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 № 6

Кубик Рубика

Выполнил студент группы № M3106
Шеин Максим Андреевич
Подпись:

Проверил:
Повышев Владислав Вячеславович

Санкт-Петербург

2020

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

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

Решение

Cubik.cpp

```
#include <iostream>
#include <fstream>
#include "Cubik.hpp"
#include "EnumerationList.hpp"
#include <cmath>

std::ifstream in("in.txt");
std::ofstream out("out.txt");

//УГЛЫ                                                                    //----------------
Element Corner№1(1.05, 1.05, -1.05, Straight, Red, Blue, Black, Yellow, Black, Black);
//| Co6| E9 |Co5 |
Element Corner№2(-1.05, 1.05, -1.05, Straight, Red, Blue, White, Black, Black, Black);
//|--------------|
Element Corner№3(-1.05, -1.05, -1.05, Straight, Red, Black, White, Black, Green, Black);
//|  E6| Ce2|E5  |
Element Corner№4(1.05, -1.05, -1.05, Straight, Red, Black, Black, Yellow, Green, Black);
//|--------------|
                                                                    //| Co2| E1 |Co1 |
Element Corner№5(1.05, 1.05, 1.05, Straight, Black, Blue, Black, Yellow, Black, Orange);
//-----------------------------------------------
Element Corner№6(-1.05, 1.05, 1.05, Straight, Black, Blue, White, Black, Black, Orange);
//| Co6| E6 |Co2 | Co2| E1 |Co1 | Co1| E5 |Co5 |
Element Corner№7(-1.05, -1.05, 1.05, Straight, Black, Black, White, Black, Green, Orange);
//-----------------------------------------------
```

```
Element Corner№8(1.05, -1.05, 1.05, Straight, Black, Black, Black, Yellow, Green, Orange);
//| E10| Ce4|E2  |  E2| Ce1|E4  |  E4| Ce5|E12 |
                                                                    //-----------------------------------
----------
//ЦЕНТР                                                                  //| Co7| E7 |Co3 | Co3| E3
|Co4 | Co4| E8 |Co8 |
Element Centre№1(0, 0, -1.05, Straight, Red, Black, Black, Black, Black, Black);
//-------------------------------------------
Element Centre№2(0, 1.05, 0, Straight, Black, Blue, Black, Black, Black, Black);
//| Co3| E3 |Co4 |
Element Centre№3(-1.05, 0, 0, Straight, Black, Black, White, Black, Black, Black);
//|--------------|
Element Centre№4(1.05, 0, 0, Straight, Black, Black, Black, Yellow, Black, Black);
//|  E7| Ce6|E8  |
Element Centre№5(0, -1.05, 0, Straight, Black, Black, Black, Black, Green, Black);
//|--------------|
Element Centre№6(0, 0, 1.05, Straight, Black, Black, Black, Black, Black, Orange);
//| Co7| E11|Co8 |
                                                                        //|--------------|
//СЕРЕДИННЫЕ ГРАНИ                                                                  //|
Co7| E11|Co8 |
Element Edge№1(0, 1.05, -1.05, Straight, Red, Blue, Black, Black, Black, Black);
//|--------------|
Element Edge№2(-1.05, 0, -1.05, Straight, Red, Black, White, Black, Black, Black);
//| E10| Ce7|E12 |
Element Edge№3(0, -1.05, -1.05, Straight, Red, Black, Black, Black, Green, Black);
//|--------------|
Element Edge№4(1.05, 0, -1.05, Straight, Red, Black, Black, Yellow, Black, Black);
//| Co6| E9 |Co5 |
                                                                        //-----------------
Element Edge№5(1.05, 1.05, 0, Straight, Black, Blue, Black, Yellow, Black, Black);
Element Edge№6(-1.05, 1.05, 0, Straight, Black, Blue, White, Black, Black, Black);
Element Edge№7(-1.05, -1.05, 0, Straight, Black, Black, White, Black, Green, Black);
Element Edge№8(1.05, -1.05, 0, Straight, Black, Black, Black, Yellow, Green, Black);

Element Edge№9(0, 1.05, 1.05, Straight, Black, Blue, Black, Black, Black, Orange);
Element Edge№10(-1.05, 0, 1.05, Straight, Black, Black, White, Black, Black, Orange);
Element Edge№11(0, -1.05, 1.05, Straight, Black, Black, Black, Black, Green, Orange);
Element Edge№12(1.05, 0, 1.05, Straight, Black, Black, Black, Yellow, Black, Orange);

CubikRubik::CubikRubik()
{

   _elements.resize(3);

   _elements[0].push_back(&Corner№1);
   _elements[0].push_back(&Corner№2);
   _elements[0].push_back(&Corner№3);
   _elements[0].push_back(&Corner№4);
   _elements[0].push_back(&Corner№5);
   _elements[0].push_back(&Corner№6);
   _elements[0].push_back(&Corner№7);
```

```
    _elements[0].push_back(&Corner№8);

    _elements[1].push_back(&Centre№1);
    _elements[1].push_back(&Centre№2);
    _elements[1].push_back(&Centre№3);
    _elements[1].push_back(&Centre№4);
    _elements[1].push_back(&Centre№5);
    _elements[1].push_back(&Centre№6);

    _elements[2].push_back(&Edge№1);
    _elements[2].push_back(&Edge№2);
    _elements[2].push_back(&Edge№3);
    _elements[2].push_back(&Edge№4);
    _elements[2].push_back(&Edge№5);
    _elements[2].push_back(&Edge№6);
    _elements[2].push_back(&Edge№7);
    _elements[2].push_back(&Edge№8);
    _elements[2].push_back(&Edge№9);
    _elements[2].push_back(&Edge№10);
    _elements[2].push_back(&Edge№11);
    _elements[2].push_back(&Edge№12);


}

void display();

void CubikRubik::getFile()
{

    std::string n;
    std::cout << n << std::endl;

}

void CubikRubik::draw()
{
    for (auto& _element : _elements)
    {
        for (auto j : _element)
        {
            j->draw();
        }
    }
}




////////////////ПРОМЕЖУТОЧНЫЕ ПОВОРОТЫ//////////////////

//Вверх и Вниз
```

```
void CubikRubik::UpAndDown(std::vector<int> corners, std::vector<int> edges, int centre,
double degree)
{
   double a = degree * (3.141592653589793238463 / 180);

   _elements[1][centre] ->_rotation = CenterUpAndDownRotation;

   _elements[1][centre] -> _angle = a;

   for (int i = 0; i < 90 / std::abs(degree); i++)
   {

      for (auto j : corners)
      {

         _elements[0][j]->_rotation = CenterUpAndDownRotation;
         double Y = _elements[0][j]->_position.y * cos(a) - _elements[0][j]->_position.z * sin(a);
         double Z = _elements[0][j]->_position.z * cos(a) + _elements[0][j]->_position.y * sin(a);
         _elements[0][j]->_position.y = Y;
         _elements[0][j]->_position.z = Z;
         _elements[0][j]->_angle = a;

      }

      for (auto j : edges)
      {

         _elements[2][j]->_rotation = CenterUpAndDownRotation;
         double Y = _elements[2][j]->_position.y * cos(a) - _elements[2][j]->_position.z * sin(a);
         double Z = _elements[2][j]->_position.z * cos(a) + _elements[2][j]->_position.y * sin(a);
         _elements[2][j]->_position.y = Y;
         _elements[2][j]->_position.z = Z;
         _elements[2][j]->_angle = a;

      }
      display();

   }
   for (auto j : corners)
   {

      _elements[0][j]->_angle = 0;
      switch (_elements[0][j]->_orientation)
      {

         case Straight:
         {
            if (degree > 0)
            {
               _elements[0][j]->_orientation = BackSide;
            }
            else
```

```
    {
        _elements[0][j]->_orientation = FrontSide;
    }

    continue;
}
case LeftSide:
{
    continue;
}
case RightSide:
{
    continue;
}
case FrontSide:
{
    if (degree > 0)
    {

        _elements[0][j]->_orientation = Straight;

    }
    else
    {

        _elements[0][j]->_orientation = BottomSide;

    }

    continue;
}
case BackSide:
{
    if (degree > 0)
    {

        _elements[0][j]->_orientation = BottomSide;

    }
    else
    {

        _elements[0][j]->_orientation = Straight;

    }

    continue;
}
case BottomSide:
{
    if (degree > 0)
    {
```

```
                _elements[0][j]->_orientation = FrontSide;

            }
            else
            {

                _elements[0][j]->_orientation = BackSide;

            }

            continue;
        }
        default:
        {

            std::cerr << "Error. Невозможная операция";
            exit(EXIT_FAILURE);

        }

    }

}
for (auto j : edges)
{

    _elements[2][j]->_angle = 0;
    switch (_elements[2][j]->_orientation)
    {

        case Straight:
        {
            if (degree > 0)
            {

                _elements[2][j]->_orientation = BackSide;

            }
            else
            {

                _elements[2][j]->_orientation = FrontSide;

            }

            continue;
        }
        case LeftSide:
        {
            continue;
        }
```

```
case RightSide:
{
   continue;
}
case FrontSide:
{
   if (degree > 0)
   {

      _elements[2][j]->_orientation = Straight;

   }
   else
   {

      _elements[2][j]->_orientation = BottomSide;

   }

   continue;
}
case BackSide:
{
   if (degree > 0)
   {

      _elements[2][j]->_orientation = BottomSide;

   }
   else
   {

      _elements[2][j]->_orientation = Straight;

   }

   continue;
}
case BottomSide:
{
   if (degree > 0)
   {

      _elements[2][j]->_orientation = FrontSide;

   }
   else
   {

      _elements[2][j]->_orientation = BackSide;

   }
```

```
            continue;
        }
        default:
        {

            std::cerr << "Error. Невозможная операция";
            exit(EXIT_FAILURE);

        }

    }
}

_elements[1][centre]->_angle = 0;

std::swap(*_elements[0][corners[0]], *_elements[0][corners[1]]);
std::swap(*_elements[0][corners[1]], *_elements[0][corners[2]]);
