**Министр науки и высшего образования Российской Федерации**

**Федеральное государственное автономное образовательное учреждение высшего образования**

**«Национальный исследовательский университет ИТМО»**

Факультет информационных технологий и программирования

Лабораторная работа №8

*Кубик Рубика.*

**Выполнил студент группы № M3116**

Тихонов Александр Дмитриевич

**Подпись:**

**Проверил:**

Повышев Владислав Вячеславович

Санкт-Петербург

2023

Задание:

Спроектировать и реализовать программу, имитирующую сборку ​Кубика Рубика​3x3. К программе предъявляются следующие функциональные требования:

* Сохранение и чтение состояния кубика рубика из файла
* Проверка корректности текущего состояния (инвариант состояний кубика)
* Вывод в консоль текущего состояния
* Вращение граней кубика рубика с помощью вводимых команд
* Генерация случайного состояния Кубика Рубика, корректного с точки зрения инварианта состояний
* Нахождения “решения” для текущего состояния в виде последовательности поворотов граней

Нефункциональные требования:

* Программа должны быть спроектирована, с использованием ОПП
* Логические сущности должны быть выделены в отдельный классы

Критерии оценки:

* Логично выстроенная архитектура приложения
* Применение возможностей языка программирования С++ включая стандартную библиотеку

В коде был реализован класс RubicCube:

1. Конструктор **RubiksCube**: инициализирует кубик Рубика.
2. Метод **CreateAllCode**: генерирует код, представляющий состояние кубика Рубика.
3. Методы **RotateFaceClockwise** и **RotateFaceCounterclockwise**: поворачивают указанную грань кубика по часовой стрелке и против часовой стрелки соответственно.
4. Методы **RotateCenterVC**, **RotateCenterVCC**, **RotateCenterHC**, **RotateCenterHCC**: выполняют вращения центрального слоя кубика вертикально и горизонтально по часовой стрелке и против часовой стрелки.
5. Метод **GetCreateCubeCode**: возвращает код, представляющий состояние кубика Рубика.
6. Метод **SetCubeState**: устанавливает состояние кубика Рубика, используя переданный код.
7. Метод **DisplaySolve**: отображает решение кубика Рубика.
8. Метод **Display**: отображает состояние кубика Рубика.
9. Метод **SpiningCube**: выполняет вращение кубика Рубика, используя данные из файла.
10. Метод **SetCubeFromFile**: устанавливает состояние кубика Рубика, используя данные из файла.
11. Метод **SolveCube**: Выводит решение кубика Рубика на консоль
12. Метод **CheckCorrectCube**: проверяет корректность состояния кубика Рубика, используя данные из файла.
13. Метод **GenerateSpinCube**: генерирует случайно перемешанный кубик Рубика и сохраняет его в файле.

Main.cpp

#include "RubicsCube.hpp"  
  
int main() {  
 RubiksCube cube;  
 std::filesystem::path FilePath1 = "CUBIC.txt";  
 std::filesystem::path FilePath2 = "CUBIC2.txt";  
 cube.GenerateSpinCube(FilePath1, 5);  
 cube.SolveCube(FilePath1);  
 return 0;  
}

RubicsCube.hpp

//  
// Created by Тихонов Александр on 01.05.2023.  
//  
#pragma once  
#include <iostream>  
#include <array>  
#include <fstream>  
#include <vector>  
#include <unistd.h>  
#include <stack>  
#include <string>  
#include <utility>  
#include <filesystem>  
#include <random>  
  
enum Color { *WHITE*, *YELLOW*, *BLUE*, *GREEN*, *RED*, *ORANGE* };  
  
class RubiksCube {  
private:  
 std::array<std::array<std::array<Color, 3>, 3>, 6> cube;  
 std::stack<int> Rotations;  
  
 std::string CreateAllCode(Color NumWise, int CurPos, char CurColor );  
 std::string CreatePosCode(int CurPos, char CurColor);  
 void RotateFaceClockwise(int face);  
 void RotateFaceCounterclockwise(int face);  
 void RotateCenterVC(int face);  
 void RotateCenterVCC(int face);  
 void RotateCenterHC(int face);  
 void RotateCenterHCC(int face);  
 std::string GetCreateCubeCode();  
 void SetCubeState(std::string RawData);  
 Color GetSTRWiseCode(const std::string &RawData, int pos);  
 Color GetSTRColor(const std::string &RawData, int pos);  
 std::pair<int, int> GetSTRPairPos(const std::string &RawData, int pos);  
 void DisplaySolve();  
 int GenerateRandomSpin();  
  
public:  
 RubiksCube();  
 void Display();  
 void SpiningCube(std::filesystem::path FilePath);  
 void SetCubeFromFile(std::filesystem::path FilePath);  
 void SolveCube(std::filesystem::path FilePath);  
 void CheckCorrectCube(std::filesystem::path FilePath);  
 void GenerateSpinCube(std::filesystem::path FilePath, int CountSpin);  
};

