```
#include<iostream>
#include<GL/glut.h>
#include<string.h>
#define HANOI_SOLVE 0
#define HANOI_ROTATE 1
#define HANOI_LIGHT 2
#define HANOI_QUIT 3
#define HANOI_FOG 4
#define HANOI_START 5
using namespace std;

/* CLASS DEFINITIONS*/

/* CLASS TO STORE THE MOVES OF THE SOLUTION */
class Solution
{
    public:
        int source;     //source peg
        int destination;//destination peg
};

/* CLASS FOR A CUBE */
class Cube
{
    public:
        float coord[24];                //X,Y,Z co-ordinate
        void initialize(float co[24]);  //function to initialize the co-
ordinates

};

void Cube::initialize(float co[24])
{
    for(int i=0;i<24;i++)
        coord[i]=co[i];
}

/* CLASS FOR PEGS */
class Peg
{
    public:
        float coord[24];            //X,Y,Z co-ordinates
        int stack[8];               //Stack to store the disks in the peg
        int top;                    //To store the index of the top most disk
        float toppixel;             //To store the top-most pixel
        float markpixel;            //To store the mid point of peg
        int gettop();               //Function to get the index of the top-
most disk
        void push(int i);           //Function to add a disk into a peg
        int pop();                  //Function to remove a disk from a peg
        void initialize(float co[]);//Function to initialize the peg co-
ordinates
        Peg(){top=-1;toppixel=-170.0;}              //Constructor to
```

```
initialize the peg
        void setmark(float mark);


};


/* PEG CLASS FUNCTION DEFINITIONS */
void Peg::setmark(float mark)
{
    markpixel=mark;
}
void Peg::initialize(float co[24])
{
    for(int i=0;i<24;i++)
        coord[i]=co[i];
}
int Peg::gettop()
{
    return stack[top];
}
void Peg::push(int i)
{
    stack[++top]=i;
    toppixel+=35.0;
}
int Peg::pop()
{
    toppixel-=35.0;
    return stack[top--];
}

int solutionlength=0;       //Contains the length of the solution
int destinationreached=0;   //To check if the disk has reached destination
int movetrack=0;            //To keep track of the moves
GLfloat mypegcolor[3]={0.0,0.0,1.0};        //Array to store colour of peg
GLfloat mycubecolor[4]={0.0,1.0,0.0};   //Array to store colour of cube
GLfloat mytitlecolor[3]={0.6481,0.8196,0.0314};//Array to store colour values
of title screen
GLfloat mymenucolor[]={1.0,0.6314,0.1524};//Array to store colour values of
menu screen
GLfloat lightOneDirection[] ={-200, 100, 1};
GLfloat lightOnePosition[] ={200, 100, 300, 1};
GLfloat lightOneColor[] ={1.0, 1.0, 0.5, 1.0};
GLfloat lightTwoDirection[] ={0, 0, 1};
GLfloat lightTwoPosition[] ={-300, 100, 300, 1};
GLfloat lightTwoColor[] ={1.0, 0.0, 0.3, 1.0};
GLfloat lightZeroPosition[] ={400, 200, 300, 1};
GLfloat lightZeroColor[] ={.3, .3, .3, .3};

Cube mycube[8];             //Creating 8 objects for 8 disks
Peg mypeg[3];               //Creating 3 objects for 3 pegs
Solution mysolution[100];   //Object to store the solution moves
```

```
bool up=true;               //Variable to control the motion of disk
bool myrotate=true;         //Variable to control the rotation
bool activesource=true;
bool won=false;
bool start=false;
bool mylight=false;
bool myfog=false;
bool fullscreen=false;

int mysource,mydest;
void *font = GLUT_BITMAP_TIMES_ROMAN_24;
void *font1 = GLUT_BITMAP_HELVETICA_18;
char defaultMessage[] = "TOWER OF HANOI";
char *message = defaultMessage;

GLfloat point1[][3]={{-100.0,180.0,10.0},{-90.0,180.0,10.0},
{-80.0,180.0,10.0}};
GLfloat point2[][3]={{100.0,135.0,10.0},{90.0,135.0,10.0},{80.0,135.0,10.0}};

GLfloat base[24]={
                    -225.0,-200.0,-100.0,
                    -225.0,-170.0,-100.0,
