```
#include <string.h>

#include<glut.h>

#include<stdio.h>


void *font = GLUT_BITMAP_TIMES_ROMAN_24;

char defaultMessage[] = "Rotation Speed:";

char *message = defaultMessage;


void

output(int x, int y, char *string)

{

  int len, i;


  glRasterPos2f(x, y);

  len = (int) strlen(string);

  for (i = 0; i < len; i++) {

    glutBitmapCharacter(font, string[i]);

  }

}



static float speed=0.0;
static int top[3][3]={{0,0,0},{0,0,0},{0,0,0}},

right[3][3]={{1,1,1},{1,1,1},{1,1,1}},

front[3][3]={{2,2,2},{2,2,2},{2,2,2}},

back[3][3]={{3,3,3},{3,3,3},{3,3,3}},

bottom[3][3]={{4,4,4},{4,4,4},{4,4,4}},

left[3][3]={{5,5,5},{5,5,5},{5,5,5}},

temp[3][3];


int solve[300];

int count=0;

int solve1=0;

static int rotation=0;

int rotationcomplete=0;

static GLfloat theta=0.0;

static GLint axis=0;

static GLfloat p=0.0,q=0.0,r=0.0;

static GLint inverse=0;

static GLfloat angle=0.0;

int beginx=0,beginy=0;

int moving=0;

static int speedmetercolor[15]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

static int speedmetercount=-1;



GLfloat vertices[][3]={{-1.0,-1.0,-1.0},
{1.0,-1.0,-1.0},
{1.0,1.0,-1.0},
{-1.0,1.0,-1.0}, //center
{-1.0,-1.0,1.0},
{1.0,-1.0,1.0},
{1.0,1.0,1.0},
{-1.0,1.0,1.0},

{-1.0,-3.0,-1.0},
{1.0,-3.0,-1.0},
{1.0,-1.0,-1.0},
{-1.0,-1.0,-1.0},  //bottom center
{-1.0,-3.0,1.0},
{1.0,-3.0,1.0},
{1.0,-1.0,1.0},
{-1.0,-1.0,1.0},

{-3.0,-1.0,-1.0},
{-1.0,-1.0,-1.0},
{-1.0,1.0,-1.0},
{-3.0,1.0,-1.0},  //left center
{-3.0,-1.0,1.0},
{-1.0,-1.0,1.0},
{-1.0,1.0,1.0},
{-3.0,1.0,1.0},
{1.0,-1.0,-1.0},

{3.0,-1.0,-1.0},

{3.0,1.0,-1.0},

{1.0,1.0,-1.0}, // right center

{1.0,-1.0,1.0},

{3.0,-1.0,1.0},

{3.0,1.0,1.0},

{1.0,1.0,1.0},




{-1.0,1.0,-1.0},

{1.0,1.0,-1.0},

{1.0,3.0,-1.0},

{-1.0,3.0,-1.0}, // top center

{-1.0,1.0,1.0},

{1.0,1.0,1.0},

{1.0,3.0,1.0},

{-1.0,3.0,1.0},


{-1.0,-1.0,1.0},

{1.0,-1.0,1.0},

{1.0,1.0,1.0},

{-1.0,1.0,1.0}, //front center
```

```
{-1.0,-1.0,3.0},

{1.0,-1.0,3.0},

{1.0,1.0,3.0},

 {-1.0,1.0,3.0},


{-1.0,-1.0,-3.0},

{1.0,-1.0,-3.0},

{1.0,1.0,-3.0},

{-1.0,1.0,-3.0}, //back center

{-1.0,-1.0,-1.0},

{1.0,-1.0,-1.0},

{1.0,1.0,-1.0},

{-1.0,1.0,-1.0},


{-3.0,1.0,-1.0},

{-1.0,1.0,-1.0},

{-1.0,3.0,-1.0},

{-3.0,3.0,-1.0}, // top left center

{-3.0,1.0,1.0},

{-1.0,1.0,1.0},

{-1.0,3.0,1.0},

{-3.0,3.0,1.0},


{1.0,1.0,-1.0},

{3.0,1.0,-1.0},
{3.0,3.0,-1.0},

{1.0,3.0,-1.0}, // top right  center

{1.0,1.0,1.0},

{3.0,1.0,1.0},

{3.0,3.0,1.0},

{1.0,3.0,1.0},


{-1.0,1.0,1.0},

{1.0,1.0,1.0},

{1.0,3.0,1.0},

{-1.0,3.0,1.0}, // top front center

{-1.0,1.0,3.0},

{1.0,1.0,3.0},

{1.0,3.0,3.0},

{-1.0,3.0,3.0},



{-1.0,1.0,-3.0},

{1.0,1.0,-3.0},

{1.0,3.0,-3.0},

{-1.0,3.0,-3.0}, // top back center

{-1.0,1.0,-1.0},

{1.0,1.0,-1.0},

{1.0,3.0,-1.0},

{-1.0,3.0,-1.0},
                               {-3.0,-3.0,-1.0},
{-1.0,-3.0,-1.0},
{-1.0,-1.0,-1.0},
{-3.0,-1.0,-1.0},  //bottom left center
{-3.0,-3.0,1.0},
{-1.0,-3.0,1.0},
{-1.0,-1.0,1.0},
{-3.0,-1.0,1.0},

                               {1.0,-3.0,-1.0},
{3.0,-3.0,-1.0},
{3.0,-1.0,-1.0},
{1.0,-1.0,-1.0},  //bottom  right center
{1.0,-3.0,1.0},
{3.0,-3.0,1.0},
{3.0,-1.0,1.0},
{1.0,-1.0,1.0},

                               {-1.0,-3.0,1.0},
{1.0,-3.0,1.0},
{1.0,-1.0,1.0},
{-1.0,-1.0,1.0},  //bottom front center
```

```
{-1.0,-3.0,3.0},

{1.0,-3.0,3.0},

{1.0,-1.0,3.0},

{-1.0,-1.0,3.0},




{-1.0,-3.0,-3.0},

{1.0,-3.0,-3.0},

{1.0,-1.0,-3.0},

{-1.0,-1.0,-3.0},  //bottom back center

{-1.0,-3.0,-1.0},

{1.0,-3.0,-1.0},

{1.0,-1.0,-1.0},

{-1.0,-1.0,-1.0},


                                    {-3.0,1.0,-3.0},

{-1.0,1.0,-3.0},

{-1.0,3.0,-3.0},

{-3.0,3.0,-3.0}, // top left back

{-3.0,1.0,-1.0},

{-1.0,1.0,-1.0},

{-1.0,3.0,-1.0},

{-3.0,3.0,-1.0},
                                        {-3.0,1.0,1.0},
{-1.0,1.0,1.0},
{-1.0,3.0,1.0},
{-3.0,3.0,1.0}, // top left front
{-3.0,1.0,3.0},
{-1.0,1.0,3.0},
{-1.0,3.0,3.0},
{-3.0,3.0,3.0},

                                        {1.0,1.0,-3.0},
{3.0,1.0,-3.0},
{3.0,3.0,-3.0},
{1.0,3.0,-3.0}, // top right  back
{1.0,1.0,-1.0},
{3.0,1.0,-1.0},
{3.0,3.0,-1.0},
{1.0,3.0,-1.0},

                                        {1.0,1.0,1.0},
{3.0,1.0,1.0},
{3.0,3.0,1.0},
{1.0,3.0,1.0}, // top right  front
{1.0,1.0,3.0},
{3.0,1.0,3.0},
{3.0,3.0,3.0},
{1.0,3.0,3.0},


                                        {-3.0,-1.0,-3.0},
{-1.0,-1.0,-3.0},
{-1.0,1.0,-3.0},
{-3.0,1.0,-3.0},  //ceneter left back
{-3.0,-1.0,-1.0},
{-1.0,-1.0,-1.0},
{-1.0,1.0,-1.0},
{-3.0,1.0,-1.0},


                                        {-3.0,-1.0,1.0},
{-1.0,-1.0,1.0},
{-1.0,1.0,1.0},
{-3.0,1.0,1.0},  //center left front
{-3.0,-1.0,3.0},
{-1.0,-1.0,3.0},
{-1.0,1.0,3.0},
{-3.0,1.0,3.0},


                                        {1.0,-1.0,-3.0},
{3.0,-1.0,-3.0},
{3.0,1.0,-3.0},
{1.0,1.0,-3.0}, // center right back
{1.0,-1.0,-1.0},
{3.0,-1.0,-1.0},
{3.0,1.0,-1.0},
{1.0,1.0,-1.0},

                    {1.0,-1.0,1.0},
