Министерство 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 № M3111
Гаврилов Алексей Евгеньевич
Подпись:

Проверил:
Повышев Владислав Вячеславович

Санкт-Петербург
2022

Текст задания

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 C++ включая стандартную библиотеку

Решение с комментариями

//Cube.h

```
//
// Created by Volirvag on 26.05.2022.
//

#pragma once
#include<vector>
#include<iostream>
#include<fstream>
#include<string>
#include <windows.h>
using namespace std;
using vector3d = vector<vector<vector<char>>>;


//sides[0] - лицевая сторона
//sides[1] - задняя сторона
//sides[2] - левая сторона
//sides[3] - правая сторона
//sides[4] - верхняя сторона
//sides[5] - нижняя сторона
class Cube {
private:

    vector3d sides;
    string str;
    short k;
    void TurnOn(vector<vector<char>> &side) {
        vector<vector<char>> side_(3, vector<char>(3));
        int ind;
        for (int i = 0; i < 3; i++) {
            ind = 2;
            for (int j = 0; j < 3; j++) {
                side_[i][j] = side[ind][i];
                ind--;
            }
        }

        side = side_;
    }

    void TurnAgainst(vector<vector<char>> &side) {
        vector<vector<char>> side_(3, vector<char>(3));
        int ind;
        for (int i = 0; i < 3; i++) {
            ind = 0;
            for (int j = 0; j < 3; j++) {
                side_[i][j] = side[ind][2 - i];
                ind++;
            }
        }

        side = side_;
    }

    void Krest() {
        while (sides[0][0][1] != 'w' || sides[0][1][0] != 'w' ||
sides[0][1][2] != 'w' || sides[0][2][1] != 'w') {
            if (sides[1][0][1] == 'w') {
                if (sides[0][0][1] == 'w') {
                    while (sides[0][0][1] == 'w') {
                        F_On();
```

```
                }
            }
            U_Twice();
        }

        if (sides[1][1][0] == 'w') {
            if (sides[0][1][2] == 'w') {
                while (sides[0][1][2] == 'w') {
                    F_On();
                }
            }
            R_Twice();
        }

        if (sides[1][1][2] == 'w') {
            if (sides[0][1][0] == 'w') {
                while (sides[0][1][0] == 'w') {
                    F_On();
                }
            }
            L_Twice();
        }

        if (sides[1][2][1] == 'w') {
            if (sides[0][2][1] == 'w') {
                while (sides[0][2][1] == 'w') {
                    F_On();
                }
            }
            D_Twice();
        }

        if (sides[2][1][2] == 'w') {
            L_On();
            if (sides[0][2][1] == 'w') {
                while (sides[0][2][1] == 'w') {
                    F_On();
                }
            }
            D_On();
        }

        if (sides[2][1][0] == 'w') {
            L_On();
            if (sides[0][0][1] == 'w') {
                while (sides[0][0][1] == 'w') {
                    F_On();
                }
            }
            U_Against();
        }

        if (sides[2][0][1] == 'w') {
            if (sides[0][0][1] == 'w') {
                while (sides[0][0][1] == 'w') {
                    F_On();
                }
            }
            U_Against();
        }

        if (sides[2][2][1] == 'w') {
            if (sides[0][2][1] == 'w') {
                while (sides[0][2][1] == 'w') {
```

```
                F_On();
            }
        }
        D_On();
    }

    if (sides[3][1][0] == 'w') {
        R_On();
        if (sides[0][0][1] == 'w') {
            while (sides[0][0][1] == 'w')
                F_On();

        }
        U_On();
    }

    if (sides[3][1][2] == 'w') {
        R_On();
        if (sides[0][2][1] == 'w') {
            while (sides[0][2][1] == 'w')
                F_On();
        }
        D_Against();
    }

    if (sides[3][0][1] == 'w') {
        if (sides[0][0][1] == 'w') {
            while (sides[0][0][1] == 'w')
                F_On();
        }
        U_On();
    }

    if (sides[3][2][1] == 'w') {
        if (sides[0][2][1] == 'w') {
            while (sides[0][2][1] == 'w')
                F_On();
        }
        D_Against();
    }

    if (sides[4][2][1] == 'w') {
        U_On();
        if (sides[0][1][0] == 'w') {
            while (sides[0][1][0] == 'w')
                F_On();
        }
        L_On();
    }

    if (sides[4][0][1] == 'w') {
        U_On();
        if (sides[0][1][2] == 'w') {
            while (sides[0][1][2] == 'w')
                F_On();
        }
        R_Against();
    }

    if (sides[4][1][0] == 'w') {
        if (sides[0][1][0] == 'w') {
            while (sides[0][1][0] == 'w')
                F_On();
        }
```

```
            L_On();
        }

        if (sides[4][1][2] == 'w') {
            if (sides[0][1][2] == 'w') {
                while (sides[0][1][2] == 'w')
                    F_On();
            }
            R_Against();
        }

        if (sides[5][0][1] == 'w') {
            D_On();
            if (sides[0][1][2] == 'w') {
                while (sides[0][1][2] == 'w')
                    F_On();
            }
            R_On();
        }

        if (sides[5][2][1] == 'w') {
            D_On();
            if (sides[0][1][0] == 'w') {
                while (sides[0][1][0] == 'w')
                    F_On();
            }
            L_Against();
        }

        if (sides[5][1][2] == 'w') {
            if (sides[0][1][2] == 'w') {
                while (sides[0][1][2] == 'w')
                    F_On();
            }
            R_On();
        }

        if (sides[5][1][0] == 'w') {
            if (sides[0][1][0] == 'w') {
                while (sides[0][1][0] == 'w')
                    F_On();
            }
            L_Against();
        }
    }
}

void PerfectKrest() {
    while (sides[2][1][1] != sides[2][1][2])
        F_On();

    if (sides[3][1][1] != sides[3][1][0]) {
        switch (sides[3][1][0]) {
            case 'r':
                R_Twice();
                B_Against();
                D_Twice();
                B_On();
                break;
            case 'o':
                R_Twice();
                B_On();
                U_Twice();
                B_Against();
```

```
                    break;
            }
            R_Twice();
