

**Федеральное государственное автономное  
образовательное учреждение высшего  
образования**

**«Национальный исследовательский  
университет ИТМО»**

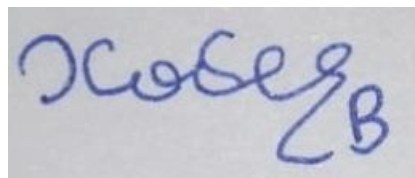
**Факультет информационных технологий  
и Программирования**

Программирование

Лабораторная работа № 4

**Выполнил студент группы № М3104**  
Хобер Владислав Алимович

Подпись:



Санкт-Петербург

2022

# Лабораторная №4

---

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

Дополнительно (за дополнительные баллы): Реализовать графический интерфейс приложения, с использование OpenGL Utility Toolkit

## main.cpp

```
#include<iostream>
#include<fstream>
#include "Cube.h"

using namespace std;

int main()
{
    Cube a;
    a.print();
    a.solve();
    a.print();

    fstream in;
    in.open("input.txt");
    Cube c(in);
    cout << c.solve() << '\n';
    c.print();
    c.out_ans();
}
```

```
}
```

## Cube.h

```
#ifndef LAB4_CUBE_H
#define LAB4_CUBE_H

#include<iostream>
#include<fstream>
#include<map>
#include<ctime>
#include<cstdlib>
#include<vector>

struct Verge
{
    Verge()
    {}

    char verge[3][3];
};

class Cube
{
public:
    Cube(std::fstream &FILE)
    {
        char colors[9];
        int k;
        for (int i = 0; i < 6; i++)
        {
            k = 0;
            for (int j = 0; j < 3; j++)
            {
                for (int q = 0; q < 3; q++)
                {
                    FILE >> colors[k];
                    k++;
                }
            }
            int verge_nubmer = colors_number[colors[4]];
            k = 0;
            for (int j = 0; j < 3; j++)
            {
                for (int q = 0; q < 3; q++)
                {
                    cube[verge_nubmer].verge[j][q] = colors[k];
                }
            }
        }
    }
};
```

```

        k++;
    }
}

}

Cube()
{
    for (auto it = colors_number.begin(); it != colors_number.end();
it++)
    {
        int verge_number = it->second;
        char verge_color = it->first;
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                cube[verge_number].verge[i][j] = verge_color;
            }
        }
        srand(std::time(NULL));
        int n = rand() % 100;
        for (int i = 0; i < n; i++)
        {
            random_rotate();
        }
    }

Cube(const Cube &other)
{
    for (int k = 0; k < 6; k++)
    {
        for (int i = 0; i < 6; i++)
        {
            for (int j = 0; j < 6; j++)
            {
                cube[k].verge[i][j] = other.cube[k].verge[i][j];
            }
        }
    }
}

~Cube()
{}

void print()
{
    for (int i = 0; i < 6; i++)
    {
        std::cout << i + 1 << ' ' << "verge" << '\n';
        for (int j = 0; j < 3; j++)
        {

```

```

        for (int k = 0; k < 3; k++)
        {
            std::cout << cube[i].verge[j][k] << ' ';
        }
        std::cout << '\n';
    }
}

void R()
{
    int verges[] = {0, 4, 2, 5};
    int row[] = {0, 1, 2, 0, 1, 2, 2, 1, 0, 0, 1, 2};
    int column[] = {2, 2, 2, 2, 2, 2, 0, 0, 0, 2, 2, 2};
    rotate(verges, row, column, 1, 1);
    ans.push_back('R');
}

void R_()
{
    int verges[] = {5, 2, 4, 0};
    int row[] = {2, 1, 0, 0, 1, 2, 2, 1, 0, 2, 1, 0};
    int column[] = {2, 2, 2, 0, 0, 0, 2, 2, 2, 2, 2, 2};
    rotate(verges, row, column, 1, 0);
    ans.push_back('R');
    ans.push_back('_');
}

void R2()
{
    this->R();
    this->R();
    ans.push_back('R');
    ans.push_back('2');
}

void L()
{
    int verges[] = {5, 2, 4, 0};
    int row[] = {0, 1, 2, 2, 1, 0, 0, 1, 2, 0, 1, 2};
    int column[] = {0, 0, 0, 2, 2, 2, 0, 0, 0, 0, 0, 0};
    rotate(verges, row, column, 3, 1);
    ans.push_back('L');
}

void L_()
{
    int verges[] = {0, 4, 2, 5};
    int row[] = {2, 1, 0, 2, 1, 0, 0, 1, 2, 2, 1, 0};
    int column[] = {0, 0, 0, 0, 0, 0, 2, 2, 2, 0, 0, 0};
    rotate(verges, row, column, 3, 0);
    ans.push_back('L');
    ans.push_back('_');
}