{3.0,-1.0,1.0},
{3.0,1.0,1.0},
{1.0,1.0,1.0}, // center right front
{1.0,-1.0,3.0},
{3.0,-1.0,3.0},
{3.0,1.0,3.0},
{1.0,1.0,3.0},

                    {-3.0,-3.0,-3.0},
{-1.0,-3.0,-3.0},
{-1.0,-1.0,-3.0},
{-3.0,-1.0,-3.0},  //bottom left back
{-3.0,-3.0,-1.0},
{-1.0,-3.0,-1.0},
{-1.0,-1.0,-1.0},
{-3.0,-1.0,-1.0},

                    {-3.0,-3.0,1.0},
{-1.0,-3.0,1.0},
{-1.0,-1.0,1.0},
{-3.0,-1.0,1.0},  //bottom left front

{-3.0,-3.0,3.0},

{-1.0,-3.0,3.0},

{-1.0,-1.0,3.0},

{-3.0,-1.0,3.0},


                                        {1.0,-3.0,-3.0},

{3.0,-3.0,-3.0},

{3.0,-1.0,-3.0},

{1.0,-1.0,-3.0},  //bottom  right back

{1.0,-3.0,-1.0},

{3.0,-3.0,-1.0},

{3.0,-1.0,-1.0},

{1.0,-1.0,-1.0},



                                        {1.0,-3.0,1.0},

{3.0,-3.0,1.0},

{3.0,-1.0,1.0},

{1.0,-1.0,1.0},  //bottom  right front

{1.0,-3.0,3.0},

{3.0,-3.0,3.0},

{3.0,-1.0,3.0},

{1.0,-1.0,3.0},
                                        {0.0,7.0,0.0},

                                        {0.0,7.5,0.0},

                                        {0.5,7.5,0.0}, //speed meter

                                        {0.5,7.0,0.0}

                                      };

GLfloat color[][3]={{1.0,1.0,1.0},  //white

        {1.0,0.5,0.0},  //orange

        {0.0,0.0,1.0},  //blue

        {0.0,1.0,0.0},  //green

        {1.0,1.0,0.0},  //yellow

        {1.0,0.0,0.0}, //red

{0.5,0.5,0.5}, //grey used to represent faces of cube without colour

{.6,.5,.6}//speed meter colour

};
```

```
void polygon(int a,int b,int c,int d,int e)

{


        glColor3f(0,0,0);

        glLineWidth(3.0);

        glBegin(GL_LINE_LOOP);

        glVertex3fv(vertices[b]);

        glVertex3fv(vertices[c]);

        glVertex3fv(vertices[d]);

        glVertex3fv(vertices[e]);

        glEnd();



        glColor3fv(color[a]);

        glBegin(GL_POLYGON);

        glVertex3fv(vertices[b]);

        glVertex3fv(vertices[c]);

        glVertex3fv(vertices[d]);

        glVertex3fv(vertices[e]);

        glEnd();

}
```

```
void colorcube1()

{

            polygon(6,0,3,2,1);

            polygon(6,2,3,7,6);

            polygon(6,0,4,7,3);    // center piece

       polygon(6,1,2,6,5);

            polygon(6,4,5,6,7);

            polygon(6,0,1,5,4);



}

void colorcube2()

{

            polygon(6,8,11,10,9);

            polygon(6,10,11,15,14);

            polygon(6,8,12,15,11);    // bottom center

   polygon(6,9,10,14,13);

            polygon(6,12,13,14,15);

            polygon(bottom[1][1],8,9,13,12);



}


void colorcube3()
```

```
{

            polygon(6,16,19,18,17);
            polygon(6,18,19,23,22);
            polygon(left[1][1],16,20,23,19);    // left center
   polygon(6,17,18,22,21);
            polygon(6,20,21,22,23);
            polygon(6,16,17,21,20);


}

void colorcube4()
{
            polygon(6,24,27,26,25);
            polygon(6,26,27,31,30);
            polygon(6,24,28,31,27);    // right center
   polygon(right[1][1],25,26,30,29);
            polygon(6,28,29,30,31);
            polygon(6,24,25,29,28);


}

void colorcube5()
```

```
{
            polygon(6,32,35,34,33);
            polygon(top[1][1],34,35,39,38);
            polygon(6,32,36,39,35);    // top center
   polygon(6,33,34,38,37);
            polygon(6,36,37,38,39);
            polygon(6,32,33,37,36);



}

void colorcube6()
{
            polygon(6,40,43,42,41);
            polygon(6,42,43,47,46);
            polygon(6,40,44,47,43);    // front center
   polygon(6,41,42,46,45);
            polygon(front[1][1],44,45,46,47);
            polygon(6,40,41,45,44);



}

void colorcube7()
{
```

```
            polygon(back[1][1],48,51,50,49);

            polygon(6,50,51,55,54);

            polygon(6,48,52,55,51);    //back center

   polygon(6,49,50,54,53);

            polygon(6,52,53,54,55);

            polygon(6,48,49,53,52);



}



void colorcube8()

{

            polygon(6,56,59,58,57);

            polygon(top[1][0],58,59,63,62);

            polygon(left[0][1],56,60,63,59);    // top left center

   polygon(6,57,58,62,61);

            polygon(6,60,61,62,63);

            polygon(6,56,57,61,60);



}


void colorcube9()

{
```

```
            polygon(6,64,67,66,65);

            polygon(top[1][2],66,67,71,70);

            polygon(6,64,68,71,67);    // top right center

   polygon(right[0][1],65,66,70,69);

            polygon(6,68,69,70,71);

            polygon(6,64,65,69,68);



}


void colorcube10()

{

            polygon(6,72,75,74,73);

            polygon(top[2][1],74,75,79,78);

            polygon(6,72,76,79,75);    // top front center

   polygon(6,73,74,78,77);

            polygon(front[0][1],76,77,78,79);

            polygon(6,72,73,77,76);



}


void colorcube11()

{

            polygon(back[0][1],80,83,82,81);
            polygon(top[0][1],82,83,87,86);

            polygon(6,80,84,87,83);    // top back center

   polygon(6,81,82,86,85);