        }

        if (sides[4][1][1] != sides[4][2][1]) {
            switch (sides[4][2][1]) {
                case 'b':
                    U_Twice();
                    B_Against();
                    R_Twice();
                    B_On();
                    break;
                case 'r':
                    U_Twice();
                    B_Twice();
                    D_Twice();
                    B_Twice();
                    break;
            }
            U_Twice();
        }

        if (sides[5][1][1] != sides[5][0][1]) {
            switch (sides[5][0][1]) {
                case 'b':
                    D_Twice();
                    B_On();
                    R_Twice();
                    B_Against();
                    break;
                case 'o':
                    D_Twice();
                    B_Twice();
                    U_Twice();
                    B_Twice();
                    break;
            }
            D_Twice();
        }

        CheckPerfectKrest();

        //Out();
        k++;
    }

    void CheckPerfectKrest() {
        while (sides[3][1][1] != sides[3][1][0] || sides[4][1][1] !=
sides[4][2][1] || sides[5][1][1] != sides[5][0][1])
            PerfectKrest();
    }

    void FirstLay() {
        while ((sides[0][0][0] != 'w' || sides[2][0][2] != sides[2][1][1] ||
sides[4][2][0] != sides[4][1][1]) ||
               (sides[0][0][2] != 'w' || sides[3][0][0] != sides[3][1][1] ||
sides[4][2][2] != sides[4][1][1]) ||
               (sides[0][2][0] != 'w' || sides[2][2][2] != sides[2][1][1] ||
sides[5][0][0] != sides[5][1][1]) ||
               (sides[0][2][2] != 'w' || sides[3][2][0] != sides[3][1][1] ||
sides[5][0][2] != sides[5][1][1])) {
            if (sides[1][0][0] == 'w' || sides[1][0][2] == 'w' ||
sides[1][2][0] == 'w' || sides[1][2][2] == 'w' ||
```

```
                sides[3][0][2] == 'w' || sides[3][2][2] == 'w' ||
sides[4][0][0] == 'w' || sides[4][0][2] == 'w' ||
                sides[2][0][0] == 'w' || sides[2][2][0] == 'w' ||
sides[5][2][0] == 'w' || sides[5][2][2] == 'w') {
                bool flag = false;
                if (sides[1][0][0] == 'w' || sides[3][0][2] == 'w' ||
sides[4][0][2] == 'w')
                    flag = true;
                while (!flag) {
                    B_On();
                    if (sides[1][0][0] == 'w' || sides[3][0][2] == 'w' ||
sides[4][0][2] == 'w')
                        flag = true;
                }

                vector<char> arr_color(2);
                if (sides[1][0][0] == 'w') {
                    arr_color[0] = sides[3][0][2];
                    arr_color[1] = sides[4][0][2];
                } else if (sides[3][0][2] == 'w') {
                    arr_color[0] = sides[1][0][0];
                    arr_color[1] = sides[4][0][2];
                } else {
                    arr_color[0] = sides[3][0][2];
                    arr_color[1] = sides[1][0][0];
                }

                if ((arr_color[0] == 'b' && arr_color[1] == 'o') ||
(arr_color[1] == 'b' && arr_color[0] == 'o')) {
                    while (sides[0][0][2] != 'w' || sides[4][2][2] != 'o' ||
sides[3][0][0] != 'b') {
                        R_On();
                        B_On();
                        R_Against();
                        B_Against();
                    }
                }

                if ((arr_color[0] == 'r' && arr_color[1] == 'g') ||
(arr_color[1] == 'r' && arr_color[0] == 'g')) {
                    B_Against();
                    B_Against();
                    while (sides[0][2][0] != 'w' || sides[5][0][0] != 'r' ||
sides[2][2][2] != 'g') {
                        L_On();
                        B_On();
                        L_Against();
                        B_Against();
                    }
                }

                if ((arr_color[0] == 'r' && arr_color[1] == 'b') ||
(arr_color[1] == 'r' && arr_color[0] == 'b')) {
                    B_Against();
                    while (sides[0][2][2] != 'w' || sides[5][0][2] != 'r' ||
sides[3][2][0] != 'b') {
                        D_On();
                        B_On();
                        D_Against();
                        B_Against();
                    }
                }

                if ((arr_color[0] == 'o' && arr_color[1] == 'g') ||
```

```
(arr_color[1] == 'o' && arr_color[0] == 'g')) {
                    B_On();
                    while (sides[0][0][0] != 'w' || sides[4][2][0] != 'o' ||
sides[2][0][2] != 'g') {
                        L_Against();
                        B_Against();
                        L_On();
                        B_On();
                    }
                }
            }

            if (sides[2][0][2] == 'w' || sides[4][2][0] == 'w' ||
                (sides[0][0][0] == 'w' && (sides[2][0][2] != sides[2][1][1]
|| sides[4][2][0] != sides[4][1][1]))) {
                U_On();
                B_On();
                U_Against();
                B_Against();
            }

            if (sides[2][2][2] == 'w' || sides[5][0][0] == 'w' ||
                (sides[0][2][0] == 'w' && (sides[2][2][2] != sides[2][1][1]
|| sides[5][0][0] != sides[5][1][1]))) {
                L_On();
                B_On();
                L_Against();
                B_Against();
            }

            if (sides[5][0][2] == 'w' || sides[3][2][0] == 'w' ||
                (sides[0][2][2] == 'w' && (sides[5][0][2] != sides[5][1][1]
|| sides[3][2][0] != sides[3][1][1]))) {
                D_On();
                B_On();
                D_Against();
                B_Against();
            }

            if (sides[3][0][0] == 'w' || sides[4][2][2] == 'w' ||
                (sides[0][0][2] == 'w' && (sides[3][0][0] != sides[3][1][1]
|| sides[4][2][2] != sides[4][1][1]))) {
                R_On();
                B_On();
                R_Against();
                B_Against();
            }
        }
        //Out();
        k++;
    }

    void SecondLay() {
        while ((sides[2][0][1] != sides[2][1][1] || sides[2][2][1] !=
sides[2][1][1]) ||
               (sides[3][0][1] != sides[3][1][1] || sides[3][2][1] !=
sides[3][1][1]) ||
               (sides[4][1][0] != sides[4][1][1] || sides[4][1][2] !=
sides[4][1][1]) ||