std::swap(*_elements[0][corners[2]], *_elements[0][corners[3]]);

std::swap(*_elements[2][edges[0]], *_elements[2][edges[1]]);
std::swap(*_elements[2][edges[1]], *_elements[2][edges[2]]);
std::swap(*_elements[2][edges[2]], *_elements[2][edges[3]]);
}

//Влево и Вправо
void CubikRubik::LeftAndRight(std::vector<int> corners, std::vector<int> edges, int centre,
double degree)
{

    double a = degree * (3.141592653589793238463 / 180);
    _elements[1][centre]->_rotation = CenterLeftAndRightRotation;
    _elements[1][centre]->_angle = a;
    for (int i = 0; i < 90 / std::abs(degree); i++)
    {

        for (auto j : corners)
        {

            _elements[0][j]->_rotation = CenterLeftAndRightRotation;
            double X = _elements[0][j]->_position.x * cos(a) - _elements[0][j]->_position.z * sin(a);
            double Z = _elements[0][j]->_position.z * cos(a) + _elements[0][j]->_position.x * sin(a);
            _elements[0][j]->_position.x = X;
            _elements[0][j]->_position.z = Z;
            _elements[0][j]->_angle = a;

        }

        for (auto j : edges)
        {

            _elements[2][j]->_rotation = CenterLeftAndRightRotation;
```

```
        double X = _elements[2][j]->_position.x * cos(a) - _elements[2][j]->_position.z * sin(a);
        double Z = _elements[2][j]->_position.z * cos(a) + _elements[2][j]->_position.x * sin(a);
        _elements[2][j]->_position.x = X;
        _elements[2][j]->_position.z = Z;
        _elements[2][j]->_angle = a;

    }
    display();
}
for (auto j : corners)
{

    _elements[0][j]->_angle = 0;
    switch (_elements[0][j]->_orientation)
    {

        case Straight:
        {
            continue;
        }
        case LeftSide:
        {
            if (degree > 0)
            {

                _elements[0][j]->_orientation = FrontSide;

            }
            else
            {

                _elements[0][j]->_orientation = BackSide;

            }

            continue;
        }
        case RightSide:
        {
            if (degree > 0)
            {

                _elements[0][j]->_orientation = BackSide;

            }
            else
            {

                _elements[0][j]->_orientation = FrontSide;

            }
```

```
            continue;
        }
        case FrontSide:
        {
            if (degree > 0)
            {

                _elements[0][j]->_orientation = RightSide;

            }
            else
            {

                _elements[0][j]->_orientation = LeftSide;

            }

            continue;
        }
        case BackSide:
        {
            if (degree > 0)
            {

                _elements[0][j]->_orientation = LeftSide;

            }
            else
            {

                _elements[0][j]->_orientation = RightSide;

            }

            continue;
        }
        case BottomSide:
        {
            continue;
        }
        default:
        {

            std::cerr << "Error. Невозможная операция";
            exit(EXIT_FAILURE);

        }
    }
}
for (auto j : edges)
{
    _elements[2][j]->_angle = 0;
```

```
switch (_elements[2][j]->_orientation)
{

   case Straight:
   {
      continue;
   }
   case LeftSide:
   {
      if (degree > 0)
      {

         _elements[2][j]->_orientation = FrontSide;

      }
      else
      {

         _elements[2][j]->_orientation = BackSide;

      }

      continue;
   }
   case RightSide:
   {
      if (degree > 0)
      {

         _elements[2][j]->_orientation = BackSide;

      }
      else
      {

         _elements[2][j]->_orientation = FrontSide;

      }

      continue;
   }
   case FrontSide:
   {
      if (degree > 0)
      {

         _elements[2][j]->_orientation = RightSide;

      }
      else
      {
```

```
                _elements[2][j]->_orientation = LeftSide;

            }

            continue;
        }
        case BackSide:
        {
            if (degree > 0)
            {

                _elements[2][j]->_orientation = LeftSide;

            }
            else
            {

                _elements[2][j]->_orientation = RightSide;

            }

            continue;
        }
        case BottomSide:
        {
            continue;
        }
        default:
        {

            std::cerr << "Error. Невозможная операция";
            exit(EXIT_FAILURE);

        }

    }
  }
  _elements[1][centre]->_angle = 0;
  std::swap(*_elements[0][corners[0]], *_elements[0][corners[1]]);
  std::swap(*_elements[0][corners[1]], *_elements[0][corners[2]]);
  std::swap(*_elements[0][corners[2]], *_elements[0][corners[3]]);

  std::swap(*_elements[2][edges[0]], *_elements[2][edges[1]]);
  std::swap(*_elements[2][edges[1]], *_elements[2][edges[2]]);
  std::swap(*_elements[2][edges[2]], *_elements[2][edges[3]]);
}

//Поврот по часовой стрелке
void CubikRubik::Clockwise(std::vector<int> corners, std::vector<int> edges, int center, double
degree)
{
```

```
double a = degree * (3.14159265358979323846 / 180);
_elements[1][center]->_rotation = ClockwiseRotation;
_elements[1][center]->_angle = a;
for (int i = 0; i < 90 / std::abs(degree); i++)
{

   for (auto j : corners)
   {

      _elements[0][j]->_rotation = ClockwiseRotation;
      double X = _elements[0][j]->_position.x * cos(a) - _elements[0][j]->_position.y * sin(a);
      double Y = _elements[0][j]->_position.y * cos(a) + _elements[0][j]->_position.x * sin(a);
      _elements[0][j]->_position.x = X;
      _elements[0][j]->_position.y = Y;
      _elements[0][j]->_angle = a;

   }
   for (auto j : edges)
   {

      _elements[2][j]->_rotation = ClockwiseRotation;
      double X = _elements[2][j]->_position.x * cos(a) - _elements[2][j]->_position.y * sin(a);
      double Y = _elements[2][j]->_position.y * cos(a) + _elements[2][j]->_position.x * sin(a);
      _elements[2][j]->_position.x = X;
      _elements[2][j]->_position.y = Y;
      _elements[2][j]->_angle = a;

   }
   display();

}
for (auto j : corners)
{

   _elements[0][j]->_angle = 0;

   switch (_elements[0][j]->_orientation)
   {
      case Straight:
      {
         if (degree > 0)
         {

            _elements[0][j]->_orientation = LeftSide;

         }
         else
         {

            _elements[0][j]->_orientation = RightSide;

         }
```

```
    continue;
}
case LeftSide:
{
    if (degree > 0)
    {

        _elements[0][j]->_orientation = BottomSide;

    }
    else
    {

        _elements[0][j]->_orientation = Straight;

    }

    continue;
}
case RightSide:
{
    if (degree > 0)
    {
        _elements[0][j]->_orientation = Straight;
    }
    else
    {
        _elements[0][j]->_orientation = BottomSide;
    }

    continue;
}
case FrontSide:
{
    continue;
}
case BackSide:
{
    continue;
}
case BottomSide:
{
    if (degree > 0)
    {

        _elements[0][j]->_orientation = RightSide;

    }
    else
    {

        _elements[0][j]->_orientation = LeftSide;
```

```
            }

            continue;
        }
        default:
        {

            std::cerr << "Error. Невозможная операция";
            exit(EXIT_FAILURE);

        }
    }

}
for (auto j : edges)
{

    _elements[2][j]->_angle = 0;
    switch (_elements[2][j]->_orientation)
    {
        case Straight:
        {
            if (degree > 0)
            {

                _elements[2][j]->_orientation = LeftSide;

            }
            else
            {

                _elements[2][j]->_orientation = RightSide;

            }
            continue;
        }
        case LeftSide:
        {
            if (degree > 0)
            {

                _elements[2][j]->_orientation = BottomSide;

            }
            else
            {

                _elements[2][j]->_orientation = Straight;

            }
```

```
            continue;
        }
        case RightSide:
        {
            if (degree > 0)
            {

                _elements[2][j]->_orientation = Straight;

            }
            else
            {

                _elements[2][j]->_orientation = BottomSide;

            }

            continue;
        }
        case FrontSide:
        {
            continue;
        }
        case BackSide:
        {
            continue;
        }
        case BottomSide:
        {
            if (degree > 0)
            {

                _elements[2][j]->_orientation = RightSide;

            }
            else
            {

                _elements[2][j]->_orientation = LeftSide;

            }

            continue;
        }
        default:
        {

            std::cerr << "Error. Невозможная операция";
            exit(EXIT_FAILURE);

        }
    }
```

```
    }
    _elements[1][center]->_angle = 0;
    std::swap(*_elements[0][corners[0]], *_elements[0][corners[1]]);
    std::swap(*_elements[0][corners[1]], *_elements[0][corners[2]]);
    std::swap(*_elements[0][corners[2]], *_elements[0][corners[3]]);

    std::swap(*_elements[2][edges[0]], *_elements[2][edges[1]]);
    std::swap(*_elements[2][edges[1]], *_elements[2][edges[2]]);
    std::swap(*_elements[2][edges[2]], *_elements[2][edges[3]]);
}

//Центральную фронтальная грань влево и вправо