            polygon(6,84,85,86,87);

            polygon(6,80,81,85,84);



}


void colorcube12()

{

            polygon(6,80+8,83+8,82+8,81+8);

            polygon(6,82+8,83+8,87+8,86+8);

            polygon(left[2][1],80+8,84+8,87+8,83+8);    // bottom left center

   polygon(6,81+8,82+8,86+8,85+8);

            polygon(6,84+8,85+8,86+8,87+8);

            polygon(bottom[1][0],80+8,81+8,85+8,84+8);



}


void colorcube13()

{

            polygon(6,80+16,83+16,82+16,81+16);

            polygon(6,82+16,83+16,87+16,86+16);
            polygon(6,80+16,84+16,87+16,83+16);    // bottom right center
    polygon(right[2][1],81+16,82+16,86+16,85+16);
            polygon(6,84+16,85+16,86+16,87+16);
            polygon(bottom[1][2],80+16,81+16,85+16,84+16);



}

void colorcube14()
{
            polygon(6,80+24,83+24,82+24,81+24);
            polygon(6,82+24,83+24,87+24,86+24);
            polygon(6,80+24,84+24,87+24,83+24);    // bottom front center
    polygon(6,81+24,82+24,86+24,85+24);
            polygon(front[2][1],84+24,85+24,86+24,87+24);
            polygon(bottom[0][1],80+24,81+24,85+24,84+24);



}
void colorcube15()
{
            polygon(back[2][1],112,115,114,113);
            polygon(6,114,115,119,118);
            polygon(6,112,116,119,115);    // bottom back center
    polygon(6,113,114,118,117);
```

```
            polygon(6,116,117,118,119);

            polygon(bottom[2][1],112,113,117,116);



}

void colorcube16()

{

            polygon(back[0][2],120,123,122,121);

            polygon(top[0][0],122,123,127,126);

            polygon(left[0][0],120,124,127,123);    // top left back

   polygon(6,121,122,126,125);

            polygon(6,124,125,126,127);

            polygon(6,120,121,125,124);



}

void colorcube17()

{

            polygon(6,128,131,130,129);

            polygon(top[2][0],130,131,135,134);

            polygon(left[0][2],128,132,135,131);    // top left front

   polygon(6,129,130,134,133);

            polygon(front[0][0],132,133,134,135);
            polygon(6,128,129,133,132);



}



void colorcube18()

{

            polygon(back[0][0],136,139,138,137);

            polygon(top[0][2],138,139,143,142);

            polygon(6,136,140,143,139);    // top right back

   polygon(right[0][2],137,138,142,141);

            polygon(6,140,141,142,143);

            polygon(6,136,137,141,140);



}



void colorcube19()

{

            polygon(6,144,147,146,145);

            polygon(top[2][2],146,147,151,150);

            polygon(6,144,148,151,147);    // top right front

   polygon(right[0][0],145,146,150,149);
```

```
            polygon(front[0][2],148,149,150,151);

            polygon(6,144,145,149,148);




}


void colorcube20()

{

            polygon(back[1][2],152,155,154,153);

            polygon(6,154,155,159,158);

            polygon(left[1][0],152,156,159,155);    //center left back

    polygon(6,153,154,158,157);

            polygon(6,156,157,158,159);

            polygon(6,152,153,157,156);




}


void colorcube21()

{

            polygon(6,160,163,162,161);

            polygon(6,162,163,167,166);

            polygon(left[1][2],160,164,167,163);    // center left front

    polygon(6,161,162,166,165);

            polygon(front[1][0],164,165,166,167);
            polygon(6,160,161,165,164);



}



void colorcube22()
{
            polygon(back[1][0],168,171,170,169);
            polygon(6,170,171,175,174);
            polygon(6,168,172,175,171);    // center right back
   polygon(right[1][2],169,170,174,173);
            polygon(6,172,173,174,175);
            polygon(6,168,169,173,172);



}



void colorcube23()
{
            polygon(6,176,179,178,177);
            polygon(6,178,179,183,182);
            polygon(6,176,180,183,179);    //center right front
   polygon(right[1][0],177,178,182,181);
```

```
            polygon(front[1][2],180,181,182,183);

            polygon(6,176,177,181,180);



}

void colorcube24()

{

            polygon(back[2][2],184,187,186,185);

            polygon(6,186,187,191,190);

            polygon(left[2][0],184,188,191,187);    // bottom left back

    polygon(6,185,186,190,189);

            polygon(6,188,189,190,191);

            polygon(bottom[2][0],184,185,189,188);



}

void colorcube25()

{

            polygon(6,192,195,194,193);

            polygon(6,194,195,199,198);

            polygon(left[2][2],192,196,199,195);    // bottom left front

    polygon(6,193,194,198,197);

            polygon(front[2][0],196,197,198,199);

            polygon(bottom[0][0],192,193,197,196);
```

```
}


void colorcube26()

{

            polygon(back[2][0],200,203,202,201);

            polygon(6,202,203,207,206);

            polygon(6,200,204,207,203);    // bottom right back

   polygon(right[2][2],201,202,206,205);

            polygon(6,204,205,206,207);

            polygon(bottom[2][2],200,201,205,204);



}