               (sides[5][1][0] != sides[5][1][1] || sides[5][1][2] !=
sides[5][1][1])) {
            while ((sides[2][1][0] != 'y' && sides[1][1][2] != 'y') ||
                   (sides[4][0][1] != 'y' && sides[1][0][1] != 'y') ||
                   (sides[3][1][2] != 'y' && sides[1][1][0] != 'y') ||
```

```
    (sides[5][2][1] != 'y' && sides[1][2][1] != 'y')) {
if (sides[3][1][2] == 'b' && sides[1][1][0] != 'y') {
    if (sides[1][1][0] == 'r') {
        B_On();
        D_On();
        B_On();
        D_Against();
        B_Against();
        R_Against();
        B_Against();
        R_On();
        B_On();
    } else if (sides[1][1][0] == 'o') {
        B_Against();
        U_Against();
        B_Against();
        U_On();
        B_On();
        R_On();
        B_On();
        R_Against();
        B_Against();
    }
}

if (sides[2][1][0] == 'g' && sides[1][1][2] != 'y') {
    if (sides[1][1][2] == 'o') {
        B_On();
        U_On();
        B_On();
        U_Against();
        B_Against();
        L_Against();
        B_Against();
        L_On();
        B_On();
    } else if (sides[1][1][2] == 'r') {
        B_Against();
        D_Against();
        B_Against();
        D_On();
        B_On();
        L_On();
        B_On();
        L_Against();
        B_Against();
    }
}

if (sides[4][0][1] == 'o' && sides[1][0][1] != 'y') {
    if (sides[1][0][1] == 'b') {
        B_On();
        R_On();
        B_On();
        R_Against();
        B_Against();
        U_Against();
        B_Against();
        U_On();
        B_On();
    } else if (sides[1][0][1] == 'g') {
        B_Against();
        L_Against();
        B_Against();
```

```
                L_On();
                B_On();
                U_On();
                B_On();
                U_Against();
                B_Against();
            }
        }

        if (sides[5][2][1] == 'r' && sides[1][2][1] != 'y') {
            if (sides[1][2][1] == 'g') {
                B_On();
                L_On();
                B_On();
                L_Against();
                B_Against();
                D_Against();
                B_Against();
                D_On();
                B_On();
            } else if (sides[1][2][1] == 'b') {
                B_Against();
                R_Against();
                B_Against();
                R_On();
                B_On();
                D_On();
                B_On();
                D_Against();
                B_Against();
            }
        }
        B_On();
    }


    if (sides[2][0][1] != 'g' || sides[4][1][0] != 'o') {
        B_On();
        U_On();
        B_On();
        U_Against();
        B_Against();
        L_Against();
        B_Against();
        L_On();
        B_On();
    }

    if (sides[2][2][1] != 'g' || sides[5][1][0] != 'r') {
        B_Against();
        D_Against();
        B_Against();
        D_On();
        B_On();
        L_On();
        B_On();
        L_Against();
        B_Against();
    }

    if (sides[3][2][1] != 'b' || sides[5][1][2] != 'r') {
        B_On();
        D_On();
        B_On();
```

```
                D_Against();
                B_Against();
                R_Against();
                B_Against();
                R_On();
                B_On();
            }

            if (sides[3][0][1] != 'b' || sides[4][1][2] != 'o') {
                B_Against();
                U_Against();
                B_Against();
                U_On();
                B_On();
                R_On();
                B_On();
                R_Against();
                B_Against();
            }
        }
        //Out();
        k++;
    }

    void UpperKrest() {
        while (sides[1][0][1] != 'y' || sides[1][1][0] != 'y' ||
sides[1][1][2] != 'y' || sides[1][2][1] != 'y') {
            if ((sides[1][1][0] == 'y' && sides[1][1][2] == 'y' &&
sides[1][0][1] != 'y' && sides[1][2][1] != 'y') ||
                (sides[1][1][0] != 'y' && sides[1][1][2] != 'y' &&
sides[1][0][1] != 'y' && sides[1][2][1] != 'y')) {
                D_On();
                L_On();
                B_On();
                L_Against();
                B_Against();
                D_Against();
            }

            if (sides[1][0][1] == 'y' && sides[1][1][0] == 'y' &&
sides[1][1][2] != 'y' && sides[1][2][1] != 'y') {
                D_On();
                L_On();
                B_On();
                L_Against();
                B_Against();
                L_On();
                B_On();
                L_Against();
                B_Against();
                D_Against();
            }
            B_On();
        }

        //Out();
        k++;
    }

    void CornerEl() {
        while (sides[1][0][0] != 'y' || sides[1][0][2] != 'y' ||
sides[1][2][0] != 'y' || sides[1][2][2] != 'y') {
            while (sides[1][0][0] != 'y') {
                F_On();
```

```
                U_On();
                F_Against();
                U_Against();
            }
            B_On();
        }

        CheckCornerEl();

        //Out();
        k++;
    }

    void CheckCornerEl() {
        bool flag = false;

        while (!flag) {
            flag = true;
            for (int i = 0; i < 3; i++) {
                for (int j = 0; j < 3; j++)
                    if (sides[0][i][j] != sides[0][1][1] || sides[1][i][j] !=
sides[1][1][1])
                        flag = false;
            }

            for (int i = 0; i < 3; i++) {
                for (int j = 1; j < 3; j++)
                    if (sides[2][i][j] != sides[2][1][1])
                        flag = false;
            }

            for (int i = 0; i < 3; i++) {
                for (int j = 0; j < 2; j++)
                    if (sides[3][i][j] != sides[3][1][1])
                        flag = false;
            }

            for (int i = 1; i < 3; i++) {
                for (int j = 0; j < 3; j++)
                    if (sides[4][i][j] != sides[4][1][1])
                        flag = false;
            }

            for (int i = 0; i < 2; i++) {
                for (int j = 0; j < 3; j++)
                    if (sides[5][i][j] != sides[5][1][1])
                        flag = false;
            }

            if (!flag) {
                F_On();
                U_On();