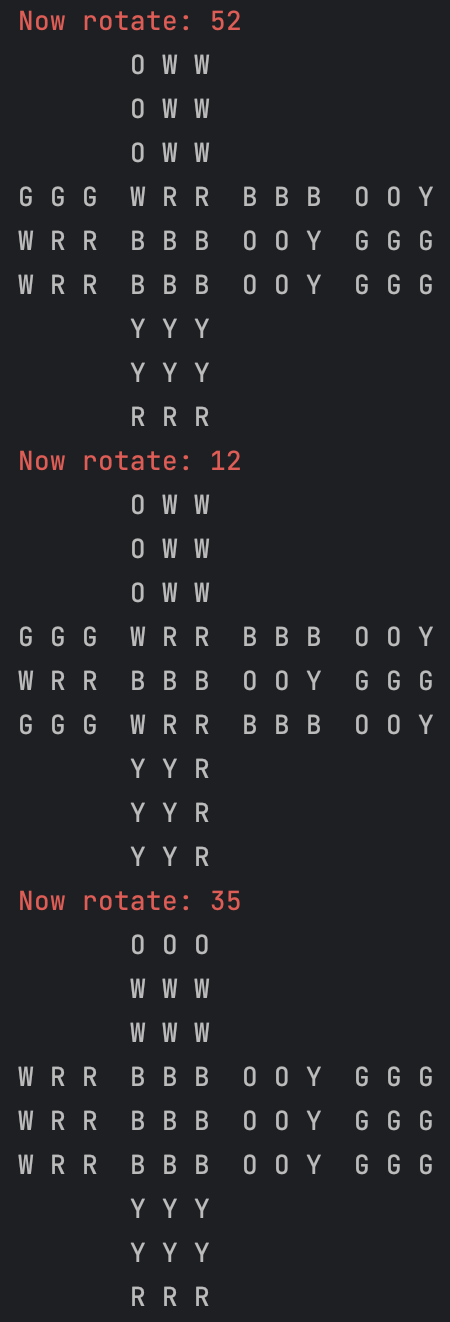
**RubicsCube.cpp**

//  
// Created by Тихонов Александр on 01.05.2023.  
//  
#include "RubicsCube.hpp"  
  
RubiksCube::RubiksCube() {  
 for (int face = 0; face < 6; ++face) {  
 for (int row = 0; row < 3; ++row) {  
 for (int col = 0; col < 3; ++col) {  
 cube[face][row][col] = static\_cast<Color>(face);  
 }  
 }  
 }  
}  
void RubiksCube::Display() {  
 std::vector<char> mColor = **{**'W', 'Y', 'B', 'G', 'R', 'O'**}**;  
 //WHITE  
 {std::string WhiteSide;  
 for (int row = 0; row < 3; ++row) {  
 for (int col = 0; col < 3; ++col) {  
 WhiteSide += mColor[cube[*WHITE*][row][col]];  
 }  
 }  
 std::reverse(WhiteSide.begin(), WhiteSide.end());  
 for (int i = 0; i < WhiteSide.size(); ++i) {  
 if (i%3 == 0){  
 if (i==0) std::cout<<" ";  
 else std::cout<<"\n ";  
 }  
 std::cout<<WhiteSide[i]<<" ";  
 }  
 std::cout<<"\n";}  
 //RED BLUE ORANGE GREEN  
 {std::string FirstLayer;  
 std::string SecondLayer;  
 std::string ThirdLayer;  
 {  
 std::string RedFirst;  
 std::string BlueFirst;  
 std::string OrangeFirst;  
 std::string GreenFirst;  
 for (int row = 0; row < 1; ++row) {  
 for (int col = 0; col < 3; ++col) {  
 RedFirst += " ";  
 RedFirst += mColor[cube[*RED*][row][col]];  
 }  
 }  
 std::reverse(RedFirst.begin(), RedFirst.end());  
 FirstLayer += RedFirst;  
 FirstLayer += " ";  
 for (int row = 0; row < 1; ++row) {  
 for (int col = 0; col < 3; ++col) {  
 BlueFirst += " ";  
 BlueFirst += mColor[cube[*BLUE*][row][col]];  
 }  
 }  
 std::reverse(BlueFirst.begin(), BlueFirst.end());  
 FirstLayer += BlueFirst;  
 FirstLayer += " ";  
 for (int row = 0; row < 1; ++row) {  
 for (int col = 0; col < 3; ++col) {  
 OrangeFirst += " ";  
 OrangeFirst += mColor[cube[*ORANGE*][row][col]];  
 }  
 }  
 std::reverse(OrangeFirst.begin(), OrangeFirst.end());  
 FirstLayer += OrangeFirst;  
 FirstLayer += " ";  
 for (int row = 0; row < 1; ++row) {  
 for (int col = 0; col < 3; ++col) {  
 GreenFirst += " ";  
 GreenFirst += mColor[cube[*GREEN*][row][col]];  
 }  
 }  
 std::reverse(GreenFirst.begin(), GreenFirst.end());  
 FirstLayer += GreenFirst;  
 std::cout << FirstLayer << "\n";  
 }  
  
 {  
 std::string RedSecond;  
 std::string BlueSecond;  
 std::string OrangeSecond;  
 std::string GreenSecond;  
  
 for (int row = 1; row < 2; ++row) {  
 for (int col = 0; col < 3; ++col) {  
 RedSecond += " ";  
 RedSecond += mColor[cube[*RED*][row][col]];  
 }  
 }  
 std::reverse(RedSecond.begin(), RedSecond.end());  
 SecondLayer += RedSecond;  
 SecondLayer += " ";  
 for (int row = 1; row < 2; ++row) {  
 for (int col = 0; col < 3; ++col) {  
 BlueSecond += " ";  
 BlueSecond += mColor[cube[*BLUE*][row][col]];  
 }  
 }  
 std::reverse(BlueSecond.begin(), BlueSecond.end());  
 SecondLayer += BlueSecond;  
 SecondLayer += " ";  
 for (int row = 1; row < 2; ++row) {  
 for (int col = 0; col < 3; ++col) {  
 OrangeSecond += " ";  
 OrangeSecond += mColor[cube[*ORANGE*][row][col]];  
 }  
 }  
 std::reverse(OrangeSecond.begin(), OrangeSecond.end());  
 SecondLayer += OrangeSecond;  
 SecondLayer += " ";  
 for (int row = 1; row < 2; ++row) {  
 for (int col = 0; col < 3; ++col) {  
 GreenSecond += " ";  
 GreenSecond += mColor[cube[*GREEN*][row][col]];  
 }  
 }  
 std::reverse(GreenSecond.begin(), GreenSecond.end());  
 SecondLayer += GreenSecond;  
 std::cout << SecondLayer << "\n";  
 }  
  