void colorcube27()

{

            polygon(6,208,211,210,209);

            polygon(6,210,211,215,214);

            polygon(6,208,212,215,211);    // bottom right front

   polygon(right[2][0],209,210,214,213);

            polygon(front[2][2],212,213,214,215);

            polygon(bottom[0][2],208,209,213,212);
```

```
}

void speedmeter()

{
   glColor3fv(color[7]);
           glBegin(GL_POLYGON);
           glVertex3f(0.0,7.2,0.0);
           glVertex3f(1.0,7.0,0.0);
           glVertex3f(1.0,7.5,0.0);
           glEnd();



           glPushMatrix();
   glTranslatef(1.0,0.0,0.0);
           polygon(speedmetercolor[0],216,217,218,219);
           glPopMatrix();

           glPushMatrix();
   glTranslatef(1.5,0.0,0.0);
           polygon(speedmetercolor[1],216,217,218,219);
           glPopMatrix();

           glPushMatrix();
```

```
glTranslatef(2.0,0.0,0.0);
        polygon(speedmetercolor[2],216,217,218,219);
        glPopMatrix();

        glPushMatrix();
glTranslatef(2.5,0.0,0.0);
        polygon(speedmetercolor[3],216,217,218,219);
        glPopMatrix();

        glPushMatrix();
glTranslatef(3.0,0.0,0.0);
        polygon(speedmetercolor[4],216,217,218,219);
        glPopMatrix();

        glPushMatrix();
glTranslatef(3.5,0.0,0.0);
        polygon(speedmetercolor[5],216,217,218,219);
        glPopMatrix();

                glPushMatrix();
glTranslatef(4.0,0.0,0.0);
        polygon(speedmetercolor[6],216,217,218,219);
        glPopMatrix();

                glPushMatrix();
```

```
glTranslatef(4.5,0.0,0.0);
        polygon(speedmetercolor[7],216,217,218,219);
        glPopMatrix();

                   glPushMatrix();
glTranslatef(5.0,0.0,0.0);
        polygon(speedmetercolor[8],216,217,218,219);
        glPopMatrix();

                   glPushMatrix();
glTranslatef(5.5,0.0,0.0);
        polygon(speedmetercolor[9],216,217,218,219);
        glPopMatrix();

                   glPushMatrix();
glTranslatef(6.0,0.0,0.0);
        polygon(speedmetercolor[10],216,217,218,219);
        glPopMatrix();

                   glPushMatrix();
glTranslatef(6.5,0.0,0.0);
        polygon(speedmetercolor[11],216,217,218,219);
        glPopMatrix();

                   glPushMatrix();
```

```
   glTranslatef(7.0,0.0,0.0);
           polygon(speedmetercolor[12],216,217,218,219);
           glPopMatrix();

                   glPushMatrix();
   glTranslatef(7.5,0.0,0.0);
           polygon(speedmetercolor[13],216,217,218,219);
           glPopMatrix();

                   glPushMatrix();
   glTranslatef(8.0,0.0,0.0);
           polygon(speedmetercolor[14],216,217,218,219);
           glPopMatrix();

           glColor3fv(color[7]);
           glBegin(GL_POLYGON);
           glVertex3f(9.5,7.2,0.0);
           glVertex3f(8.5,7.0,0.0);
           glVertex3f(8.5,7.5,0.0);
           glEnd();

}
```

```
void display()

{



            glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
            glLoadIdentity();

            speedmeter();

glColor3fv(color[0]);
            output(0,8,message);

            glPushMatrix();
   glRotatef(25.0+p,1.0,0.0,0.0);
   glRotatef(-30.0+q,0.0,1.0,0.0);
   glRotatef(0.0+r,0.0,0.0,1.0);



if(rotation==0)
{

colorcube1();
```

```
colorcube2();

colorcube3();

colorcube4();

colorcube5();

colorcube6();

colorcube7();

colorcube8();

colorcube9();

colorcube10();

colorcube11();

colorcube12();

colorcube13();

colorcube14();

colorcube15();

colorcube16();

colorcube17();

colorcube18();

colorcube19();

colorcube20();

colorcube21();

colorcube22();

colorcube23();

colorcube24();

colorcube25();

colorcube26();
```

```
colorcube27();

}

if(rotation==1)

{



colorcube1();

colorcube2();

colorcube3();

colorcube4();

colorcube6();

colorcube7();

colorcube12();

colorcube13();

colorcube14();

colorcube15();

colorcube20();

colorcube21();

colorcube22();

colorcube23();

colorcube24();

colorcube25();

colorcube26();

colorcube27();
```

```
if(inverse==0)
{glPushMatrix();
glColor3fv(color[0]);
            output(-11,6,"Top");
            glPopMatrix();
glRotatef(-theta,0.0,1.0,0.0);
}
else
{glPushMatrix();
glColor3fv(color[0]);
            output(-11,6,"TopInverted");
            glPopMatrix();
glRotatef(theta,0.0,1.0,0.0);
}
colorcube5();
colorcube8();
colorcube9();
colorcube10();
colorcube11();
colorcube16();
colorcube17();
colorcube18();
colorcube19();


}
```

```
if(rotation==2)

{

colorcube1();

colorcube2();

colorcube3();

colorcube5();

colorcube6();

colorcube7();

colorcube8();

colorcube10();

colorcube11();

colorcube12();

colorcube14();

colorcube15();

colorcube16();

colorcube17();

colorcube20();

colorcube21();

colorcube24();

colorcube25();

if(inverse==0)

{

glPushMatrix();

glColor3fv(color[0]);
```

```
            output(-11,6,"Right");

            glPopMatrix();

glRotatef(-theta,1.0,0.0,0.0);

}

else

{

            glPushMatrix();

glColor3fv(color[0]);