                F_Against();
                U_Against();
            }
        }
    }

    void UpperEdge() {
        while (sides[2][1][0] != sides[2][1][1] || sides[3][1][2] !=
sides[3][1][1] ||
               sides[4][0][1] != sides[4][1][1] || sides[5][2][1] !=
sides[5][1][1]) {
            while ((sides[2][1][0] != sides[2][1][1] || sides[4][0][1] !=
```

```
sides[4][1][1]) &&
                    (sides[2][1][0] != sides[2][1][1] || sides[5][2][1] !=
sides[5][1][1]) &&
                    (sides[3][1][2] != sides[3][1][1] || sides[4][0][1] !=
sides[4][1][1]) &&
                    (sides[3][1][2] != sides[3][1][1] || sides[5][2][1] !=
sides[5][1][1]) &&
                    (sides[4][0][1] != sides[4][1][1] || sides[5][2][1] !=
sides[5][1][1]) &&
                    (sides[3][1][2] != sides[3][1][1] || sides[2][1][0] !=
sides[2][1][1])) {
                B_On();
            }

            if ((sides[2][1][0] == sides[2][1][1] && sides[4][0][1] ==
sides[4][1][1]) ||
                (sides[3][1][2] == sides[3][1][1] && sides[2][1][0] ==
sides[2][1][1])) {
                D_Against();
                B_On();
                B_On();
                D_On();
                B_On();
                B_On();
                D_Against();
                R_On();
                D_On();
                B_On();
                D_Against();
                B_Against();

                D_Against();
                R_Against();
                D_On();
                D_On();
                B_Against();
            }

            if ((sides[2][1][0] == sides[2][1][1] && sides[5][2][1] ==
sides[5][1][1]) ||
                (sides[4][0][1] == sides[4][1][1] && sides[5][2][1] ==
sides[5][1][1])) {
                R_Against();
                B_On();
                B_On();
                R_On();
                B_On();
                B_On();
                R_Against();
                U_On();
                R_On();
                B_On();
                R_Against();
                B_Against();

                R_Against();
                U_Against();
                R_On();
                R_On();
                B_Against();
            }

            if (sides[3][1][2] == sides[3][1][1] && sides[4][0][1] ==
sides[4][1][1]) {
```

```
                L_Against();
                B_On();
                B_On();
                L_On();
                B_On();
                B_On();
                L_Against();
                D_On();
                L_On();
                B_On();
                L_Against();
                B_Against();

                L_Against();
                D_Against();
                L_On();
                L_On();
                B_Against();
            }

            if (sides[3][1][2] == sides[3][1][1] && sides[5][2][1] ==
sides[5][1][1]) {
                U_Against();
                B_On();
                B_On();
                U_On();
                B_On();
                B_On();
                U_Against();
                L_On();
                U_On();
                B_On();
                U_Against();
                B_Against();

                U_Against();
                L_Against();
                U_On();
                U_On();
                B_Against();
            }
        }

        //Out();
        k++;
    }

    void FinalStage() {
        while (sides[2][0][0] != sides[2][1][1] || sides[2][0][2] !=
sides[2][1][1] ||
                sides[3][0][2] != sides[3][1][1] || sides[3][2][2] !=
sides[3][1][1] ||
                sides[4][0][0] != sides[4][1][1] || sides[4][0][2] !=
sides[4][1][1] ||
                sides[5][2][0] != sides[5][1][1] || sides[5][2][2] !=
sides[5][1][1]) {
            if (sides[3][2][2] == sides[3][1][1] && sides[5][2][2] ==
sides[5][1][1]) {
                while (sides[2][0][0] != sides[2][1][1] || sides[2][0][2] !=
sides[2][1][1] ||
                        sides[3][0][2] != sides[3][1][1] || sides[3][2][2] !=
sides[3][1][1] ||
                        sides[4][0][0] != sides[4][1][1] || sides[4][0][2] !=
sides[4][1][1] ||
```

```
                        sides[5][2][0] != sides[5][1][1] || sides[5][2][2] !=
sides[5][1][1]) {
                    U_On();
                    R_Against();
                    U_On();
                    L_Twice();
                    U_Against();
                    R_On();
                    U_On();
                    L_Twice();
                    U_Twice();
                }
            } else if (sides[3][0][2] == sides[3][1][1] && sides[4][0][2] ==
sides[4][1][1]) {
                while (sides[2][0][0] != sides[2][1][1] || sides[2][0][2] !=
sides[2][1][1] ||
                        sides[3][0][2] != sides[3][1][1] || sides[3][2][2] !=
sides[3][1][1] ||
                        sides[4][0][0] != sides[4][1][1] || sides[4][0][2] !=
sides[4][1][1] ||
                        sides[5][2][0] != sides[5][1][1] || sides[5][2][2] !=
sides[5][1][1]) {
                    L_On();
                    U_Against();
                    L_On();
                    D_Twice();
                    L_Against();
                    U_On();
                    L_On();
                    D_Twice();
                    L_Twice();
                }
            } else if (sides[2][0][0] == sides[2][1][1] && sides[4][0][0] ==
sides[4][1][1]) {
                while (sides[2][0][0] != sides[2][1][1] || sides[2][0][2] !=
sides[2][1][1] ||
                        sides[3][0][2] != sides[3][1][1] || sides[3][2][2] !=
sides[3][1][1] ||
                        sides[4][0][0] != sides[4][1][1] || sides[4][0][2] !=
sides[4][1][1] ||
                        sides[5][2][0] != sides[5][1][1] || sides[5][2][2] !=
sides[5][1][1]) {
                    D_On();
                    L_Against();
                    D_On();
                    R_Twice();
                    D_Against();
                    L_On();
                    D_On();
                    R_Twice();
                    D_Twice();
                }
            } else if (sides[2][0][2] == sides[2][1][1] && sides[5][2][0] ==
sides[5][1][1]) {
                while (sides[2][0][0] != sides[2][1][1] || sides[2][0][2] !=
sides[2][1][1] ||
                        sides[3][0][2] != sides[3][1][1] || sides[3][2][2] !=
