МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ»

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Факультет №3 «Системы управления, информатика и электроэнергетика»

Кафедра 304 «Вычислительные машины, системы и сети»

**КУРСОВАЯ РАБОТА**

**на тему «Синтез шарнирного куба 2:2:2»**

по дисциплине «Объектно-ориентированное программирование»

Выполнил:

Студент группы 3О-210Б

Сомов Д. Н.

Принял:

доц. каф. 304, к.т.н.

Новиков П. В.

Москва, 2016

# СОДЕРЖАНИЕ

[Задание 3](#_Toc451007903)

[Общее понятие объектно-ориентированного программирования 4](#_Toc451007904)

[Проектирование класса «Кубик» 6](#_Toc451007905)

[Проектирование класса «Шарнирный куб» 9](#_Toc451007906)

[Листинг исходного кода, реализующего синтез шарнирного куба 11](#_Toc451007907)

[Результаты работы 31](#_Toc451007908)

[Заключение 36](#_Toc451007909)

[Список использованных источников 36](#_Toc451007910)

# Задание

Составить проект класса(-ов), позволяющих синтезировать подвижный шарнирный куб 2:2:2. Написать программу на языке ООП С++, реализующую данный(-е) класс(-ы), показать синтез куба и его движение.

# Общее понятие объектно-ориентированного программирования

Объектно-ориентированная парадигма впервые была использована в языке Симула67. Позже данный подход был поддержан и другими языками программирования. Верным будет утверждение, что объектно-ориентированный подход в языке программирования может как сочетаться с другими – функциональным (Common Lisp), логическим (Visual Prolog 5.0) или императивным (Visual Basic 4.0, C++, Object Pascal), – так и использоваться в чистом виде (пример – чисто объектно-ориентированные языки Java, C#).

В программе, использующей объектно-ориентированный подход, присутствуют классы и объекты. Итак, класс – это объектный пользовательский тип данных, в котором описываются имена полей данных (по сути, переменные класса), их типы и возможность доступа, а также описываются методы (объявляются и определяются). Объект же – это самостоятельный программный компонент, имеющий собственные характеристики (поля данных), а также собственное поведение, определяемое методами (функциями, членами класса). Процессу ООП обычно предшествует процесс проектирования классов, направленный на построение грамотной и удобной иерархии проекта.

ООП использует (но не ограничивается ими) три присущих только ему приёма: наследование, инкапсуляцию и полиморфизм.

Наследование – способ построения иерархии классов, когда класс, стоящий ниже в иерархической «лестнице» – класс-потомок – наследует все открытые для доступа поля данных и методы классов, стоящих выше – классов-предков. Этот приём может быть использован в целях экономии кода при наличии классов, мало отличающихся друг от друга (например, классы двухмерного и трехмерного векторов отличаются только добавлением одной координаты, что позволяет эффективно унаследовать второй класс от первого), но имеет свои побочные эффекты; один из них – увеличенный расход оперативной памяти.

Инкапсуляцией был назван приём закрытия доступа к некоторым полям данных и методам класса, позволяя защититься от нежелательного доступа к ним. Также закрытые поля не наследуются классом-потомком, что позволяет использовать инкапсуляцию для большего контроля над процессом наследования.

Полиморфизм наглядно показывает, насколько широко могут быть использованы объекты при ООП. Полиморфные (или многоформенные) аргументы функций могут принимать объекты сразу нескольких типов, а полиморфизм методов и конструкторов еще больше подчеркивает, что объекты одного и того же класса могут иметь абсолютно разное поведение.

Весьма полезным приемом является композиция классов и объектов, дающая возможность также создать иерархию классов, но несколько иную, чем при наследовании. При этом в качестве полей данных одного класса используются объекты другого класса. В некоторых случаях композиция как нельзя лучше подчеркивает устройство большого сложного объекта, выделяя в нём в виде полей данных более простые объекты, а в них, в свою очередь, еще более простые, и так далее. В таком случае с иерархией довольно просто работать, используя логику управления объектами и/или их частями.

# Проектирование класса «Кубик»

Проект класса Cube (один из кубиков целого шарнирного куба) составлялся согласно той идее, что полный куб будет состоять из восьми объектов данного класса. Он имеет поля данных, обозначающие координаты точек кубика и цвет граней, и методы для рисования, стирания и вращения кубика в трех плоскостях.

Использование объектно-ориентированной парадигмы здесь заключается в том, что каждый кубик (т.е. каждый объект) будет иметь собственное поведение при вращении и отрисовке в зависимости от места этого кубика в составе большого, что наглядно показывает присутствие методов Draw1÷Draw8.

**Имя класса**: Cube

**Поля данных** (открытые для упрощения доступа к ним):

pos – целочисленное значение от 1 до 8, означающее позицию кубика в составе большого куба (необходимо для правильного вращения и отрисовки). Расположение кубиков в зависимости от значения этого поля см. на рис. 1;

cx, cy – целочисленные координаты середины верхней грани кубика (необходимы для правильного вращения и отрисовки);

x[8], y[8] – целочисленные координаты всех точек – вершин кубика. Расположение вершин см. на рис. 2;

col – цвет ребер кубика;

edgcol[3] (edge color) – цвета внешних граней кубика. Расположение граней см. на рис. 3.

**Методы** (открытые):

а) Конструктор: входные параметры – два массива целочисленных координат точек – вершин кубика; цвета граней по порядку их следования в массиве edgcol; позиция pos в большом кубе; координаты центра верхней грани.

б) Прочие методы:

Draw – в зависимости от позиции pos вызывает один из методов отрисовки Draw1÷Draw8;

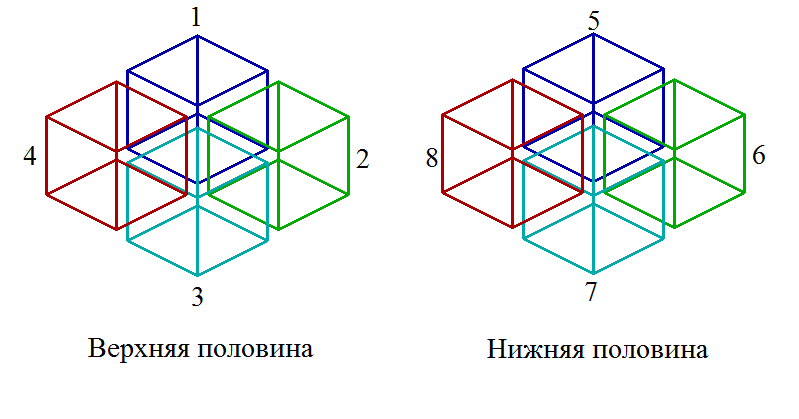
Hide – рисует кубик цветом фона; для этого запоминаются текущие цвета ребер и граней кубика, затем они меняются на цвет фона, вызывается метод Show, и после его выполнения цвета меняются обратно на те, что были изначально;

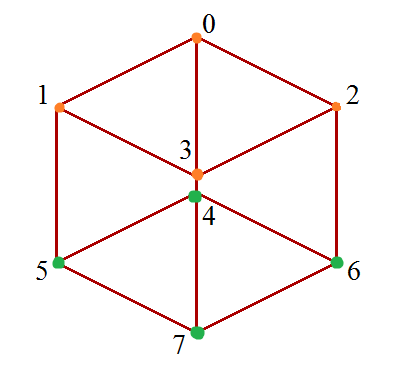
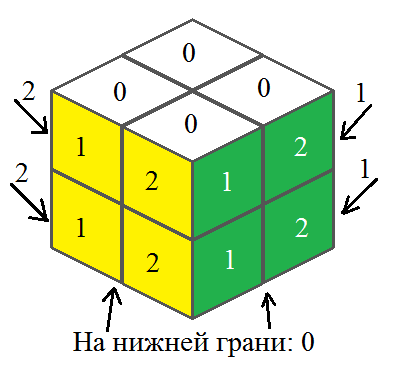
RotSide, RotULDR (from Up-Left to Down-Right), RotURDL (from Up-Right to Down-Left) (целочисл. индекс i, целочисл. флаг направления dir) – методы, отвечающие за пошаговое вращение кубика в одной из трех плоскостей. Каждый вызов одного из этих методов выполняет ровно одну итерацию процесса поворота. Для этого кубик стирается, затем соответствующим образом меняются координаты каждой из точек в зависимости от шага итерации i и направления вращения dir (изменение координат центра, происходящее перед изменением координат точек, дополнительно зависит от позиции кубика pos). После этого кубик рисуется методом Draw. Метод, вызывающий методы RotSide, RotULDR и RotURDL, делает ровно столько итераций, сколько нужно для полного поворота кубика. Если обнаруживается последняя итерация поворота, после рисования выполняется переприсваивание вершин кубика согласно тому, как он вращался, для возможности продолжать вращение по тому же алгоритму при следующем вызове методов вращения большого куба. Фактически это значит то, что при вращении точки меняли свои координаты, и после процесса вращения их нужно вернуть обратно на место, чтобы было возможно повторение процесса.