 {  
 std::string RedThird;  
 std::string BlueThird;  
 std::string OrangeThird;  
 std::string GreenThird;  
 for (int row = 2; row < 3; ++row) {  
 for (int col = 0; col < 3; ++col) {  
 RedThird += " ";  
 RedThird += mColor[cube[*RED*][row][col]];  
 }  
 }  
 std::reverse(RedThird.begin(), RedThird.end());  
 ThirdLayer += RedThird;  
 ThirdLayer += " ";  
 for (int row = 2; row < 3; ++row) {  
 for (int col = 0; col < 3; ++col) {  
 BlueThird += " ";  
 BlueThird += mColor[cube[*BLUE*][row][col]];  
 }  
 }  
 std::reverse(BlueThird.begin(), BlueThird.end());  
 ThirdLayer += BlueThird;  
 ThirdLayer += " ";  
 for (int row = 2; row < 3; ++row) {  
 for (int col = 0; col < 3; ++col) {  
 OrangeThird += " ";  
 OrangeThird += mColor[cube[*ORANGE*][row][col]];  
 }  
 }  
 std::reverse(OrangeThird.begin(), OrangeThird.end());  
 ThirdLayer += OrangeThird;  
 ThirdLayer += " ";  
 for (int row = 2; row < 3; ++row) {  
 for (int col = 0; col < 3; ++col) {  
 GreenThird += " ";  
 GreenThird += mColor[cube[*GREEN*][row][col]];  
 }  
 }  
 std::reverse(GreenThird.begin(), GreenThird.end());  
 ThirdLayer += GreenThird;  
 std::cout << ThirdLayer << "\n";  
 }  
 }  
 //YELLOW  
 {std::string YellowSide;  
 for (int row = 0; row < 3; ++row) {  
 for (int col = 0; col < 3; ++col) {  
 YellowSide+=mColor[cube[*YELLOW*][row][col]];  
 }  
 }  
 std::reverse(YellowSide.begin(), YellowSide.end());  
 for (int i = 0; i < YellowSide.size(); ++i) {  
 if (i%3 == 0){  
 if (i==0) std::cout<<" ";  
 else std::cout<<"\n ";  
 }  
 std::cout<<YellowSide[i]<<" ";  
 }  
 std::cout<<"\n";}  
}  
  
//Разборка  
//enum Color { WHITE, YELLOW, BLUE, GREEN, RED, ORANGE };  
void rotate\_matrix\_clockwise(std::array<std::array<Color, 3>, 3>& matrix) {  
 std::array<std::array<Color, 3>, 3> temp = matrix;  
 for (int i = 0; i < 3; ++i) {  
 for (int j = 0; j < 3; ++j) {  
 matrix[i][j] = temp[2 - j][i];  
 }  
 }  
}  
void rotate\_matrix\_counterclockwise(std::array<std::array<Color, 3>, 3>& matrix) {  
 std::array<std::array<Color, 3>, 3> temp = matrix;  
 for (int i = 0; i < 3; ++i) {  
 for (int j = 0; j < 3; ++j) {  
 matrix[i][j] = temp[j][2 - i];  
 }  
 }  
}  
void RubiksCube::RotateFaceClockwise(int face) {  
 // Поворачиваем грань по часовой стрелке  
 rotate\_matrix\_clockwise(cube[face]);  
  
 // Обновляем смежные грани  
 std::array<Color, 3> temp;  
 switch (face) {  
 case *WHITE*:  
 temp = {cube[*RED*][0][0], cube[*RED*][0][1], cube[*RED*][0][2]};  
 for (int i = 0; i < 3; ++i) {  
 cube[*RED*][0][i] = cube[*BLUE*][0][i];  
 cube[*BLUE*][0][i] = cube[*ORANGE*][0][i];  
 cube[*ORANGE*][0][i] = cube[*GREEN*][0][i];  
 cube[*GREEN*][0][i] = temp[i];  
 }  
 break;  
 case *YELLOW*:  
 temp = {cube[*BLUE*][2][0], cube[*BLUE*][2][1], cube[*BLUE*][2][2]};  
 cube[*BLUE*][2] = cube[*ORANGE*][2];  
 cube[*ORANGE*][2] = cube[*GREEN*][2];  
 cube[*GREEN*][2] = cube[*RED*][2];  
 cube[*RED*][2] = temp;  
 break;  
 case *BLUE*:  
 temp = {cube[*WHITE*][0][0], cube[*WHITE*][0][1], cube[*WHITE*][0][2]};  
 cube[*WHITE*][0] = {cube[*ORANGE*][2][2], cube[*ORANGE*][1][2], cube[*ORANGE*][0][2]};  
 for (int i = 0; i < 3; ++i) {  
 cube[*ORANGE*][i][2] = cube[*YELLOW*][2][i];  
 }  
 cube[*YELLOW*][2] = {cube[*RED*][0][0], cube[*RED*][1][0], cube[*RED*][2][0]};  
 for (int i = 0; i < 3; ++i) {  
 cube[*RED*][i][0] = temp[i];  
 }  
 break;  
 case *GREEN*:  
 temp = {cube[*WHITE*][2][0], cube[*WHITE*][2][1], cube[*WHITE*][2][2]};  
 cube[*WHITE*][2] = {cube[*RED*][2][2], cube[*RED*][1][2], cube[*RED*][0][2]};  
 for (int i = 0; i < 3; ++i) {  
 cube[*RED*][i][2] = cube[*YELLOW*][0][i];  
 }  
 cube[*YELLOW*][0] = {cube[*ORANGE*][0][0], cube[*ORANGE*][1][0], cube[*ORANGE*][2][0]};  
 for (int i = 0; i < 3; ++i) {  
 cube[*ORANGE*][i][0] = temp[i];  
 }  
 break;  
 case *RED*:  
 temp = {cube[*WHITE*][0][0], cube[*WHITE*][1][0], cube[*WHITE*][2][0]};  
 for (int i = 0; i < 3; ++i) {  
 cube[*WHITE*][i][0] = cube[*GREEN*][i][2];  
 }  
 for (int i = 0; i < 3; ++i) {  
 cube[*GREEN*][i][2] = cube[*YELLOW*][i][0];  
 }  
 for (int i = 0; i < 3; ++i) {  
 cube[*YELLOW*][i][0] = cube[*BLUE*][i][0];  
 }  
 for (int i = 0; i < 3; ++i) {  
 cube[*BLUE*][i][0] = temp[i];  
 }  
 break;  
  