sides[3][1][1] ||
                        sides[4][0][0] != sides[4][1][1] || sides[4][0][2] !=
sides[4][1][1] ||
                        sides[5][2][0] != sides[5][1][1] || sides[5][2][2] !=
sides[5][1][1]) {
                    R_On();
                    D_Against();
```

```
                    R_On();
                    U_Twice();
                    R_Against();
                    D_On();
                    R_On();
                    U_Twice();
                    R_Twice();
                }
            } else if ((sides[3][2][2] != sides[3][1][1] || sides[5][2][2] !=
sides[5][1][1]) &&
                       (sides[3][0][2] != sides[3][1][1] || sides[4][0][2] !=
sides[4][1][1]) &&
                       (sides[2][0][0] != sides[2][1][1] || sides[4][0][0] !=
sides[4][1][1]) &&
                       (sides[2][0][2] != sides[2][1][1] || sides[5][2][0] !=
sides[5][1][1])) {
                while ((sides[2][0][2] != sides[2][1][1] || sides[5][2][0] !=
sides[5][1][1]) &&
                       (sides[2][0][0] != sides[2][1][1] || sides[4][0][0] !=
sides[4][1][1]) &&
                       (sides[3][0][2] != sides[3][1][1] || sides[4][0][2] !=
sides[4][1][1]) &&
                       (sides[3][2][2] != sides[3][1][1] || sides[5][2][2] !=
sides[5][1][1])) {
                    U_On();
                    R_Against();
                    U_On();
                    L_Twice();
                    U_Against();
                    R_On();
                    U_On();
                    L_Twice();
                    U_Twice();
                }
            }
        }

        //Out();
        k++;
    }

public:
    Cube() {
        sides.assign(6, vector<vector<char>>(3, vector<char>(3, 'b')));
        str = "";
        k = 0;
    }

    Cube(const vector<vector<vector<char>>> &sides) {
        this->sides = sides;
        str = "";
        k = 0;
    }

    Cube(const Cube &other) {
        this->sides = other.sides;
    }

    char get_el(int i, int j, int k) const {
        return this->sides[i][j][k];
    }

    void set_el(int i, int j, int k, char el) {
        this->sides[i][j][k] = el;
```

```
    }


    void F_On() {
        TurnOn(this->sides[0]);

        vector<vector<char>> side_(2, vector<char>(3));
        for (int i = 0; i < 3; i++) {
            side_[0][i] = sides[2][2 - i][2];
            side_[1][i] = sides[3][2 - i][0];
        }

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

            sides[2][2 - i][2] = sides[5][0][2 - i];
            sides[3][2 - i][0] = sides[4][2][2 - i];
        }

        sides[4][2] = side_[0];
        sides[5][0] = side_[1];

        str += "F ";
    }

    void F_Against() {
        TurnAgainst(this->sides[0]);

        vector<vector<char>> side_(2, vector<char>(3));
        for (int i = 0; i < 3; i++) {
            side_[0][i] = sides[2][i][2];
            side_[1][i] = sides[3][i][0];
        }

        for (int i = 0; i < 3; i++) {
            sides[2][i][2] = sides[4][2][2 - i];
            sides[3][i][0] = sides[5][0][2 - i];
        }

        sides[4][2] = side_[1];
        sides[5][0] = side_[0];

        str += "F' ";
    }

    void F_Twice() {
        F_On();
        F_On();

        str += "F2 ";
    }


    void B_On() {
        TurnOn(this->sides[1]);

        vector<vector<char>> side_(2, vector<char>(3));
        for (int i = 0; i < 3; i++) {
            side_[0][i] = sides[3][i][2];
            side_[1][i] = sides[2][i][0];
        }

        for (int i = 0; i < 3; i++) {
            sides[3][i][2] = sides[5][2][2 - i];
            sides[2][i][0] = sides[4][0][2 - i];
```

```
        }

        sides[4][0] = side_[0];
        sides[5][2] = side_[1];

        str += "B ";
    }

    void B_Against() {
        TurnAgainst(this->sides[1]);

        vector<vector<char>> side_(2, vector<char>(3));
        for (int i = 0; i < 3; i++) {
            side_[0][i] = sides[3][2 - i][2];
            side_[1][i] = sides[2][2 - i][0];
        }

        for (int i = 0; i < 3; i++) {
            sides[3][i][2] = sides[4][0][i];
            sides[2][i][0] = sides[5][2][i];
        }

        sides[4][0] = side_[1];
        sides[5][2] = side_[0];

        str += "B' ";
    }

    void B_Twice() {
        B_On();
        B_On();

        str += "B2 ";
    }


    void L_On() {
        TurnOn(this->sides[2]);

        vector<vector<char>> side_(2, vector<char>(3));
        for (int i = 0; i < 3; i++) {
            side_[0][i] = sides[1][2 - i][2];
            side_[1][i] = sides[0][2 - i][0];
        }

        for (int i = 0; i < 3; i++) {
            sides[1][2 - i][2] = sides[5][i][0];
            sides[0][2 - i][0] = sides[4][2 - i][0];
        }

        for (int i = 0; i < 3; i++) {
            sides[5][2 - i][0] = side_[1][i];
            sides[4][i][0] = side_[0][i];
        }

        str += "L ";
    }

    void L_Against() {
        TurnAgainst(this->sides[2]);

        vector<vector<char>> side_(2, vector<char>(3));
        for (int i = 0; i < 3; i++) {
            side_[0][i] = sides[1][2 - i][2];
```

```
            side_[1][i] = sides[0][2 - i][0];
        }

        for (int i = 0; i < 3; i++) {
            sides[1][2 - i][2] = sides[4][i][0];
            sides[0][2 - i][0] = sides[5][2 - i][0];
        }

        for (int i = 0; i < 3; i++) {
            sides[5][i][0] = side_[0][i];
            sides[4][2 - i][0] = side_[1][i];
        }

        str += "L' ";
    }

    void L_Twice() {
        L_On();
        L_On();

        str += "L2 ";
    }


    void R_On() {
        TurnOn(this->sides[3]);

        vector<vector<char>> side_(2, vector<char>(3));
        for (int i = 0; i < 3; i++) {
            side_[0][i] = sides[0][2 - i][2];
            side_[1][i] = sides[1][2 - i][0];
        }

        for (int i = 0; i < 3; i++) {
            sides[0][2 - i][2] = sides[5][2 - i][2];
            sides[1][2 - i][0] = sides[4][i][2];
        }

        for (int i = 0; i < 3; i++) {
            sides[4][2 - i][2] = side_[0][i];
            sides[5][i][2] = side_[1][i];
        }

        str += "R ";
    }

    void R_Against() {
        TurnAgainst(this->sides[3]);

        vector<vector<char>> side_(2, vector<char>(3));
        for (int i = 0; i < 3; i++) {
