**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**КЫРГЫЗСКОЙ РЕСПУБЛИКИ**

**КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМ И.РАЗЗАКОВА**

**ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ**

**КАФЕДРА ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ КОМПЬЮТЕРНЫХ СИСТЕМ**

**КУРСОВОЙ ПРОЕКТ**

ПО ДИСЦИПЛИНЕ:

**Функционально-ориентированное проектирование**

НА ТЕМУ: Функционально-ориентированное программирование

**ВЫПОЛНИЛ:** СТУДЕНТ ГР ПИ(англ)-1-16

АБАКИРОВ НУРСУЛТАН

**РУКОВОДИТЕЛЬ**: доц. МАКИЕВА З.Д.

**БИШКЕК 2016**

1. Условие задачи[[1]](#footnote-1).
2. Постановка задачи[[2]](#footnote-2).
3. Графическое представление алгоритма решения в виде блок-схемы (блок-схема для каждой функции, в том числе и для main).
4. Словесный (пошаговый алгоритм решения).
5. Программа на языке С++.
6. Тестовый пример: файл с исходными данными и файл с результатом.
7. Список литературы[[3]](#footnote-3).

Условие задачи

Программа генерирует два набора по 5 карт, определяет их комбинации и сравнивает их по правилам Покера.

Правила игры Покер

В покер играют стандартной колодой из 52 листов с равнозначными мастями. Туз может рассматриваться и как младшая карта для образования последовательности (стрит) до 5 включительно, и как старшая (в комбинации туз- король— дама — валет — 10). Победителем считается тот, чья комбинация из пяти карт окажется лучшей, или тот, кто сможет вытеснить из игры других игроков с помощью ставок или блеф-ставок и останется один до вскрытия карт.

**Роял-флэш**

* ***Роял-флэш*** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *royal flush* — «королевская масть»): не является отдельной комбинацией, а является частным случаем стрит-флэша, старшим из всех возможных, и состоит из 5 старших (туз, король, дама, валет, десять) карт одной масти, например: **Т♥ К♥ Д♥ В♥ 10♥**.

**Стрит-флэш**

* **Стрит-флэш** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *straight flush* — «масть по порядку»): любые пять карт одной масти по порядку, например: **9♠ 8♠ 7♠ 6♠ 5♠**. Туз может как начинать порядок, так и заканчивать его. Самый младший стрит-флэш (от туза до пятёрки) иногда именуют «стальное колесо».

**Каре**

* **Каре**/**Четвёрка**/**Покер** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *four of a kind, quads* — «четыре одинаковых»): четыре карты одного достоинства, например: **3♥ 3♦ 3♣ 3♠**.

**Фулл-хауз**

* **Фулл-хауз**/**Полный дом**/**Три плюс два** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *full house, full boat* — «полный дом», «полная лодка»): одна тройка и одна пара, например: **10♥ 10♦ 10♠ 8♣ 8♥**.

**Флэш**

* **Флэш** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *flush* — «масть»): пять карт одной масти, например: **К♠ В♠ 8♠ 4♠ 3♠**.

**Стрит**

* **Стрит** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *straight* — «порядок») : пять карт по порядку любых мастей, например: **5♦ 4♥ 3♠ 2♦ Т♦**. Туз может как начинать порядок, так и заканчивать его. В данном примере **Т♦** начинает комбинацию и его достоинство оценивается в единицу, а **5♦** считается старшей картой. Самый младший стрит (от туза до пятёрки) иногда именуют «колесо».

**Сет/Триплет/Трипс/Тройка**

* **Сет/Триплет/Трипс/Тройка** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *three of a kind, set* — «три одинаковых», «набор»): три карты одного достоинства, например: **7♣ 7♥ 7♠** .

**Две пары**

* **Две пары** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *two pairs*): две пары карт, например: **8♣ 8♠ 4♥ 4♣**.

**Одна пара**

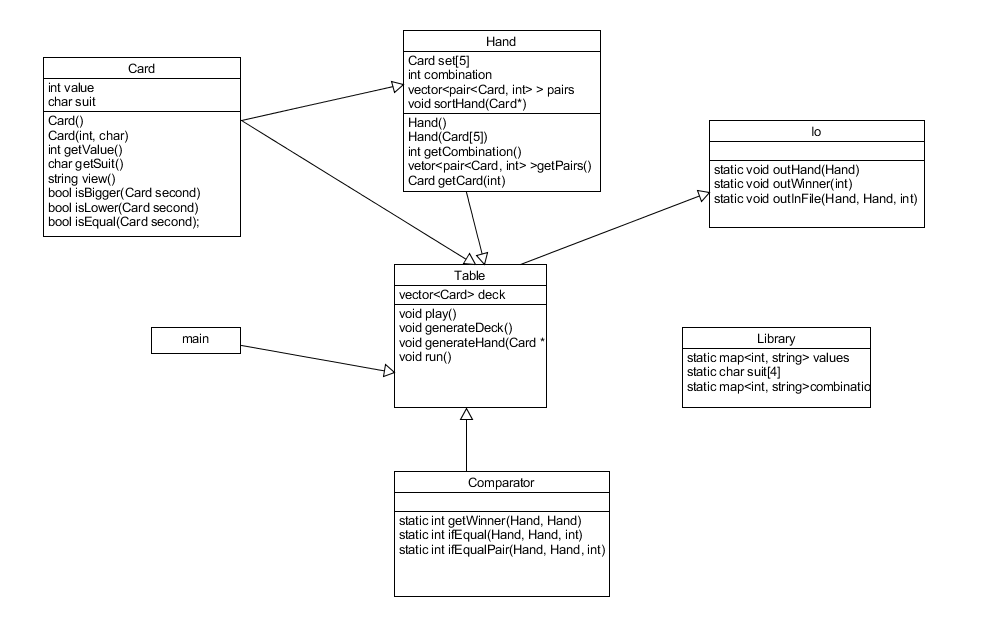
* **Одна пара** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *one pair*): две карты одного достоинства, например: **9♥ 9♠**.

**Старшая карта**

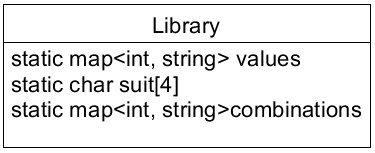
* **Старшая карта** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *high card*): ни одна из вышеописанных комбинаций, например (комбинация называется «старший туз»): **Т♦ 10♦ 9♠ 5♣ 4♣**.

При совпадении комбинаций более сильной является комбинация со старшими картами, например, **8♣ 8♠ 4♥ 4♣ 2♠** старше, чем **7♣ 7♠ 5♥ 5♣ K♠**, а комбинация **6♠ 5♦ 4♥ 3♠ 2♦** старше, чем **5♦ 4♥ 3♠ 2♦ Т♦**.

Постановка задачи

Классы, используемые в проекте:

В библиотеке мы храним данные о карте и о комбинациях. Каждое значение карты и комбинации имеют свой индекс.



**Library –** объект библиотека, доступная всем модулям программы, с

Параметрами:

**values** = 0 - "2" 1 - "3" **suit** = 'C', 'H', 'S', 'D'

2 - "4" 3 - "5" ('C'- **♣** 'H'**- ♥** 'S'**- ♠** 'D'**- ♦**)

4 - "6" 5 - "7"

6 - "8" 7 - "9"

8 - "10" 9 - "J"

10 - "Q" 11 - "K"

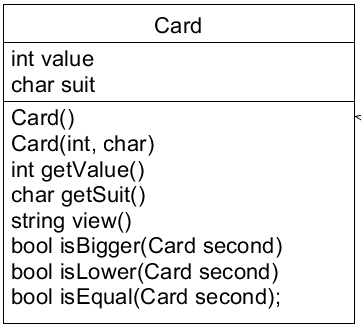
12 - "A"

**Combinations** = 0 - "HIGT CARD" 1 - "PAIR" 2 - "TWO PAIR"

3 - "THREE OF A KIND" 4 - "STRAIGHT" 5 - "FLUSH"

6 - "FULL HOUSE" 7 - "FOUR OF A KIND" 8 - "STRAIGHT FLUSH"

9 - "ROYAL FLUSH"

**Card** – это объект одной карты с

параметрами:

**value**(ключ)

**suit**(масть)

методами:

**Card(int, char)** – конструктор для заполнения объекта

**getValue()** – возвращает value объекта

**getSuit()** – возвращает suit объекта

**view()** – возвращает строку “value”(значение)+“suit”

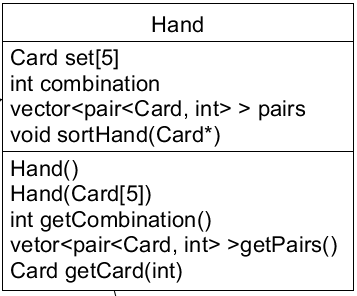
**isBigger(Card)** – возвращает true если ключ текущего объекта больше ключа объекта параметра, иначе false

**isLower(Card)** – возвращает true если ключ текущего объекта меньше

ключа объекта параметра, иначе false

**isEqual(Card)** – возвращает true если ключ текущего объекта равен

ключа объекта параметра, иначе false



**Hand** – это объект руки игрока с

параметрами:

**set[5]** – массив объектов Card

**combination** - ключ комбинации

**pairs** - массив из пар (карта, кол-во повторений этой карты), для определения комбинации

методами:

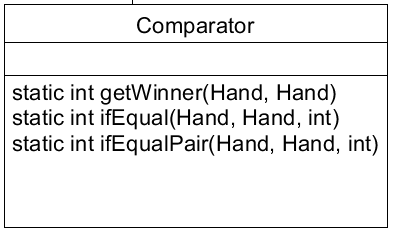
**Hand(Card[5])** – конструктор для заполнения объекта

**getCombination()** – возвращает combination объекта

**getPair()** – возвращает массив pairs объекта

**sortHand(Card\*)** – сортирует массив объектов Card по ключу параметра value

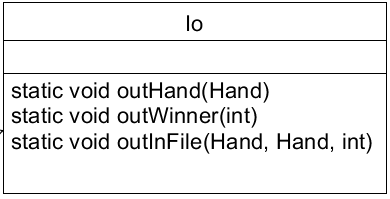
**getCard(int)** - возвращает объект Card из массива set по параметру(индекс)



**Comparator** – это объект сравнения со статическими

методами:

**getWinner(Hand, Hand)** – возвращает 1, если комбинация первого параметра выше комбинации второго параметра объектов Hand. Если же комбинация первого параметра ниже комбинации второго параметра возвращает 2. Если комбинация первого параметра равна комбинации второго параметра возвращает 0.