void CubikRubik::CenterLeftAndRight(std::vector<int> edges, std::vector<int> centers, double
degree)
{
    double a = degree * (3.141592653589793238463 / 180);
    for (int i = 0; i < 90 / std::abs(degree); i++)
    {

        for (auto j : edges)
        {

            _elements[2][j]->_rotation = CenterLeftAndRightRotation;
            double X = _elements[2][j]->_position.x * cos(a) - _elements[2][j]->_position.z * sin(a);
            double Z = _elements[2][j]->_position.z * cos(a) + _elements[2][j]->_position.x * sin(a);
            _elements[2][j]->_position.x = X;
            _elements[2][j]->_position.z = Z;
            _elements[2][j]->_angle = a;

        }
        for (auto j : centers)
        {

            _elements[1][j]->_rotation = CenterLeftAndRightRotation;
            double X = _elements[1][j]->_position.x * cos(a) - _elements[1][j]->_position.z * sin(a);
            double Z = _elements[1][j]->_position.z * cos(a) + _elements[1][j]->_position.x * sin(a);
            _elements[1][j]->_position.x = X;
            _elements[1][j]->_position.z = Z;
            _elements[1][j]->_angle = a;

        }
        display();
    }
    for (auto j : edges)
    {

        _elements[2][j]->_angle = 0;
        switch (_elements[2][j]->_orientation)
        {
            case Straight:
            {
```

```
    continue;
}
case LeftSide:
{
    if (degree > 0)
    {

        _elements[2][j]->_orientation = FrontSide;

    }
    else
    {

        _elements[2][j]->_orientation = BackSide;

    }

    continue;
}
case RightSide:
{
    if (degree > 0)
    {

        _elements[2][j]->_orientation = BackSide;

    }
    else
    {

        _elements[2][j]->_orientation = FrontSide;

    }

    continue;
}
case FrontSide:
{
    if (degree > 0)
    {

        _elements[2][j]->_orientation = RightSide;

    }
    else
    {

        _elements[2][j]->_orientation = LeftSide;

    }

    continue;
```

```
            }
            case BackSide:
            {
               if (degree > 0)
               {

                  _elements[2][j]->_orientation = LeftSide;

               }
               else
               {

                  _elements[2][j]->_orientation = RightSide;

               }

               continue;
            }
            case BottomSide:
            {
               continue;
            }
            default:
            {

               std::cerr << "Error. Невозможная операция";
               exit(EXIT_FAILURE);

            }
         }

      }

      for (auto j : centers)
      {

         _elements[1][j]->_angle = 0;

      }
      std::swap(*_elements[2][edges[0]], *_elements[2][edges[1]]);
      std::swap(*_elements[2][edges[1]], *_elements[2][edges[2]]);
      std::swap(*_elements[2][edges[2]], *_elements[2][edges[3]]);

      std::swap(*_elements[1][centers[0]], *_elements[1][centers[1]]);
      std::swap(*_elements[1][centers[1]], *_elements[1][centers[2]]);
      std::swap(*_elements[1][centers[2]], *_elements[1][centers[3]]);
   }

   //Центральную боковую грань вверх и вниз
   void CubikRubik::CenterUpAndDown(std::vector<int> edges, std::vector<int> centers, double
   degree)
   {
```

```
double a = degree * (3.14159265358979323846 / 180);
for (int i = 0; i < 90 / std::abs(degree); i++)
{

   for (auto j : edges)
   {

      _elements[2][j]->_rotation = CenterUpAndDownRotation;
      double Y = _elements[2][j]->_position.y * cos(a) - _elements[2][j]->_position.z * sin(a);
      double Z = _elements[2][j]->_position.z * cos(a) + _elements[2][j]->_position.y * sin(a);
      _elements[2][j]->_position.y = Y;
      _elements[2][j]->_position.z = Z;
      _elements[2][j]->_angle = a;

   }
   for (auto j : centers)
   {

      _elements[1][j]->_rotation = CenterUpAndDownRotation;
      double Y = _elements[1][j]->_position.y * cos(a) - _elements[1][j]->_position.z * sin(a);
      double Z = _elements[1][j]->_position.z * cos(a) + _elements[1][j]->_position.y * sin(a);
      _elements[1][j]->_position.y = Y;
      _elements[1][j]->_position.z = Z;
      _elements[1][j]->_angle = a;

   }
   display();
}

for (auto j : edges)
{

   _elements[2][j]->_angle = 0;
   switch (_elements[2][j]->_orientation)
   {
      case Straight:
      {
         if (degree > 0)
         {

            _elements[2][j]->_orientation = BackSide;

         }
         else
         {

            _elements[2][j]->_orientation = FrontSide;

         }

         continue;
      }
```

```
case LeftSide:
{
   continue;
}
case RightSide:
{
   continue;
}
case FrontSide:
{
   if (degree > 0)
   {

      _elements[2][j]->_orientation = Straight;

   }
   else
   {

      _elements[2][j]->_orientation = BottomSide;

   }

   continue;
}
case BackSide:
{
   if (degree > 0)
   {

      _elements[2][j]->_orientation = BottomSide;

   }
   else
   {

      _elements[2][j]->_orientation = Straight;

   }

   continue;
}
case BottomSide:
{
   if (degree > 0)
   {

      _elements[2][j]->_orientation = FrontSide;

   }
   else
   {
```

```
                _elements[2][j]->_orientation = BackSide;

            }

            continue;
        }
        default:
        {

            std::cerr << "Error. Невозможная операция";
            exit(EXIT_FAILURE);

        }
    }

}

for (auto j : centers)
{

    _elements[1][j]->_angle = 0;

}
std::swap(*_elements[2][edges[0]], *_elements[2][edges[1]]);
std::swap(*_elements[2][edges[1]], *_elements[2][edges[2]]);
std::swap(*_elements[2][edges[2]], *_elements[2][edges[3]]);

std::swap(*_elements[1][centers[0]], *_elements[1][centers[1]]);
std::swap(*_elements[1][centers[1]], *_elements[1][centers[2]]);
std::swap(*_elements[1][centers[2]], *_elements[1][centers[3]]);
}
```

```
////////////////ПОВОРОТЫ ГРАНЕЙ//////////////////////
double degree = 6;

//Правую боковую грань вверх
void CubikRubik::RIGHT_UP()
{

    std::vector<int> corners = { 0, 3, 7, 4 };
    std::vector<int> edges = { 3, 7, 11, 4 };
    UpAndDown(corners, edges, 3, degree);
    std::cout << "RIGHT->up; ";
```

```
}

//Правую боковую грань вниз
void CubikRubik::RIGHT_DOWN()
{

    std::vector<int> corners = { 0, 4, 7, 3 };
    std::vector<int> edges = { 3, 4, 11, 7 };
    UpAndDown(corners, edges, 3, -degree);
    std::cout << "RIGHT->down; ";

}

//Левую боковую грань вверх
void CubikRubik::LEFT_UP()
{

    std::vector<int> corners = { 1, 2, 6, 5 };
    std::vector<int> edges = { 1, 6, 9, 5 };
    UpAndDown(corners, edges, 2, degree);
    std::cout << "LEFT->up; ";

}

//Левую боковую грань вниз
void CubikRubik::LEFT_DOWN()
{

    std::vector<int> corners = { 1, 5, 6, 2 };
    std::vector<int> edges = { 1, 5, 9, 6 };
    UpAndDown(corners, edges, 2, -degree);
    std::cout << "LEFT->down; ";

}

//Верхнюю грань вправо
void CubikRubik::UP_LEFT()
{

    std::vector<int> corners = { 0, 4, 5, 1 };
    std::vector<int> edges = { 0, 4, 8, 5 };
    LeftAndRight(corners, edges, 1, -degree);
    std::cout << "UP->left; ";

}

//Верхнюю грань влево
void CubikRubik::UP_RIGHT()
{

    std::vector<int> corners = { 0, 1, 5, 4 };
    std::vector<int> edges = { 0, 5, 8, 4 };
```

```
    LeftAndRight(corners, edges, 1, degree);
    std::cout << "UP->right; ";

}

//Нижнюю нрань вправо
void CubikRubik::DOWN_RIGHT()
{

    std::vector<int> corners = { 2, 6, 7, 3 };
    std::vector<int> edges = { 2, 6, 10, 7 };
    LeftAndRight(corners, edges, 4, degree);
    std::cout << "DOWN->right; ";

}

//Нижнюю грань влево
void CubikRubik::DOWN_LEFT()
{

    std::vector<int> corners = { 2, 3, 7, 6 };
    std::vector<int> edges = { 2, 7, 10, 6 };
    LeftAndRight(corners, edges, 4, -degree);
    std::cout << "DOWN->left; ";

}

//Фронтальную грань направо
void CubikRubik::FRONT_RIGHT()
{

    std::vector<int> corners = { 0, 1, 2, 3 };
    std::vector<int> edges = { 0, 1, 2, 3 };
    Clockwise(corners, edges, 0, -degree);
    std::cout << "FRONT->right; ";

}

//Фронтальную грань налево
void CubikRubik::FRONT_LEFT()
{

    std::vector<int> corners = { 0, 3, 2, 1 };
    std::vector<int> edges = { 0, 3, 2, 1 };
    Clockwise(corners, edges, 0, degree);
    std::cout << "FRONT->left; ";

}

//Центральную фронтальную грань влево
void CubikRubik::CENTER_RIGHT()
{
```

```
    std::vector<int> edges = { 3, 1, 9, 11 };
    std::vector<int> centers = { 0, 2, 5, 3 };
    CenterLeftAndRight(edges, centers, degree);
    std::cout << "CENTER->right; ";

}

//Центральную фронтальную грань вправо
void CubikRubik::CENTER_LEFT()
{

    std::vector<int> edges = { 3, 11, 9, 1 };
    std::vector<int> centers = { 0, 3, 5, 2 };
    CenterLeftAndRight(edges, centers, -degree);
    std::cout << "CENTER->left; ";

}

//Центральную боковую грань вверх
void CubikRubik::CENTER_UP()
{

    std::vector<int> edges = { 0, 2, 10, 8 };
    std::vector<int> centers = { 0, 4, 5, 1 };
    CenterUpAndDown(edges, centers, degree);
    std::cout << "CENTER->up; ";

}

//Цетральную боковую грань вниз
void CubikRubik::CENTER_DOWN()
{

    std::vector<int> edges = { 0, 8, 10, 2 };
    std::vector<int> centers = { 0, 1, 5, 4 };
    CenterUpAndDown(edges, centers, -degree);
    std::cout << "CENTER->down; ";

}
```