            side_[0][i] = sides[0][2 - i][2];
            side_[1][i] = sides[1][2 - i][0];
        }

        for (int i = 0; i < 3; i++) {
            sides[0][2 - i][2] = sides[4][2 - i][2];
            sides[1][2 - i][0] = sides[5][i][2];
        }

        for (int i = 0; i < 3; i++) {
            sides[5][2 - i][2] = side_[0][i];
            sides[4][i][2] = side_[1][i];
        }
```

```
        str += "R' ";
    }

    void R_Twice() {
        R_On();
        R_On();

        str += "R2 ";
    }


    void U_On() {
        TurnOn(this->sides[4]);

        vector<vector<char>> side_(2, vector<char>(3));
        for (int i = 0; i < 3; i++) {
            side_[0][i] = sides[2][0][i];
            side_[1][i] = sides[3][0][i];

        }

        for (int i = 0; i < 3; i++) {
            sides[2][0][i] = sides[0][0][i];
            sides[3][0][i] = sides[1][0][i];
        }

        sides[0][0] = side_[1];
        sides[1][0] = side_[0];

        str += "U ";
    }

    void U_Against() {
        TurnAgainst(this->sides[4]);

        vector<vector<char>> side_(2, vector<char>(3));
        for (int i = 0; i < 3; i++) {
            side_[0][i] = sides[2][0][i];
            side_[1][i] = sides[3][0][i];
        }

        for (int i = 0; i < 3; i++) {
            sides[2][0][i] = sides[1][0][i];
            sides[3][0][i] = sides[0][0][i];
        }

        sides[1][0] = side_[1];
        sides[0][0] = side_[0];

        str += "U' ";
    }

    void U_Twice() {
        U_On();
        U_On();

        str += "U2 ";
    }


    void D_On() {
        TurnOn(this->sides[5]);
```

```
        vector<vector<char>> side_(2, vector<char>(3));
        for (int i = 0; i < 3; i++) {
            side_[0][i] = sides[2][2][i];
            side_[1][i] = sides[3][2][i];
        }

        for (int i = 0; i < 3; i++) {
            sides[2][2][i] = sides[1][2][i];
            sides[3][2][i] = sides[0][2][i];
        }

        sides[1][2] = side_[1];
        sides[0][2] = side_[0];

        str += "D ";
    }

    void D_Against() {
        TurnAgainst(this->sides[5]);

        vector<vector<char>> side_(2, vector<char>(3));
        for (int i = 0; i < 3; i++) {
            side_[0][i] = sides[2][2][i];
            side_[1][i] = sides[3][2][i];
        }

        for (int i = 0; i < 3; i++) {
            sides[2][2][i] = sides[0][2][i];
            sides[3][2][i] = sides[1][2][i];
        }

        sides[1][2] = side_[0];
        sides[0][2] = side_[1];

        str += "D' ";
    }

    void D_Twice() {
        D_On();
        D_On();

        str += "D2 ";
    }


    void str_out() const {
        cout << "ASSEMBLY STEPS:" << endl;
        for (int i = 0; i < str.length(); i++)
            cout << str[i];
    }


    void Out() const
    {
        for (int j = 2; j >= 0; j--)
            cout << "      " << sides[1][j][2] << " " << sides[1][j][1] << "
" << sides[1][j][0] << endl;
        for (int j = 0; j < 3; j++)
            cout << "      " << sides[4][j][0] << " " << sides[4][j][1] << "
" << sides[4][j][2] << endl;
        for (int j = 0; j < 3; j++) {
            for (int k = 0; k < 3; k++)
                cout << sides[2][j][k] << " ";
            for (int k = 0; k < 3; k++)
```

```
                cout << sides[0][j][k] << " ";
            for (int k = 0; k < 3; k++)
                cout << sides[3][j][k] << " ";
            cout << endl;
        }
        for (int j = 0; j < 3; j++)
            cout << "      " << sides[5][j][0] << " " << sides[5][j][1] << "
" << sides[5][j][2] << endl;
    }

    void SaveCube()
    {
        ofstream fout("cube_out.txt");
        for (int j = 2; j >= 0; j--)
            fout << "      " << sides[1][j][2] << " " << sides[1][j][1] << "
" << sides[1][j][0] << endl;
        for (int j = 0; j < 3; j++)
            fout << "      " << sides[4][j][0] << " " << sides[4][j][1] << "
" << sides[4][j][2] << endl;
        for (int j = 0; j < 3; j++) {
            for (int k = 0; k < 3; k++)
                fout << sides[2][j][k] << " ";
            for (int k = 0; k < 3; k++)
                fout << sides[0][j][k] << " ";
            for (int k = 0; k < 3; k++)
                fout << sides[3][j][k] << " ";
            fout << endl;
        }
        for (int j = 0; j < 3; j++)
            fout << "      " << sides[5][j][0] << " " << sides[5][j][1] << "
" << sides[5][j][2] << endl;
//        for (int i = 0; i < 6; i++)
//        {
//            for (int j = 0; j < 3; j++)
//            {
//                for (int k = 0; k < 3; k++)
//                    fout << sides[i][j][k] << ' ';
//                fout << endl;
//            }
//
//            fout << endl;
//        }
    }



    void ReadCube()
    {
        ifstream fin("Cube_in.txt");
        for (int i = 0; i < 6; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                for (int k = 0; k < 3; k++)
                {
                    fin >> this->sides[i][j][k];
                }
            }
        }

        for (int i = 0; i < 6; i++)
        {
            switch (sides[i][1][1])
```

```
            {
                case 'w':
                    if (i != 0)
                        swap(sides[i], sides[0]);
                    break;
                case 'y':
                    if (i != 1)
                        swap(sides[i], sides[1]);
                    break;
                case 'g':
                    if (i != 2)
                        swap(sides[i], sides[2]);
                    break;
                case 'b':
                    if (i != 3)
                        swap(sides[i], sides[3]);
                    break;
                case 'o':
                    if (i != 4)
                        swap(sides[i], sides[4]);
                    break;
                case 'r':
                    if (i != 5)
                        swap(sides[i], sides[5]);
                    break;
            }
        }
        //SaveCube();
        fin.close();
    }


    void AllSteps()
    {
        Krest();
        PerfectKrest();
        FirstLay();
        SecondLay();
        UpperKrest();
        CornerEl();
        UpperEdge();
        FinalStage();
    }

};
```

//Algo.h

```
//
// Created by Volirvag on 26.05.2022.