 case *ORANGE*:  
 temp = {cube[*WHITE*][0][2], cube[*WHITE*][1][2], cube[*WHITE*][2][2]};  
 for (int i = 0; i < 3; ++i) {  
 cube[*WHITE*][i][2] = cube[*BLUE*][i][2];  
 }  
 for (int i = 0; i < 3; ++i) {  
 cube[*BLUE*][i][2] = cube[*YELLOW*][i][2];  
 }  
 for (int i = 0; i < 3; ++i) {  
 cube[*YELLOW*][i][2] = cube[*GREEN*][i][0];  
 }  
 for (int i = 0; i < 3; ++i) {  
 cube[*GREEN*][i][0] = temp[i];  
 }  
 break;  
 }  
}  
void RubiksCube::RotateFaceCounterclockwise(int face) {  
 // Поворачиваем грань против часовой стрелки  
 rotate\_matrix\_counterclockwise(cube[face]);  
  
 // Обновляем смежные грани  
 std::array<Color, 3> temp;  
 switch (face) {  
 case *WHITE*:  
 temp = {cube[*RED*][0][0], cube[*RED*][0][1], cube[*RED*][0][2]};  
 for (int i = 0; i < 3; ++i) {  
 cube[*RED*][0][i] = cube[*GREEN*][0][i];  
 cube[*GREEN*][0][i] = cube[*ORANGE*][0][i];  
 cube[*ORANGE*][0][i] = cube[*BLUE*][0][i];  
 cube[*BLUE*][0][i] = temp[i];  
 }  
 break;  
 case *YELLOW*:  
 temp = {cube[*BLUE*][2][0], cube[*BLUE*][2][1], cube[*BLUE*][2][2]};  
 cube[*BLUE*][2] = cube[*RED*][2];  
 cube[*RED*][2] = cube[*GREEN*][2];  
 cube[*GREEN*][2] = cube[*ORANGE*][2];  
 cube[*ORANGE*][2] = temp;  
 break;  
  
 case *BLUE*:  
 temp = {cube[*WHITE*][0][0], cube[*WHITE*][0][1], cube[*WHITE*][0][2]};  
 cube[*WHITE*][0] = {cube[*RED*][0][0], cube[*RED*][1][0], cube[*RED*][2][0]};  
 for (int i = 0; i < 3; ++i) {  
 cube[*RED*][i][0] = cube[*YELLOW*][2][i];  
 }  
 cube[*YELLOW*][2] = {cube[*ORANGE*][2][2], cube[*ORANGE*][1][2], cube[*ORANGE*][0][2]};  
 for (int i = 0; i < 3; ++i) {  
 cube[*ORANGE*][i][2] = temp[i];  
 }  
 break;  
  
 case *GREEN*:  
 temp = {cube[*WHITE*][2][0], cube[*WHITE*][2][1], cube[*WHITE*][2][2]};  
 cube[*WHITE*][2] = {cube[*ORANGE*][0][0], cube[*ORANGE*][1][0], cube[*ORANGE*][2][0]};  
 for (int i = 0; i < 3; ++i) {  
 cube[*ORANGE*][i][0] = cube[*YELLOW*][0][i];  
 }  
 cube[*YELLOW*][0] = {cube[*RED*][2][2], cube[*RED*][1][2], cube[*RED*][0][2]};  
 for (int i = 0; i < 3; ++i) {  
 cube[*RED*][i][2] = temp[i];  
 }  
 break;  
 case *RED*:  
 temp = {cube[*WHITE*][0][0], cube[*WHITE*][1][0], cube[*WHITE*][2][0]};  
 for (int i = 0; i < 3; ++i) {  
 cube[*WHITE*][i][0] = cube[*BLUE*][i][0];  
 }  
 for (int i = 0; i < 3; ++i) {  
 cube[*BLUE*][i][0] = cube[*YELLOW*][i][0];  
 }  
 for (int i = 0; i < 3; ++i) {  
 cube[*YELLOW*][i][0] = cube[*GREEN*][i][2];  
 }  
 for (int i = 0; i < 3; ++i) {  
 cube[*GREEN*][i][2] = temp[i];  
 }  
 break;  
  
 case *ORANGE*:  
 temp = {cube[*WHITE*][0][2], cube[*WHITE*][1][2], cube[*WHITE*][2][2]};  
 for (int i = 0; i < 3; ++i) {  
 cube[*WHITE*][i][2] = cube[*GREEN*][i][0];  
 }  
 for (int i = 0; i < 3; ++i) {  
 cube[*GREEN*][i][0] = cube[*YELLOW*][i][2];  
 }  
 for (int i = 0; i < 3; ++i) {  
 cube[*YELLOW*][i][2] = cube[*BLUE*][i][2];  
 }  
 for (int i = 0; i < 3; ++i) {  
 cube[*BLUE*][i][2] = temp[i];  
 }  
 break;  
 default:  
 std::cerr<<"CHECK WISE\n";  
 }  
}  
void RubiksCube::RotateCenterVC(int face) {  
 switch (face) {  
 case *WHITE*:  
 case *YELLOW*:  
 case *BLUE*:  
 case *GREEN*:  
 this->RotateFaceCounterclockwise(*RED*);  
 this->RotateFaceCounterclockwise(*ORANGE*);  
 break;  
 case *RED*:  
 case *ORANGE*:  
 this->RotateFaceCounterclockwise(*GREEN*);  
 this->RotateFaceCounterclockwise(*BLUE*);  
 break;  
 }  
}  
void RubiksCube::RotateCenterVCC(int face) {  
 switch(face) {  
 case *WHITE*:  
 case *YELLOW*:  
 case *BLUE*:  
 case *GREEN*:  
 this->RotateFaceClockwise(*RED*);  
 this->RotateFaceClockwise(*ORANGE*);  
 break;  
 case *RED*:  
 case *ORANGE*:  
 this->RotateFaceClockwise(*GREEN*);  
 this->RotateFaceClockwise(*BLUE*);  
 break;  
 }  
  