**Методы** (закрытые):

Двойной точности sind, cosd(двойной точности аргумент x) – принимают меру угла x в градусах и возвращают, соответственно, значение синуса и косинуса для той же меры угла в радианах, переводя градусы в радианы по известному соотношению: градусы = ( радианы \* π ) / 180;

Draw1÷Draw8 – методы для отрисовки (согласно позиции кубика pos) именно тех ребер и граней, которые в данной позиции видны наблюдателю, т.е. сначала отрисовываются все видимые наблюдателю ребра, затем по точкам заливаются все грани, видимые наблюдателю, используя цвета из массива edgcol.

*Рис. 1. Значение поля pos в зависимости от положения кубика*

* Рис. 2. Расположение вершин кубика Рис. 3. Расположение граней кубика*

# Проектирование класса «Шарнирный куб»

Проект класса BigCube, отвечающего за показ на экране и управление полным шарнирным кубом, составлялся с использованием принципа композиции: подобный куб состоит из восьми отдельных кубиков, каждому из которых нужно организовывать собственное поведение для корректного вращения и отрисовки. Поэтому большой куб как объект является композицией восьми маленьких кубиков-объектов, имея их в качестве своих полей данных. В его классе присутствуют методы для комплексного управления всеми объектами сразу.

**Имя класса**: BigCube

**Поля данных** (открытые):

c[8] – маленькие кубики, из которых состоит большой куб; объекты класса Cube. Расположение в кубе идет согласно их позициям pos.

**Методы** (открытые):

а) Конструктор: входные параметры – восемь кубиков – объектов класса Cube – в порядке их следования в массиве c[8].

б) Прочие методы:

FullDraw – отрисовка всего куба сразу; фактически, вызов у каждого объекта из c[8] метода Draw;

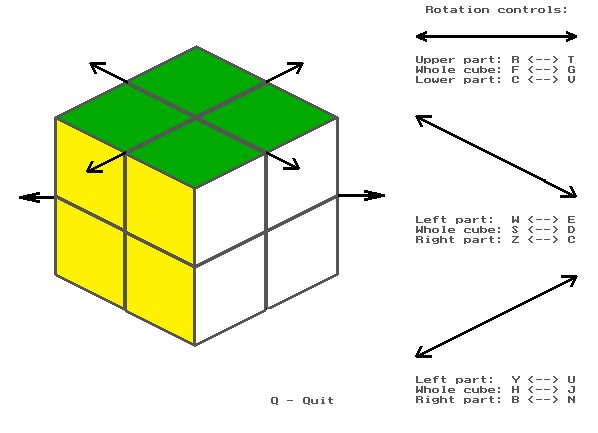
Целочисл. RotationCommand(символьный c) – вызывает один из методов комплексного вращения данного класса в зависимости от поступившего символа (фактически, это введенная с клавиатуры буква). «Управление» кубом представлено на рис. 4. Возвращает 0, если была нажата Q (что соответствует сигналу выхода из программы) и 1 в любом другом случае.

ShowControls – отображает список клавиш для управления вращением куба в виде линий-стрелок и надписей.

**Методы** (закрытые):

SortByPosition – после выполнения вращения номера позиций кубов оказываются «сбиты» относительно начальных, что приводит к невозможности применения тех же самых методов для выполнения дальнейших вращений. Данный метод сортирует объекты-кубики в массиве c[8] таким образом, чтобы их позиции pos, как и до вращения, соответствовали их индексам в массиве.

RotateSideways, RotateUpLeftDownRight, RotateUp- RightDownLeft (целочисл. режим поворота mode, целочисл. направление поворота dir) – методы, вызывающие методы RotSide, RotULDR и RotURDL у соответствующих кубиков, пользуясь тем, что, благодаря методу SortByPosition, индексы кубиков в массиве с[8] соответствуют номерам их позиций. Другими словами, то, какие методы будут вызваны у маленьких кубиков, зависит от аргументов mode и dir, а также от позиции pos. Это значит, что от значения аргумента режима mode будет выбрано, какие из кубиков будут участвовать в итерационном процессе вращения, а какие будут просто перерисовываться, оставаясь неподвижными. Аргумент режима принимает три значения – поворот левой части, правой части или всего куба целиком (и левой, и правой частей). После выполнения вращения переприсваиваются цвета граней и позиции кубиков (для правильного отображения), и методом FullDraw выполняется отрисовка всего куба. В конце кубики сортируются в массиве методом SortByPosition, т.к. их позиции перестали соответствовать их расположению в нем.



*Рис. 4. Управление вращением куба в трех плоскостях*

# Листинг исходного кода, реализующего синтез шарнирного куба

***/\* Автор – Сомов Дмитрий, группа – 3О-210Б***

***Реализация классов Cube и BigCube \*/***

#include <graphics.h>

#include <conio.h>

#include <math.h>

#include <dos.h>

class Cube

{ public:

int pos;

int cx, cy;

int x[8], y[8];

int col;

int edgcol[3];

Cube(int\*,int\*,int,int,int,int,int,int);

Cube() {};

void Draw();

void Hide();

void RotSide(int, int); //sideways

void RotULDR(int, int); //up-left-down-right

void RotURDL(int, int); //up-right-down-left

private:

double cosd(double);

double sind(double);

void Draw1();

void Draw2();

void Draw3();

void Draw4();

void Draw6();

void Draw7();

void Draw8();

};

class BigCube

{ public:

Cube c[8];

BigCube(Cube,Cube,Cube,Cube,Cube,Cube,Cube,Cube);

void FullDraw();

int RotationCommand(char);

void ShowControls();

private:

void RotateSideways(int, int);

void RotateUpLeftDownRight(int, int);

void RotateUpRightDownLeft(int, int);

void SortByPosition();

};

Cube::Cube(int \*ix,int \*iy,int ec1,int ec2,

int ec3,int pos,int cx,int cy)

{ for(int i = 0; i < 8; i++)

{ x[i] = ix[i];

y[i] = iy[i];

}

col = 8;

edgcol[0] = ec1;

edgcol[1] = ec2;

edgcol[2] = ec3;

this -> pos = pos;

this->cx = cx;

this->cy = cy;

}

void Cube::Draw()

{ if(pos == 1) Draw1();

else if(pos == 2) Draw2();

else if(pos == 3) Draw3();

else if(pos == 4) Draw4();

else if (pos == 6) Draw6();

else if (pos == 7) Draw7();

else if (pos == 8) Draw8();

}

void Cube::Draw1()

{ setcolor(col);

line(x[0], y[0], x[1], y[1]);

line(x[0], y[0], x[2], y[2]);

line(x[2], y[2], x[3], y[3]);

line(x[1], y[1], x[3], y[3]);

setfillstyle(1, edgcol[0]);

int t3[10]={x[0],y[0],x[1],y[1],x[3],y[3],x[2],y[2],x[0],y[0]};

fillpoly(4, t3);

}

void Cube::Draw2()