            output(-11,6,"RightInverted");

            glPopMatrix();

glRotatef(theta,1.0,0.0,0.0);

}

colorcube4();

colorcube9();

colorcube13();

colorcube18();

colorcube19();

colorcube22();

colorcube23();

colorcube26();

colorcube27();

}
```

```
if(rotation==3)
{
colorcube1();
colorcube2();
colorcube3();
colorcube4();
colorcube5();
colorcube7();
colorcube8();
colorcube9();
colorcube11();
colorcube12();
colorcube13();
colorcube15();
colorcube16();
colorcube18();
colorcube20();
colorcube22();
colorcube24();
colorcube26();
if(inverse==0)
{
        glPushMatrix();
glColor3fv(color[0]);
        output(-11,6,"Front");
        glPopMatrix();
glRotatef(-theta,0.0,0.0,1.0);
}

else
{
        glPushMatrix();
glColor3fv(color[0]);
        output(-11,6,"FrontInverted");
        glPopMatrix();
glRotatef(theta,0.0,0.0,1.0);
}

colorcube6();
colorcube10();
colorcube14();
colorcube17();
colorcube19();
colorcube21();
colorcube23();
colorcube25();
colorcube27();
}

if(rotation==4)
```

```
{
colorcube1();
colorcube2();
colorcube4();
colorcube5();
colorcube6();
colorcube7();
colorcube9();
colorcube10();
colorcube11();
colorcube13();
colorcube14();
colorcube15();
colorcube18();
colorcube19();
colorcube22();
colorcube23();
colorcube26();
colorcube27();
if(inverse==0)
{glPushMatrix();
glColor3fv(color[0]);
            output(-11,6,"Left");
            glPopMatrix();
glRotatef(theta,1.0,0.0,0.0);
}
else
{glPushMatrix();
glColor3fv(color[0]);
            output(-11,6,"LeftInverted");
    glPopMatrix();
            glRotatef(-theta,1.0,0.0,0.0);
}
colorcube3();
colorcube8();
colorcube12();
colorcube16();
colorcube17();
colorcube20();
colorcube21();
colorcube24();
colorcube25();
}


if(rotation==5)
{
colorcube1();
colorcube2();
colorcube3();
```

```
colorcube4();

colorcube5();

colorcube6();

colorcube8();

colorcube9();

colorcube10();

colorcube12();

colorcube13();

colorcube14();

colorcube17();

colorcube19();

colorcube21();

colorcube23();

colorcube25();

colorcube27();

if(inverse==0)

{glPushMatrix();

glColor3fv(color[0]);

            output(-11,6,"Back");

            glPopMatrix();

glRotatef(theta,0.0,0.0,1.0);


}

else

{
```

```
            glPushMatrix();
glColor3fv(color[0]);
            output(-11,6,"BackInverted");
            glPopMatrix();
            glRotatef(-theta,0.0,0.0,1.0);
}
colorcube7();
colorcube11();
colorcube15();
colorcube16();
colorcube18();
colorcube20();
colorcube22();
colorcube24();
colorcube26();
}

if(rotation==6)
{
colorcube1();
colorcube3();
colorcube4();
colorcube5();
colorcube6();
colorcube7();
```

```
colorcube8();

colorcube9();

colorcube10();

colorcube11();

colorcube16();

colorcube17();

colorcube18();

colorcube19();

colorcube20();

colorcube21();

colorcube22();

colorcube23();


if(inverse==0)

{glPushMatrix();

glColor3fv(color[0]);

            output(-11,6,"Bottom");

            glPopMatrix();

            glRotatef(theta,0.0,1.0,0.0);

}

else

{glPushMatrix();

glColor3fv(color[0]);

            output(-11,6,"BottomInverted");

            glPopMatrix();
glRotatef(-theta,0.0,1.0,0.0);

}

colorcube2();

colorcube12();

colorcube13();

colorcube14();

colorcube15();

colorcube24();

colorcube25();

colorcube26();

colorcube27();


}

glPopMatrix();



/*glPushMatrix();

glTranslatef(-.5,-4,0);

glScalef(speed/4.5,1.0,1.0);

glTranslatef(0.5,4,0);

polygon(5,216,217,218,219);


glPopMatrix();

*/
```

```
            glFlush();

            glutSwapBuffers();


}



void transpose(char a)

{


 if(a=='r')

            {

            int temp;

            temp=right[0][0];

            right[0][0]=right[2][0];

            right[2][0]=right[2][2];

            right[2][2]=right[0][2];

            right[0][2]=temp;

            temp=right[1][0];

   right[1][0]=right[2][1];

            right[2][1]=right[1][2];

            right[1][2]=right[0][1];

            right[0][1]=temp;

 }

            if(a=='t')
            {
            int temp;
            temp=top[0][0];
            top[0][0]=top[2][0];
            top[2][0]=top[2][2];
            top[2][2]=top[0][2];
            top[0][2]=temp;
            temp=top[1][0];
   top[1][0]=top[2][1];
            top[2][1]=top[1][2];
            top[1][2]=top[0][1];
            top[0][1]=temp;
}
                    if(a=='f')
            {
            int temp;
            temp=front[0][0];
            front[0][0]=front[2][0];
            front[2][0]=front[2][2];
            front[2][2]=front[0][2];
            front[0][2]=temp;
            temp=front[1][0];
   front[1][0]=front[2][1];
            front[2][1]=front[1][2];
            front[1][2]=front[0][1];
```

```
        front[0][1]=temp;
        }
        if(a=='l')
        {
        int temp;
        temp=left[0][0];
        left[0][0]=left[2][0];
        left[2][0]=left[2][2];
        left[2][2]=left[0][2];
        left[0][2]=temp;
        temp=left[1][0];
left[1][0]=left[2][1];