////////////////КОМБИНАЦИИ//////////////////////////

//Классический пиф-паф. Правую от себя, верхнюю влево, правую на себя, верхнюю вправо.

```
void CubikRubik::PifPaf()
```

```
{

   RIGHT_UP();
   UP_LEFT();
   RIGHT_DOWN();
   UP_RIGHT();

}

//Ревёрснутый пиф-паф. То же самое, но для левой стороны.
void CubikRubik::ReversedPifPaf()
{

   LEFT_UP();
   UP_RIGHT();
   LEFT_DOWN();
   UP_LEFT();

}

//Всё налево
void CubikRubik::AllLeft()
{

   UP_LEFT();
   CENTER_LEFT();
   DOWN_LEFT();

}

//Всё направо
void CubikRubik::AllRight()
{

   UP_RIGHT();
   CENTER_RIGHT();
   DOWN_RIGHT();

}

//Всё вверх
void CubikRubik::AllUp()
{

   LEFT_UP();
   RIGHT_UP();
   CENTER_UP();

}

//Всё вниз
void CubikRubik::AllDown()
```

```
{

   LEFT_DOWN();
   RIGHT_DOWN();
   CENTER_DOWN();

}









////////////////СБОРКА И РАЗБОРКА//////////////////////////

//Собиратель кубика
void CubikRubik::RubiksCubeAssembler()
{

   std::cout << "\n\nСБОРКА:\n\n";
   StepOne(); //Правильный крест
   StepTwo(); //Ребро первого слоя
   StepThree(); //Углы первого слоя
   StepFour(); //Рёбра среднего слоя
   StepFive(); //Крест последнего слоя
   StepSix(); //Правильный крест последнего слоя
   StepSeven(); //Расстановка углов последнего слоя
   StepEight();//Разворот углов третьего слоя

}

//Разбиратель кубика
void CubikRubik::RubiksCubeDisassembler()
{

   std::cout << "\n\nРАЗБОРКА:\n\n";
   srand(time(NULL));
   for (int i = 0; i < 50; i++)
   {

      int rotation = rand() % 10 + 0;
      switch (rotation)
      {

         case Right1Rotation://Правую боковую грань вниз
         {
            RIGHT_UP();
            continue;
```

```
        }
        case Right2Rotation://Правую боковую грань вверх
        {
            RIGHT_DOWN();
            continue;
        }
        case Left1Rotation://Левую боковую грань вверх
        {
            LEFT_DOWN();
            continue;
        }
        case Left2Rotation://Левую боковую грань вниз
        {
            LEFT_UP();
            continue;
        }
        case Up1Rotation://Верхнюю грань вправо
        {
            UP_LEFT();
            continue;
        }
        case Down1Rotation://Нижнюю нрань вправо
        {
            DOWN_RIGHT();
            continue;
        }
        case Down2Rotation://Нижнюю грань влево
        {
            DOWN_LEFT();
            continue;
        }
        case Front2Rotation://Фронтальную грань налево
        {
            FRONT_LEFT();
            continue;
        }
        case Front1Rotation://Фронтальную грань направо
        {
            FRONT_RIGHT();
            continue;
        }

    }

}

for (auto& i : _elements)
{

    for (auto& j : i)
    {
```

```
            j->RightPosition = false;
            j->CornersPosition = false;

        }
    }
}
```

```
////////////////АЛГОРИТМ СБОРКИ//////////////////////////

//Правильный крест
void CubikRubik::StepOne()
{

    while (!_elements[2][2]->RightPosition || !_elements[2][7]->RightPosition || !_elements[2][6]->RightPosition || !_elements[2][10]->RightPosition)
    {

        int element_index = FindElement(Green, _Edge);

        if (element_index == 2 || element_index == 6 || element_index == 7 || element_index == 10)
        {

            if (element_index == 6)
            {

                AllRight();

            }
            else if (element_index == 7)
            {

                AllLeft();

            }
            else if (element_index == 10)
            {

                AllLeft();
                AllLeft();

            }
```

```
        element_index = 2;

        if (_elements[2][element_index]->_orientation == Straight)
        {

            bool right_center = false;
            for (int i = 0; i < 6; i++)
            {

                for (int j = 0; j < 6; j++)
                {

                    if (_elements[1][0]->FieldColour[i] == _elements[2][element_index]-
>FieldColour[j] && _elements[1][0]->FieldColour[i] != Black)
                    {

                        right_center = true;

                    }

                }

            }
            if (!right_center)
            {

                FRONT_LEFT();
                FRONT_LEFT();
                continue;

            }
            else
            {

                _elements[2][element_index]->RightPosition = true;
                continue;

            }
        }
        else
        {

            FRONT_LEFT();
            PifPaf();
            UP_LEFT();
            FRONT_LEFT();
            FRONT_LEFT();
            continue;

        }
    }
    if (element_index == 3)
```

```
{

   int turn_amount = 0;

   if (!_elements[2][2]->RightPosition)
   {

      FRONT_LEFT();

   }
   else
   {

      while (_elements[2][2]->RightPosition)
      {

         DOWN_LEFT();
         turn_amount++;

      }

      FRONT_LEFT();

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

         DOWN_RIGHT();

      }
   }

}
else if (element_index == 1)
{

   AllRight();
   int turn_amount = 0;

   if (!_elements[2][2]->RightPosition)
   {

      FRONT_LEFT();

   }
   else
   {

      while (_elements[2][2]->RightPosition)
      {

         DOWN_LEFT();
         turn_amount++;
```

```
        }

        FRONT_LEFT();

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

            DOWN_RIGHT();

        }
    }

}
else if (element_index == 9)
{

    AllRight();
    AllRight();
    int turn_amount = 0;

    if (!_elements[2][2]->RightPosition)
    {

        FRONT_LEFT();

    }
    else
    {

        while (_elements[2][2]->RightPosition)
        {

            DOWN_LEFT();
            turn_amount++;

        }

        FRONT_LEFT();

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

            DOWN_RIGHT();

        }
    }

}
else if (element_index == 11)
{
```

```
    AllLeft();
    int turn_amount = 0;

    if (!_elements[2][2]->RightPosition)
    {

        FRONT_LEFT();

    }
    else
    {

        while (_elements[2][2]->RightPosition)
        {

            DOWN_LEFT();
            turn_amount++;

        }

        FRONT_LEFT();

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

            DOWN_RIGHT();

        }
    }

}
if (element_index == 4)
{

    UP_LEFT();

}
else if (element_index == 5)
{

    UP_RIGHT();

}
else if (element_index == 8)
{

    UP_LEFT();
    UP_LEFT();

}

element_index = 0;
```

```
        bool right_center = false;

        while (!right_center)
        {

            for (int i = 0; i < 6; i++)
            {

                for (int j = 0; j < 6; j++)
                {

                    if (_elements[1][0]->FieldColour[i] == _elements[2][element_index]-
>FieldColour[j] && _elements[1][0]->FieldColour[i] != Black)
                    {

                        right_center = true;

                    }

                }

            }
            if (!right_center)
            {

                DOWN_RIGHT();
                CENTER_RIGHT();

            }
        }
        if (_elements[2][element_index]->_orientation == BottomSide)
        {

            FRONT_LEFT();
            FRONT_LEFT();

        }
        else
        {

            FRONT_RIGHT();
            CENTER_LEFT();
            FRONT_LEFT();
            CENTER_RIGHT();

        }

        _elements[2][2]->RightPosition = true;
    }

}
```

```
//Ребро первого слоя
void CubikRubik::StepTwo()
{
   while (!_elements[0][2]->RightPosition || !_elements[0][3]->RightPosition || !_elements[0][6]->RightPosition || !_elements[0][7]->RightPosition)
   {

      int element_index = FindElement(Green, _Corner);

      if (element_index == 2 || element_index == 6 || element_index == 7 || element_index == 3)
      {

         if (element_index == 2)
         {

            AllRight();

         }
         else if (element_index == 7)
         {

            AllLeft();

         }
         else if (element_index == 6)
         {