{ setcolor(col);

line(x[0], y[0], x[1], y[1]);

line(x[0], y[0], x[2], y[2]);

line(x[2], y[2], x[3], y[3]);

line(x[1], y[1], x[3], y[3]);

line(x[3], y[3], x[7], y[7]);

line(x[2], y[2], x[6], y[6]);

line(x[6], y[6], x[7], y[7]);

setfillstyle(1, edgcol[0]);

int t3[10]={x[0],y[0],x[1],y[1],x[3],y[3],x[2],y[2],x[0],y[0]};

fillpoly(4, t3);

setfillstyle(1, edgcol[2]);

int t2[10]={x[2],y[2],x[3],y[3],x[7],y[7],x[6],y[6],x[2],y[2]};

fillpoly(4, t2);

}

void Cube::Draw3()

{ setcolor(col);

line(x[0], y[0], x[1], y[1]);

line(x[0], y[0], x[2], y[2]);

line(x[2], y[2], x[3], y[3]);

line(x[1], y[1], x[3], y[3]);

line(x[1], y[1], x[5], y[5]);

line(x[2], y[2], x[6], y[6]);

line(x[3], y[3], x[7], y[7]);

line(x[7], y[7], x[5], y[5]);

line(x[7], y[7], x[6], y[6]);

setfillstyle(1, edgcol[0]);

int t3[10]={x[0],y[0],x[1],y[1],x[3],y[3],x[2],y[2],x[0],y[0]};

fillpoly(4, t3);

setfillstyle(1, edgcol[2]);

int t2[10]={x[1],y[1],x[3],y[3],x[7],y[7],x[5],y[5],x[1],y[1]};

fillpoly(4, t2);

setfillstyle(1, edgcol[1]);

int t1[10]={x[2],y[2],x[3],y[3],x[7],y[7],x[6],y[6],x[2],y[2]};

fillpoly(4, t1);

}

void Cube::Draw4()

{ setcolor(col);

line(x[0], y[0], x[1], y[1]);

line(x[0], y[0], x[2], y[2]);

line(x[2], y[2], x[3], y[3]);

line(x[1], y[1], x[3], y[3]);

line(x[3], y[3], x[7], y[7]);

line(x[1], y[1], x[5], y[5]);

line(x[7], y[7], x[5], y[5]);

setfillstyle(1, edgcol[0]);

int t3[10]={x[0],y[0],x[1],y[1],x[3],y[3],x[2],y[2],x[0],y[0]};

fillpoly(4, t3);

setfillstyle(1, edgcol[1]);

int t1[10]={x[1],y[1],x[3],y[3],x[7],y[7],x[5],y[5],x[1],y[1]};

fillpoly(4, t1);

}

void Cube::Draw6()

{ setcolor(col);

line(x[2], y[2], x[3], y[3]);

line(x[3], y[3], x[7], y[7]);

line(x[2], y[2], x[6], y[6]);

line(x[7], y[7], x[6], y[6]);

setfillstyle(1, edgcol[2]);

int t2[10]={x[2],y[2],x[3],y[3],x[7],y[7],x[6],y[6],x[2],y[2]};

fillpoly(4, t2);

}

void Cube::Draw7()

{ setcolor(col);

line(x[3], y[3], x[1], y[1]);

line(x[3], y[3], x[2], y[2]);

line(x[3], y[3], x[7], y[7]);

line(x[1], y[1], x[5], y[5]);

line(x[2], y[2], x[6], y[6]);

line(x[7], y[7], x[5], y[5]);

line(x[7], y[7], x[6], y[6]);

setfillstyle(1, edgcol[2]);

int t2[10]={x[1],y[1],x[3],y[3],x[7],y[7],x[5],y[5],x[1],y[1]};

fillpoly(4, t2);

setfillstyle(1, edgcol[1]);

int t1[10]={x[3],y[3],x[2],y[2],x[6],y[6],x[7],y[7],x[3],y[3]};

fillpoly(4, t1);

}

void Cube::Draw8()

{ setcolor(col);

line(x[3], y[3], x[1], y[1]);

line(x[1], y[1], x[5], y[5]);

line(x[3], y[3], x[7], y[7]);

line(x[7], y[7], x[5], y[5]);

setfillstyle(1, edgcol[1]);

int t1[10]={x[1],y[1],x[3],y[3],x[7],y[7],x[5],y[5],x[1],y[1]};

fillpoly(4, t1);

}

double Cube::sind(double x) { return sin((x\*M\_PI) / 180); }

double Cube::cosd(double x) { return cos((x\*M\_PI) / 180); }

void Cube::Hide()

{ int tcol, tedgcol[3];

tcol = col;

tedgcol[0] = edgcol[0];

tedgcol[1] = edgcol[1];

tedgcol[2] = edgcol[2];

col = getbkcolor();

edgcol[0] = col;

edgcol[1] = col;

edgcol[2] = col;

Draw();

col = tcol;

edgcol[0] = tedgcol[0];

edgcol[1] = tedgcol[1];

edgcol[2] = tedgcol[2];

}

void Cube::RotSide(int i, int d)

{ Hide();

if(pos == 1)

{ cx = 200 + d\*(71\*cosd(90-6\*i));

cy = 121 - 36\*sind(90-6\*i);

}

if(pos == 2)

{ cx = 200 + (71\*cosd(6\*i));

cy = 121 + d\*(36\*sind(6\*i));

}

if(pos == 3)

{ cx = 200 - d\*71\*cosd(90-6\*i);

cy = 121 + 36\*sind(90-6\*i);

}

if(pos == 4)

{ cx = 200 - 71\*cosd(6\*i);

cy = 121 - d\*36\*sind(6\*i);

}

if (pos == 5)

{ cx = 200 + d\*(71 \* cosd(90 - 6 \* i));

cy = 200 - 36 \* sind(90 - 6 \* i);

}

if (pos == 6)

{ cx = 200 + (71 \* cosd(6 \* i));

cy = 200 + d\*(36 \* sind(6 \* i));

}

if (pos == 7)

{ cx = 200 - d \* 71 \* cosd(90 - 6 \* i);

cy = 200 + 36 \* sind(90 - 6 \* i);

}

if (pos == 8)

{ cx = 200 - 71 \* cosd(6 \* i);

cy = 200 - d \* 36 \* sind(6 \* i);

}

x[0] = cx + d\*(70 \* cosd(90 - 6 \* i));

y[0] = cy - (35 \* sind(90 - 6 \* i));

x[1] = cx - (70 \* cosd(6 \* i));

y[1] = cy - d\*(35 \* sind(6 \* i));

x[2] = cx + (70 \* cosd(6 \* i));

y[2] = cy + d\*(35 \* sind(6 \* i));

x[3] = cx - d\*(70 \* cosd(90 - 6 \* i));

y[3] = cy + (35 \* sind(90 - 6 \* i));

x[4] = cx + d\*(70 \* cosd(90 - 6 \* i));

y[4] = cy + 78 - (35 \* sind(90 - 6 \* i));

x[5] = cx - (70 \* cosd(6 \* i));

y[5] = cy + 78 - d\*(35 \* sind(6 \* i));

x[6] = cx + (70 \* cosd(6 \* i));

y[6] = cy + 78 + d\*(35 \* sind(6 \* i));

x[7] = cx - d\*(70 \* cosd(90 - 6 \* i));

y[7] = cy + 78 + (35 \* sind(90 - 6 \* i));

Draw();

if (i == 15)

{ if(d == 1)

{ int tx = x[0], ty = y[0];

x[0] = x[1];

y[0] = y[1];

x[1] = x[3];

y[1] = y[3];

x[3] = x[2];

y[3] = y[2];

x[2] = tx;

y[2] = ty;

tx = x[4];

ty = y[4];

x[4] = x[5];

y[4] = y[5];

x[5] = x[7];

y[5] = y[7];

x[7] = x[6];

y[7] = y[6];

x[6] = tx;

y[6] = ty;

}

else if(d == -1)

{ int tx = x[0], ty = y[0];

x[0] = x[2];

y[0] = y[2];

x[2] = x[3];

y[2] = y[3];

x[3] = x[1];

y[3] = y[1];

x[1] = tx;

y[1] = ty;

tx = x[4];

ty = y[4];

x[4] = x[6];

y[4] = y[6];

x[6] = x[7];

y[6] = y[7];

x[7] = x[5];

y[7] = y[5];

x[5] = tx;

y[5] = ty;