        left[2][1]=left[1][2];
        left[1][2]=left[0][1];
        left[0][1]=temp;
        }
        if(a=='k')
        {
        int temp;
        temp=back[0][0];
        back[0][0]=back[2][0];
        back[2][0]=back[2][2];
        back[2][2]=back[0][2];
        back[0][2]=temp;
        temp=back[1][0];
```

```
   back[1][0]=back[2][1];
            back[2][1]=back[1][2];
            back[1][2]=back[0][1];
            back[0][1]=temp;
            }


                        if(a=='b')
            {
            int temp;
            temp=bottom[0][0];
            bottom[0][0]=bottom[2][0];
            bottom[2][0]=bottom[2][2];
            bottom[2][2]=bottom[0][2];
            bottom[0][2]=temp;
            temp=bottom[1][0];
   bottom[1][0]=bottom[2][1];
            bottom[2][1]=bottom[1][2];
            bottom[1][2]=bottom[0][1];
            bottom[0][1]=temp;
            }
}



void topc()
{
        transpose('t');
int temp1=front[0][0];
int temp2=front[0][1];
int temp3=front[0][2];

front[0][0]=right[0][0];
front[0][1]=right[0][1];
front[0][2]=right[0][2];

right[0][0]=back[0][0];
right[0][1]=back[0][1];
right[0][2]=back[0][2];

back[0][0]=left[0][0];
back[0][1]=left[0][1];
back[0][2]=left[0][2];

left[0][0]=temp1;
left[0][1]=temp2;
left[0][2]=temp3;

}

void frontc()
{
```

```
transpose('f');

int temp1=left[0][2];

int temp2=left[1][2];

int temp3=left[2][2];


left[0][2]=bottom[0][0];

left[1][2]=bottom[0][1];

left[2][2]=bottom[0][2];


bottom[0][0]=right[2][0];

bottom[0][1]=right[1][0];

bottom[0][2]=right[0][0];


right[2][0]=top[2][2];

right[1][0]=top[2][1];

right[0][0]=top[2][0];


top[2][2]=temp1;

top[2][1]=temp2;

top[2][0]=temp3;

}


void rightc()

{

        transpose('r');
```

```
            int temp1=top[0][2];
   int temp2=top[1][2];
            int temp3=top[2][2];

            top[0][2]=front[0][2];
            top[1][2]=front[1][2];
            top[2][2]=front[2][2];

            front[0][2]=bottom[0][2];
            front[1][2]=bottom[1][2];
            front[2][2]=bottom[2][2];



            bottom[0][2]=back[2][0];
            bottom[1][2]=back[1][0];
            bottom[2][2]=back[0][0];

            back[2][0]=temp1;
            back[1][0]=temp2;
            back[0][0]=temp3;

}

void leftc()
{
```

```
transpose('l');
            int temp1=front[0][0];
            int temp2=front[1][0];
            int temp3=front[2][0];

            front[0][0]=top[0][0];
            front[1][0]=top[1][0];
            front[2][0]=top[2][0];

            top[0][0]=back[2][2];
            top[1][0]=back[1][2];
            top[2][0]=back[0][2];

            back[2][2]=bottom[0][0];
            back[1][2]=bottom[1][0];
            back[0][2]=bottom[2][0];


            bottom[0][0]=temp1;
            bottom[1][0]=temp2;
            bottom[2][0]=temp3;
}

void backc()
{
            transpose('k');
            int temp1=top[0][0];
            int temp2=top[0][1];
            int temp3=top[0][2];

            top[0][0]=right[0][2];
            top[0][1]=right[1][2];
            top[0][2]=right[2][2];

            right[0][2]=bottom[2][2];
            right[1][2]=bottom[2][1];
            right[2][2]=bottom[2][0];

            bottom[2][2]=left[2][0];
            bottom[2][1]=left[1][0];
            bottom[2][0]=left[0][0];

            left[2][0]=temp1;
            left[1][0]=temp2;
            left[0][0]=temp3;
}


void bottomc()
{
            transpose('b');
            int temp1=front[2][0];
            int temp2=front[2][1];
            int temp3=front[2][2];

            front[2][0]=left[2][0];
            front[2][1]=left[2][1];
            front[2][2]=left[2][2];

            left[2][0]=back[2][0];
            left[2][1]=back[2][1];
            left[2][2]=back[2][2];

            back[2][0]=right[2][0];
            back[2][1]=right[2][1];
            back[2][2]=right[2][2];

            right[2][0]=temp1;
            right[2][1]=temp2;
            right[2][2]=temp3;

}


void spincube()
```

```
{   theta+=0.5+speed;

             if(theta==360.0)

            theta-=360.0;

            if(theta>=90.0)

            {

                        rotationcomplete=1;

            glutIdleFunc(NULL);


if(rotation==1&&inverse==0)

            {

            topc();

}

if(rotation==1&&inverse==1)

            {

            topc();

            topc();

            topc();

}


if(rotation==2&&inverse==0)

            {

  rightc();



  }
```

```
if(rotation==2&&inverse==1)
            {
rightc();
rightc();
rightc();


}
if(rotation==3&&inverse==0)
            {
            frontc();
}
if(rotation==3&&inverse==1)
            {
            frontc();
            frontc();
            frontc();
}
if(rotation==4&&inverse==0)
{
            leftc();
}
if(rotation==4&&inverse==1)
{
            leftc();
            leftc();
            leftc();


}
if(rotation==5&&inverse==0)
{
            backc();