}  
void RubiksCube::RotateCenterHC(int face) {  
 switch (face) {  
 case *WHITE*:  
 case *YELLOW*:  
 this->RotateFaceCounterclockwise(*GREEN*);  
 this->RotateFaceCounterclockwise(*BLUE*);  
 break;  
 case *BLUE*:  
 case *GREEN*:  
 case *RED*:  
 case *ORANGE*:  
 this->RotateFaceCounterclockwise(*WHITE*);  
 this->RotateFaceCounterclockwise(*YELLOW*);  
 break;  
 }  
}  
void RubiksCube::RotateCenterHCC(int face) {  
 switch (face) {  
 case *WHITE*:  
 case *YELLOW*:  
 this->RotateFaceClockwise(*GREEN*);  
 this->RotateFaceClockwise(*BLUE*);  
 break;  
 case *BLUE*:  
 case *GREEN*:  
 case *RED*:  
 case *ORANGE*:  
 this->RotateFaceClockwise(*WHITE*);  
 this->RotateFaceClockwise(*YELLOW*);  
 break;  
 }  
}  
  
void RubiksCube::SpiningCube(std::filesystem::path FilePath) {  
 int RotCode = 1;  
 std::ofstream oCube(FilePath, std::ios::trunc);  
 if (!oCube.is\_open()) {  
 std::cerr << "File doesn't open1\n";  
 std::abort();  
 }  
 oCube << GetCreateCubeCode();  
 oCube.close();  
  
 while (RotCode!=0) {  
 std::cout << "\033[31m" << "Cube state now:" << "\033[0m" << std::endl;  
 this->Display();  
  
 std::cout << "\n\033[31m ROTATIONS CODE:\n\033[0m"  
 " | ClockWise | CounterClockWise | Center HC | Center HCC | Center VC | Center VCC |\n"  
 "WHITE | 11 | 21 | 31 | 41 | 51 | 61 |\n"  
 "YELLOW | 12 | 22 | 32 | 42 | 52 | 62 |\n"  
 "BLUE | 13 | 23 | 33 | 43 | 53 | 63 |\n"  
 "GREEN | 14 | 24 | 34 | 44 | 54 | 64 |\n"  
 "RED | 15 | 25 | 35 | 45 | 55 | 65 |\n"  
 "ORANGE | 16 | 26 | 36 | 46 | 56 | 66 |\n"  
 "\n\033[31mFor exit press: 0 (Programm has autosave)\n\n\033[0m"  
 "Please, enter code: ";  
 std::cin >> RotCode;  
 if (RotCode == 0) break;  
 switch (RotCode) {  
 case 11:  
 this->RotateFaceClockwise(*WHITE*);  
 break;  
 case 12:  
 this->RotateFaceClockwise(*YELLOW*);  
 break;  
 case 13:  
 this->RotateFaceClockwise(*BLUE*);  
 break;  
 case 14:  
 this->RotateFaceClockwise(*GREEN*);  
 break;  
 case 15:  
 this->RotateFaceClockwise(*RED*);  
 break;  
 case 16:  
 this->RotateFaceClockwise(*ORANGE*);  
 break;  
 case 21:  
 this->RotateFaceCounterclockwise(*WHITE*);  
 break;  
 case 22:  
 this->RotateFaceCounterclockwise(*YELLOW*);  
 break;  
 case 23:  
 this->RotateFaceCounterclockwise(*BLUE*);  
 break;  
 case 24:  
 this->RotateFaceCounterclockwise(*GREEN*);  
 break;  
 case 25:  
 this->RotateFaceCounterclockwise(*RED*);  
 break;  
 case 26:  
 this->RotateFaceCounterclockwise(*ORANGE*);  
 break;  
 case 31:  
 this->RotateCenterHC(*WHITE*);  
 break;  
 case 32:  
 this->RotateCenterHC(*YELLOW*);  
 break;  
 case 33:  
 this->RotateCenterHC(*BLUE*);  
 break;  
 case 34:  
 this->RotateCenterHC(*GREEN*);  
 break;  
 case 35:  
 this->RotateCenterHC(*RED*);  
 break;  
 case 36:  
 this->RotateCenterHC(*ORANGE*);  
 break;  
 case 41:  
 this->RotateCenterHCC(*WHITE*);  
 break;  
 case 42:  
 this->RotateCenterHCC(*YELLOW*);  
 break;  
 case 43:  
 this->RotateCenterHCC(*BLUE*);  
 break;  
 case 44:  
 this->RotateCenterHCC(*GREEN*);  
 break;  
 case 45:  
 this->RotateCenterHCC(*RED*);  
 break;  
 case 46:  
 this->RotateCenterHCC(*ORANGE*);  
 break;  
 case 51:  
 this->RotateCenterVC(*WHITE*);  
 break;  
 case 52:  
 this->RotateCenterVC(*YELLOW*);  
 break;  
 case 53:  
 this->RotateCenterVC(*BLUE*);  
 break;  
 case 54:  
 this->RotateCenterVC(*GREEN*);  
 break;  
 case 55:  
 this->RotateCenterVC(*RED*);  
 break;  
 case 56:  
 this->RotateCenterVC(*ORANGE*);  
 break;  
 case 61:  
 this->RotateCenterVCC(*WHITE*);  
 break;  
 case 62:  
 this->RotateCenterVCC(*YELLOW*);  
 break;  
 case 63:  
 this->RotateCenterVCC(*BLUE*);  
 break;  
 case 64:  
 this->RotateCenterVCC(*GREEN*);  
 break;  
 case 65:  
 this->RotateCenterVCC(*RED*);  
 break;  
 case 66:  
 this->RotateCenterVCC(*ORANGE*);  
 break;  
 default:  
 std::cout<<"Incorrect code!\n";  
 continue;  
 }  
  
 std::ifstream iCubeFile(FilePath);  
 std::string CubeContent;  
 if (!iCubeFile.is\_open()) {  
 std::cerr << "File doesn't open2\n";  
 std::abort();  
 }  
 iCubeFile>>CubeContent;  
 iCubeFile.close();  
  