            AllLeft();
            AllLeft();

         }

         element_index = 3;

         if (_elements[0][element_index]->_orientation == Straight)
         {

            bool right_center_1 = false;
            bool right_center_2 = false;

            for (int i = 0; i < 6; i++)
            {

               for (int j = 0; j < 6; j++)
               {

                  if (_elements[1][0]->FieldColour[i] == _elements[0][element_index]->FieldColour[j] && _elements[1][0]->FieldColour[i] != Black)
                  {

                     right_center_1 = true;
```

```
                }

            }

        }
        for (int i = 0; i < 6; i++)
        {

            for (int j = 0; j < 6; j++)
            {

                if (_elements[1][3]->FieldColour[i] == _elements[0][element_index]-
>FieldColour[j] && _elements[1][3]->FieldColour[i] != Black)
                {

                    right_center_2 = true;

                }

            }

        }
        if (!right_center_1 || !right_center_2)
        {

            PifPaf();

        }
        else
        {

            _elements[0][element_index]->RightPosition = true;

        }
    }
    else
    {
        if (element_index == 2)
        {

            AllRight();
            element_index = 3;

        }
        else if (element_index == 6)
        {

            AllRight();
            AllRight();
            element_index = 3;

        }
```

```
        else if (element_index == 7)
        {

            AllLeft();
            element_index = 3;

        }
        switch (_elements[0][element_index]->_orientation)
        {
            case BackSide:
            {
                for (int i = 0; i < 4; i++)
                {

                    PifPaf();

                }

                _elements[0][3]->RightPosition = true;
                continue;
            }
            case LeftSide:
            {
                for (int i = 0; i < 2; i++)
                {

                    PifPaf();

                }
                _elements[0][3]->RightPosition = true;
                continue;
            }
            case Straight:
            {

                _elements[0][3]->RightPosition = true;
                continue;

            }

        }

    }

    continue;
}
if (element_index == 4)
{

    AllLeft();

}
```

```
        else if (element_index == 5)
        {

            AllRight();
            AllRight();

        }
        else if (element_index == 1)
        {

            AllRight();

        }

        element_index = 0;

        bool right_center_1 = false;
        bool right_center_2 = false;

        while (!right_center_2 || !right_center_1)
        {

            right_center_1 = false;
            right_center_2 = false;

            for (int i = 0; i < 6; i++)
            {

                for (int j = 0; j < 6; j++)
                {

                    if (_elements[1][0]->FieldColour[i] == _elements[0][element_index]-
>FieldColour[j] && _elements[1][0]->FieldColour[i] != Black)
                    {

                        right_center_1 = true;

                    }

                }

            }
            for (int i = 0; i < 6; i++)
            {

                for (int j = 0; j < 6; j++)
                {

                    if (_elements[1][3]->FieldColour[i] == _elements[0][element_index]-
>FieldColour[j] && _elements[1][3]->FieldColour[i] != Black)
                    {
```

```
            right_center_2 = true;

        }

    }

}
if (!right_center_1 || !right_center_2)
{

    DOWN_RIGHT();
    CENTER_RIGHT();

}
}
switch (_elements[0][0]->_orientation)
{
    case FrontSide:
    {

        std::cerr << "ERROR" << '\n';
        exit(EXIT_FAILURE);

    }
    case RightSide:
    {

        std::cerr << "ERROR" << '\n';
        exit(EXIT_FAILURE);

    }
    case BottomSide:
    {

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

            PifPaf();

        }
        _elements[0][3]->RightPosition = true;
        continue;

    }
    case Straight:
    {

        std::cerr << "ERROR" << '\n';
        exit(EXIT_FAILURE);

    }
    case LeftSide:
```

```
            {

                PifPaf();
                _elements[0][3]->RightPosition = true;
                continue;

            }
            case BackSide:
            {

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

                    PifPaf();

                }

                _elements[0][3]->RightPosition = true;
                continue;

            }
            default:
            {

                std::cerr << "ERROR" << '\n';
                exit(EXIT_FAILURE);

            }
        }
    }
}

//Углы первого слоя
void CubikRubik::StepThree()
{

    while (!_elements[2][1]->RightPosition || !_elements[2][3]->RightPosition || !_elements[2][9]->RightPosition || !_elements[2][11]->RightPosition)
    {

        if (_elements[2][0]->FieldColour[1] == Black)
        {

            bool right_center = false;

            while (!right_center)
            {

                for (int i = 0; i < 6; i++)
                {

                    for (int j = 0; j < 6; j++)
```

```
            {

                if (_elements[2][0]->FieldColour[i] == _elements[1][0]->FieldColour[j] &&
_elements[2][0]->FieldColour[i] != Black)
                {

                    right_center = true;

                }

            }

        }
        if (!right_center)
        {

            DOWN_RIGHT();
            CENTER_RIGHT();

        }

    }

    bool color = false;

    switch (_elements[2][0]->_orientation)
    {

        case LeftSide:
        {

            for (int i = 0; i < 6; i++)
            {

                if (_elements[2][0]->FieldColour[i] == _elements[1][3]->FieldColour[i] &&
_elements[2][0]->FieldColour[i] != Black)
                {

                    color = true;

                }

            }
            if (!color)
            {

                DOWN_RIGHT();
                CENTER_RIGHT();

            }

            UP_LEFT();
```

```
                PifPaf();
                AllLeft();
                ReversedPifPaf();
                AllRight();

                _elements[2][3]->RightPosition = true;

                continue;
            }
            case RightSide:
            {

                for (int i = 0; i < 6; i++)
                {

                    if (_elements[2][0]->FieldColour[i] == _elements[1][2]->FieldColour[i] &&
_elements[2][0]->FieldColour[i] != Black)
                    {

                        color = true;

                    }

                }
                if (!color)
                {

                    DOWN_LEFT();
                    CENTER_LEFT();

                }

                UP_RIGHT();
                ReversedPifPaf();
                AllRight();
                PifPaf();

                _elements[2][3]->RightPosition = true;

                continue;
            }
            default:
            {

                std::cout << "ERROR" << '\n';
                continue;

            }
        }
    }
    else
    {
```

```
            bool not_blue = false;

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

               UP_LEFT();

               if (_elements[2][0]->FieldColour[1] == Black)
               {

                  not_blue = true;
                  break;

               }

            }
            if (not_blue)
            {

               continue;

            }
            else
            {

               while (_elements[2][3]->RightPosition)
               {

                  DOWN_RIGHT();
                  CENTER_RIGHT();

               }

               PifPaf();
               AllLeft();
               ReversedPifPaf();
               AllRight();
            }

         }

      }

}

//Рёбра среднего слоя
void CubikRubik::StepFour()
{

   while (true)
   {
```