}
if(rotation==5&&inverse==1)
{
            backc();
            backc();
            backc();
}
if(rotation==6&&inverse==0)
{
            bottomc();
}
if(rotation==6&&inverse==1)
{
            bottomc();
            bottomc();
            bottomc();


}
```

```
rotation=0;

theta=0;

}




glutPostRedisplay();

}



void

motion(int x, int y)

{

  if (moving) {


    q=q + (x - beginx);

    beginx = x;

    p=p + (y - beginy);

              beginy=y;

              glutPostRedisplay();

  }

}



void mouse(int btn,int state,int x,int y)
```

```
{
        if(btn==GLUT_MIDDLE_BUTTON && state==GLUT_DOWN)
        {
                 //printf("%d %d",x,y);

        }
        if(btn==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
        {
                /*printf("%d %d\n",x,y);
                if(x>=0&&x<=2&&y>=7&&y<=9)
                {
                        printf("colour red\n");
                }
                */
  moving = 1;
  beginx = x;
  beginy=y;

        }
}


static void keyboard(unsigned char key,int x,int y)
{
```

```
            if(key=='a'&&rotationcomplete==1)
            {    rotationcomplete=0;
        rotation=1;
                        inverse=0;
                        solve[++count]=1;
                        glutIdleFunc(spincube);


            }
            if(key=='q'&&rotationcomplete==1)
            {    rotationcomplete=0;
        rotation=1;
                        inverse=1;
                        solve[++count]=-1;
                        glutIdleFunc(spincube);


            }
            if(key=='s'&&rotationcomplete==1)
            {rotationcomplete=0;
                       rotation=2;
                       inverse=0;
                       solve[++count]=2;
            glutIdleFunc(spincube);
            }
    if(key=='w'&&rotationcomplete==1)
            {rotationcomplete=0;
```

```
                rotation=2;
                inverse=1;
                solve[++count]=-2;
        glutIdleFunc(spincube);
        }
if(key=='d'&&rotationcomplete==1)
        {rotationcomplete=0;
                rotation=3;
                inverse=0;
                solve[++count]=3;
        glutIdleFunc(spincube);

        }
if(key=='e'&&rotationcomplete==1)
        {rotationcomplete=0;
                rotation=3;
                inverse=1;
                solve[++count]=-3;
        glutIdleFunc(spincube);

        }

if(key=='f'&&rotationcomplete==1)
        {rotationcomplete=0;
                rotation=4;
                inverse=0;

                solve[++count]=4;

        glutIdleFunc(spincube);


        }
if(key=='r'&&rotationcomplete==1)

        {rotationcomplete=0;

                rotation=4;

                inverse=1;

                solve[++count]=-4;

        glutIdleFunc(spincube);


        }
if(key=='g'&&rotationcomplete==1)

        {rotationcomplete=0;

                rotation=5;

                inverse=0;

                solve[++count]=5;

        glutIdleFunc(spincube);


        }
if(key=='t'&&rotationcomplete==1)

        {rotationcomplete=0;

                rotation=5;

                inverse=1;
```

```
                    solve[++count]=-5;
          glutIdleFunc(spincube);


          }


if(key=='h'&&rotationcomplete==1)
          {rotationcomplete=0;
                    rotation=6;
                    inverse=0;
                    solve[++count]=6;
          glutIdleFunc(spincube);


          }


if(key=='y'&&rotationcomplete==1)
          {rotationcomplete=0;
                    rotation=6;
                    inverse=1;
                    solve[++count]=-6;
          glutIdleFunc(spincube);


          }
if(key=='2'&&rotationcomplete==1)
          {
p=p+2.0;
```

```
glutIdleFunc(spincube);

}

if(key=='8'&&rotationcomplete==1)

            {

p=p-2.0;

glutIdleFunc(spincube);

}

if(key=='6'&&rotationcomplete==1)

            {

q=q+2.0;

glutIdleFunc(spincube);

}

if(key=='4'&&rotationcomplete==1)

            {

q=q-2.0;

glutIdleFunc(spincube);

}


if(key=='9'&&rotationcomplete==1)

            {

r=r+2.0;

glutIdleFunc(spincube);

}


if(key=='1'&&rotationcomplete==1)
```

```
            {
r=r-2.0;
glutIdleFunc(spincube);
}
if(key=='5'&&rotationcomplete==1)
            {
p=0.0;
q=0.0;
r=0.0;
glutIdleFunc(spincube);
}

if(key=='m'&&rotationcomplete==1)
            {
                        if(speed<=1.3)
                        {
                        //for(speed=0;speed<1.3;speed++)

      speed=speed+0.3;
                        speedmetercolor[++speedmetercount]=3;

                        }
                        glutPostRedisplay();
}
if(key=='m'&&rotationcomplete==1)
```

```
        {
                if(speed>1.3)
                {                   if(speed<=2.9)
                {
                //for(speed=0;speed<1.3;speed++)

     speed=speed+0.3;
                speedmetercolor[++speedmetercount]=4;
                }