 CubeContent += std::to\_string(RotCode);  
 CubeContent.replace(0, 162, GetCreateCubeCode());  
 std::ofstream oCubeFile(FilePath);  
 if (!oCubeFile.is\_open()) {  
 std::cerr << "File doesn't open3\n";  
 std::abort();  
 }  
 oCubeFile << CubeContent;  
 oCubeFile.close();  
 std::cout << "\033[2J\033[H";  
 }  
}  
  
std::string RubiksCube::GetCreateCubeCode() {  
 std::vector<char> mColor = **{**'W', 'Y', 'B', 'G', 'R', 'O'**}**;  
 std::string CubeState;  
 int count = 1;  
 for (int row = 0; row < 3; ++row) {  
 for (int col = 0; col < 3; ++col) {  
 CubeState+=CreateAllCode(*WHITE*, count, mColor[cube[*WHITE*][row][col]]); count++;  
 }  
 }  
 count = 1;  
 for (int row = 0; row < 3; ++row) {  
 for (int col = 0; col < 3; ++col) {  
 CubeState+=CreateAllCode(*YELLOW*, count, mColor[cube[*YELLOW*][row][col]]); count++;  
 }  
 }  
 count = 1;  
 for (int row = 0; row < 3; ++row) {  
 for (int col = 0; col < 3; ++col) {  
 CubeState+=CreateAllCode(*BLUE*, count, mColor[cube[*BLUE*][row][col]]); count++;  
 }  
 }  
 count = 1;  
 for (int row = 0; row < 3; ++row) {  
 for (int col = 0; col < 3; ++col) {  
 CubeState+=CreateAllCode(*GREEN*, count, mColor[cube[*GREEN*][row][col]]); count++;  
 }  
 }  
 count = 1;  
 for (int row = 0; row < 3; ++row) {  
 for (int col = 0; col < 3; ++col) {  
 CubeState+=CreateAllCode(*RED*, count, mColor[cube[*RED*][row][col]]); count++;  
 }  
 }  
 count = 1;  
 for (int row = 0; row < 3; ++row) {  
 for (int col = 0; col < 3; ++col) {  
 CubeState+=CreateAllCode(*ORANGE*, count, mColor[cube[*ORANGE*][row][col]]); count++;  
 }  
 }  
 if (!Rotations.empty()){  
 std::stack<int> TempST;  
 while (!Rotations.empty()){  
 TempST.push(Rotations.top());  
 Rotations.pop();  
 }  
 while (!TempST.empty()){  
 CubeState+=std::to\_string(TempST.top());  
 TempST.pop();  
 }  
 }  
 return CubeState;  
}  
std::string RubiksCube::CreateAllCode(Color NumWise, int CurPos, char CurColor) {  
 std::string Code;  
 Code+=std::to\_string(NumWise+1);  
 switch (NumWise) {  
 case *WHITE*:  
 Code+=CreatePosCode(CurPos, CurColor);  
 break;  
 case *YELLOW*:  
 Code+=CreatePosCode(CurPos, CurColor);  
 break;  
 case *BLUE*:  
 Code+=CreatePosCode(CurPos, CurColor);  
 break;  
 case *GREEN*:  
 Code+=CreatePosCode(CurPos, CurColor);  
 break;  
 case *RED*:  
 Code+=CreatePosCode(CurPos, CurColor);  
 break;  
 case *ORANGE*:  
 Code+=CreatePosCode(CurPos, CurColor);  
 break;  
 }  
 return Code;  
}  
std::string RubiksCube::CreatePosCode(int CurPos, char CurColor){  
 std::string Code;  
 Code+=std::to\_string(CurPos);  
 switch (CurColor){  
 case ('W'):  
 Code+="1";  
 break;  
 case ('Y'):  
 Code+="2";  
 break;  
 case ('B'):  
 Code+="3";  
 break;  
 case ('G'):  
 Code+="4";  
 break;  
 case ('R'):  
 Code+="5";  
 break;  
 case ('O'):  
 Code+="6";  
 break;  
 }  
 return Code;  
}  
  
void RubiksCube::SetCubeState(std::string RawData) {  
 for (int pos = 0; pos < 162; pos += 3) {  
 cube[GetSTRWiseCode(RawData, pos)][GetSTRPairPos(RawData, pos).first][GetSTRPairPos(RawData, pos).second] = GetSTRColor(  
 RawData, pos);  
 }  
 for (int pos = 162; pos < RawData.size(); pos+=2) {  
 std::string RotTemp;  
 RotTemp += RawData[pos];  
 RotTemp += RawData[pos+1];  
 Rotations.push(std::stoi(RotTemp));  
 }  
}  
Color RubiksCube::GetSTRWiseCode(const std::string &RawData, int pos) {  
 switch(RawData[pos]){  
 case '1': return *WHITE*;  
 case '2': return *YELLOW*;  
 case '3': return *BLUE*;  
 case '4': return *GREEN*;  
 case '5': return *RED*;  
 case '6': return *ORANGE*;  
 }  
}  
std::pair<int, int> RubiksCube::GetSTRPairPos(const std::string &RawData, int pos) {  
 switch(RawData[pos+1]){  
 case '1': return std::make\_pair(0, 0);  
 case '2': return std::make\_pair(0, 1);  
 case '3': return std::make\_pair(0, 2);  
 case '4': return std::make\_pair(1, 0);  
 case '5': return std::make\_pair(1, 1);  
 case '6': return std::make\_pair(1, 2);  
 case '7': return std::make\_pair(2, 0);  
 case '8': return std::make\_pair(2, 1);  
 case '9': return std::make\_pair(2, 2);  
 }  
}  
Color RubiksCube::GetSTRColor(const std::string &RawData, int pos) {  
 switch(RawData[pos+2]){  
 case '1': return *WHITE*;  
 case '2': return *YELLOW*;  
 case '3': return *BLUE*;  
 case '4': return *GREEN*;  
 case '5': return *RED*;  
 case '6': return *ORANGE*;  
 }  
}  
  
