**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

# «ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ» ИНЖЕНЕРНО- ТЕХНОЛОГИЧЕСКАЯ АКАДЕМИЯ

**Институт компьютерных технологий и информационной безопасности**

**Кафедра математического обеспечения и применения ЭВМ**

Отчет по лабораторной работе №1

по курсу «Объектно ориентированное программирование»

«Классы и объекты в С++»

Выполнил: студент гр. КТбо2-6

Калитин А.В.

Таганрог 2020

**Оглавление**

[Техническое задание 3](#_TOC_250003)

[Выполнение задания 3](#_TOC_250002)

1. [Спецификация классов 3](#_TOC_250001)
2. [Диаграмма классов 4](#_TOC_250000)
3. Листинг 5

**Вариант № 15**

# ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Определить классы Карта (Card) и Колода\_карт (Cardatch). Поля первого

* масть (suit) и достоинство (rank). Методы класса возвращают масть и достоинство. Второй включает массив (32) объектов первого.

Методы класса:

* + перемешивания колоды;
  + сравнения 2-х карт по достоинству при условии, что масти одинаковы;
  + создания 4-х мест и раздачи равного количества карт;

−моделирования упрощенного розыгрыша взятки: на стол выкладываются по одной карте от каждого из 4-х игроков; первая выложенная карта определяет масть; выигрывает карта, старшая по достоинству (картинки старше простых карт; козырной масти нет).

Список карт для инициализации программы хранить в файле.

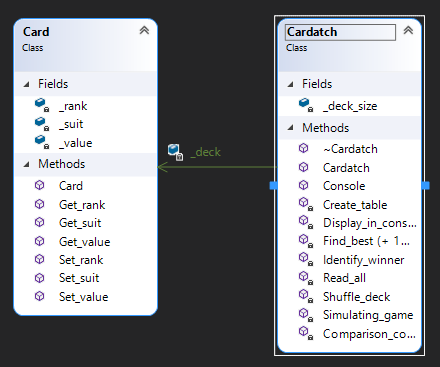
# ВЫПОЛНЕНИЕ ЗАДАНИЯ

# Спецификация классов.

Класс **Cardatch** содержит публичный метод для консольного

взаимодействия с объектом. В приватной секции имеется массив из объектов класса **Card**, пара методов для ввода из файла(**Read**\_**all**) и вывода карт(**Display**\_**in**\_**console**) в консоль, метод для сравнения двух карт(**Сomparison**\_**couple**\_**cards**), который принимает две ссылки на объекты и возвращает результат в виде целого числа. Метод для перемешивания карт(**Shuffle**\_**deck**) на основе случайных чисел. Метод для симуляции игры четырех человек(**Simulating**\_**game**), который создает четыре динамических массива из элементов **Card** и с помощью побочных функций определяется победитель. Класс **Card** имеет три приватных поля для ранга, масти карты и «веса», а также сеттеры и геттеры для этих полей.

# Диаграмма классов.



* 1. **Листинг**

1. Main.cpp

#include <iostream>

#include "Cardatch.h"

#include "vld.h"

int main()

{

Cardatch deck;

deck.Console();

return 0;

}

1. Cardatch.h

#pragma once

#include <iostream>

#include <ctime>

#include <exception>

#include <fstream>

#include "Card.h"

class Cardatch

{

public:

Cardatch()=default;

~Cardatch();

void Console();

private:

Card\* \_deck=nullptr;

int \_deck\_size=0;

void Display\_in\_console(const Card &card);

void Display\_in\_console(const Card &card,int number);

void Read\_all(const std::string& path);

void Shuffle\_deck();

int Сomparison\_couple\_cards(const Card &first, const Card &second);

Card\* Create\_table(int how);

Card Find\_best(int first, int last,const Card\* table);

Card Find\_best(int first, int last, Card\* table, const std::string& suit);

int Identify\_winner(Card\* table, int how);

int Simulating\_game(int n);

};

1. Cardatch.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "Cardatch.h"

Cardatch::~Cardatch()

{

if (\_deck != nullptr)

{

delete[] \_deck;

}

}

void Cardatch::Display\_in\_console(const Card &card)

{

std::cout << card.Get\_rank() << " " << card.Get\_suit() << ' ' << '\n';

}

void Cardatch::Display\_in\_console(const Card &card, int number)

{

std::cout<<number<<": "<< card.Get\_rank() << " " << card.Get\_suit() << ' ' << '\n';

}

void Cardatch::Console()

{

srand(time(0));

std::cout << "Enter the number of cards in the deck(32 or 52) or 104 but it's illegal:\n";

std::string path;

std::cin >> path;

path += ".txt";

try

{

Read\_all(path);

}

catch (const std::exception &err)

{

std::cerr << "Error: " << err.what();

exit(1);

}

int command = 1;

std::cout << "1 - Shuffle\n2 - Comparison couple cards\n3 - Simulation game\n4 - Print all cards\n0 - Exit.\n";

while (command)