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

            if (_elements[2][5]->_orientation == Straight && _elements[2][4]->_orientation ==
Straight)
            {

                FRONT_RIGHT();
                PifPaf();
                FRONT_LEFT();
                return;

            }

            UP_LEFT();

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

            if (_elements[2][5]->_orientation == Straight && _elements[2][8]->_orientation ==
Straight)
            {

                FRONT_RIGHT();
                PifPaf();
                PifPaf();
                FRONT_LEFT();
                return;

            }

            UP_LEFT();
        }

        FRONT_RIGHT();
        PifPaf();
        FRONT_LEFT();
    }

}

//Крест последнего слоя
void CubikRubik::StepFive()
{
    while (!_elements[2][0]->RightPosition || !_elements[2][4]->RightPosition || !_elements[2][5]-
>RightPosition || !_elements[2][8]->RightPosition)
    {
        bool first_one = false;
        bool second_one = false;
        bool third_one = false;
        bool fourth_one = false;
```

```
        for (int j = 0; j < 6; j++)
        {

            if (_elements[2][4]->FieldColour[j] == _elements[1][3]->FieldColour[j] &&
_elements[2][4]->FieldColour[j] != Black)
            {

                first_one = true;

            }

        }
        for (int j = 0; j < 6; j++)
        {

            if (_elements[2][0]->FieldColour[j] == _elements[1][0]->FieldColour[j] &&
_elements[2][0]->FieldColour[j] != Black)
            {

                second_one = true;

            }

        }
        for (int j = 0; j < 6; j++)
        {

            if (_elements[2][5]->FieldColour[j] == _elements[1][2]->FieldColour[j] &&
_elements[2][5]->FieldColour[j] != Black)
            {

                third_one = true;

            }

        }
        for (int j = 0; j < 6; j++)
        {

            if (_elements[2][8]->FieldColour[j] == _elements[1][5]->FieldColour[j] &&
_elements[2][8]->FieldColour[j] != Black)
            {

                fourth_one = true;

            }

        }
        if (first_one && second_one && third_one && fourth_one)
        {
```

```
            _elements[2][0]->RightPosition = true;
            _elements[2][4]->RightPosition = true;
            _elements[2][5]->RightPosition = true;
            _elements[2][8]->RightPosition = true;
            break;

        }

        CENTER_RIGHT();
        DOWN_RIGHT();

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

            bool first_element = false;
            bool second_element = false;
            bool third_element = false;

            for (int j = 0; j < 6; j++)
            {

                if (_elements[2][4]->FieldColour[j] == _elements[1][3]->FieldColour[j] &&
_elements[2][4]->FieldColour[j] != Black)
                {

                    first_element = true;

                }

            }
            if (!first_element)
            {

                UP_LEFT();
                continue;

            }
            for (int j = 0; j < 6; j++)
            {

                if (_elements[2][8]->FieldColour[j] == _elements[1][5]->FieldColour[j] &&
_elements[2][8]->FieldColour[j] != Black)
                {

                    second_element = true;

                }

            }
            if (second_element)
            {
```

```
            RIGHT_UP();
            UP_LEFT();
            RIGHT_DOWN();
            UP_LEFT();
            RIGHT_UP();
            UP_LEFT();
            UP_LEFT();
            RIGHT_DOWN();
            UP_LEFT();
            break;

        }
        for (int j = 0; j < 6; j++)
        {

            if (_elements[2][5]->FieldColour[j] == _elements[1][2]->FieldColour[j] &&
_elements[2][5]->FieldColour[j] != Black)
            {

                third_element = true;

            }

        }
        if (third_element)
        {

            RIGHT_UP();
            UP_LEFT();
            RIGHT_DOWN();
            UP_LEFT();
            RIGHT_UP();
            UP_LEFT();
            UP_LEFT();
            RIGHT_DOWN();
            UP_LEFT();
            break;

        }

      }

   }

}

//Правильный крест последнего слоя
void CubikRubik::StepSix()
{
   while (!_elements[0][0]->CornersPosition || !_elements[0][4]->CornersPosition ||
!_elements[0][5]->CornersPosition || !_elements[0][1]->CornersPosition)
   {
```

```
        bool first_col = false;
        bool second_col = false;
        bool first_corner = false;
        bool second_corner = false;
        bool third_corner = false;
        bool fourth_corner = false;

        for (int j = 0; j < 6; j++)
        {

            if (_elements[0][1]->FieldColour[j] == _elements[1][0]->FieldColour[j] &&
_elements[0][1]->FieldColour[j] != Black)
            {

                first_col = true;

            }

        }
        for (int j = 0; j < 6; j++)
        {

            if (_elements[0][1]->FieldColour[j] == _elements[1][2]->FieldColour[j] &&
_elements[0][1]->FieldColour[j] != Black)
            {

                second_col = true;

            }

        }
        if (second_col && first_col)
        {

            first_corner = true;

        }

        first_col = false;
        second_col = false;

        for (int j = 0; j < 6; j++)
        {

            if (_elements[0][0]->FieldColour[j] == _elements[1][0]->FieldColour[j] &&
_elements[0][0]->FieldColour[j] != Black)
            {

                first_col = true;

            }
```

```
        }
        for (int j = 0; j < 6; j++)
        {

            if (_elements[0][0]->FieldColour[j] == _elements[1][3]->FieldColour[j] &&
_elements[0][0]->FieldColour[j] != Black)
            {

                second_col = true;

            }

        }
        if (second_col && first_col)
        {

            second_corner = true;

        }

        first_col = false;
        second_col = false;

        for (int j = 0; j < 6; j++)
        {

            if (_elements[0][4]->FieldColour[j] == _elements[1][5]->FieldColour[j] &&
_elements[0][4]->FieldColour[j] != Black)
            {

                first_col = true;

            }

        }
        for (int j = 0; j < 6; j++)
        {

            if (_elements[0][4]->FieldColour[j] == _elements[1][3]->FieldColour[j] &&
_elements[0][4]->FieldColour[j] != Black)
            {

                second_col = true;