void RubiksCube::SetCubeFromFile(std::filesystem::path FilePath) {  
 std::ifstream iCubeFile(FilePath);  
 std::string CubeContent;  
 if (!iCubeFile.is\_open()) {  
 std::cerr << "File doesn't open4\n";  
 std::abort();  
 }  
 iCubeFile>>CubeContent;  
 iCubeFile.close();  
 SetCubeState(CubeContent);  
}  
  
void RubiksCube::SolveCube(std::filesystem::path FilePath) {  
 std::ifstream iCubeFile(FilePath);  
 std::string CubeContent;  
 if (!iCubeFile.is\_open()) {  
 std::cerr << "File doesn't open4\n";  
 std::abort();  
 }  
 iCubeFile>>CubeContent;  
 iCubeFile.close();  
 SetCubeState(CubeContent);  
 DisplaySolve();  
}  
void RubiksCube::DisplaySolve() {  
 std::cout << "\033[31m" << "Cube to be solved:" << "\033[0m" << std::endl;  
 this->Display();  
 usleep(1000000);  
 std::cout << "\033[2J\033[H";  
 while (!Rotations.empty()){  
 int RotCode = Rotations.top();  
 Rotations.pop();  
 std::cout << "\033[31m" << "Now rotate: " << RotCode <<"\033[0m" << std::endl;  
 switch (RotCode) {  
 case 11:  
 this->RotateFaceCounterclockwise(*WHITE*);  
 break;  
 case 12:  
 this->RotateFaceCounterclockwise(*YELLOW*);  
 break;  
 case 13:  
 this->RotateFaceCounterclockwise(*BLUE*);  
 break;  
 case 14:  
 this->RotateFaceCounterclockwise(*GREEN*);  
 break;  
 case 15:  
 this->RotateFaceCounterclockwise(*RED*);  
 break;  
 case 16:  
 this->RotateFaceCounterclockwise(*ORANGE*);  
 break;  
 case 21:  
 this->RotateFaceClockwise(*WHITE*);  
 break;  
 case 22:  
 this->RotateFaceClockwise(*YELLOW*);  
 break;  
 case 23:  
 this->RotateFaceClockwise(*BLUE*);  
 break;  
 case 24:  
 this->RotateFaceClockwise(*GREEN*);  
 break;  
 case 25:  
 this->RotateFaceClockwise(*RED*);  
 break;  
 case 26:  
 this->RotateFaceClockwise(*ORANGE*);  
 break;  
 case 31:  
 this->RotateCenterHCC(*WHITE*);  
 break;  
 case 32:  
 this->RotateCenterHCC(*YELLOW*);  
 break;  
 case 33:  
 this->RotateCenterHCC(*BLUE*);  
 break;  
 case 34:  
 this->RotateCenterHCC(*GREEN*);  
 break;  
 case 35:  
 this->RotateCenterHCC(*RED*);  
 break;  
 case 36:  
 this->RotateCenterHCC(*ORANGE*);  
 break;  
 case 41:  
 this->RotateCenterHC(*WHITE*);  
 break;  
 case 42:  
 this->RotateCenterHC(*YELLOW*);  
 break;  
 case 43:  
 this->RotateCenterHC(*BLUE*);  
 break;  
 case 44:  
 this->RotateCenterHC(*GREEN*);  
 break;  
 case 45:  
 this->RotateCenterHC(*RED*);  
 break;  
 case 46:  
 this->RotateCenterHC(*ORANGE*);  
 break;  
 case 51:  
 this->RotateCenterVCC(*WHITE*);  
 break;  
 case 52:  
 this->RotateCenterVCC(*YELLOW*);  
 break;  
 case 53:  
 this->RotateCenterVCC(*BLUE*);  
 break;  
 case 54:  
 this->RotateCenterVCC(*GREEN*);  
 break;  
 case 55:  
 this->RotateCenterVCC(*RED*);  
 break;  
 case 56:  
 this->RotateCenterVCC(*ORANGE*);  
 break;  
 case 61:  
 this->RotateCenterVC(*WHITE*);  
 break;  
 case 62:  
 this->RotateCenterVC(*YELLOW*);  
 break;  
 case 63:  
 this->RotateCenterVC(*BLUE*);  
 break;  
 case 64:  
 this->RotateCenterVC(*GREEN*);  
 break;  
 case 65:  
 this->RotateCenterVC(*RED*);  
 break;  
 case 66:  
 this->RotateCenterVC(*ORANGE*);  
 break;  
 default:  
 std::cout<<"Incorrect code!\n";  
 continue;  
 }  
 this->Display();  
 usleep(1000000);  
 std::cout << "\033[2J\033[H";  
 }  
 std::cout << "\033[31m" << "Cube solved: \033[0m" << std::endl;  
 this->Display();  
}  
  
void RubiksCube::CheckCorrectCube(std::filesystem::path FilePath) {  
 std::ifstream iCubeFile(FilePath);  
 std::string CubeContent;  
 if (!iCubeFile.is\_open()) {  
 std::cerr << "File doesn't open4\n";  
 std::abort();  
 }  
 iCubeFile>>CubeContent;  
 iCubeFile.close();  
 if (CubeContent.size() >= 162 && CubeContent.size() %2 == 0){  
 std::cout<<"Cube is correct\n";  
 }  
 else{  
 std::cout<<"Cube is incorrect!\n";  
 }  
}  
  
  
int RubiksCube::GenerateRandomSpin() {  
 std::random\_device rd;  
 std::mt19937 gen(rd());  
 std::uniform\_int\_distribution<int> SDist(1, 6);  
 std::uniform\_int\_distribution<int> BDist(1, 6);  
 int SmallDist = SDist(gen);  
 int BigDist = BDist(gen);  
 int res = (BigDist \* 10) + SmallDist;  
 return res;  
}  
  