{

std::cout << "Enter a command: ";

std::cin >> command;

switch (command)

{

case 0:

break;

case 1:

Shuffle\_deck();

std::cout << "The deck is shuffled" << std::endl;

break;

case 2:

{

Card first = \_deck[rand() % 32];

Card second = \_deck[rand() % 32];

Display\_in\_console(first);

Display\_in\_console(second);

int res = Сomparison\_couple\_cards(first,second);

if (res == 1)

{

std::cout << "First win!\n";

}

if (res == 2)

{

std::cout << "Second win!\n";

}

if (res == 3)

{

std::cout << "Diffrent suits\n";

}

}

break;

case 3:

{

int num;

std::cout << "How much cards per player?\n";

std::cin >> num;

if (num > 0 && num <= \_deck\_size / 4)

{

std::cout<<"Winning player: "<<Simulating\_game(num)<<"\n";

}

else

{

std::cout << "Invalid number. Try again\n";

}

}

break;

case 4:

for (int i = 0; i < \_deck\_size; i++)

{

Display\_in\_console(\_deck[i]);

}

break;

default:

std::cout << "Invalid command. Try again.\n";

break;

}

}

}

void Cardatch::Read\_all(const std::string &path)

{

std::ifstream file;

file.open(path);

std::string suit, rank;

int value;

if (file.is\_open())

{

file >> \_deck\_size;

\_deck = new Card[\_deck\_size];

for (int i = 0; i < \_deck\_size; i++)

{

file >> rank >> suit >> value;

\_deck[i].Set\_rank(rank);

\_deck[i].Set\_suit(suit);

\_deck[i].Set\_value(value);

}

}

else

{

file.close();

throw std::exception("File not found.");

}

}

void Cardatch::Shuffle\_deck()

{

int n = \_deck\_size;

while (n > 1)

{

int ind = rand() % \_deck\_size;

n--;

Card temp = \_deck[n];

\_deck[n] = \_deck[ind];

\_deck[ind] = temp;

}

}

int Cardatch::Сomparison\_couple\_cards(const Card &first,const Card &second)

{

if (first.Get\_suit()==second.Get\_suit())

{

if (first.Get\_value()>second.Get\_value())

{

return 1;

}

else

{

return 2;

}

}

else

{

return 3;

}

}

Card\* Cardatch::Create\_table(int how)

{

Card\* table = new Card[how \* 4];

int j = 0;

for (int i = \_deck\_size - 1; i > \_deck\_size - 1 - how \* 4; i--, j++)

{

table[j] = \_deck[i];

}

return table;

}

Card Cardatch::Find\_best(int first,int last,const Card\* table)

{

int max\_value = table[first].Get\_value();

Card temp= table[first];

for (int i = first; i <= last; i++)

{

if (table[i].Get\_value() > max\_value)

{

temp = table[i];

max\_value = temp.Get\_value();

}

}

return temp;

}

Card Cardatch::Find\_best(int first, int last, Card\* table,const std::string& suit)

{

int max\_value = table[first].Get\_value();

Card temp = table[first];

for (int i = first; i <= last; i++)

{

if (table[i].Get\_suit() == suit && table[i].Get\_value() > max\_value)

{

temp = table[i];

max\_value = temp.Get\_value();

}

}

return temp;

}

int Cardatch::Identify\_winner(Card\* table,int how)

{

int winner = 1;

Card \*best\_cards=new Card[4];

best\_cards[0] = Find\_best(0, how-1,table);

std::string suit = best\_cards[0].Get\_suit();

for (int i = 1; i < 4; i++)

{

best\_cards[i] = Find\_best(i\*how, (i+1) \* how - 1, table, suit);

}

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

{

Display\_in\_console(best\_cards[i],i+1);

}

int max\_value = -1;

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

{

if (best\_cards[i].Get\_suit() == suit && best\_cards[i].Get\_value() > max\_value)

{

winner = i + 1;

max\_value = best\_cards[i].Get\_value();

}

}

delete[] best\_cards;

return winner;

}

int Cardatch::Simulating\_game(int how)

{

Card\* table=Create\_table(how);

int result = Identify\_winner(table,how);

delete[] table;

return result;

}

1. Card.h

#pragma once

class Card

{

public:

Card() = default;

std::string Get\_suit() const;

std::string Get\_rank() const;

int Get\_value() const;

void Set\_suit(const std::string &suit);

void Set\_rank(const std::string& suit);

void Set\_value(int value);

private:

std::string \_suit;

std::string \_rank;

int \_value=0;

};

1. Card.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <string>

#include "Card.h"

int Card::Get\_value() const

{

return \_value;

}

void Card::Set\_value(int value)

{

\_value = value;

}

std::string Card::Get\_suit() const

{

return \_suit;

}

std::string Card::Get\_rank() const

{

return \_rank;

}

void Card::Set\_suit(const std::string& suit)

{

\_suit = suit;

}

void Card::Set\_rank(const std::string& rank)

{

\_rank = rank;

}