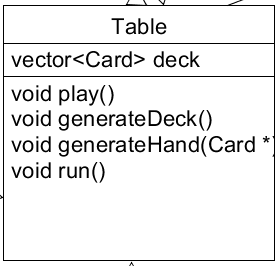
**Io** – это объект вывода со статическими

методами:

**outHand(Hand)** – выводит на экран карты и комбинацию Hand

**outWinner(int)** – выводит победителя. Если параметр равен 1, то победил первый игрок. Если 2, то второй. Если 0, то ничья.

**outInFile(Hand, Hand, int)** – записывает карты игроков, их комбинации и победителя в файл.



**Table** – это объект стола с

параметром:

**deck** – колода, массив объектов Card

методами:

**play()** – связывающий элемент, создает колоду, игроков, раздает карты, сравнивает их, выводит победителя.

**generateDeck()** – создает колоду из 52 карт

**generateHand(Card\*)** – берет 5 случайных карт с колоды и записывает их в параметр Card

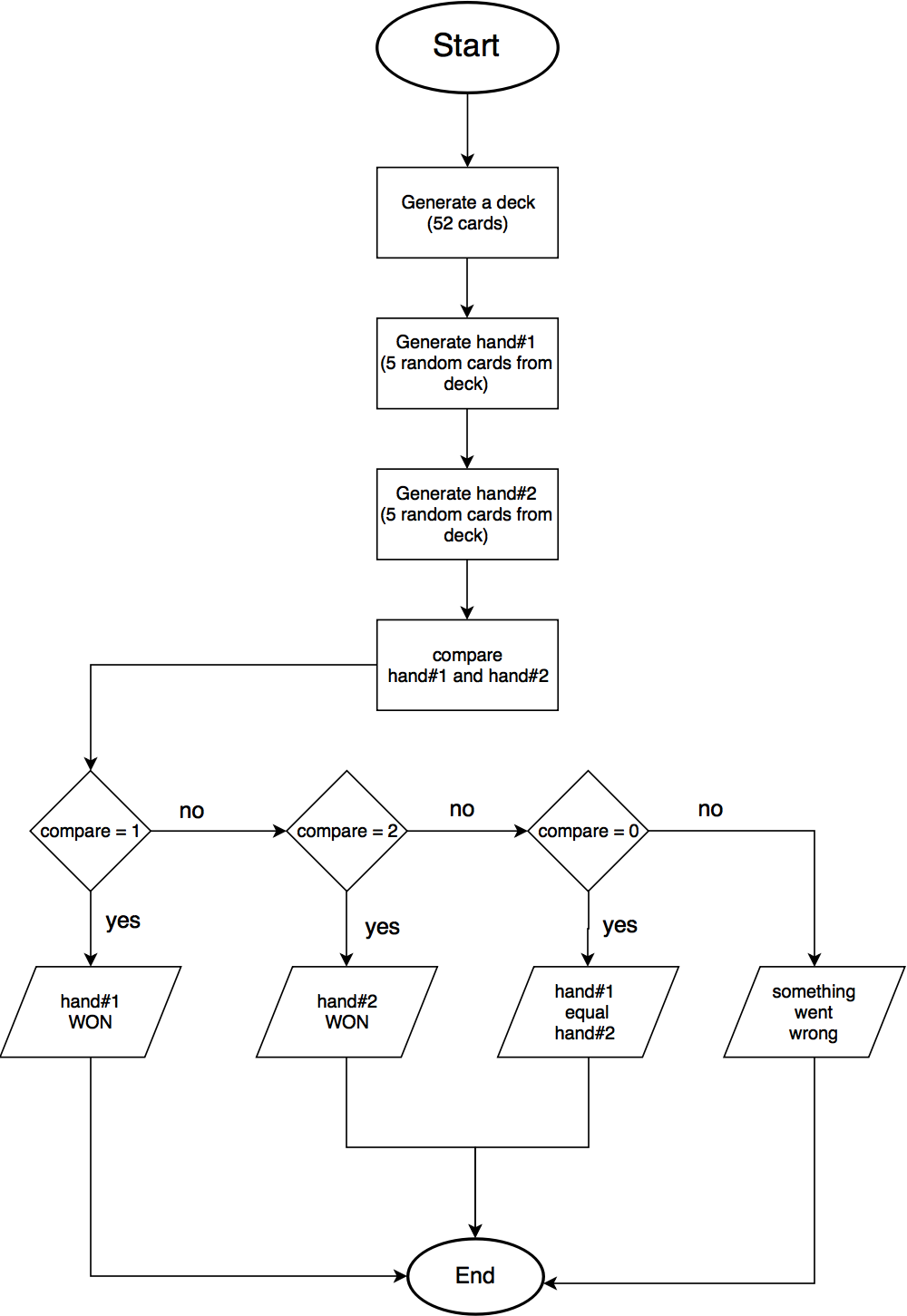
**run()** – дает пользователю выбор: 1 – пройти тесты

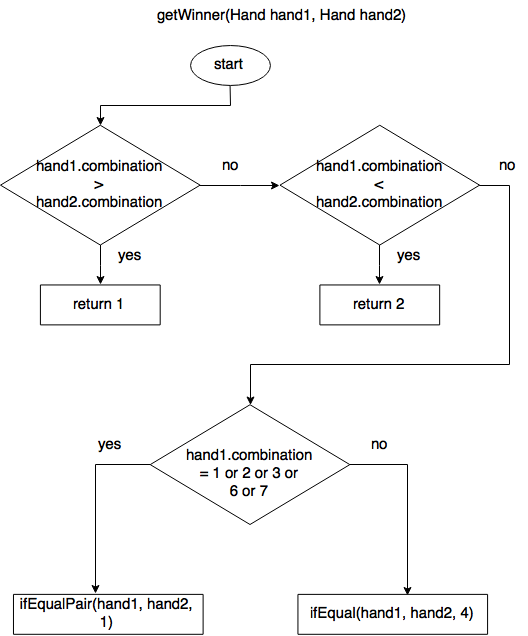
2 – начать игру

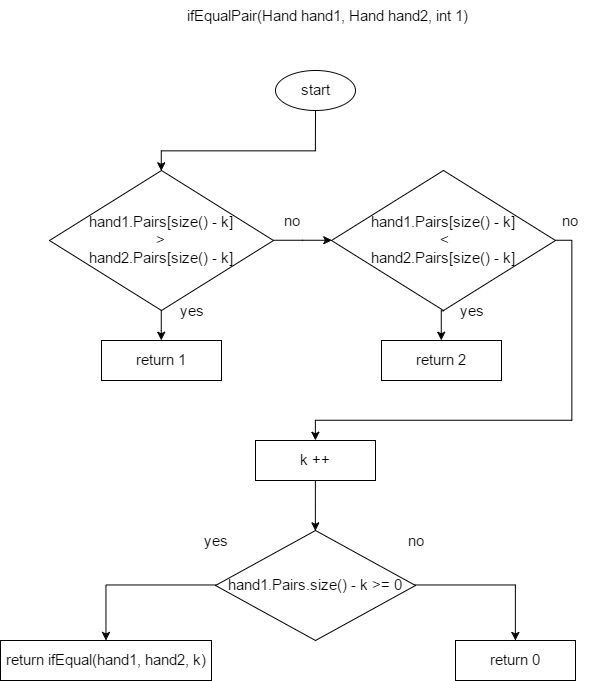
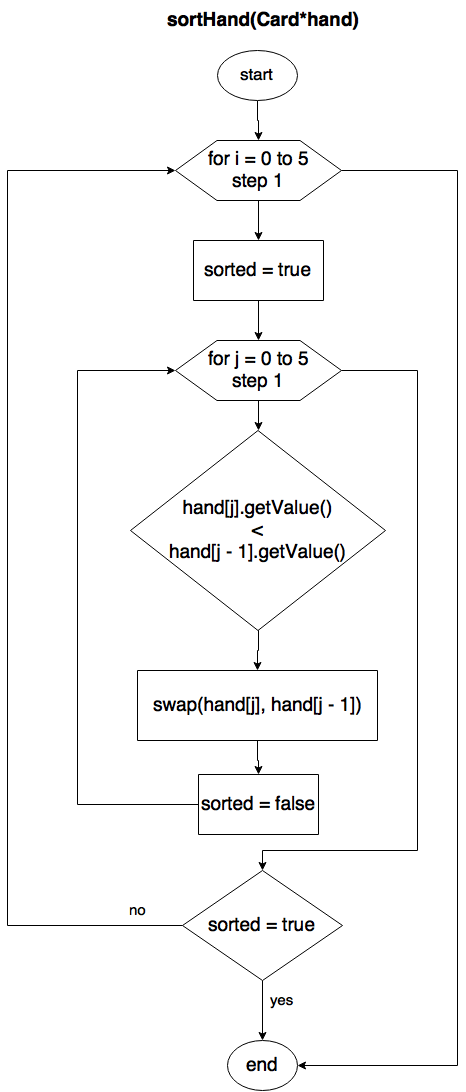
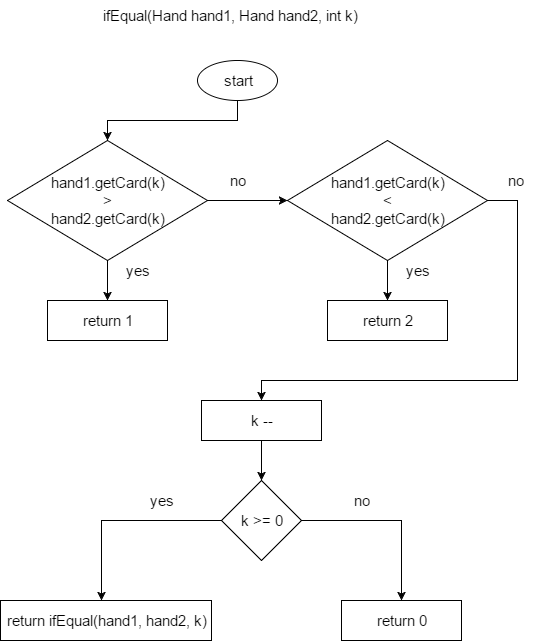
Словесный (пошаговый алгоритм решения):