}

}

}

void Cube::RotULDR(int i, int d)

{ Hide();

if (pos == 1 || pos == 4)

{ for (int j = 0; j < 8; j++)

{ if (d == 1)

{ x[j] += 10;

y[j] += 5;

}

else if (d == -1) y[j] += 11;

}

}

if (pos == 2 || pos == 3)

{ for (int j = 0; j < 8; j++)

{ if (d == 1) y[j] += 11;

else if (d == -1)

{ x[j] -= 10;

y[j] -= 5;

}

}

}

if (pos == 5 || pos == 8)

{ for (int j = 0; j < 8; j++)

{ if (d == 1) y[j] -= 11;

else if (d == -1)

{ x[j] += 10;

y[j] += 5;

}

}

}

if (pos == 6 || pos == 7)

{ for (int j = 0; j < 8; j++)

{ if (d == 1)

{ x[j] -= 10;

y[j] -= 5;

}

else if (d == -1) y[j] -= 11;

}

}

if (d == 1)

{ x[0] += 10;

y[0] += 5;

x[1] += 10;

y[1] += 5;

y[2] += 11;

y[3] += 11;

y[4] -= 11;

y[5] -= 11;

x[6] -= 10;

y[6] -= 5;

x[7] -= 10;

y[7] -= 5;

}

else if (d == -1)

{ x[4] += 10;

y[4] += 5;

x[5] += 10;

y[5] += 5;

y[0] += 11;

y[1] += 11;

y[6] -= 11;

y[7] -= 11;

x[2] -= 10;

y[2] -= 5;

x[3] -= 10;

y[3] -= 5;

}

if (i == 6)

{ if (d == 1)

{ int tx = x[0], ty = y[0];

x[0] = x[4];

y[0] = y[4];

x[4] = x[6];

y[4] = y[6];

x[6] = x[2];

y[6] = y[2];

x[2] = tx;

y[2] = ty;

tx = x[1];

ty = y[1];

x[1] = x[5];

y[1] = y[5];

x[5] = x[7];

y[5] = y[7];

x[7] = x[3];

y[7] = y[3];

x[3] = tx;

y[3] = ty;

}

else if (d == -1)

{ int tx = x[0], ty = y[0];

x[0] = x[2];

y[0] = y[2];

x[2] = x[6];

y[2] = y[6];

x[6] = x[4];

y[6] = y[4];

x[4] = tx;

y[4] = ty;

tx = x[1];

ty = y[1];

x[1] = x[3];

y[1] = y[3];

x[3] = x[7];

y[3] = y[7];

x[7] = x[5];

y[7] = y[5];

x[5] = tx;

y[5] = ty;

}

}

Draw();

}

void Cube::RotURDL(int i, int d)

{ Hide();

if (pos == 1 || pos == 2)

{ for (int j = 0; j < 8; j++)

{ if (d == 1)

{ x[j] -= 10;

y[j] += 5;

}

else if (d == -1) y[j] += 11;

}

}

if (pos == 4 || pos == 3)

{ for (int j = 0; j < 8; j++)

{ if (d == 1) y[j] += 11;

else if (d == -1)

{ x[j] += 10;

y[j] -= 5;

}

}

}

if (pos == 5 || pos == 6)

{ for (int j = 0; j < 8; j++)

{ if (d == 1) y[j] -= 11;

else if (d == -1)

{ x[j] -= 10;

y[j] += 5;

}

}

}

if (pos == 8 || pos == 7)

{ for (int j = 0; j < 8; j++)

{ if (d == 1)

{ x[j] += 10;

y[j] -= 5;

}

else if (d == -1) y[j] -= 11;

}

}

if (d == 1)

{ x[0] -= 10;

y[0] += 5;

x[2] -= 10;

y[2] += 5;

y[1] += 11;

y[3] += 11;

y[4] -= 11;

y[6] -= 11;

x[5] += 10;

y[5] -= 5;

x[7] += 10;

y[7] -= 5;

}

else if (d == -1)

{ x[4] -= 10;

y[4] += 5;

x[6] -= 10;

y[6] += 5;

y[0] += 11;

y[2] += 11;

y[5] -= 11;

y[7] -= 11;

x[1] += 10;

y[1] -= 5;

x[3] += 10;

y[3] -= 5;

}

if (i == 6)

{ if (d == 1)

{ int tx = x[0], ty = y[0];

x[0] = x[4];

y[0] = y[4];

x[4] = x[5];

y[4] = y[5];

x[5] = x[1];

y[5] = y[1];

x[1] = tx;

y[1] = ty;

tx = x[2];

ty = y[2];

x[2] = x[6];

y[2] = y[6];

x[6] = x[7];

y[6] = y[7];

x[7] = x[3];

y[7] = y[3];

x[3] = tx;

y[3] = ty;

}

else if (d == -1)

{ int tx = x[0], ty = y[0];

x[0] = x[1];

y[0] = y[1];

x[1] = x[5];

y[1] = y[5];

x[5] = x[4];

y[5] = y[4];

x[4] = tx;

y[4] = ty;

tx = x[2];

ty = y[2];

x[2] = x[3];

y[2] = y[3];

x[3] = x[7];

y[3] = y[7];

x[7] = x[6];

y[7] = y[6];

x[6] = tx;

y[6] = ty;

}

}

Draw();

}

BigCube::BigCube(Cube c1,Cube c2,Cube c3,Cube c4,

Cube c5,Cube c6,Cube c7,Cube c8)

{ c[0] = c1;

c[1] = c2;

c[2] = c3;

c[3] = c4;

c[4] = c5;

c[5] = c6;

c[6] = c7;

c[7] = c8;

}

void BigCube::FullDraw()

{ for (int j = 0; j < 8; j++) c[j].Draw(); }

void BigCube::RotateSideways(int mode, int dir)

{ int j;

for(int i = 0; i < 16; i++)

{ if (mode == 0 || mode == 1)

for (j = 0; j < 4; j++) c[j].RotSide(i, dir);

if (mode == 0 || mode == 2)

for (j = 4; j < 8; j++) c[j].RotSide(i, dir);

if (mode == 1)

for (j = 4; j < 8; j++) c[j].Draw();

if (mode == 2)

for (j = 0; j < 4; j++) c[j].Draw();

delay(20);

}

if (mode == 0 || mode == 1)

{ for (j = 0; j < 4; j++)

{ c[j].pos += dir;

if (c[j].pos > 4) c[j].pos = 1;

else if (c[j].pos < 1) c[j].pos = 4;

}

}

if (mode == 0 || mode == 2)

{ for (j = 4; j < 8; j++)

{ c[j].pos += dir;

if (c[j].pos > 8) c[j].pos = 5;

else if (c[j].pos < 5) c[j].pos = 8;

}

}

for (j = 0; j < 8; j++) c[j].Hide();

FullDraw();

SortByPosition();

}

void BigCube::RotateUpLeftDownRight(int mode, int dir)

{ int j;

for (int i = 0; i < 7; i++)

{ if (mode == 0 || mode == 1)

{ c[3].RotULDR(i, dir);

c[2].RotULDR(i, dir);

c[6].RotULDR(i, dir);

c[7].RotULDR(i, dir);

}

if (mode == 0 || mode == 2)

{ c[0].RotULDR(i, dir);

c[1].RotULDR(i, dir);

c[5].RotULDR(i, dir);

c[4].RotULDR(i, dir);

}

if (mode == 1)

{ c[0].Draw();

c[1].Draw();

c[5].Draw();

c[4].Draw();

}

if (mode == 2)

{ c[3].Draw();

c[2].Draw();

c[6].Draw();

c[7].Draw();

}

}

if (mode == 0 || mode == 1)

{ for (j = 0; j < 8; j++)

{ if (dir == 1)

{ if (c[j].pos == 4)

{ c[j].pos = 3;

int t = c[j].edgcol[0];

c[j].edgcol[0] = c[j].edgcol[2];

c[j].edgcol[2] = c[j].edgcol[1];

c[j].edgcol[1] = t;