//
#ifndef RUBICK_ALGO_H
#define RUBICK_ALGO_H
#pragma once
#include<iostream>
#include <experimental/random>
#include"Cube.h"

class Algo{
private:
    Cube cube;

    bool CheckColor(char a, char b) const
    {
        if (a == 'w' && (b == 'o' || b == 'g' || b == 'b' || b == 'r'))
            return false;

        if (a == 'y' && (b == 'o' || b == 'g' || b == 'b' || b == 'r'))
            return false;

        if (a == 'g' && (b == 'o' || b == 'r' || b == 'w' || b == 'y'))
            return false;

        if (a == 'b' && (b == 'o' || b == 'r' || b == 'w' || b == 'y'))
            return false;

        if (a == 'o' && (b == 'b' || b == 'g' || b == 'w' || b == 'y'))
            return false;

        if (a == 'r' && (b == 'b' || b == 'g' || b == 'w' || b == 'y'))
            return false;

        return true;
    }

    bool CheckCornerColor(char a, char b, char c) const
    {
        if ((a == 'w' && b == 'g' && c == 'o') || (a == 'w' && b == 'g' && c
== 'r') || (a == 'w' && b == 'b' && c == 'o') || (a == 'w' && b == 'b' && c
== 'r') || (a == 'y' && b == 'b' && c == 'o') || (a == 'y' && b == 'b' && c
== 'r') || (a == 'y' && b == 'g' && c == 'o') || (a == 'y' && b == 'g' && c
== 'r'))
            return false;

        return true;
    }

    //CheckCornerColorElements
    bool CCCE(char a, char b, char c) const
    {
        return CheckCornerColor(a, b, c) * CheckCornerColor(a, c, b) *
CheckCornerColor(b, a, c) * CheckCornerColor(b, c, a) * CheckCornerColor(c,
a, b) * CheckCornerColor(c, b, a);
    }

public:

    Algo() = default;

    Algo(const Cube &other)
```

```
    {
        this->cube = other;
    }

    Cube get_cube() const
    {
        return this->cube;
    }

    bool CheckDate()
    {
        if (CheckColor(cube.get_el(0, 0, 1), cube.get_el(4, 2, 1)) ||
CheckColor(cube.get_el(0, 1, 0), cube.get_el(2, 1, 2)) ||
CheckColor(cube.get_el(0, 2, 1), cube.get_el(5, 0, 1)) ||
CheckColor(cube.get_el(0, 1, 2), cube.get_el(3, 1, 0)) ||
CheckColor(cube.get_el(1, 0, 1), cube.get_el(4, 0, 1)) ||
CheckColor(cube.get_el(1, 1, 0), cube.get_el(3, 1, 2)) ||
CheckColor(cube.get_el(1, 2, 1), cube.get_el(5, 2, 1)) ||
CheckColor(cube.get_el(1, 1, 2), cube.get_el(2, 1, 0)) ||
CheckColor(cube.get_el(2, 0, 1), cube.get_el(4, 1, 0)) ||
CheckColor(cube.get_el(2, 2, 1), cube.get_el(5, 1, 0)) ||
CheckColor(cube.get_el(3, 0, 1), cube.get_el(4, 1, 2)) ||
CheckColor(cube.get_el(3, 2, 1), cube.get_el(5, 1, 2)))
        {
            return false;
        }

        if (CCCE(cube.get_el(0, 0, 0), cube.get_el(2, 0, 2), cube.get_el(4,
2, 0)) || CCCE(cube.get_el(0, 0, 2), cube.get_el(3, 0, 0), cube.get_el(4, 2,
2)) || CCCE(cube.get_el(0, 2, 0), cube.get_el(2, 2, 2), cube.get_el(5, 0, 0))
|| CCCE(cube.get_el(0, 2, 2), cube.get_el(3, 2, 0), cube.get_el(5, 0, 2)) ||
CCCE(cube.get_el(1, 0, 0), cube.get_el(3, 0, 2), cube.get_el(4, 0, 2)) ||
CCCE(cube.get_el(1, 0, 2), cube.get_el(2, 0, 0), cube.get_el(4, 0, 0)) ||
CCCE(cube.get_el(1, 2, 0), cube.get_el(3, 2, 2), cube.get_el(5, 2, 2)) ||
CCCE(cube.get_el(1, 2, 2), cube.get_el(2, 2, 0), cube.get_el(5, 2, 0)))
        {
            return false;
        }

        cube.AllSteps();

        for (int j = 0; j < 3; j++)
            for (int k = 0; k < 3; k++)
            {
                if (cube.get_el(0, j, k) != 'w' || cube.get_el(1, j, k) !=
'y' || cube.get_el(2, j, k) != 'g' || cube.get_el(3, j, k) != 'b' ||
cube.get_el(4, j, k) != 'o' || cube.get_el(5, j, k) != 'r')
                {
                    return false;
                }
            }

        return true;
    }

    Cube RandomCube()
    {
        short num;
        for (int i = 0; i < 6; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                for (int k = 0; k < 3; k++)
                {
```

```
                switch (i)
                {
                    case 0:
                        cube.set_el(i, j, k, 'w');
                        break;
                    case 1:
                        cube.set_el(i, j, k, 'y');
                        break;
                    case 2:
                        cube.set_el(i, j, k, 'g');
                        break;
                    case 3:
                        cube.set_el(i, j, k, 'b');
                        break;
                    case 4:
                        cube.set_el(i, j, k, 'o');
                        break;
                    case 5:
                        cube.set_el(i, j, k, 'r');
                        break;
                }
            }
        }
    }

    for (int i = 0; i < experimental::randint(1, 100); i++)
    {
        num = experimental::randint(1, 12);
        if (num == 1)
        {
            cube.F_On();
        }
        else if (num == 2)
        {
            cube.B_On();
        }
        else if (num == 3)
        {
            cube.L_On();
        }
        else if (num == 4)
        {
            cube.R_On();
        }
        else if (num == 5)
        {
            cube.U_On();
        }
        else if (num == 6)
        {
            cube.D_On();
        }
        else if (num == 7)
        {
            cube.B_On();
        }

        else if (num == 8)
        {
            cube.L_On();
        }
        else if (num == 9)
        {
            cube.R_On();
