 1; // only call srand once

srand(seed); // randomize

}

}

RandGen::RandGen()

// postcondition: system srand() used to initialize seed

// once per program

{

if (0 == ourInitialized)

{ ourInitialized = 1; // only call srand once

srand(unsigned(time(0))); // randomize

}

}

int RandGen::RandInt(int max)

// precondition: max > 0

// postcondition: returns int in [0..max)

{

return int(RandReal() \* max);

}

int RandGen::RandInt(int low, int max)

// precondition: low <= max

// postcondition: returns int in [low..max]

{

return low + RandInt(max-low+1);

}

double RandGen::RandReal()

// postcondition: returns double in [0..1)

{

return rand() / (double(RAND\_MAX) + 1);

}

double RandGen::RandReal(double low, double high)

{

double width = fabs(high-low);

double thelow = low < high ? low : high;

return RandReal()\*width + thelow;

}