```

```

void L2()
{
    this->L();
    this->L();
    ans.push_back('L');
    ans.push_back('2');
}

void F()
{
    int verges[] = {1, 5, 3, 4};
    int row[] = {0, 1, 2, 0, 0, 0, 2, 1, 0, 2, 2, 2};
    int column[] = {0, 0, 0, 2, 1, 0, 2, 2, 2, 0, 1, 2};
    rotate(verges, row, column, 0, 1);
    ans.push_back('F');
}

void F_()
{
    int verges[] = {4, 3, 5, 1};
    int row[] = {2, 2, 2, 0, 1, 2, 0, 0, 0, 2, 1, 0};
    int column[] = {2, 1, 0, 2, 2, 2, 0, 1, 2, 0, 0, 0};
    rotate(verges, row, column, 0, 0);
    ans.push_back('F');
    ans.push_back('_');
}

void F2()
{
    this->F();
    this->F();
    ans.push_back('F');
    ans.push_back('2');
}

void B()
{
    int verges[] = {1, 4, 3, 5};
    int row[] = {0, 1, 2, 0, 0, 0, 2, 1, 0, 2, 2, 2};
    int column[] = {2, 2, 2, 0, 1, 2, 0, 0, 0, 2, 1, 0};
    rotate(verges, row, column, 2, 1);
    ans.push_back('B');
}

void B_()
{
    int verges[] = {5, 3, 4, 1};
    int row[] = {2, 2, 2, 0, 1, 2, 0, 0, 0, 2, 1, 0};
    int column[] = {0, 1, 2, 0, 0, 0, 2, 1, 0, 2, 2, 2};
    rotate(verges, row, column, 2, 0);
    ans.push_back('B');
    ans.push_back('_');
}

```

```

void B2()
{
    this->B();
    this->B();
    ans.push_back('B');
    ans.push_back('2');
}

void U()
{
    int verges[] = {3, 2, 1, 0};
    int row[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    int column[] = {2, 1, 0, 2, 1, 0, 2, 1, 0, 2, 1, 0};
    rotate(verges, row, column, 4, 1);
    ans.push_back('U');
}

void U_()
{
    int verges[] = {0, 1, 2, 3};
    int row[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    int column[] = {0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2};
    rotate(verges, row, column, 4, 0);
    ans.push_back('U');
    ans.push_back('_');
}

void U2()
{
    this->U();
    this->U();
    ans.push_back('U');
    ans.push_back('2');
}

void D()
{
    int verges[] = {0, 1, 2, 3};
    int row[] = {2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2};
    int column[] = {0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2};
    rotate(verges, row, column, 5, 1);
    ans.push_back('D');
}

void D_()
{
    int verges[] = {3, 2, 1, 0};
    int row[] = {2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2};
    int column[] = {2, 1, 0, 2, 1, 0, 2, 1, 0, 2, 1, 0};
    rotate(verges, row, column, 5, 0);
    ans.push_back('D');
    ans.push_back('_');
}

```

```

void D2()
{
    this->D();
    this->D();
    ans.push_back('D');
    ans.push_back('2');
}

void rp()
{
    this->R();
    this->U();
    this->R_();
    this->U_();
}

void lp()
{
    this->L_();
    this->U_();
    this->L();
    this->U();
}

void test()
{
    this->horizontal_rotate_l();
}

bool is_correct()
{
    Cube copy(*this);
    if (copy.solve())
        return true;
    else
        return false;
}

void out_ans()
{
    std::ofstream FILE;
    FILE.open("output.txt");
    for (auto it = ans.begin(); it != ans.end(); it++)
    {
        FILE << *it;
    }
}

bool solve()
{
    srand(std::time(NULL));
    ans.clear();
    if (this->cross())