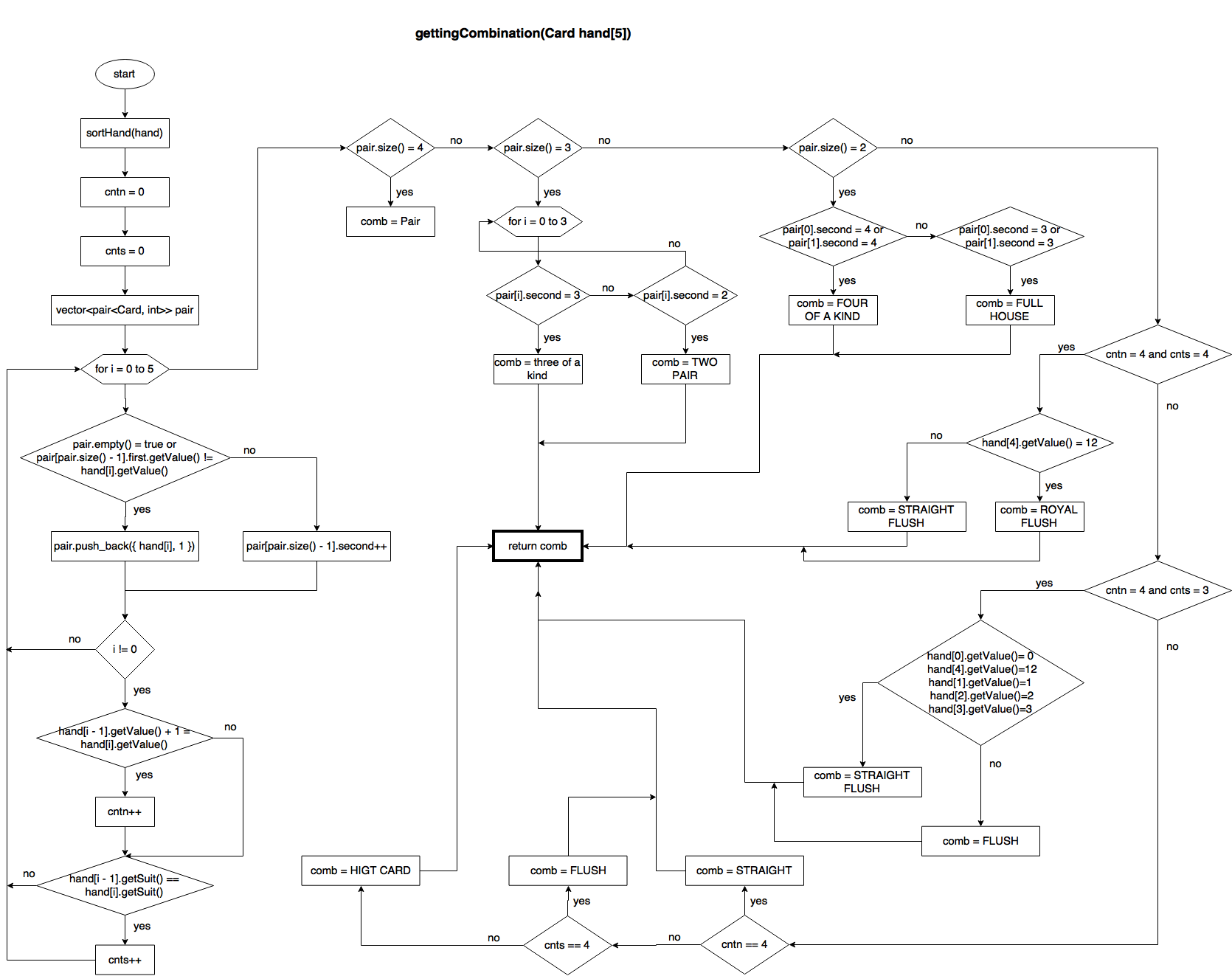
1. Создать колоду из 52 карт.
2. Раздать двум игрокам по пять случайных карт из колоды.
3. Сравнить карты двух игроков по правилам игры «Покер».
4. Вывести карты двух игроков и победителя или ничью.

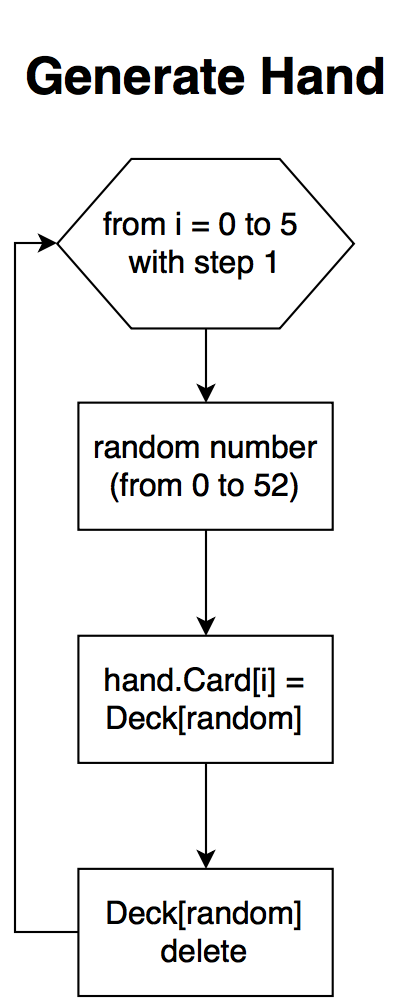
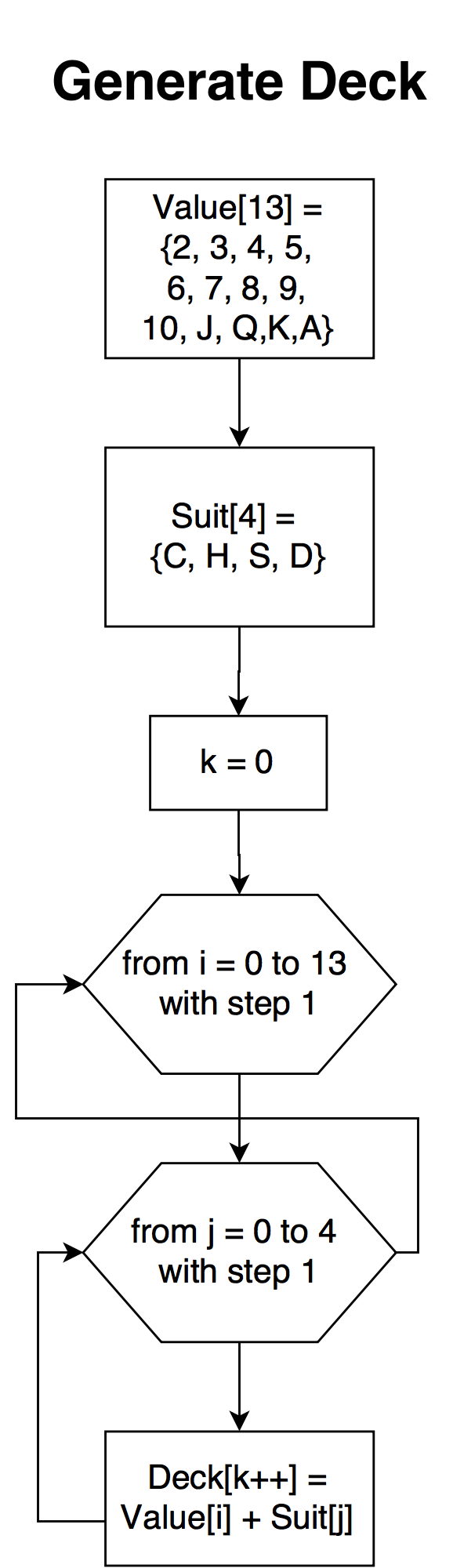
Графическое представление алгоритма решения в виде блок-схемы