            }

        }
        if (second_col && first_col)
        {

            third_corner = true;
```

```
        }

        first_col = false;
        second_col = false;

        for (int j = 0; j < 6; j++)
        {

            if (_elements[0][5]->FieldColour[j] == _elements[1][2]->FieldColour[j] &&
_elements[0][5]->FieldColour[j] != Black)
            {

                first_col = true;

            }

        }
        for (int j = 0; j < 6; j++)
        {

            if (_elements[0][5]->FieldColour[j] == _elements[1][5]->FieldColour[j] &&
_elements[0][5]->FieldColour[j] != Black)
            {

                second_col = true;

            }

        }
        if (second_col && first_col)
        {

            fourth_corner = true;

        }
        if (first_corner && second_corner && third_corner && fourth_corner)
        {

            _elements[0][0]->CornersPosition = true;
            _elements[0][4]->CornersPosition = true;
            _elements[0][5]->CornersPosition = true;
            _elements[0][1]->CornersPosition = true;
            break;

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

            bool first_color = false;
            bool second_color = false;

            for (int j = 0; j < 6; j++)
```

```
        {

            if (_elements[0][1]->FieldColour[j] == _elements[1][0]->FieldColour[j] &&
_elements[0][1]->FieldColour[j] != Black)
            {

                first_color = true;

            }
        }
        if (!first_color)
        {

            AllRight();
            continue;

        }
        for (int j = 0; j < 6; j++)
        {

            if (_elements[0][1]->FieldColour[j] == _elements[1][2]->FieldColour[j] &&
_elements[0][1]->FieldColour[j] != Black)
            {

                second_color = true;

            }

        }
        if (!second_color)
        {

            AllRight();
            continue;

        }

        break;
    }

    RIGHT_UP();
    UP_RIGHT();
    LEFT_UP();
    UP_LEFT();
    RIGHT_DOWN();
    UP_RIGHT();
    LEFT_DOWN();
    UP_LEFT();

  }
}
```

```
//Расстановка углов последнего слоя
void CubikRubik::StepSeven()
{

    if (_elements[0][0]->_orientation == Straight)
    {

        _elements[0][0]->RightPosition = true;

    }
    if (_elements[0][4]->_orientation == Straight)
    {

        _elements[0][4]->RightPosition = true;

    }
    if (_elements[0][5]->_orientation == Straight)
    {

        _elements[0][5]->RightPosition = true;

    }
    if (_elements[0][1]->_orientation == Straight)
    {

        _elements[0][1]->RightPosition = true;

    }

}

//Разворот углов третьего слоя
void CubikRubik::StepEight()
{

    AllUp();
    AllUp();

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

        while (_elements[0][3]->_orientation != BottomSide)
        {

            PifPaf();

        }

        DOWN_RIGHT();

    }
```

```
    _elements[0][2]->RightPosition = true;
    _elements[0][3]->RightPosition = true;
    _elements[0][6]->RightPosition = true;
    _elements[0][7]->RightPosition = true;

    AllUp();
    AllUp();
}

int CubikRubik::FindElement(Colour colour, ElementType type)
{

    int temp_index = 0;
    switch (type)
    {

        case _Edge:
        {

            temp_index = 2;
            break;

        }
        case _Corner:
        {

            temp_index = 0;
            break;

        }
        case _Centre:
        {

            temp_index = 1;
            break;

        }
        default:
        {

            std::cerr << "Error" << '\n';
            exit(EXIT_FAILURE);

        }
    }
    for (int i = 0; i < _elements[temp_index].size(); i++)
    {

        if (!_elements[temp_index][i]->RightPosition && (_elements[temp_index][i]-
>FieldColour[0] == colour || _elements[temp_index][i]->FieldColour[1] == colour ||
_elements[temp_index][i]->FieldColour[2] == colour || _elements[temp_index][i]-
```

```
>FieldColour[3] == colour || _elements[temp_index][i]->FieldColour[4] == colour ||
_elements[temp_index][i]->FieldColour[5] == colour))
      {

         return i;

      }

   }

   return 0;
}

CubikRubik::~CubikRubik()
{
   for (auto& _element : _elements)
   {

      _element.clear();

   }

   _elements.clear();
}
```

ELEMENTS.CPP

```
#pragma once
#include <glut.h>
#include "Elements.hpp"
#include <cmath>

//Цвета составных кубов
colours Element::BoxColours(Colour It_is_colour)
{
    colours colour;

    switch (It_is_colour)
    {
        case White:
        {
            colour.red = 1;
            colour.green = 1;
            colour.blue = 1;
            return colour;
        }
        case Yellow:
        {
            colour.red = 1;
            colour.green = 0.9;
            colour.blue = 0;
            return colour;
        }
        case Green:
        {
            colour.red = 0.2;
            colour.green = 0.7;
            colour.blue = 0;
            return colour;
        }
```

```
        case Red:
        {
            colour.red = 0.9;
            colour.green = 0.1;
            colour.blue = 0.1;
            return colour;
        }
        case Blue:
        {
            colour.red = 0.2;
            colour.green = 0;
            colour.blue = 0.7;
            return colour;
        }
        case Orange:
        {
            colour.red = 1;
            colour.green = 0.6;
            colour.blue = 0;
            return colour;
        }
        default:
        {
            colour.red = 0.0;
            colour.green = 0.0;
            colour.blue = 0.0;
            return colour;
        }
    }
}

//Размеры и структура составных кубов
Element::Element(float position_x, float position_y, float position_z, Orientation
orientation, Colour colour1, Colour colour2, Colour colour3, Colour colour4, Colour
colour5, Colour colour6)
{
    _orientation = orientation;
    Points position(position_x, position_y, position_z);
    _position = position;
    _surface.resize(6);

    Points miniCube(0.5, -0.5, -0.5);
    _surface[0].push_back(miniCube);
    miniCube.set(0.5, 0.5, -0.5);
    _surface[0].push_back(miniCube);
    miniCube.set(-0.5, 0.5, -0.5);
    _surface[0].push_back(miniCube);
    miniCube.set(-0.5, -0.5, -0.5);
    _surface[0].push_back(miniCube);

    miniCube.set(0.5, 0.5, 0.5);
    _surface[1].push_back(miniCube);
    miniCube.set(0.5, 0.5, -0.5);
    _surface[1].push_back(miniCube);
    miniCube.set(-0.5, 0.5, -0.5);
    _surface[1].push_back(miniCube);
    miniCube.set(-0.5, 0.5, 0.5);
    _surface[1].push_back(miniCube);

    miniCube.set(-0.5, -0.5, 0.5);
    _surface[2].push_back(miniCube);
    miniCube.set(-0.5, 0.5, 0.5);
    _surface[2].push_back(miniCube);
    miniCube.set(-0.5, 0.5, -0.5);
    _surface[2].push_back(miniCube);
```

```
    miniCube.set(-0.5, -0.5, -0.5);
    _surface[2].push_back(miniCube);

    miniCube.set(0.5, -0.5, -0.5);
    _surface[3].push_back(miniCube);
    miniCube.set(0.5, 0.5, -0.5);
    _surface[3].push_back(miniCube);
    miniCube.set(0.5, 0.5, 0.5);
    _surface[3].push_back(miniCube);
    miniCube.set(0.5, -0.5, 0.5);
    _surface[3].push_back(miniCube);

    miniCube.set(0.5, -0.5, -0.5);
    _surface[4].push_back(miniCube);
    miniCube.set(0.5, -0.5, 0.5);
    _surface[4].push_back(miniCube);
    miniCube.set(-0.5, -0.5, 0.5);
    _surface[4].push_back(miniCube);
    miniCube.set(-0.5, -0.5, -0.5);
    _surface[4].push_back(miniCube);

    miniCube.set(0.5, -0.5, 0.5);
    _surface[5].push_back(miniCube);
    miniCube.set(0.5, 0.5, 0.5);
    _surface[5].push_back(miniCube);
    miniCube.set(-0.5, 0.5, 0.5);
    _surface[5].push_back(miniCube);
    miniCube.set(-0.5, -0.5, 0.5);
    _surface[5].push_back(miniCube);

    FieldColour.push_back(colour1);
    FieldColour.push_back(colour2);
    FieldColour.push_back(colour3);
    FieldColour.push_back(colour4);
    FieldColour.push_back(colour5);
    FieldColour.push_back(colour6);
}