```



```

    {
        this->white_verge();
        this->second_verges();
        this->yellow_cross();
        this->yellow_verge();
        this->finish_solve();
        while (cube[0].verge[0][1] != cube[0].verge[1][1])
            this->U();
        return is_solved(0) && is_solved(1) && is_solved(2) &&
is_solved(3) && is_solved(4) && is_solved(5);
    }
}

private:
    Verge cube[6];
    std::map<char, int> colors_number = {{'R', 0},
                                         {'G', 1},
                                         {'O', 2},
                                         {'B', 3},
                                         {'Y', 4},
                                         {'W', 5}};

    std::vector<char> ans;
    const int max_iteration = 1e6;

    bool is_solved(int number_verge)
    {
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                if (cube[number_verge].verge[i][j] !=
cube[number_verge].verge[1][j])
                {
                    return false;
                }
            }
        }
        return true;
    }

    void finish_solve()
    {
        for (int x = 0; x < max_iteration; x++)
        {
            for (int k = 0; k < 4; k++)
            {
                int num1 = rand() % 2;
                if (num1 == 1)
                {
                    this->rp();
                    this->lp();
                    for (int i = 0; i < 5; i++)
                        this->rp();
                }
            }
        }
    }
}

```

&&

```
for (int i = 0; i < 5; i++)
    this->lp();
int number;
for (int i = 0; i < 4; i++)
{
    number = 0;
    for (int j = 0; j < 4; j++)
    {
        if (cube[0].verge[0][0] == cube[0].verge[0][1]
            cube[0].verge[0][2] == cube[0].verge[0][1])
        {
            number++;
        }
        this->horizontal_rotate();
    }
    if (number == 4)
        return;
}
this->rp();
this->lp();
for (int i = 0; i < 5; i++)
    this->rp();
for (int i = 0; i < 5; i++)
    this->lp();
for (int i = 0; i < 4; i++)
{
    number = 0;
    for (int j = 0; j < 4; j++)
    {
        if (cube[0].verge[0][0] == cube[0].verge[0][1]
            cube[0].verge[0][2] == cube[0].verge[0][1])
        {
            number++;
        }
        this->horizontal_rotate();
    }
    if (number == 4)
        return;
}
}
this->horizontal_rotate();
}
```

&&

```
void yellow_verge()
{
    for (int x = 0; x < max_iteration; x++)
    {
        for (int k = 0; k < 4; k++)
        {
            int num1 = rand() % 2;
```

```

if (num1 == 0)
{
    this->rp();
    this->rp();
    this->rp();
    this->horizontal_rotate();
    this->lp();
    this->lp();
    this->lp();
    this->horizontal_rotate_l();
    int num2 = rand() % 5;
    for (int i = 0; i < num2; i++)
    {
        this->U();
    }
    for (int i = 0; i < 4; i++)
    {
        while (cube[4].verge[2][0] != 'Y')
        {
            this->L();
            this->D();
            this->L_();
            this->D_();
        }
        this->U();
    }
    for (int i = 0; i < 4; i++)
    {
        int number = 0;
        for (int j = 0; j < 4; j++)
        {
            if (cube[0].verge[0][0] == cube[0].verge[1][1]
                && cube[0].verge[0][2] == cube[0].verge[1][1])
            {
                number++;
            }
            this->horizontal_rotate();
        }
        if (number == 4)
        {
            return;
        }
        this->U();
    }
}
this->horizontal_rotate();
for (int i = 0; i < 4; i++)
{
    for (int j = 0; j < 4; j++)
    {

```

```

        }
    }

}

void yellow_cross()
{
    for (int x = 0; x < max_iteration; x++)
    {
        for (int k = 0; k < 4; k++)
        {
            this->F();
            int num1 = rand() % 3;
            for (int i = 0; i < num1; i++)
            {
                this->rp();
            }
            this->F_();
            this->horizontal_rotate();
        }
        if (cube[4].verge[0][1] == 'Y' && cube[4].verge[1][0] == 'Y' &&
cube[4].verge[2][1] == 'Y' &&
        cube[4].verge[1][2] == 'Y')
        {
            break;
        }
    }
}

void second_verges()
{
    for (int x = 0; x < max_iteration; x++)
    {
        for (int k = 0; k < 4; k++)
        {
            if (cube[0].verge[1][1] == cube[0].verge[0][1])
            {
                if (cube[4].verge[2][1] == cube[1].verge[1][1])
                {
                    this->U();
                    this->rp();
                    this->horizontal_rotate();
                    this->lp();
                    this->horizontal_rotate_l();
                }
                if (cube[4].verge[2][1] == cube[3].verge[1][1])
                {
                    this->U_();
                    this->lp();
                    this->horizontal_rotate_l();
                    this->rp();
                    this->horizontal_rotate();
                }
            }
        }
    }
}