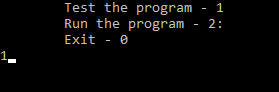


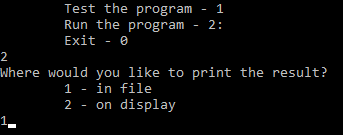


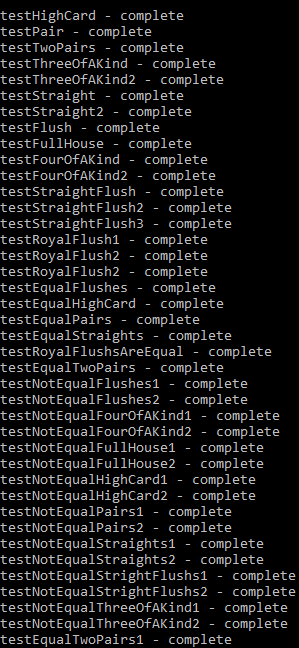




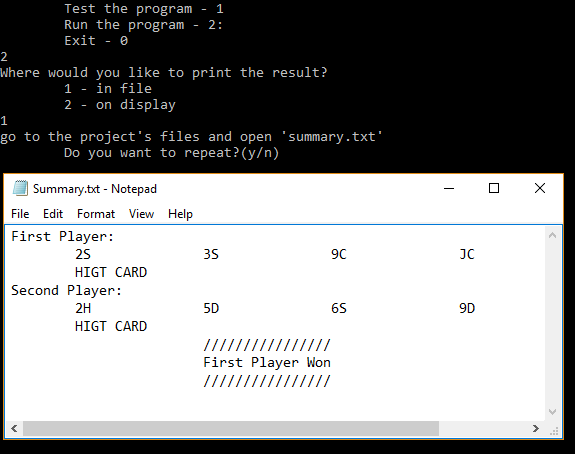


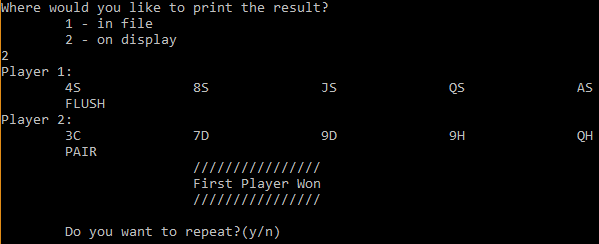
****

****

****

****

****

****

**Main.cpp**

#include "Table.h"

int main()

{

Table game;

game.run();

}

**Table.h**

#pragma once

#include <vector>

#include <ctime>

#include <stdlib.h>

#include <iostream>

#include <conio.h>

#include <string>

#include "Card.h"

#include "Hand.h"

#include "Library.h"

#include "Comparator.h"

#include "Io.h"

#include "test.h"

class Table

{

private:

std::vector<Card> deck;

void play();

public:

void generateDeck();

void genetareHand(Card \*);

void run();

};

**Table.cpp**

#include "Table.h"

void Table::generateDeck()

{

for (int i = 0; i < 13; i++)

{

for (int j = 0; j < 4; j++)

{

Table::deck.push\_back(Card(i, Library::suit[j]));

}

}

}

void Table::genetareHand(Card \*cards)

{

srand(time(NULL));

for (int i = 0; i < 5; i++)

{

int randNum = rand() % Table::deck.size();

cards[i] = Table::deck[randNum];

std::swap(Table::deck[randNum], Table::deck[Table::deck.size() - 1]);

Table::deck.pop\_back();

}

}

void Table::play()

{

std::string n;

while (n != "n")

{

Table game;

game.generateDeck();

Card cards[5];

game.genetareHand(cards);

Hand hand1(cards);

game.genetareHand(cards);

Hand hand2(cards);

int compare = Comparator::getWinner(hand1, hand2);

int chosen;

std::cout << "Where would you like to print the result?\n\t1 - in file\n\t2 - on display\n";

std::cin >> chosen;

if (chosen == 2)

{

std::cout << "Player 1:";

Io::outHand(hand1);

std::cout << "Player 2:";

Io::outHand(hand2);

Io::outWinner(compare);

}

else if (chosen == 1)

{

Io::outInFile(hand1, hand2, compare);

}

std::cout << "\n\tDo you want to repeat?(y/n)\n";

std::cin >> n;

while (n != "y")

{

if (n == "n")

break;

std::cout << "\n\t'y' - yes\n\t'n' - no\n";

std::cin >> n;

}

}

}

void Table::run()

{

int n = 9;

std::cout << "\tTest the program - 1\n\tRun the program - 2:\n\tExit - 0\n";

std::cin >> n;

if (n == 1)

{

test::combTest();

test::comparatorTest();

system("pause");

}

else if (n == 2)

{

Table::play();

}

}

**Card.h**

#pragma once

#include <string>

#include "Library.h"

class Card

{

private:

int value;

char suit;

public:

Card();

Card(int, char);

int getValue();

char getSuit();

std::string view();

bool isBigger(Card);

bool isLower(Card);

bool isEqual(Card);

};

**Card.cpp**

#include "Card.h"

Card::Card() {};

Card::Card(int value, char suit)

{

Card::value = value;

Card::suit = suit;

}

int Card::getValue()

{

return value;

}

char Card::getSuit()

{

return suit;

}

std::string Card::view()

{

std::string str = Library::values.at(value) + suit;

return str;

}

bool Card::isBigger(Card second)

{

return value > second.value;

}

bool Card::isLower(Card second)

{

return value < second.value;

}

bool Card::isEqual(Card second)

{

return value == second.value;

}

**Comparator.h**

#pragma once

#include "Hand.h"

class Comparator

{

public:

static int getWinner(Hand, Hand);

static int ifEqual(Hand, Hand, int);

static int ifEqualPair(Hand, Hand, int);

};

**Comparator.cpp**

#include "Comparator.h"

int Comparator::getWinner(Hand hand1, Hand hand2)

{

if (hand1.getCombination() > hand2.getCombination())

{

return 1;

}

else if (hand1.getCombination() < hand2.getCombination())

{

return 2;

}

else

{

//If hand1 equals to hand2 by pair

if (hand1.getCombination() == 2 || hand1.getCombination() == 1 ||

hand1.getCombination() == 3 || hand1.getCombination() == 6 ||

hand1.getCombination() == 7)

{

return ifEqualPair(hand1, hand2, 1);

}

else

{

return ifEqual(hand1, hand2, 4);

}

}

}

int Comparator::ifEqual(Hand hand1, Hand hand2, int k) {

if (hand1.getCard(k).isBigger(hand2.getCard(k)))

{

return 1;

}

else if (hand1.getCard(k).isLower(hand2.getCard(k)))

{

return 2;

}

else if (hand1.getCard(k).isEqual(hand2.getCard(k)))

{

k--;

if (k >= 0)

{

return ifEqual(hand1, hand2, k);

}

else

{

return 0;

}

}

}

int Comparator::ifEqualPair(Hand hand1, Hand hand2, int k)

{

if (hand1.getPairs()[hand1.getPairs().size() -k].first.isBigger(hand2.getPairs()[hand2.getPairs().size() - k].first))

{

return 1;

}

else if (hand1.getPairs()[hand1.getPairs().size() - k].first.isLower(hand2.getPairs()[hand2.getPairs().size() - k].first))

{

return 2;

}

else if (hand1.getPairs()[hand1.getPairs().size() - k].first.isEqual(hand2.getPairs()[hand2.getPairs().size() - k].first))

{

k++;

if (hand1.getPairs().size() - k >= 0)

{

return ifEqual(hand1, hand2, k);

}

else

{

return 0;

}

}

}

**Hand.h**

#pragma once

#include "Card.h"

#include <vector>

class Hand

{

private:

Card set[5];

int combination;

std::vector<std::pair<Card, int> > pairs;

void sortHand(Card\*);

public:

Hand();

Hand(Card[5]);

int getCombination();

std::vector<std::pair<Card, int> > getPairs();

Card getCard(int);

};

**Hand.cpp**

#include "Hand.h"

Hand::Hand() {};

void Hand::sortHand(Card \*hand)

{

for (int i = 0; i < 5; i++)

{

bool sorted = true;

for (int j = 1; j < 5; j++)

{

if (hand[j].getValue() < hand[j - 1].getValue())

{

std::swap(hand[j], hand[j - 1]);

sorted = false;

}

}

if (sorted)

{

break;

}

}

}

Hand::Hand(Card hand[5])

{

//sorting the hand

sortHand(hand);

for (int i = 0; i < 5; i++)

{

Hand::set[i] = hand[i];

}

std::vector<std::pair<Card, int> > pair;

//Geting combination

int comb;

int cntn = 0;

int cnts = 0;

for (int i = 0; i < 5; i++)

{

if (pair.empty() || pair[pair.size() - 1].first.getValue() != hand[i].getValue())

{

pair.push\_back({ hand[i], 1 });

}

else

{

pair[pair.size() - 1].second++;

}

if (i != 0)

{

if (hand[i - 1].getValue() + 1 == hand[i].getValue())

{

cntn++;

}

if (hand[i - 1].getSuit() == hand[i].getSuit())

{

cnts++;

}

}

}

Hand::pairs = pair;

if (pair.size() == 4)

{

//Pair

comb = 1;

}

else if (pair.size() == 3)

{

for (int i = 0; i < 3; i++)

{

if (pair[i].second == 3)

{

//three of a kind

comb = 3;

break;

}

else if (pair[i].second == 2)

{

//TWO PAIR

comb = 2;

break;

}

}

}

else if (pair.size() == 2)

{

if (pair[0].second == 4 || pair[1].second == 4)

{

//FOUR OF A KIND

comb = 7;

}

else if (pair[0].second == 3 || pair[1].second == 3)

{

//FULL HOUSE

comb = 6;

}

}

else

{

if (cntn == 4 && cnts == 4)

{

if (hand[4].getValue() == 12)

{

//ROYAL FLUSH

comb = 9;

}

else

{

//STRAIGHT FLUSH

comb = 8;

}

}

else if (cnts == 4 && cntn == 3)

{

if (hand[0].getValue() == 0 && hand[4].getValue() == 12 && hand[1].getValue() == 1 && hand[2].getValue() == 2 && hand[3].getValue() == 3)

{

//STRAIGHT FLUSH

comb = 8;

}

else

{

//FLUSH

comb = 5;

}

}

else if (cntn == 4)

{

//STRAIGHT

comb = 4;

}

else if (cnts == 4)

{

//FLUSH

comb = 5;

}

else

{

//HIGT CARD

comb = 0;

}

}

Hand::combination = comb;

}

int Hand::getCombination()

{

return Hand::combination;

}

std::vector<std::pair<Card, int>> Hand::getPairs()

{

return Hand::pairs;

}

Card Hand::getCard(int index)