//Прорисовка и анимация проворотов
void Element::draw()
{

    colours colour;
    double x = 0, y = 0, z = 0;

    for (int i = 0; i < _surface.size(); i++)
    {

        colour = BoxColours(FieldColour[i]);
        glBegin(GL_POLYGON);

        for (auto& j : _surface[i])
        {

            glColor3f(colour.red, colour.green, colour.blue);
            switch (_rotation)
            {

                case Nothing:
                {

                    x = j.x;
                    y = j.y;
                    z = j.z;
                    break;
```

```
                }
                case CenterUpAndDownRotation:
                {

                    x = j.x;
                    y = (j.y) * cos(_angle) - (j.z) * sin(_angle);
                    z = (j.z) * cos(_angle) + (j.y) * sin(_angle);
                    break;

                }
                case CenterLeftAndRightRotation:
                {

                    x = (j.x) * cos(_angle) - (j.z) * sin(_angle);
                    y = j.y;
                    z = (j.z) * cos(_angle) + (j.x) * sin(_angle);
                    break;

                }
                case ClockwiseRotation:
                {

                    x = (j.x) * cos(_angle) - (j.y) * sin(_angle);
                    y = (j.y) * cos(_angle) + (j.x) * sin(_angle);
                    z = j.z;
                    break;

                }
                default:
                {

                    x = j.x;
                    y = j.y;
                    z = j.z;
                    break;

                }

            }

            glVertex3f(x + _position.x, y + _position.y, z + _position.z);

            j.x = x;
            j.y = y;
            j.z = z;
        }

        glEnd();

    }

}
```

Rubik'sCube.cpp

```
#include <iostream>
#include <glut.h>
#include "Cubik.hpp"

double Xrotate = 0;
double Yrotate = 0;
CubikRubik start;
```

```
void Keys(int k, int x, int y)
{
    //Заданное действие(для примера)
    if (k == GLUT_KEY_F1)
    {
        start.PifPaf();
    }
    //сборка
    else if (k == GLUT_KEY_F2)
    {

        start.RubiksCubeAssembler();

    }
    //разборка
    else if (k == GLUT_KEY_F3)
    {

        start.RubiksCubeDisassembler();

    }
    //Закрыть
    else if (k == GLUT_KEY_F4)
    {

        std::cout << "\nКонец работы программы\n";
        exit(EXIT_SUCCESS);

    }
    //Вращение кубика вокруг оси вправо
    else if (k == GLUT_KEY_RIGHT)
    {

    Yrotate = Yrotate + 8;

    }
    //Вращение кубика вокруг оси влево
    else if (k == GLUT_KEY_LEFT)
    {

    Yrotate = Yrotate - 8;

    }
    //Вращение кубика вокруг оси вверх
    else if (k == GLUT_KEY_UP)
    {

    Xrotate = Xrotate + 8;

    }
    //Вращение кубика вокруг оси вниз
    else if (k == GLUT_KEY_DOWN)
    {

    Xrotate = Xrotate - 8;

    }

    glutPostRedisplay();

}

void display()
{
```

```
    glClearColor(0.9, 0.8, 0.6, 1);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glRotatef(Xrotate, 1.0, 0.0, 0.0);
    glRotatef(Yrotate, 0.0, 1.0, 0.0);
    glScalef(0.2, 0.2, 0.2);
    start.draw();
    glFlush();
    glutSwapBuffers();


}

int main(int argc, char** argv)
{
    setlocale(LC_ALL, "Russian");
    start.getFile();
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(800, 800);
    glutInitWindowPosition(GLUT_INIT_WINDOW_WIDTH / 3 , GLUT_INIT_WINDOW_HEIGHT / 7);
    glutCreateWindow("LAB №8 Maks Shein M3106");
    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LESS);
    glutDisplayFunc(display);
    glutSpecialFunc(Keys);
    glutMainLoop();

    return 0;

}
```

Elements.hpp

```
#pragma once

#include "EnumerationList.hpp"
#include <vector>

//Цвета
class colours
{

public:
    float red = 0;
    float green = 0;
    float blue = 0;

};
//Координаты
struct Points
{

    Points() : x(0), y(0), z(0)
    {

    }
    void set(float _x, float _y, float _z)
    {

        x = _x;
        y = _y;
        z = _z;

    }
```

```
    Points(float x, float y, float z) : x(x), y(y), z(z)
    {

    }
    Points& operator=(const Points& right)
    {

        if (this == &right)
        {

            return *this;

        }

        x = right.x;
        y = right.y;
        z = right.z;

        return *this;
    }

    double x = 0, y = 0, z = 0;
};
//Структура составных кубов
class Element
{

public:
    Element(float, float, float, Orientation, Colour, Colour, Colour, Colour, Colour,
Colour);

    void draw();

    colours BoxColours(Colour);

    std::vector<Colour>FieldColour;
    std::vector<std::vector<Points>>_surface;

    Orientation _orientation;

    Rotation _rotation = Nothing;

    Points _position;

    double _angle = 0;

    bool RightPosition = true;
    bool CornersPosition = true;

};
```

EnumirationList.hpp

```
#pragma once
//Список цветов
enum Colour
{

    White, Yellow, Green, Red, Blue, Orange, Black

};

//Список элементов
enum ElementType
{
```

```
    _Corner, _Edge, _Centre

};

//Список распложений
enum Orientation
{

    Straight, LeftSide, RightSide, FrontSide, BackSide, BottomSide

};

//Список поворотов
enum Rotation
{

    Nothing, CenterUpAndDownRotation, CenterLeftAndRightRotation, ClockwiseRotation

};

//Список операций для разбирателя
enum Disassembly
{

    Right1Rotation, Right2Rotation, Left1Rotation, Left2Rotation, Up1Rotation,
Up2Rotation, Down1Rotation, Down2Rotation, Front1Rotation, Front2Rotation

};
```

Cubik.hpp

```
#pragma once
#include "EnumerationList.hpp"
#include "Elements.hpp"
#include <vector>
class CubikRubik
{

public:
    CubikRubik();

    void getFile();

    void UpAndDown(std::vector<int>, std::vector<int>, int, double);//Вверх и Вниз
    void LeftAndRight(std::vector<int>, std::vector<int>, int, double);//Влево и Вправо
    void Clockwise(std::vector<int>, std::vector<int>, int, double);//Поворот по часовой
стрелке
    void CenterUpAndDown(std::vector<int>, std::vector<int>, double);//Центральную
фронтальная грань влево и вправо
    void CenterLeftAndRight(std::vector<int>, std::vector<int>, double);//Центральную
боковую грань вверх и вниз

    void UP_LEFT();//Верхнюю грань вправо
    void UP_RIGHT();//Верхнюю грань влево
    void RIGHT_UP();//Правую боковую грань вверх
    void RIGHT_DOWN();//Правую боковую грань вниз
    void LEFT_DOWN();//Левую боковую грань вниз
    void LEFT_UP();//Левую боковую грань вверх
    void DOWN_RIGHT();//Нижнюю нрань вправо
    void DOWN_LEFT();//Нижнюю грань влево
    void FRONT_RIGHT();//Фронтальную грань направо
    void FRONT_LEFT();//Фронтальную грань налево
    void CENTER_UP();//Центральную боковую грань вверх
    void CENTER_DOWN();//Цетральную боковую грань вниз
```

```
    void CENTER_LEFT();//Центральную фронтальную грань вправо
    void CENTER_RIGHT();//Центральную фронтальную грань влево

    void PifPaf();//Классический пиф-паф. Правую от себя, верхнюю влево, правую на себя,
верхнюю вправо.
    void ReversedPifPaf();//Ревёрснутый пиф-паф. То же самое, но для левой стороны.
    void AllLeft();//Всё налево
    void AllRight();//Всё направо
    void AllUp();//Всё вверх
    void AllDown();//Всё вниз

    void RubiksCubeDisassembler();//Собиратель кубика
    void RubiksCubeAssembler();//Разбиратель кубика

    void StepOne();//Правильный крест
    void StepTwo();//Ребро первого слоя
    void StepThree();//Углы первого слоя
    void StepFour();//Рёбра среднего слоя
    void StepFive();//Крест последнего слоя
    void StepSix();//Правильный крест последнего слоя
    void StepSeven();//Расстановка углов последнего слоя
    void StepEight();//Разворот углов третьего слоя

    int FindElement(Colour, ElementType);
    void draw();

    ~CubikRubik();

private:

    std::vector<std::vector<Element*>>_elements;
};
```