```

```

    }
}
int num1 = rand() % 2, num2 = rand() % 2, num3 = rand() %
4;
if (num1 == 0)
{
    this->U();
    this->rp();
    this->horizontal_rotate();
    this->lp();
    this->horizontal_rotate_l();
}
if (num2 == 0)
{
    this->U_();
    this->lp();
    this->horizontal_rotate_l();
    this->rp();
    this->horizontal_rotate();
}
for (int i = 0; i < num3; i++)
{
    this->U();
}
this->horizontal_rotate();

}
bool check = true;
for (int k = 0; k < 4; k++)
{
    if (cube[k].verge[1][0] != cube[k].verge[1][1] ||
cube[k].verge[1][2] != cube[k].verge[1][1])
    {
        check = false;
    }
}
if (check)
    break;
}

}

void white_verge()
{
    for (int x = 0; x < max_iteration; x++)
    {
        for (int k = 0; k < 4; k++)
        {

            int num1 = rand() % 7, num2 = rand() % 7, num3 = rand() %
4;

            for (int i = 0; i < num1; i++)
            {

```

```

        this->rp();
    }
    for (int i = 0; i < num2; i++)
    {
        this->lp();
    }
    for (int i = 0; i < num3; i++)
    {
        this->U();
    }
    this->horizontal_rotate();
}

if (is_solved(5))
{
    bool check = true;
    for (int k = 0; k < 4; k++)
    {
        if (cube[0].verge[2][1] != cube[0].verge[2][0] ||
cube[0].verge[2][1] != cube[0].verge[2][2])
        {
            check = false;
            break;
        }
        this->horizontal_rotate();
    }
    if (check)
    {
        return;
    }
}
}

bool cross()
{
    bool check = false;
    for (int x = 0; x < max_iteration; x++)
    {
        if (cube[4].verge[0][1] == 'W' && cube[4].verge[1][0] == 'W' &&
cube[4].verge[2][1] == 'W' &&
        cube[4].verge[1][2] == 'W')
        {
            check = true;
            break;
        }
    }
    for (int k = 0; k < 4; k++)
    {
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                if (cube[0].verge[i][j] == 'W')
                {

```

```

        if (i == 0 && j == 1)
        {
            this->F();
            while (cube[4].verge[1][2] == 'W')
            {
                this->U();
            }
            this->R();
        }
        if (i == 1 && j == 0)
        {
            while (cube[4].verge[1][0] == 'W')
            {
                this->U();
            }
            this->L_();
        }
        if (i == 1 && j == 2)
        {
            while (cube[4].verge[1][2] == 'W')
            {
                this->U();
            }
            this->R();
        }
        if (i == 2 && j == 1)
        {
            this->F();
            while (cube[4].verge[1][0] == 'W')
            {
                this->U();
            }
            this->L_();
        }
    }
}

this->horizontal_rotate();
}
for (int i = 0; i < 3; i++)
{
    for (int j = 0; j < 3; j++)
    {
        if (cube[5].verge[i][j] == 'W')
        {
            int k1 = i, k2 = j;
            if (i == 0 && j == 1)
            {
                k1 = 2;
                k2 = 1;
            }
            if (i == 2 && j == 1)
            {
                k1 = 0;

```

```

        k2 = 1;
    }
    while (cube[4].verge[k1][k2] == 'W')
        this->U();
    if (i == 0 && j == 1)
    {
        this->F2();
    }
    if (i == 1 && j == 0)
    {
        this->L2();
    }
    if (i == 1 && j == 2)
    {
        this->R2();
    }
    if (i == 2 && j == 1)
    {
        this->B2();
    }
}
}
}
}
if (check)
{
    for (int k = 0; k < 4; k++)
    {
        while (cube[0].verge[1][1] != cube[0].verge[0][1])
        {
            this->U();
        }
        this->F2();
        this->horizontal_rotate();
    }
    return true;
}
return false;
}

void rotate(int verges[], int row[], int column[], int roll_verge, bool
direction)
{
    int current, k = 0;
    char save[3] = {cube[verges[3]].verge[row[9]][column[9]],
cube[verges[3]].verge[row[10]][column[10]],
        cube[verges[3]].verge[row[11]][column[11]]};
    for (int i = 0; i < 4; i++)
    {
        current = verges[i];
        for (int j = 0; j < 3; j++)
        {