}

else if (c[j].pos == 3)

{ c[j].pos = 7;

int t = c[j].edgcol[0];

c[j].edgcol[0] = c[j].edgcol[1];

c[j].edgcol[1] = t;

}

else if (c[j].pos == 7)

{ c[j].pos = 8;

int t = c[j].edgcol[0];

c[j].edgcol[0] = c[j].edgcol[1];

c[j].edgcol[1] = c[j].edgcol[2];

c[j].edgcol[2] = t;

}

else if (c[j].pos == 8)

{ c[j].pos = 4;

int t = c[j].edgcol[0];

c[j].edgcol[0] = c[j].edgcol[2];

c[j].edgcol[2] = t;

}

}

if (dir == -1)

{ if (c[j].pos == 4)

{ c[j].pos = 8;

int t = c[j].edgcol[0];

c[j].edgcol[0] = c[j].edgcol[2];

c[j].edgcol[2] = t;

}

else if (c[j].pos == 8)

{ c[j].pos = 7;

int t = c[j].edgcol[0];

c[j].edgcol[0] = c[j].edgcol[2];

c[j].edgcol[2] = c[j].edgcol[1];

c[j].edgcol[1] = t;

}

else if (c[j].pos == 7)

{ c[j].pos = 3;

int t = c[j].edgcol[0];

c[j].edgcol[0] = c[j].edgcol[1];

c[j].edgcol[1] = t;

}

else if (c[j].pos == 3)

{ c[j].pos = 4;

int t = c[j].edgcol[0];

c[j].edgcol[0] = c[j].edgcol[1];

c[j].edgcol[1] = c[j].edgcol[2];

c[j].edgcol[2] = t;

}

}

}

}

if (mode == 0 || mode == 2)

{ for (j = 0; j < 8; j++)

{ if (dir == 1)

{ if (c[j].pos == 1)

{ c[j].pos = 2;

int t = c[j].edgcol[0];

c[j].edgcol[0] = c[j].edgcol[1];

c[j].edgcol[1] = c[j].edgcol[2];

c[j].edgcol[2] = t;

}

else if (c[j].pos == 2)

{ c[j].pos = 6;

int t = c[j].edgcol[0];

c[j].edgcol[0] = c[j].edgcol[2];

c[j].edgcol[2] = t;

}

else if (c[j].pos == 6)

{ c[j].pos = 5;

int t = c[j].edgcol[0];

c[j].edgcol[0] = c[j].edgcol[2];

c[j].edgcol[2] = c[j].edgcol[1];

c[j].edgcol[1] = t;

}

else if (c[j].pos == 5)

{ c[j].pos = 1;

int t = c[j].edgcol[0];

c[j].edgcol[0] = c[j].edgcol[1];

c[j].edgcol[1] = t;

}

}

if (dir == -1)

{ if (c[j].pos == 1)

{ c[j].pos = 5;

int t = c[j].edgcol[0];

c[j].edgcol[0] = c[j].edgcol[1];

c[j].edgcol[1] = t;

}

else if (c[j].pos == 5)

{ c[j].pos = 6;

int t = c[j].edgcol[0];

c[j].edgcol[0] = c[j].edgcol[1];

c[j].edgcol[1] = c[j].edgcol[2];

c[j].edgcol[2] = t;

}

else if (c[j].pos == 6)

{ c[j].pos = 2;

int t = c[j].edgcol[0];

c[j].edgcol[0] = c[j].edgcol[2];

c[j].edgcol[2] = t;

}

else if (c[j].pos == 2)

{ c[j].pos = 1;

int t = c[j].edgcol[0];

c[j].edgcol[0] = c[j].edgcol[2];

c[j].edgcol[2] = c[j].edgcol[1];

c[j].edgcol[1] = t;

}

}

}

}

for (j = 0; j < 8; j++) c[j].Hide();

FullDraw();

SortByPosition();

}

void BigCube::RotateUpRightDownLeft(int mode, int dir)

{ int j;

for (int i = 0; i < 7; i++)

{ if (mode == 0 || mode == 1)

{ c[0].RotURDL(i, dir);

c[3].RotURDL(i, dir);

c[4].RotURDL(i, dir);

c[7].RotURDL(i, dir);

}

if (mode == 0 || mode == 2)

{ c[1].RotURDL(i, dir);

c[2].RotURDL(i, dir);

c[5].RotURDL(i, dir);

c[6].RotURDL(i, dir);

}

if (mode == 1)

{ c[1].Draw();

c[2].Draw();

c[5].Draw();

c[6].Draw();

}

if (mode == 2)

{ c[0].Draw();

c[3].Draw();

c[4].Draw();

c[7].Draw();

}

}

if (mode == 0 || mode == 1)

{ for (j = 0; j < 8; j++)

{ if (dir == 1)

{ if (c[j].pos == 1)

{ c[j].pos = 4;

int t = c[j].edgcol[0];

c[j].edgcol[0] = c[j].edgcol[2];

c[j].edgcol[2] = c[j].edgcol[1];

c[j].edgcol[1] = t;

}

else if (c[j].pos == 4)

{ c[j].pos = 8;

int t = c[j].edgcol[0];

c[j].edgcol[0] = c[j].edgcol[1];

c[j].edgcol[1] = t;

}

else if (c[j].pos == 8)

{ c[j].pos = 5;

int t = c[j].edgcol[0];

c[j].edgcol[0] = c[j].edgcol[1];

c[j].edgcol[1] = c[j].edgcol[2];

c[j].edgcol[2] = t;

}

else if (c[j].pos == 5)

{ c[j].pos = 1;

int t = c[j].edgcol[0];

c[j].edgcol[0] = c[j].edgcol[2];

c[j].edgcol[2] = t;

}

}

if (dir == -1)

{ if (c[j].pos == 1)

{ c[j].pos = 5;

int t = c[j].edgcol[0];

c[j].edgcol[0] = c[j].edgcol[2];

c[j].edgcol[2] = t;

}

else if (c[j].pos == 5)

{ c[j].pos = 8;

int t = c[j].edgcol[0];

c[j].edgcol[0] = c[j].edgcol[2];

c[j].edgcol[2] = c[j].edgcol[1];

c[j].edgcol[1] = t;

}

else if (c[j].pos == 8)

{ c[j].pos = 4;

int t = c[j].edgcol[0];

c[j].edgcol[0] = c[j].edgcol[1];

c[j].edgcol[1] = t;

}

else if (c[j].pos == 4)

{ c[j].pos = 1;

int t = c[j].edgcol[0];

c[j].edgcol[0] = c[j].edgcol[1];

c[j].edgcol[1] = c[j].edgcol[2];

c[j].edgcol[2] = t;

}

}

}

}

if (mode == 0 || mode == 2)

{ for (j = 0; j < 8; j++)

{ if (dir == 1)

{ if (c[j].pos == 2)

{ c[j].pos = 3;

int t = c[j].edgcol[0];

c[j].edgcol[0] = c[j].edgcol[1];

c[j].edgcol[1] = c[j].edgcol[2];

c[j].edgcol[2] = t;

}

else if (c[j].pos == 3)

{ c[j].pos = 7;

int t = c[j].edgcol[0];

c[j].edgcol[0] = c[j].edgcol[2];

c[j].edgcol[2] = t;

}

else if (c[j].pos == 7)

{ c[j].pos = 6;

int t = c[j].edgcol[0];

c[j].edgcol[0] = c[j].edgcol[2];

c[j].edgcol[2] = c[j].edgcol[1];

c[j].edgcol[1] = t;

}

else if (c[j].pos == 6)

{ c[j].pos = 2;

int t = c[j].edgcol[0];

c[j].edgcol[0] = c[j].edgcol[1];

c[j].edgcol[1] = t;

}

}

if (dir == -1)

{ if (c[j].pos == 2)

{ c[j].pos = 6;

int t = c[j].edgcol[0];

c[j].edgcol[0] = c[j].edgcol[1];

c[j].edgcol[1] = t;

}

else if (c[j].pos == 6)