```

```
            }
            else if (num == 10)
            {
                cube.U_On();
            }
            else if (num == 11)
            {
                cube.D_On();
            }
            else if (num == 12)
            {
                cube.F_On();
            }
        }
        return cube;
    }

    Cube HandTurn()
    {
        string step;
        cout << "Print your commands: / If you want to finish, print - ." <<
endl;
        cin >> step;

        while (step != "."){
            if (step == "L")
                cube.L_On();
            else if (step == "L'")
                cube.L_Against();
            else if (step == "L'")
                cube.L_Against();
            else if (step == "B")
                cube.B_On();
            else if (step == "B'")
                cube.B_Against();
            else if (step == "F")
                cube.F_On();
            else if (step == "F'")
                cube.F_Against();
            else if (step == "R")
                cube.R_On();
            else if (step == "R'")
                cube.R_Against();
            else if (step == "U")
                cube.U_On();
            else if (step == "U'")
                cube.U_Against();
            else if (step == "D")
                cube.D_On();
            else if (step == "D'")
                cube.D_Against();
            else if (step == ".")
                cout << "";
            else
                cout << endl << "Unknown command!" << endl;
            cin >> step;
        }
        return cube;
    }

    void make_cube(){
        cube.AllSteps();
    }
    void show_commands(){
```

```
        cube.str_out();
    }
};


#endif //RUBICK_ALGO_H
```

//Menu.h

```
//
// Created by Volirvag on 29.05.2022.
//

#ifndef LAST_VERSION_OF_CUBIK_MENU_H
#define LAST_VERSION_OF_CUBIK_MENU_H
#include <iostream>
#include "Cube.h"
#include "ALgo.h"
class Menu{
private:
    Cube cube;
    bool flag = false;
    bool correct_cube = false;
public:
    Menu(){};
    void Show_Menu(){
        cout << "For show menu again print - Show_Menu" << endl;
        cout << "For read a cube from file print - Read_Cube" << endl;
        cout << "For check a cube print - Check_Cube" << endl;
        cout << "For save a cube in file print - Save_Cube" << endl;
        cout << "For show a cube in console print - Show_Cube" << endl;
        cout << "For start handle making of cube print - Handle" << endl;
        cout << "For generate random cube print - Random_Cube (Cube will be
correct!)" << endl;
        cout << "For sole a cube print - Solve_Cube" << endl;
        cout << "For exit print - Exit" << endl;
    }
    bool Check_Cube(Cube &cube){
        Algo al(cube);
        if (al.CheckDate()) {
            return true;
        }else{
            return false;
        }
    }
    void Read_Cube(){
        cube.ReadCube();
        flag = true;
        if (Check_Cube(cube))
            correct_cube = true;
        else
            correct_cube = false;
    }
    void Save_Cube(){
        if (flag)
            cube.SaveCube();
        else
            cout << "Please read cube or generate random cube!" << endl;
    }
    void ShowCube(){
        if (flag)
            cube.Out();
```

```
        else
            cout << "Please read cube or generate random cube!" << endl;
    }

    void Random_Cube(){
        Algo al(cube);
        cube = al.RandomCube();
        if (Check_Cube(cube))
            correct_cube = true;
        else
            correct_cube = false;
        flag = true;
    }
    void Handle(){
        if(flag){
            Algo al(cube);
            cube = al.HandTurn();
        }
        else
            cout << "Please read cube or generate random cube!" << endl;
    }
    void Solve_Cube(){
        if(Check_Cube(cube)) {
            cube.AllSteps();
            cube.str_out();
        }else
            cout << "Please read cube or generate random cube!" << endl;
        cout << endl;
    }

    void check(){
        if (correct_cube)
            cout << "Cube is correct!" << endl;
        else
            cout << "Cube isn't correct!" << endl;
    }

};
#endif //LAST_VERSION_OF_CUBIK_MENU_H
```

//Main

```
#include<iostream>
#include<vector>
#include "Menu.h"
#include <fstream>
using namespace std;
    int main() {
        Menu m;
        m.Show_Menu();
        string command;
        cin >> command;
        while (command != "Exit") {
            if (command == "Show_Menu")
                m.Show_Menu();
            else if (command == "Read_Cube")
                m.Read_Cube();
            else if (command == "Save_Cube")
                m.Save_Cube();
            else if (command == "Check_Cube")
                m.check();
            else if (command == "Show_Cube")
```

```
                m.ShowCube();
            else if (command == "Handle_Solve")
                m.Handle();
            else if (command == "Random_Cube")
                m.Random_Cube();
            else if (command == "Solve_Cube")
                m.Solve_Cube();
            else if (command == "Exit")
                break;
            else
                cout << "Unknown command!" << endl;
            cin >> command;
        }
        return 0;
}
```