```



```

        int pos1 = row[k];
        int pos2 = column[k];
        k++;
        char s;
        s = cube[current].verge[pos1][pos2];
        cube[current].verge[pos1][pos2] = save[j];
        save[j] = s;
    }
}
char updated[3][3];
if (direction)
{
    updated[0][2] = cube[roll_verge].verge[0][0];
    updated[1][2] = cube[roll_verge].verge[0][1];
    updated[2][2] = cube[roll_verge].verge[0][2];
    updated[2][1] = cube[roll_verge].verge[1][2];
    updated[2][0] = cube[roll_verge].verge[2][2];
    updated[1][0] = cube[roll_verge].verge[2][1];
    updated[0][0] = cube[roll_verge].verge[2][0];
    updated[0][1] = cube[roll_verge].verge[1][0];
} else
{
    updated[0][0] = cube[roll_verge].verge[0][2];
    updated[1][0] = cube[roll_verge].verge[0][1];
    updated[2][0] = cube[roll_verge].verge[0][0];
    updated[2][1] = cube[roll_verge].verge[1][0];
    updated[2][2] = cube[roll_verge].verge[2][0];
    updated[1][2] = cube[roll_verge].verge[2][1];
    updated[0][2] = cube[roll_verge].verge[2][2];
    updated[0][1] = cube[roll_verge].verge[1][2];
}
updated[1][1] = cube[roll_verge].verge[1][1];
for (int i = 0; i < 3; i++)
{
    for (int j = 0; j < 3; j++)
    {
        cube[roll_verge].verge[i][j] = updated[i][j];
    }
}
}

void horizontal_rotate()
{
    int verges[] = {3, 2, 1, 0};
    char save[3];
    for (int i = 0; i < 3; i++)
    {
        save[i] = cube[0].verge[1][i];
    }
    for (int q = 0; q < 4; q++)
    {
        for (int i = 0; i < 3; i++)
        {

```

```

        char saved = cube[verges[q]].verge[1][i];
        cube[verges[q]].verge[1][i] = save[i];
        save[i] = saved;
    }
}
ans.push_back('M');
ans.push_back('_');
this->D();
this->U();
}

void horizontal_rotate_l()
{
    int verges[] = {0, 1, 2, 3};
    char save[3];
    for (int i = 0; i < 3; i++)
    {
        save[i] = cube[3].verge[1][i];
    }
    for (int q = 0; q < 4; q++)
    {
        for (int i = 0; i < 3; i++)
        {
            char saved = cube[verges[q]].verge[1][i];
            cube[verges[q]].verge[1][i] = save[i];
            save[i] = saved;
        }
    }
    ans.push_back('M');
    this->D();
    this->U_();
}

void random_rotate()
{
    int value = rand() % 18;
    switch (value)
    {
        case 0:
            this->R();
            break;
        case 1:
            this->R_();
            break;
        case 2:
            this->R2();
        case 3:
            this->U();
            break;
        case 4:
            this->U_();
            break;
        case 5:
            this->U2();
    }
}

```

```

        break;
    case 6:
        this->L();
        break;
    case 7:
        this->L_();
        break;
    case 8:
        this->L2();
        break;
    case 9:
        this->D();
        break;
    case 10:
        this->D_();
        break;
    case 11:
        this->D2();
        break;
    case 12:
        this->F();
        break;
    case 13:
        this->F_();
        break;
    case 14:
        this->F2();
        break;
    case 15:
        this->B();
        break;
    case 16:
        this->B_();
        break;
    case 17:
        this->B2();
        break;
    }
}

};

#endif

```

## Ввод и вывод

---

Программа реализует класс кубика, находит "решение" кубика с помощью стандартного алгоритма его сборки, может генерировать его случайные корректные состояния, считывать его состояние из файла.

# Вывод

---

Используя принципы ООП, я выполнил предложенное мне задание, реализовал класс кубика, научился его собирать и реализовал алгоритм его сборки программно.