{ c[j].pos = 7;

int t = c[j].edgcol[0];

c[j].edgcol[0] = c[j].edgcol[1];

c[j].edgcol[1] = c[j].edgcol[2];

c[j].edgcol[2] = t;

}

else if (c[j].pos == 7)

{ c[j].pos = 3;

int t = c[j].edgcol[0];

c[j].edgcol[0] = c[j].edgcol[2];

c[j].edgcol[2] = t;

}

else if (c[j].pos == 3)

{ c[j].pos = 2;

int t = c[j].edgcol[0];

c[j].edgcol[0] = c[j].edgcol[2];

c[j].edgcol[2] = c[j].edgcol[1];

c[j].edgcol[1] = t;

}

}

}

}

for (j = 0; j < 8; j++) c[j].Hide();

FullDraw();

SortByPosition();

}

void BigCube::SortByPosition()

{ int i, j;

Cube t;

for (i = 0; i < 8; i++)

for (j = 0; j < 7; j++)

if (c[j].pos > c[j+1].pos)

{ t = c[j];

c[j] = c[j+1];

c[j+1] = t;

}

}

int BigCube::RotationCommand(char c)

{ if (c == 'q') return 0; //quit

if (c == 'r') RotateSideways(1, 1); //left upper turn

if (c == 't') RotateSideways(1, -1); //right upper turn

if (c == 'f') RotateSideways(0, 1); //left full turn

if (c == 'g') RotateSideways(0, -1); //right full turn

if (c == 'c') RotateSideways(2, 1); //left lower turn

if (c == 'v') RotateSideways(2, -1); //right lower turn

if (c == 'e') RotateUpLeftDownRight(1, 1);

if (c == 'w') RotateUpLeftDownRight(1, -1);

if (c == 'd') RotateUpLeftDownRight(0, 1);

if (c == 's') RotateUpLeftDownRight(0, -1);

if (c == 'x') RotateUpLeftDownRight(2, 1);

if (c == 'z') RotateUpLeftDownRight(2, -1);

if (c == 'y') RotateUpRightDownLeft(1, 1);

if (c == 'u') RotateUpRightDownLeft(1, -1);

if (c == 'h') RotateUpRightDownLeft(0, 1);

if (c == 'j') RotateUpRightDownLeft(0, -1);

if (c == 'b') RotateUpRightDownLeft(2, 1);

if (c == 'n') RotateUpRightDownLeft(2, -1);

return 1;

}

void BigCube::ShowControls()

{ outtextxy(430, 10, "Rotation controls:");

line(420, 40, 580, 40);

line(420, 40, 430, 44);

line(420, 40, 430, 36);

line(580, 40, 570, 44);

line(580, 40, 570, 36);

outtextxy(420, 60, "Upper part: R <--> T");

outtextxy(420, 70, "Whole cube: F <--> G");

outtextxy(420, 80, "Lower part: C <--> V");

line(420, 120, 580, 200);

line(420, 120, 435, 120);

line(420, 120, 425, 130);

line(580, 200, 565, 200);

line(580, 200, 575, 190);

outtextxy(420, 220, "Left part: W <--> E");

outtextxy(420, 230, "Whole cube: S <--> D");

outtextxy(420, 240, "Right part: Z <--> X");

line(420, 360, 580, 280);

line(420, 360, 435, 360);

line(420, 360, 425, 350);

line(580, 280, 565, 280);

line(580, 280, 575, 290);

outtextxy(420, 380, "Left part: Y <--> U");

outtextxy(420, 390, "Whole cube: H <--> J");

outtextxy(420, 400, "Right part: B <--> N");

outtextxy(450, 460, "Q - Quit");

}

int x1[8] = {200, 130, 270, 200, 200, 130, 270, 200};

int y1[8] = {50, 85, 85, 120, 128, 163, 163, 198};

int x2[8] = { 271, 201, 341, 271, 271, 201, 341, 271 };

int y2[8] = { 86, 121, 121, 156, 164, 199, 199, 234 };

int x3[8] = { 200, 130, 270, 200, 200, 130, 270, 200 };

int y3[8] = {122, 157, 157, 192, 200, 235, 235, 270};

int x4[8] = { 129, 59, 199, 129, 129, 59, 199, 129 };

int y4[8] = { 86, 121, 121, 156, 164, 199, 199, 234 };

int x5[8] = {200, 130, 270, 200, 200, 130, 270, 200};

int y5[8] = {129, 164, 164, 199, 207, 242, 242, 277};

int x6[8] = { 271, 201, 341, 271, 271, 201, 341, 271 };

int y6[8] = {165, 200, 200, 235, 243, 278, 278, 313};

int x7[8] = { 200, 130, 270, 200, 200, 130, 270, 200 };

int y7[8] = {201, 236, 236, 271, 279, 314, 314, 349};

int x8[8] = { 129, 59, 199, 129, 129, 59, 199, 129 };

int y8[8] = { 165, 200, 200, 235, 243, 278, 278, 313 };

int x11[8] = { 500, 460, 540, 500, 500, 460, 540, 500 };

int y11[8] = { 50, 70, 70, 90, 95, 115, 115, 135 };

int x12[8] = { 541, 501, 581, 541, 541, 501, 581, 541 };

int y12[8] = { 71, 91, 91, 111, 116, 136, 136, 156 };

int x13[8] = { 500, 460, 540, 500, 500, 460, 540, 500 };

int y13[8] = { 92, 112, 112, 132, 137, 157, 157, 177 };

int x14[8] = { 459, 419, 499, 459, 459, 419, 499, 459 };

int y14[8] = { 71, 91, 91, 111, 116, 136, 136, 156 };

int x15[8] = { 500, 460, 540, 500, 500, 460, 540, 500 };

int y15[8] = { 97, 117, 117, 137, 143, 163, 163, 183 };

int x16[8] = { 541, 501, 581, 541, 541, 501, 581, 541 };

int y16[8] = { 118, 138, 138, 158, 163, 183, 183, 203 };

int x17[8] = { 500, 460, 540, 500, 500, 460, 540, 500 };

int y17[8] = { 139, 159, 159, 179, 184, 204, 204, 224 };

int x18[8] = { 459, 419, 499, 459, 459, 419, 499, 459 };

int y18[8] = { 118, 138, 138, 158, 163, 183, 183, 203 };

int x21[8] = { 500, 460, 540, 500, 500, 460, 540, 500 };

int y21[8] = { 250, 270, 270, 290, 295, 315, 315, 335 };

int x22[8] = { 541, 501, 581, 541, 541, 501, 581, 541 };

int y22[8] = { 271, 291, 291, 311, 316, 336, 336, 356 };

int x23[8] = { 500, 460, 540, 500, 500, 460, 540, 500 };

int y23[8] = { 292, 312, 312, 332, 337, 357, 357, 377 };

int x24[8] = { 459, 419, 499, 459, 459, 419, 499, 459 };

int y24[8] = { 271, 291, 291, 311, 316, 336, 336, 356 };

int x25[8] = { 500, 460, 540, 500, 500, 460, 540, 500 };

int y25[8] = { 297, 317, 317, 337, 343, 363, 363, 383 };

int x26[8] = { 541, 501, 581, 541, 541, 501, 581, 541 };

int y26[8] = { 318, 338, 338, 358, 363, 383, 383, 403 };

int x27[8] = { 500, 460, 540, 500, 500, 460, 540, 500 };

int y27[8] = { 339, 359, 359, 379, 384, 404, 404, 424 };

int x28[8] = { 459, 419, 499, 459, 459, 419, 499, 459 };

int y28[8] = { 318, 338, 338, 358, 363, 383, 383, 403 };

void main()

{ int gmode, gdriver = DETECT, show = 0;

initgraph(&gdriver, &gmode, "");

setbkcolor(15);

setlinestyle(0, 0, 3);

Cube cube1(x1, y1, 14, 1, 2, 1, 200, 85);

Cube cube2(x2, y2, 14, 2, 4, 2, 271, 121);

Cube cube3(x3, y3, 14, 4, 6, 3, 200, 157);