{

return Hand::set[index];

}

**Io.h**

#pragma once

#include "Hand.h"

#include <iostream>

#include "Library.h"

#include <fstream>

#include <stdlib.h>

class Io

{

public:

static void outHand(Hand);

static void outWinner(int);

static void outInFile(Hand, Hand, int);

};

**Io.cpp**

#include "Io.h"

void Io::outHand(Hand hand)

{

std::cout << std::endl;

for (int i = 0; i < 5; i++)

{

std::cout << "\t" << hand.getCard(i).view() << "\t";

}

std::cout << std::endl << "\t" <<Library::combination.at(hand.getCombination()) << std::endl;

}

void Io::outWinner(int compare)

{

if (compare == 1)

{

std::cout << "\t\t\t//////////\n\t\t\tFirst Player Won\n\t\t\t//////////" << std::endl;

}

else if (compare == 2)

{

std::cout << "\t\t\t//////////\n\t\t\tSecond Player Won\n\t\t\t//////////" << std::endl;

}

else if (compare == 0)

{

std::cout << "\t\t\t//////////\n\t\t\tEqual\n\t\t\t//////////" << std::endl;

}

else

{

std::cout << "Something Went Wrong" << std::endl;

}

}

void Io::outInFile(Hand hand1, Hand hand2, int compare)

{

std::ofstream out;

out.open("Summary.txt", \_IOS\_Nocreate);

if (out.fail())

{

std::cout << std::endl << "File does not exist! Create the file 'Summary.txt' in the project's files and repeat\n";

}

else

{

out << "First Player: \n";

for (int i = 0; i < 5; i++)

{

out << "\t" << hand1.getCard(i).view() << "\t";

}

out << std::endl << "\t" << Library::combination.at(hand1.getCombination()) << std::endl;

out << "Second Player: \n";

for (int i = 0; i < 5; i++)

{

out << "\t" << hand2.getCard(i).view() << "\t";

}

out << std::endl << "\t" << Library::combination.at(hand2.getCombination()) << std::endl;

if (compare == 1)

{

out << "First Player Won" << std::endl;

}

else if (compare == 2)

{

out << "Second Player Won" << std::endl;

}

else if (compare == 0)

{

out << "Equal" << std::endl;

}

else

{

out << "Something Went Wrong" << std::endl;

}

std::cout << "go to the project's files and open 'summary.txt'";

}

}

**Library.h**

#pragma once

#include <string>

#include <map>

class Library

{

public:

static std::map<int, std::string> values;

static char suit[4];

static std::map<int, std::string> combination;

};

**Library.cpp**

#include "Library.h"

std::map<int, std::string> Library::values = {

{ 0, "2" },

{ 1, "3" },

{ 2, "4" },

{ 3, "5" },

{ 4, "6" },

{ 5, "7" },

{ 6, "8" },

{ 7, "9" },

{ 8, "10" },

{ 9, "J" },

{ 10, "Q" },

{ 11, "K" },

{ 12, "A" }

};

char Library::suit[4] = { 'C', 'H', 'S', 'D' };

std::map<int, std::string> Library::combination = {

{ 0, "HIGT CARD" },

{ 1, "PAIR" },

{ 2, "TWO PAIR" },

{ 3, "THREE OF A KIND" },

{ 4, "STRAIGHT" },

{ 5, "FLUSH" },

{ 6, "FULL HOUSE" },

{ 7, "FOUR OF A KIND" },

{ 8, "STRAIGHT FLUSH" },

{ 9, "ROYAL FLUSH" }

};

**Test.h**

#pragma once

#include "HandEvaluatorTest.h"

#include "TestComparator.h"

class test

{

public:

static void combTest();

static void comparatorTest();

};

**Test.cpp**

#include "test.h"

void test::combTest()

{

HandEvaluatorTest test1;

test1.testHighCard();

test1.testPair();

test1.testTwoPairs();

test1.testThreeOfAKind();

test1.testThreeOfAKind2();

test1.testStraight();

test1.testStraight2();

test1.testFlush();

test1.testFullHouse();

test1.testFourOfAKind();

test1.testFourOfAKind2();

test1.testStraightFlush();

test1.testStraightFlush2();

test1.testStraightFlush3();

test1.testRoyalFlush();

test1.testRoyalFlush2();

test1.testRoyalFlush3();

}

void test::comparatorTest()

{

TestComparator test;

test.testEqualFlushes();

test.testEqualHighCard();

test.testEqualPairs();

test.testEqualStraights();

test.testEqualStrightFlushs();

test.testEqualTwoPairs();

test.testNotEqualFlushes();

test.testNotEqualFourOfAKind();

test.testNotEqualFullHouse();

test.testNotEqualHighCard();

test.testNotEqualPairs();

test.testNotEqualStraights();

test.testNotEqualStrightFlushs();

test.testNotEqualThreeOfAKind();

test.testNotEqualTwoPairs();

test.testRoyalFlushsAreEqual();

}

**HandEvaluatorTest.h**

#pragma once

#include "Hand.h"

#include "Card.h"

#include "Library.h"

#include <iostream>

class HandEvaluatorTest

{

public:

void testRoyalFlush();

void testRoyalFlush2();

void testRoyalFlush3();

void testStraightFlush();

void testStraightFlush2();

void testStraightFlush3();

void testFourOfAKind();

void testFourOfAKind2();

void testFullHouse();

void testFlush();

void testStraight();

void testStraight2();

void testThreeOfAKind();

void testThreeOfAKind2();

void testTwoPairs();

void testPair();

void testHighCard();

};

**HandEvaluatorTest.cpp**

#include "HandEvaluatorTest.h"

void HandEvaluatorTest::testRoyalFlush()

{

Hand hand;

Card cards[5];

cards[0] = Card(12, 'C'); //A

cards[1] = Card(11, 'C'); //K

cards[2] = Card(10, 'C'); //Q

cards[3] = Card(9, 'C'); //J

cards[4] = Card(8, 'C'); //10

hand = Hand(cards);

std::string k = Library::combination.at(hand.getCombination());

if (k == "ROYAL FLUSH")

{

std::cout << "testRoyalFlush1 \t complete\n";

}

else

{

std::cout << "testRoyalFlush1 \t failed\n";

}

}

void HandEvaluatorTest::testRoyalFlush2()

{

Hand hand;

Card cards[5];

cards[0] = Card(12, 'C'); //A

cards[1] = Card(2, 'C'); //4

cards[2] = Card(10, 'C'); //Q

cards[3] = Card(9, 'C'); //J

cards[4] = Card(8, 'C'); //10

hand = Hand(cards);

std::string k = Library::combination.at(hand.getCombination());

if (k != "ROYAL FLUSH")

{

std::cout << "testRoyalFlush2 \t complete\n";

}

else

{

std::cout << "testRoyalFlush2 \t failed\n";

}

}

void HandEvaluatorTest::testRoyalFlush3()

{

Hand hand;

Card cards[5];

cards[0] = Card(12, 'S'); //A

cards[1] = Card(11, 'S'); //K

cards[2] = Card(10, 'S'); //Q

cards[3] = Card(9, 'S'); //J

cards[4] = Card(8, 'S'); //10

hand = Hand(cards);

std::string k = Library::combination.at(hand.getCombination());

if (k == "ROYAL FLUSH")

{

std::cout << "testRoyalFlush2 \t complete\n";

}

else

{

std::cout << "testRoyalFlush2 \t failed\n";

}

}

void HandEvaluatorTest::testStraightFlush()

{

Hand hand;

Card cards[5];

cards[0] = Card(12, 'C'); //A

cards[1] = Card(0, 'C'); //2

cards[2] = Card(1, 'C'); //3

cards[3] = Card(2, 'C'); //4

cards[4] = Card(3, 'C'); //5

hand = Hand(cards);

std::string k = Library::combination.at(hand.getCombination());

if (k == "STRAIGHT FLUSH")

{

std::cout << "testStraightFlush \t complete\n";

}

else

{

std::cout << "testStraightFlush \t failed\n";

}

}

void HandEvaluatorTest::testStraightFlush2()

{

Hand hand;

Card cards[5];

cards[0] = Card(7, 'D'); //9

cards[1] = Card(11, 'D'); //K

cards[2] = Card(10, 'D'); //Q

cards[3] = Card(9, 'D'); //J

cards[4] = Card(8, 'D'); //10

hand = Hand(cards);

std::string k = Library::combination.at(hand.getCombination());

if (k == "STRAIGHT FLUSH")

{

std::cout << "testStraightFlush2 \t complete\n";

}

else

{

std::cout << "testStraightFlush2 \t failed\n";

}

}

void HandEvaluatorTest::testStraightFlush3()

{

Hand hand;

Card cards[5];

cards[0] = Card(7, 'D'); //9

cards[1] = Card(11, 'D'); //K

cards[2] = Card(10, 'D'); //Q

cards[3] = Card(12, 'D'); //A

cards[4] = Card(8, 'D'); //10

hand = Hand(cards);

std::string k = Library::combination.at(hand.getCombination());

if (k != "STRAIGHT FLUSH")

{

std::cout << "testStraightFlush3 \t complete\n";

}

else

{

std::cout << "testStraightFlush3 \t failed\n";

}

}

void HandEvaluatorTest::testFourOfAKind()

{

Hand hand;

Card cards[5];

cards[0] = Card(7, 'D'); //9

cards[1] = Card(7, 'C'); //9

cards[2] = Card(7, 'S'); //9

cards[3] = Card(7, 'H'); //9

cards[4] = Card(8, 'D'); //10

hand = Hand(cards);

std::string k = Library::combination.at(hand.getCombination());

if (k == "FOUR OF A KIND")

{

std::cout << "testFourOfAKind \t complete\n";

}

else

{

std::cout << "testFourOfAKind \t failed\n";

}

}

void HandEvaluatorTest::testFourOfAKind2()