                }
                glutPostRedisplay();
}
if(key=='m'&&rotationcomplete==1)
        {

                if(speed>2.9)
                {
                        if(speed<=4.2)
                {
                //r(speed=0;speed<=4.3;speed+=0.1)
                //{
     speed=speed+0.3;
                speedmetercolor[++speedmetercount]=5;
                        }
                }
```

```
                    glutPostRedisplay();

}

if(key=='n'&&rotationcomplete==1)

          {

                    if(speed>=0.3)

                    {

     speed=speed-0.3;

                    speedmetercolor[speedmetercount--]=0;

                    }

                    glutPostRedisplay();


}



if(key=='o'&&rotationcomplete==1)

          {

                    rotationcomplete=0;

          if(count>=0)

          {

                    if(solve[count]<0)

                              {

                                        rotation=-1*solve[count];

                                        inverse=0;

          glutIdleFunc(spincube);

                              }
```

```
                                        if(solve[count]>0)
                                        {
                                        rotation=solve[count];
                                        inverse=1;
        glutIdleFunc(spincube);
                                        }


        count--;
}


glutIdleFunc(spincube);


}

}

void myreshape(int w,int h)
{
        glViewport(0,0,w,h);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        if (w <= h)
```

```
            glOrtho(-10.0,10.0,-10.0*(GLfloat)h/(GLfloat)w, 10.0*(GLfloat)h/(GLfloat)w,-10.0,10.0);

            else

            glOrtho(-10.0*(GLfloat)w/(GLfloat)h, 10.0*(GLfloat)w/(GLfloat)h,-10.0,10.0,-10.0,10.0);

            glMatrixMode(GL_MODELVIEW);

}

void mymenu(int id)

{

            if(rotationcomplete==1)

            {rotationcomplete=0;

switch(id)

{

   case 1:

                        rotation=1;

            inverse=0;

                        solve[++count]=1;

               glutIdleFunc(spincube);

            break;


            case 2:

                        rotation=1;

                        inverse=1;

                        solve[++count]=-1;

                        glutIdleFunc(spincube);

            break;
```

```
case 3:
        rotation=2;
        inverse=0;
        solve[++count]=2;
        glutIdleFunc(spincube);
break;

case 4:
        rotation=2;
        inverse=1;
        solve[++count]=-2;
        glutIdleFunc(spincube);
break;
        case 5:
        rotation=3;
        inverse=0;
        solve[++count]=3;
        glutIdleFunc(spincube);
break;
        case 6:
        rotation=3;
        inverse=1;
        solve[++count]=-3;
        glutIdleFunc(spincube);
break;
```

```
            case 7:
            rotation=4;
            inverse=0;
            solve[++count]=4;
            glutIdleFunc(spincube);
    break;
            case 8:
            rotation=4;
            inverse=1;
            solve[++count]=-4;
            glutIdleFunc(spincube);
    break;
            case 9:
            rotation=5;
            inverse=0;
            solve[++count]=5;
            glutIdleFunc(spincube);
    break;
            case 10:
            rotation=5;
            inverse=1;
            solve[++count]=-5;
            glutIdleFunc(spincube);
    break;
            case 11:
                rotation=6;
                inverse=0;
                solve[++count]=6;
                glutIdleFunc(spincube);
        break;
                case 12:
                rotation=6;
                inverse=1;
                solve[++count]=-6;
                glutIdleFunc(spincube);
        break;


                        case 13:
                        exit(0);
                        break;
}
        }
}
int main(int argc, char** argv)
{
glutInit(&argc, argv);
```

```
glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);

glutInitWindowSize (500, 500);

glutCreateWindow ("RUBIK'S CUBE");

glutReshapeFunc (myreshape);

glutIdleFunc(spincube);

glutMouseFunc(mouse);

  glutMotionFunc(motion);

glutCreateMenu(mymenu);

glutAddMenuEntry("Top                 :a",1);

glutAddMenuEntry("Top Inverted    :q",2);

glutAddMenuEntry("Right               :s",3);

glutAddMenuEntry("Right Inverted  :w",4);

glutAddMenuEntry("Front               :d",5);

glutAddMenuEntry("Front Inverted  :e",6);

glutAddMenuEntry("Left                 :f",7);

glutAddMenuEntry("Left Inverted    :r",8);

glutAddMenuEntry("Back                :g",9);

glutAddMenuEntry("Back Inverted   :t",10);

glutAddMenuEntry("Bottom            :h",11);

glutAddMenuEntry("Bottom Inverted :y",12);


glutAddMenuEntry("Exit",13);

glutAttachMenu(GLUT_RIGHT_BUTTON);


glutKeyboardFunc(keyboard);
```

```
glutDisplayFunc (display);

glEnable(GL_DEPTH_TEST);

glutMainLoop();

//return 0;

}
```