Cube cube4(x4, y4, 14, 6, 1, 4, 129, 121);

Cube cube5(x5, y5, 15, 1, 2, 5, 200, 164);

Cube cube6(x6, y6, 15, 2, 4, 6, 271, 200);

Cube cube7(x7, y7, 15, 4, 6, 7, 200, 236);

Cube cube8(x8, y8, 15, 6, 1, 8, 129, 200);

Cube cube11(x11, y11, 6, 2, 6, 1, 500, 70);

Cube cube12(x12, y12, 6, 2, 15, 2, 541, 91);

Cube cube13(x13, y13, 4, 6, 1, 3, 500, 112);

Cube cube14(x14, y14, 4, 15, 14, 4, 459, 91);

Cube cube15(x15, y15, 15, 2, 6, 5, 500, 117);

Cube cube16(x16, y16, 15, 2, 15, 6, 541, 138);

Cube cube17(x17, y17, 15, 6, 1, 7, 500, 159);

Cube cube18(x18, y18, 15, 2, 14, 8, 459, 138);

Cube cube21(x21, y21, 14, 2, 6, 1, 500, 270);

Cube cube22(x22, y22, 15, 2, 4, 2, 541, 291);

Cube cube23(x23, y23, 15, 4, 14, 3, 500, 312);

Cube cube24(x24, y24, 14, 2, 6, 4, 459, 291);

Cube cube25(x25, y25, 15, 2, 6, 5, 500, 317);

Cube cube26(x26, y26, 15, 2, 15, 6, 541, 338);

Cube cube27(x27, y27, 15, 15, 14, 7, 500, 359);

Cube cube28(x28, y28, 15, 2, 14, 8, 459, 338);

BigCube cubRub(cube1,cube2,cube3,cube4,cube5,cube6,cube7,cube8);

BigCube addCub1(cube11,cube12,cube13,cube14,

cube15,cube16,cube17,cube18);

BigCube addCub2(cube21,cube22,cube23,cube24,

cube25,cube26,cube27,cube28);

cubRub.FullDraw();

int t[10]={370,0,370,480,640,480,640,0,70,0};

setfillstyle(1, getbkcolor());

fillpoly(4, t);

cubRub.ShowControls();

while(1)

{ char c;

c = getch();

if (c == 'i')

{ if(show == 0)

{ int t[10]={370,0,370,480,640,480,640,0,370,0};

setfillstyle(1, getbkcolor());

fillpoly(4, t);

addCub1.FullDraw();

addCub2.FullDraw();

show = 1;

}

else if (show == 1)

{ int t[10]={370,0,370,480,640,480,640,0,370,0};

setfillstyle(1, getbkcolor());

fillpoly(4, t);

cubRub.ShowControls();

show = 0;

}

}

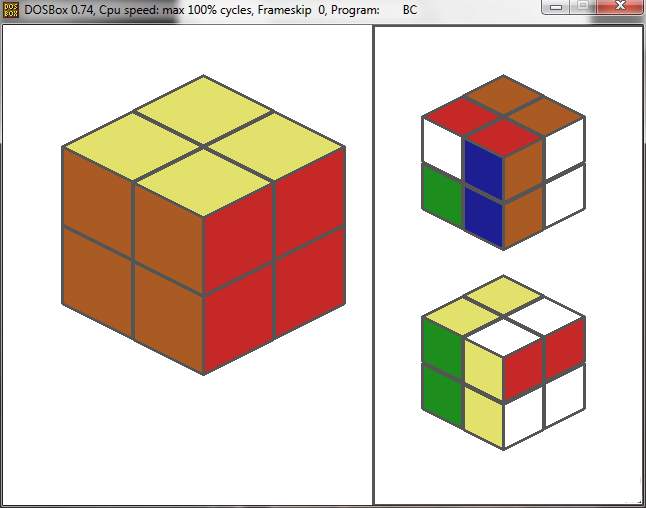
if(!cubRub.RotationCommand(c)) break;

}

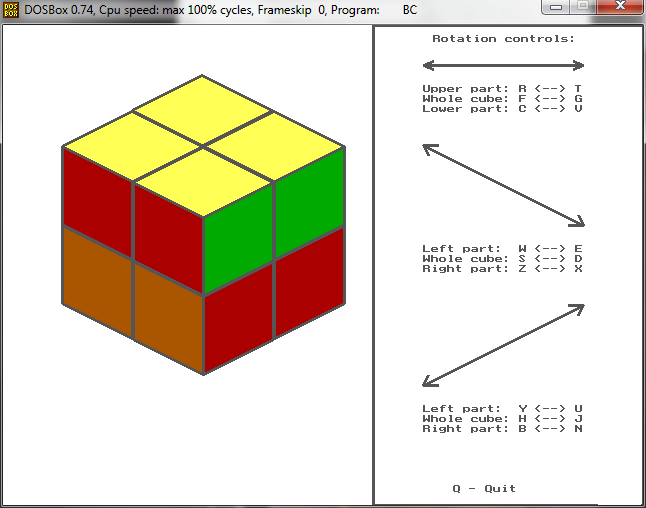
closegraph();

}

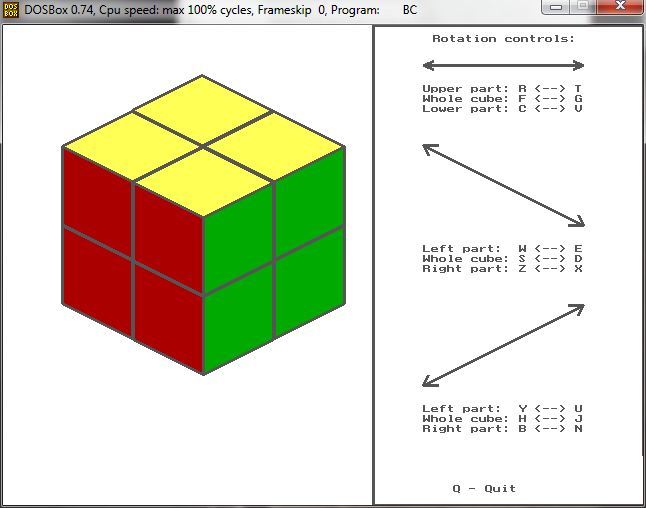
# Результаты работы



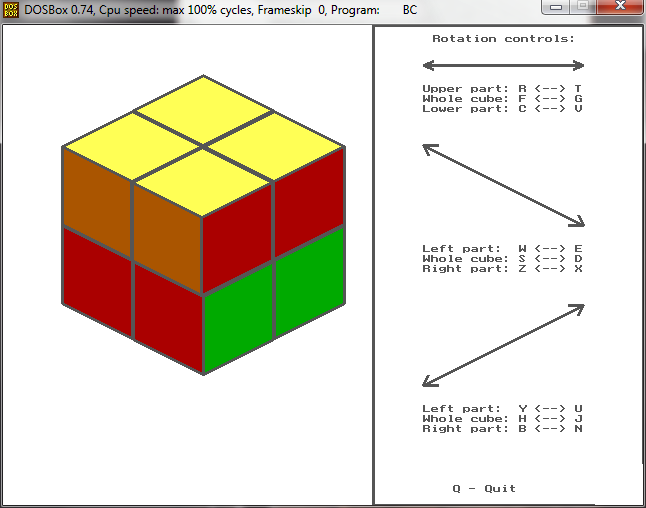
*Рис. 5. Показ дополнительных шарнирных кубиков*



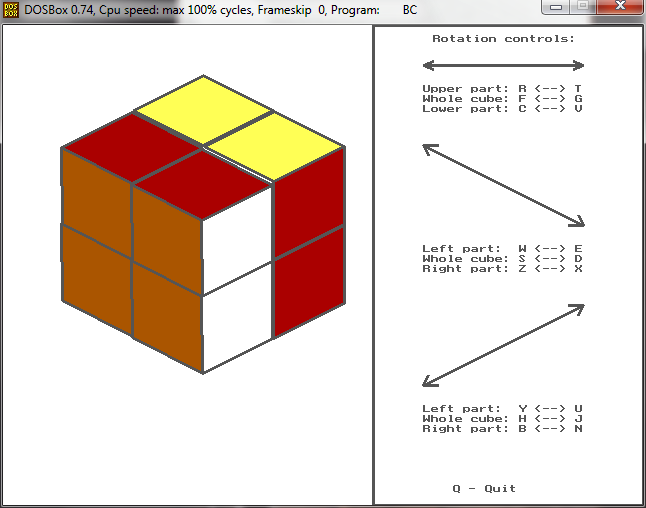
*Рис. 6. Вращение верхней части куба справа налево (клавиша R)*