{

Hand hand;

Card cards[5];

cards[0] = Card(7, 'D'); //9

cards[1] = Card(11, 'C'); //K

cards[2] = Card(7, 'S'); //9

cards[3] = Card(7, 'H'); //9

cards[4] = Card(8, 'D'); //10

hand = Hand(cards);

std::string k = Library::combination.at(hand.getCombination());

if (k != "FOUR OF A KIND")

{

std::cout << "testFourOfAKind2 \t complete\n";

}

else

{

std::cout << "testFourOfAKind2 \t failed\n";

}

}

void HandEvaluatorTest::testFullHouse()

{

Hand hand;

Card cards[5];

cards[0] = Card(7, 'D'); //9

cards[1] = Card(9, 'C'); //J

cards[2] = Card(7, 'S'); //9

cards[3] = Card(9, 'H'); //J

cards[4] = Card(9, 'D'); //J

hand = Hand(cards);

std::string k = Library::combination.at(hand.getCombination());

if (k == "FULL HOUSE")

{

std::cout << "testFullHouse \t complete\n";

}

else

{

std::cout << "testFullHouse \t failed\n";

}

}

void HandEvaluatorTest::testFlush()

{

Hand hand;

Card cards[5];

cards[0] = Card(7, 'H'); //9

cards[1] = Card(12, 'H'); //A

cards[2] = Card(5, 'H'); //7

cards[3] = Card(9, 'H'); //J

cards[4] = Card(3, 'H'); //5

hand = Hand(cards);

std::string k = Library::combination.at(hand.getCombination());

if (k == "FLUSH")

{

std::cout << "testFlush \t complete\n";

}

else

{

std::cout << "testFlush \t failed\n";

}

}

void HandEvaluatorTest::testStraight()

{

Hand hand;

Card cards[5];

cards[0] = Card(8, 'H'); //10

cards[1] = Card(9, 'C'); //J

cards[2] = Card(10, 'S'); //Q

cards[3] = Card(11, 'D'); //K

cards[4] = Card(7, 'H'); //9

hand = Hand(cards);

std::string k = Library::combination.at(hand.getCombination());

if (k == "STRAIGHT")

{

std::cout << "testStraight \t complete\n";

}

else

{

std::cout << "testStraight \t failed\n";

}

}

void HandEvaluatorTest::testStraight2()

{

Hand hand;

Card cards[5];

cards[0] = Card(8, 'H'); //10

cards[1] = Card(9, 'C'); //J

cards[2] = Card(1, 'S'); //3

cards[3] = Card(11, 'D'); //K

cards[4] = Card(7, 'H'); //9

hand = Hand(cards);

std::string k = Library::combination.at(hand.getCombination());

if (k != "STRAIGHT")

{

std::cout << "testStraight2 \t complete\n";

}

else

{

std::cout << "testStraight2 \t failed\n";

}

}

void HandEvaluatorTest::testThreeOfAKind()

{

Hand hand;

Card cards[5];

cards[0] = Card(8, 'H'); //10

cards[1] = Card(9, 'C'); //J

cards[2] = Card(10, 'S'); //Q

cards[3] = Card(9, 'D'); //J

cards[4] = Card(9, 'H'); //J

hand = Hand(cards);

std::string k = Library::combination.at(hand.getCombination());

if (k == "THREE OF A KIND")

{

std::cout << "testThreeOfAKind \t complete\n";

}

else

{

std::cout << "testThreeOfAKind \t failed\n";

}

}

void HandEvaluatorTest::testThreeOfAKind2()

{

Hand hand;

Card cards[5];

cards[0] = Card(8, 'H'); //10

cards[1] = Card(8, 'C'); //10

cards[2] = Card(10, 'S'); //Q

cards[3] = Card(9, 'D'); //J

cards[4] = Card(9, 'H'); //J

hand = Hand(cards);

std::string k = Library::combination.at(hand.getCombination());

if (k != "THREE OF A KIND")

{

std::cout << "testThreeOfAKind2 \t complete\n";

}

else

{

std::cout << "testThreeOfAKind2 \t failed\n";

}

}

void HandEvaluatorTest::testTwoPairs()

{

Hand hand;

Card cards[5];

cards[0] = Card(8, 'H'); //10

cards[1] = Card(8, 'C'); //10

cards[2] = Card(10, 'S'); //Q

cards[3] = Card(9, 'D'); //J

cards[4] = Card(9, 'H'); //J

hand = Hand(cards);

std::string k = Library::combination.at(hand.getCombination());

if (k == "TWO PAIR")

{

std::cout << "testTwoPairs \t complete\n";

}

else

{

std::cout << "testTwoPairs \t failed\n";

}

}

void HandEvaluatorTest::testPair()

{

Hand hand;

Card cards[5];

cards[0] = Card(8, 'H'); //10

cards[1] = Card(7, 'C'); //9

cards[2] = Card(10, 'S'); //Q

cards[3] = Card(9, 'D'); //J

cards[4] = Card(9, 'H'); //J

hand = Hand(cards);

std::string k = Library::combination.at(hand.getCombination());

if (k == "PAIR")

{

std::cout << "testPair \t complete\n";

}

else

{

std::cout << "testPair \t failed\n";

}

}

void HandEvaluatorTest::testHighCard()

{

Hand hand;

Card cards[5];

cards[0] = Card(8, 'H'); //10

cards[1] = Card(7, 'C'); //9

cards[2] = Card(10, 'S'); //Q

cards[3] = Card(9, 'D'); //J

cards[4] = Card(0, 'H'); //2

hand = Hand(cards);

std::string k = Library::combination.at(hand.getCombination());

if (k == "HIGT CARD")

{

std::cout << "testHighCard \t complete\n";

}

else

{

std::cout << "testHighCard \t failed\n";

}

}

**TestComparator.h**

#pragma once

#include "Hand.h"

#include "Card.h"

#include "Comparator.h"

#include <iostream>

class TestComparator

{

public:

void testRoyalFlushsAreEqual();

void testEqualStrightFlushs();

void testNotEqualStrightFlushs();

void testNotEqualFourOfAKind();

void testNotEqualFullHouse();

void testEqualFlushes();

void testNotEqualFlushes();

void testEqualStraights();

void testNotEqualStraights();

void testNotEqualThreeOfAKind();

void testEqualTwoPairs();

void testNotEqualTwoPairs();

void testEqualPairs();

void testNotEqualPairs();

void testEqualHighCard();

void testNotEqualHighCard();

};

**TestComparator.cpp**

#include "TestComparator.h"

void TestComparator::testRoyalFlushsAreEqual()

{

Hand hand1;

Card cards1[5];

cards1[0] = Card(8, 'C'); //10

cards1[1] = Card(9, 'C'); //J

cards1[2] = Card(10, 'C'); //Q

cards1[3] = Card(11, 'C'); //K

cards1[4] = Card(12, 'C'); //A

hand1 = Hand(cards1);

Hand hand2;

Card cards2[5];

cards2[0] = Card(12, 'H'); //A

cards2[1] = Card(11, 'H'); //K

cards2[2] = Card(10, 'H'); //Q

cards2[3] = Card(9, 'H'); //J

cards2[4] = Card(8, 'H'); //10

hand2 = Hand(cards2);

if (Comparator::getWinner(hand1, hand2) == 0)

{

std::cout << "testRoyalFlushsAreEqual \t complete\n";

}

else

{

std::cout << "testRoyalFlushsAreEqual \t failed\n";

}

}

void TestComparator::testEqualStrightFlushs()

{

Hand hand1;

Card cards1[5];

cards1[0] = Card(12, 'C'); //A

cards1[1] = Card(0, 'C'); //2

cards1[2] = Card(1, 'C'); //3

cards1[3] = Card(2, 'C'); //4

cards1[4] = Card(3, 'C'); //5

hand1 = Hand(cards1);

Hand hand2;

Card cards2[5];

cards2[0] = Card(12, 'S'); //A

cards2[1] = Card(0, 'S'); //2

cards2[2] = Card(1, 'S'); //3

cards2[3] = Card(2, 'S'); //4

cards2[4] = Card(3, 'S'); //5

hand2 = Hand(cards2);

if (Comparator::getWinner(hand1, hand2) == 0)

{

std::cout << "testRoyalFlushsAreEqual \t complete\n";

}

else

{

std::cout << "testRoyalFlushsAreEqual \t failed\n";

}

}

void TestComparator::testNotEqualStrightFlushs()

{

Hand hand1;

Card cards1[5];

cards1[0] = Card(12, 'C'); //A

cards1[1] = Card(0, 'C'); //2

cards1[2] = Card(1, 'C'); //3

cards1[3] = Card(2, 'C'); //4

cards1[4] = Card(3, 'C'); //5

hand1 = Hand(cards1);

Hand hand2;

Card cards2[5];

cards2[0] = Card(4, 'H'); //6

cards2[1] = Card(0, 'H'); //2

cards2[2] = Card(1, 'H'); //3

cards2[3] = Card(2, 'H'); //4

cards2[4] = Card(3, 'H'); //5

hand2 = Hand(cards2);

if (Comparator::getWinner(hand1, hand2) == 1)

{

std::cout << "testNotEqualStrightFlushs1 \t complete\n";

}

else

{

std::cout << "testNotEqualStrightFlushs1 \t failed\n";

}

Hand hand3;

Card cards3[5];

cards3[0] = Card(4, 'C'); //6

cards3[1] = Card(0, 'C'); //2

cards3[2] = Card(1, 'C'); //3

cards3[3] = Card(2, 'C'); //4

cards3[4] = Card(3, 'C'); //5

hand3 = Hand(cards3);

Hand hand4;

Card cards4[5];

cards4[0] = Card(4, 'H'); //6

cards4[1] = Card(5, 'H'); //7

cards4[2] = Card(6, 'H'); //8

cards4[3] = Card(7, 'H'); //9

cards4[4] = Card(8, 'H'); //10

hand4 = Hand(cards4);

if (Comparator::getWinner(hand3, hand4) == 2)