void RubiksCube::GenerateSpinCube(std::filesystem::path FilePath, int CountSpin) {  
 int RotCode = 1;  
 std::ofstream oCube(FilePath, std::ios::trunc);  
 if (!oCube.is\_open()) {  
 std::cerr << "File doesn't open1\n";  
 std::abort();  
 }  
 oCube << GetCreateCubeCode();  
 oCube.close();  
 for (int i = 0; i < CountSpin; ++i){  
 RotCode = GenerateRandomSpin();  
 switch (RotCode) {  
 case 11:  
 this->RotateFaceClockwise(*WHITE*);  
 break;  
 case 12:  
 this->RotateFaceClockwise(*YELLOW*);  
 break;  
 case 13:  
 this->RotateFaceClockwise(*BLUE*);  
 break;  
 case 14:  
 this->RotateFaceClockwise(*GREEN*);  
 break;  
 case 15:  
 this->RotateFaceClockwise(*RED*);  
 break;  
 case 16:  
 this->RotateFaceClockwise(*ORANGE*);  
 break;  
 case 21:  
 this->RotateFaceCounterclockwise(*WHITE*);  
 break;  
 case 22:  
 this->RotateFaceCounterclockwise(*YELLOW*);  
 break;  
 case 23:  
 this->RotateFaceCounterclockwise(*BLUE*);  
 break;  
 case 24:  
 this->RotateFaceCounterclockwise(*GREEN*);  
 break;  
 case 25:  
 this->RotateFaceCounterclockwise(*RED*);  
 break;  
 case 26:  
 this->RotateFaceCounterclockwise(*ORANGE*);  
 break;  
 case 31:  
 this->RotateCenterHC(*WHITE*);  
 break;  
 case 32:  
 this->RotateCenterHC(*YELLOW*);  
 break;  
 case 33:  
 this->RotateCenterHC(*BLUE*);  
 break;  
 case 34:  
 this->RotateCenterHC(*GREEN*);  
 break;  
 case 35:  
 this->RotateCenterHC(*RED*);  
 break;  
 case 36:  
 this->RotateCenterHC(*ORANGE*);  
 break;  
 case 41:  
 this->RotateCenterHCC(*WHITE*);  
 break;  
 case 42:  
 this->RotateCenterHCC(*YELLOW*);  
 break;  
 case 43:  
 this->RotateCenterHCC(*BLUE*);  
 break;  
 case 44:  
 this->RotateCenterHCC(*GREEN*);  
 break;  
 case 45:  
 this->RotateCenterHCC(*RED*);  
 break;  
 case 46:  
 this->RotateCenterHCC(*ORANGE*);  
 break;  
 case 51:  
 this->RotateCenterVC(*WHITE*);  
 break;  
 case 52:  
 this->RotateCenterVC(*YELLOW*);  
 break;  
 case 53:  
 this->RotateCenterVC(*BLUE*);  
 break;  
 case 54:  
 this->RotateCenterVC(*GREEN*);  
 break;  
 case 55:  
 this->RotateCenterVC(*RED*);  
 break;  
 case 56:  
 this->RotateCenterVC(*ORANGE*);  
 break;  
 case 61:  
 this->RotateCenterVCC(*WHITE*);  
 break;  
 case 62:  
 this->RotateCenterVCC(*YELLOW*);  
 break;  
 case 63:  
 this->RotateCenterVCC(*BLUE*);  
 break;  
 case 64:  
 this->RotateCenterVCC(*GREEN*);  
 break;  
 case 65:  
 this->RotateCenterVCC(*RED*);  
 break;  
 case 66:  
 this->RotateCenterVCC(*ORANGE*);  
 break;  
 default:  
 std::cout<<"Incorrect code!\n";  
 continue;  
 }  
  
 std::ifstream iCubeFile(FilePath);  
 std::string CubeContent;  
 if (!iCubeFile.is\_open()) {  
 std::cerr << "File doesn't open2\n";  
 std::abort();  
 }  
 iCubeFile>>CubeContent;  
 iCubeFile.close();  
  
 CubeContent += std::to\_string(RotCode);  
 CubeContent.replace(0, 162, GetCreateCubeCode());  
 std::ofstream oCubeFile(FilePath);  
 if (!oCubeFile.is\_open()) {  
 std::cerr << "File doesn't open3\n";  
 std::abort();  
 }  
 oCubeFile << CubeContent;  
 oCubeFile.close();  
 std::cout << "\033[2J\033[H";  
 }  
 std::cout << "\033[31m" << "Generated cube is:" << "\033[0m" << std::endl;  
 this->Display();  
}  
  
// ROTATIONS CODE:  
// | ClockWise | CounterClockWise | Center HC | Center HCC | Center VC | Center VCC |  
//WHITE | 11 | 21 | 31 | 41 | 51 | 61 |  
//YELLOW | 12 | 22 | 32 | 42 | 52 | 62 |  
//BLUE | 13 | 23 | 33 | 43 | 53 | 63 |  
//GREEN | 14 | 24 | 34 | 44 | 54 | 64 |  
//RED | 15 | 25 | 35 | 45 | 55 | 65 |  
//ORANGE | 16 | 26 | 36 | 46 | 56 | 66 |  
  
// FILE STRUCT:  
//Need 54 symbols (for 1 cube state)  
//Have 6 faces, one face - one color  
  
// NUM FACE:| NUM COLOR:|  
//WHITE | 1 | 1 |  
//YELLOW | 2 | 2 |  
//BLUE | 3 | 3 |  
//GREEN | 4 | 4 |  
//RED | 5 | 5 |  
//ORANGE | 6 | 6 |  
  
//NUM POSITIONS:  
// 1 2 3  
// 4 5 6  
// 7 8 9  
  
//FIRST - num face, SECOND - num pos, THIRD - num color

**Программа запустилась и работает корректно:**

**Изображение выглядит как текст, снимок экрана, дизайн

Автоматически созданное описание**

****