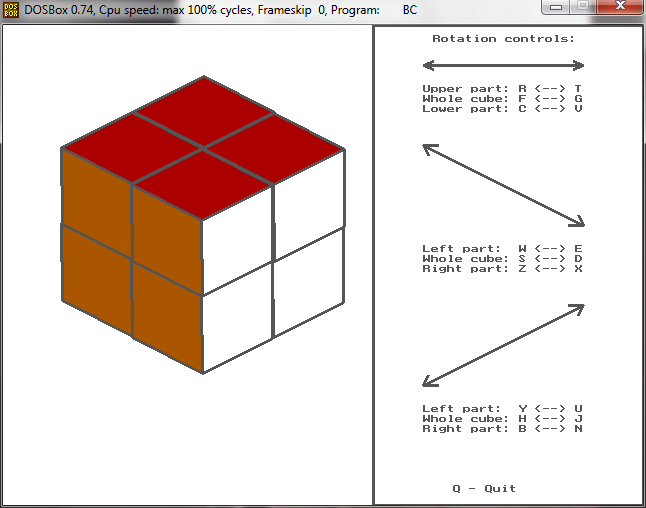
*Рис. 7. Вращение куба целиком справа налево (клавиша F)*



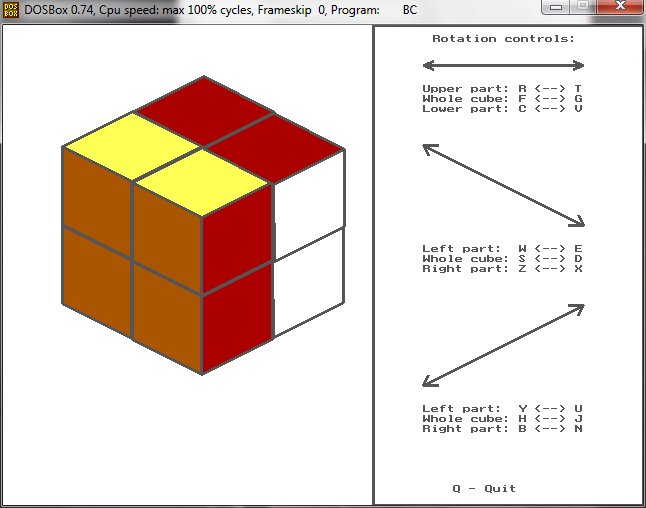
*Рис. 8. Вращение нижней части куба справа налево (клавиша С)*

**

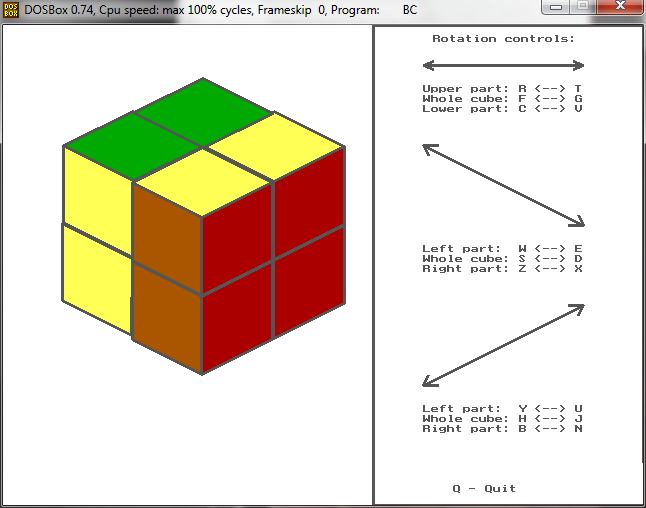
*Рис. 9. Вращение левой части куба влево вверх (клавиша W)*

**

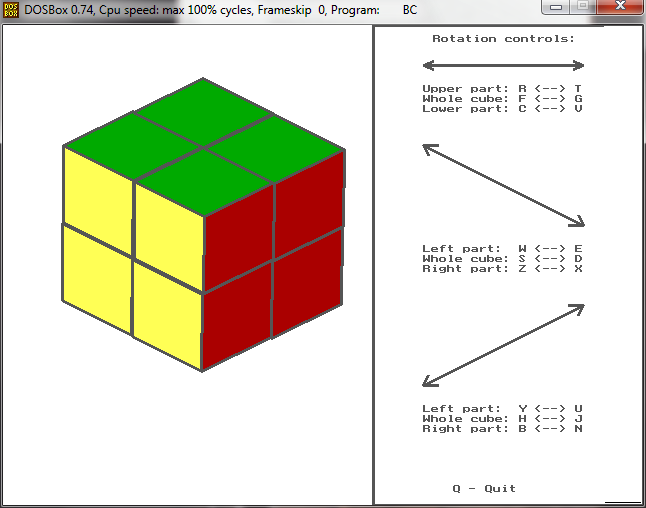
*Рис. 10. Вращение всего куба влево вверх (клавиша S)*

**

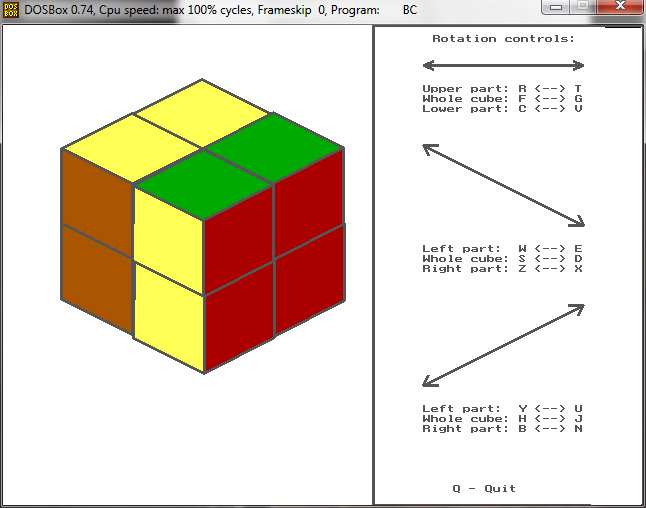
*Рис. 11. Вращение правой части куба влево вверх (клавиша Z)*

**

*Рис. 12. Вращение левой части куба влево вниз (клавиша Y)*

**

*Рис. 13. Вращение всего куба влево вниз (клавиша H)*

**

*Рис. 14. Вращение правой части куба влево вниз (клавиша B)*

# Заключение

В данной курсовой работе на примере подвижного шарнирного куба были рассмотрены такие аспекты объектно-ориентированного подхода, как композиция классов и объектов (объект «Большой куб» состоит из маленьких объектов «Кубик») и управление объектами одного класса, имеющими разное поведение (кубик будет вращаться по-разному в зависимости от его места в большом кубе). В частности, композиция объектов позволила легко управлять вращением сразу всего куба через его методы, не прибегая к отдельному вызову методов вращения у маленьких кубиков. Это один из примеров того, как с помощью объектно-ориентированного программирования можно организовывать иерархии и структуры сложных объектов, состоящих из более простых объектов, и управлять ими, не прилагая больших усилий и избегая написания лишнего кода.

# Список использованных источников

1. П.В. Новиков «Объектно-ориентированное программирование. Учебное пособие к лабораторным работам» (издание 2). URL: <https://www.academia.edu/22701705/Методическое_пособие_по_курсу_Объектно_ориентированное_программирование7>
2. Graphics.h | Программирование на С и С++. URL: <http://www.c-cpp.ru/funkcii/graphicsh> (дата последнего обращения - 24 апреля 2016).