{

std::cout << "testNotEqualStrightFlushs2 \t complete\n";

}

else

{

std::cout << "testNotEqualStrightFlushs2 \t failed\n";

}

}

void TestComparator::testNotEqualFourOfAKind()

{

Hand hand1;

Card cards1[5];

cards1[0] = Card(7, 'D'); //9

cards1[1] = Card(11, 'C'); //K

cards1[2] = Card(7, 'S'); //9

cards1[3] = Card(7, 'H'); //9

cards1[4] = Card(7, 'С'); //9

hand1 = Hand(cards1);

Hand hand2;

Card cards2[5];

cards2[0] = Card(6, 'D'); //8

cards2[1] = Card(11, 'H'); //K

cards2[2] = Card(6, 'S'); //8

cards2[3] = Card(6, 'H'); //8

cards2[4] = Card(6, 'C'); //8

hand2 = Hand(cards2);

if (Comparator::getWinner(hand1, hand2) == 1)

{

std::cout << "testNotEqualFourOfAKind1 \t complete\n";

}

else

{

std::cout << "testNotEqualFourOfAKind1 \t failed\n";

}

Hand hand3;

Card cards3[5];

cards3[0] = Card(7, 'D'); //9

cards3[1] = Card(11, 'C'); //K

cards3[2] = Card(7, 'S'); //9

cards3[3] = Card(7, 'H'); //9

cards3[4] = Card(7, 'C'); //9

hand3 = Hand(cards3);

Hand hand4;

Card cards4[5];

cards4[0] = Card(11, 'D'); //K

cards4[1] = Card(6, 'C'); //8

cards4[2] = Card(11, 'S'); //K

cards4[3] = Card(11, 'H'); //K

cards4[4] = Card(11, 'C'); //K

hand4 = Hand(cards4);

if (Comparator::getWinner(hand3, hand4) == 2)

{

std::cout << "testNotEqualFourOfAKind2 \t complete\n";

}

else

{

std::cout << "testNotEqualFourOfAKind2 \t failed\n";

}

}

void TestComparator::testNotEqualFullHouse()

{

Hand hand1;

Card cards1[5];

cards1[0] = Card(8, 'D'); //10

cards1[1] = Card(8, 'C'); //10

cards1[2] = Card(8, 'S'); //10

cards1[3] = Card(11, 'H'); //K

cards1[4] = Card(11, 'D'); //K

hand1 = Hand(cards1);

Hand hand2;

Card cards2[5];

cards2[0] = Card(8, 'D'); //10

cards2[1] = Card(8, 'C'); //10

cards2[2] = Card(8, 'S'); //10

cards2[3] = Card(7, 'H'); //9

cards2[4] = Card(7, 'D'); //9

hand2 = Hand(cards2);

if (Comparator::getWinner(hand1, hand2) == 1)

{

std::cout << "testNotEqualFullHouse1 \t complete\n";

}

else

{

std::cout << "testNotEqualFullHouse1 \t failed\n";

}

Hand hand3;

Card cards3[5];

cards3[0] = Card(7, 'D'); //9

cards3[1] = Card(7, 'C'); //9

cards3[2] = Card(7, 'S'); //9

cards3[3] = Card(2, 'H'); //4

cards3[4] = Card(2, 'D'); //4

hand3 = Hand(cards3);

Hand hand4;

Card cards4[5];

cards4[0] = Card(3, 'D'); //5

cards4[1] = Card(3, 'C'); //5

cards4[2] = Card(3, 'S'); //5

cards4[3] = Card(11, 'H'); //K

cards4[4] = Card(11, 'D'); //K

hand4 = Hand(cards4);

if (Comparator::getWinner(hand3, hand4) == 2)

{

std::cout << "testNotEqualFullHouse2 \t complete\n";

}

else

{

std::cout << "testNotEqualFullHouse2 \t failed\n";

}

}

void TestComparator::testEqualFlushes()

{

Hand hand1;

Card cards1[5];

cards1[0] = Card(7, 'H'); //9

cards1[1] = Card(12, 'H'); //A

cards1[2] = Card(5, 'H'); //7

cards1[3] = Card(9, 'H'); //J

cards1[4] = Card(3, 'H'); //5

hand1 = Hand(cards1);

Hand hand2;

Card cards2[5];

cards2[0] = Card(7, 'D'); //9

cards2[1] = Card(12, 'D'); //A

cards2[2] = Card(5, 'D'); //7

cards2[3] = Card(9, 'D'); //J

cards2[4] = Card(3, 'D'); //5

hand2 = Hand(cards2);

if (Comparator::getWinner(hand1, hand2) == 0)

{

std::cout << "testEqualFlushes \t complete\n";

}

else

{

std::cout << "testEqualFlushes \t failed\n";

}

}

void TestComparator::testNotEqualFlushes()

{

Hand hand1;

Card cards1[5];

cards1[0] = Card(7, 'H'); //9

cards1[1] = Card(12, 'H'); //A

cards1[2] = Card(5, 'H'); //7

cards1[3] = Card(9, 'H'); //J

cards1[4] = Card(3, 'H'); //5

hand1 = Hand(cards1);

Hand hand2;

Card cards2[5];

cards2[0] = Card(7, 'C'); //9

cards2[1] = Card(11, 'C'); //K

cards2[2] = Card(5, 'C'); //7

cards2[3] = Card(9, 'C'); //J

cards2[4] = Card(3, 'C'); //5

hand2 = Hand(cards2);

if (Comparator::getWinner(hand1, hand2) == 1)

{

std::cout << "testNotEqualFlushes1 \t\t complete\n";

}

else

{

std::cout << "testNotEqualFlushes1 \t\t failed\n";

}

Hand hand3;

Card cards3[5];

cards3[0] = Card(12, 'H'); //A

cards3[1] = Card(11, 'H'); //K

cards3[2] = Card(2, 'H'); //4

cards3[3] = Card(1, 'H'); //3

cards3[4] = Card(0, 'H'); //2

hand3 = Hand(cards3);

Hand hand4;

Card cards4[5];

cards4[0] = Card(2, 'S'); //4

cards4[1] = Card(3, 'S'); //5

cards4[2] = Card(1, 'S'); //3

cards4[3] = Card(11, 'S'); //K

cards4[4] = Card(12, 'S'); //A

hand4 = Hand(cards4);

if (Comparator::getWinner(hand3, hand4) == 2)

{

std::cout << "testNotEqualFlushes2 \t\t complete\n";

}

else

{

std::cout << "testNotEqualFlushes2 \t\t failed\n";

}

}

void TestComparator::testEqualStraights()

{

Hand hand1;

Card cards1[5];

cards1[0] = Card(8, 'H'); //10

cards1[1] = Card(9, 'C'); //J

cards1[2] = Card(10, 'S'); //Q

cards1[3] = Card(11, 'D'); //K

cards1[4] = Card(7, 'H'); //9

hand1 = Hand(cards1);

Hand hand2;

Card cards2[5];

cards2[0] = Card(8, 'D'); //10

cards2[1] = Card(9, 'H'); //J

cards2[2] = Card(10, 'C'); //Q

cards2[3] = Card(11, 'H'); //K

cards2[4] = Card(7, 'D'); //9

hand2 = Hand(cards2);

if (Comparator::getWinner(hand1, hand2) == 0)

{

std::cout << "testEqualStraights \t\t complete\n";

}

else

{

std::cout << "testEqualStraights \t\t failed\n";

}

}

void TestComparator::testNotEqualStraights()

{

Hand hand1;

Card cards1[5];

cards1[0] = Card(7, 'H'); //9

cards1[1] = Card(8, 'H'); //10

cards1[2] = Card(9, 'C'); //J

cards1[3] = Card(10, 'C'); //Q

cards1[4] = Card(11, 'D'); //K

hand1 = Hand(cards1);

Hand hand2;

Card cards2[5];

cards2[0] = Card(7, 'S'); //9

cards2[1] = Card(8, 'C'); //10

cards2[2] = Card(6, 'S'); //8

cards2[3] = Card(9, 'S'); //J

cards2[4] = Card(10, 'C'); //Q

hand2 = Hand(cards2);

if (Comparator::getWinner(hand1, hand2) == 1)

{

std::cout << "testNotEqualStraights1 \t complete\n";

}

else

{

std::cout << "testNotEqualStraights1 \t failed\n";

}

Hand hand3;

Card cards3[5];

cards3[0] = Card(12, 'H'); //A

cards3[1] = Card(0, 'S'); //2

cards3[2] = Card(1, 'D'); //3

cards3[3] = Card(2, 'C'); //4

cards3[4] = Card(3, 'H'); //5

hand3 = Hand(cards3);

Hand hand4;

Card cards4[5];

cards4[0] = Card(4, 'C'); //6

cards4[1] = Card(5, 'H'); //7

cards4[2] = Card(6, 'S'); //8

cards4[3] = Card(7, 'D'); //9

cards4[4] = Card(8, 'C'); //10

hand4 = Hand(cards4);

if (Comparator::getWinner(hand3, hand4) == 2)

{

std::cout << "testNotEqualStraights2 \t complete\n";

}

else

{

std::cout << "testNotEqualStraights2 \t failed\n";

}

}

void TestComparator::testNotEqualThreeOfAKind()

{

Hand hand1;

Card cards1[5];

cards1[0] = Card(8, 'C'); //10

cards1[1] = Card(8, 'H'); //10

cards1[2] = Card(8, 'S'); //10

cards1[3] = Card(10, 'D'); //Q

cards1[4] = Card(11, 'C'); //K

hand1 = Hand(cards1);

Hand hand2;

Card cards2[5];

cards2[0] = Card(6, 'S'); //8

cards2[1] = Card(5, 'C'); //7

cards2[2] = Card(7, 'S'); //9

cards2[3] = Card(7, 'S'); //9

cards2[4] = Card(7, 'C'); //9

hand2 = Hand(cards2);

if (Comparator::getWinner(hand1, hand2) == 1)

{

std::cout << "testNotEqualThreeOfAKind1 \t complete\n";

}

else

{

std::cout << "testNotEqualThreeOfAKind1 \t failed\n";

}

Hand hand3;

Card cards3[5];

cards3[0] = Card(8, 'C'); //10

cards3[1] = Card(8, 'H'); //10

cards3[2] = Card(8, 'S'); //10

cards3[3] = Card(4, 'D'); //6

cards3[4] = Card(11, 'C'); //K

hand3 = Hand(cards3);

Hand hand4;

Card cards4[5];

cards4[0] = Card(6, 'H'); //8

cards4[1] = Card(12, 'C'); //A

cards4[2] = Card(11, 'H'); //K

cards4[3] = Card(12, 'S'); //A

cards4[4] = Card(12, 'D'); //A

hand4 = Hand(cards4);

if (Comparator::getWinner(hand3, hand4) == 2)

{

std::cout << "testNotEqualThreeOfAKind2 \t complete\n";

}

else

{

std::cout << "testNotEqualThreeOfAKind2 \t failed\n";

}

}

void TestComparator::testEqualTwoPairs()

{

Hand hand1;

Card cards1[5];

cards1[0] = Card(8, 'H'); //10

cards1[1] = Card(8, 'C'); //10

cards1[2] = Card(10, 'S'); //Q

cards1[3] = Card(9, 'D'); //J

cards1[4] = Card(9, 'H'); //J

hand1 = Hand(cards1);

Hand hand2;

Card cards2[5];

cards2[0] = Card(8, 'D'); //10

cards2[1] = Card(8, 'S'); //10

cards2[2] = Card(10, 'D'); //Q

cards2[3] = Card(9, 'C'); //J

cards2[4] = Card(9, 'S'); //J

hand2 = Hand(cards2);

if (Comparator::getWinner(hand1, hand2) == 0)

{

std::cout << "testEqualTwoPairs \t complete\n";

}

else

{

std::cout << "testEqualTwoPairs \t failed\n";

}

}

void TestComparator::testNotEqualTwoPairs()

{

Hand hand1;

Card cards1[5];

cards1[0] = Card(8, 'C'); //10

cards1[1] = Card(8, 'H'); //10

cards1[2] = Card(10, 'S'); //Q

cards1[3] = Card(10, 'D'); //Q

cards1[4] = Card(11, 'C'); //K

hand1 = Hand(cards1);

Hand hand2;

Card cards2[5];

cards2[0] = Card(6, 'H'); //8

cards2[1] = Card(5, 'C'); //7

cards2[2] = Card(5, 'H'); //7

cards2[3] = Card(7, 'S'); //9

cards2[4] = Card(7, 'D'); //9

hand2 = Hand(cards2);

if (Comparator::getWinner(hand1, hand2) == 1)

{

std::cout << "testEqualTwoPairs1 \t complete\n";

}

else

{

std::cout << "testEqualTwoPairs1 \t failed\n";

}

Hand hand3;

Card cards3[5];

cards3[0] = Card(8, 'C'); //10

cards3[1] = Card(8, 'H'); //10

cards3[2] = Card(3, 'S'); //5

cards3[3] = Card(3, 'C'); //5

cards3[4] = Card(0, 'H'); //2

hand3 = Hand(cards3);

Hand hand4;

Card cards4[5];

cards4[0] = Card(8, 'H'); //10

cards4[1] = Card(8, 'S'); //10

cards4[2] = Card(4, 'C'); //6

cards4[3] = Card(4, 'D'); //6

cards4[4] = Card(6, 'D'); //8

hand4 = Hand(cards4);

if (Comparator::getWinner(hand3, hand4) == 2)

{

std::cout << "testEqualTwoPairs2 \t complete\n";

}

else

{

std::cout << "testEqualTwoPairs2 \t failed\n";

}

}

void TestComparator::testEqualPairs()

{

Hand hand1;

Card cards1[5];

cards1[0] = Card(8, 'H'); //10

cards1[1] = Card(7, 'C'); //9

cards1[2] = Card(10, 'S'); //Q

cards1[3] = Card(9, 'D'); //J

cards1[4] = Card(9, 'H'); //J

hand1 = Hand(cards1);

Hand hand2;

Card cards2[5];

cards2[0] = Card(8, 'C'); //10

cards2[1] = Card(7, 'S'); //9

cards2[2] = Card(10, 'H'); //Q

cards2[3] = Card(9, 'C'); //J

cards2[4] = Card(9, 'S'); //J

hand2 = Hand(cards2);

if (Comparator::getWinner(hand1, hand2) == 0)

{

std::cout << "testEqualPairs \t\t\t complete\n";

}

else

{

std::cout << "testEqualPairs \t\t\t failed\n";

}

}

void TestComparator::testNotEqualPairs()

{

Hand hand1;

Card cards1[5];

cards1[0] = Card(8, 'C'); //10

cards1[1] = Card(8, 'S'); //10

cards1[2] = Card(4, 'D'); //6

cards1[3] = Card(7, 'S'); //9

cards1[4] = Card(11, 'H'); //K

hand1 = Hand(cards1);

Hand hand2;

Card cards2[5];

cards2[0] = Card(6, 'H'); //8

cards2[1] = Card(1, 'D'); //3

cards2[2] = Card(11, 'D'); //K

cards2[3] = Card(6, 'S'); //8

cards2[4] = Card(5, 'S'); //7

hand2 = Hand(cards2);

if (Comparator::getWinner(hand1, hand2) == 1)

{

std::cout << "testNotEqualPairs1 \t\t complete\n";

}

else

{

std::cout << "testNotEqualPairs1 \t\t failed\n";

}

Hand hand3;

Card cards3[5];

cards3[0] = Card(8, 'C'); //10

cards3[1] = Card(7, 'S'); //9

cards3[2] = Card(4, 'H'); //6

cards3[3] = Card(4, 'D'); //6

cards3[4] = Card(11, 'D'); //K

hand3 = Hand(cards3);

Hand hand4;

Card cards4[5];

cards4[0] = Card(6, 'D'); //8

cards4[1] = Card(3, 'D'); //5

cards4[2] = Card(12, 'S'); //A

cards4[3] = Card(8, 'D'); //10

cards4[4] = Card(8, 'H'); //10

hand4 = Hand(cards4);

if (Comparator::getWinner(hand3, hand4) == 2)

{

std::cout << "testNotEqualPairs2 \t\t complete\n";

}

else

{

std::cout << "testNotEqualPairs2 \t\t failed\n";

}

}

void TestComparator::testEqualHighCard()

{

Hand hand1;

Card cards1[5];

cards1[0] = Card(8, 'H'); //10

cards1[1] = Card(7, 'C'); //9

cards1[2] = Card(10, 'S'); //Q

cards1[3] = Card(9, 'D'); //J

cards1[4] = Card(0, 'H'); //2

hand1 = Hand(cards1);

Hand hand2;

Card cards2[5];

cards2[0] = Card(8, 'C'); //10

cards2[1] = Card(7, 'S'); //9

cards2[2] = Card(10, 'H'); //Q

cards2[3] = Card(9, 'C'); //J

cards2[4] = Card(0, 'S'); //2

hand2 = Hand(cards2);

if (Comparator::getWinner(hand1, hand2) == 0)

{

std::cout << "testEqualHighCard \t complete\n";

}

else

{

std::cout << "testEqualHighCard \t failed\n";

}

}

void TestComparator::testNotEqualHighCard()

{

Hand hand1;

Card cards1[5];

cards1[0] = Card(8, 'C'); //10

cards1[1] = Card(12, 'H'); //A

cards1[2] = Card(5, 'D'); //7

cards1[3] = Card(7, 'S'); //9

cards1[4] = Card(11, 'C'); //K

hand1 = Hand(cards1);

Hand hand2;

Card cards2[5];

cards2[0] = Card(8, 'S'); //10

cards2[1] = Card(12, 'S'); //A

cards2[2] = Card(4, 'D'); //6

cards2[3] = Card(7, 'D'); //9

cards2[4] = Card(11, 'H'); //K

hand2 = Hand(cards2);

if (Comparator::getWinner(hand1, hand2) == 1)

{

std::cout << "testNotEqualHighCard1 \t\t complete\n";

}

else

{

std::cout << "testNotEqualHighCard1 \t\t failed\n";

}

Hand hand3;

Card cards3[5];

cards3[0] = Card(9, 'S'); //J

cards3[1] = Card(0, 'D'); //2

cards3[2] = Card(10, 'H'); //Q

cards3[3] = Card(6, 'H'); //8

cards3[4] = Card(8, 'C'); //10

hand3 = Hand(cards3);

Hand hand4;

Card cards4[5];

cards4[0] = Card(10, 'H'); //Q

cards4[1] = Card(9, 'H'); //J

cards4[2] = Card(8, 'S'); //10

cards4[3] = Card(6, 'S'); //8

cards4[4] = Card(1, 'D'); //3

hand4 = Hand(cards4);

if (Comparator::getWinner(hand3, hand4) == 2)

{

std::cout << "testNotEqualHighCard2 \t\t complete\n";

}

else

{

std::cout << "testNotEqualHighCard2 \t\t failed\n";

}

}

1. [1] Условие задачи в том виде, который приведен в учебнике или задании [↑](#footnote-ref-1)
2. [2] Постановка задачи включает подробное описание задачи: приводятся имена исходных данных (массивов, переменных), имена файлов, файлов, имена промежуточных данных и результата. Если необходимо по условию – формулы, которые будут использованы в вычислениях. [↑](#footnote-ref-2)
3. [3] Необязательный пункт [↑](#footnote-ref-3)