                    225.0,-170.0,-100.0,
                    225.0,-200.0,-100.0,
                    -225.0,-200.0,100.0,
                    -225.0,-170.0,100.0,
                    225.0,-170.0,100.0,
                    225.0,-200.0,100.0};
GLfloat peg1[24]={-120.0,-200.0,-10.0,
                    -120.0,150.0,-10.0,
                    -100.0,150.0,-10.0,
                    -100.0,-200.0,-10.0,
                    -120.0,-200.0,10.0,
                    -120.0,150.0,10.0,
                    -100.0,150.0,10.0,
                    -100.0,-200.0,10.0};
GLfloat peg2[24]={-10.0,-200.0,-10.0,
                    -10.0,150.0,-10.0,
                    10.0,150.0,-10.0,
                    10.0,-200.0,-10.0,
                    -10.0,-200.0,10.0,
                    -10.0,150.0,10.0,
                    10.0,150.0,10.0,
                    10.0,-200.0,10.0};
GLfloat peg3[24]={100.0,-200.0,-10.0,
                    100.0,150.0,-10.0,
                    120.0,150.0,-10.0,
                    120.0,-200.0,-10.0,
                    100.0,-200.0,10.0,
                    100.0,150.0,10.0,
                    120.0,150.0,10.0,
                    120.0,-200.0,10.0};
```

```
GLfloat cube1[24]={-160.0,-170.0,-55.0,
                   -160.0,-135.0,-55.0,
                   -50.0,-135.0,-55.0,
                   -50.0,-170.0,-55.0,
                   -160.0,-170.0,55.0,
                   -160.0,-135.0,55.0,
                   -50.0,-135.0,55.0,
                   -50.0,-170.0,55.0};
GLfloat cube2[]={-150.0,-135.0,-45.0,
                   -150.0,-100.0,-45.0,
                   -60.0,-100.0,-45.0,
                   -60.0,-135.0,-45.0,
                   -150.0,-135.0,45.0,
                   -150.0,-100.0,45.0,
                   -60.0,-100.0,45.0,
                   -60.0,-135.0,45.0};
GLfloat cube3[]={-140.0,-100.0,-35.0,
                   -140.0,-65.0,-35.0,
                   -70.0,-65.0,-35.0,
                   -70.0,-100.0,-35.0,
                   -140.0,-100.0,35.0,
                   -140.0,-65.0,35.0,
                   -70.0,-65.0,35.0,
                   -70.0,-100.0,35.0};
GLfloat cube4[]={-130.0,-65.0,-25.0,
                   -130.0,-30.0,-25.0,
                   -80.0,-30.0,-25.0,
                   -80.0,-65.0,-25.0,
                   -130.0,-65.0,25.0,
                   -130.0,-30.0,25.0,
                   -80.0,-30.0,25.0,
                   -80.0,-65.0,25.0};
void keyFunc(unsigned char,int,int);
void myinit();
void titledisplay();
void movepoints();
void restart(unsigned char,int,int);
void activekeyFunc(unsigned char,int,int);
void display();
void gameover();
void update();
void hanoimenu(int);
void idleFunction();
bool mycolour=true;
void special(GLint key,int x,int y)
{
    if (key == GLUT_KEY_F1)     // Toggle FullScreen
     {
        fullscreen = !fullscreen; // Toggle FullScreenBool
        if (fullscreen)
        {
            glutFullScreen();
            glutSetCursor(GLUT_CURSOR_NONE);// Enable Fullscreen
```

```
        }
        else
        {
            glutReshapeWindow(1350,700);
            glutPositionWindow(10,30);
            //glutSetCursor(GLUT_CURSOR_BOTTOM_LEFT_CORNER);
        }
    }
    if(key==GLUT_KEY_F2)//exit
    {
        glutDestroyWindow(1);
        exit(0);
    }
    if(key==GLUT_KEY_F3)//solution
    {
        myinit();
        glutDisplayFunc(display);
        glutIdleFunc(update);
        glutKeyboardFunc(activekeyFunc);
        glutMouseFunc(NULL);
    }
    if(key==GLUT_KEY_F4)//quit
    {
        glutIdleFunc(movepoints);
            glutKeyboardFunc(restart);
            glutDisplayFunc(gameover);
            myinit();
    }
    if(key==GLUT_KEY_F5)
        if(mycolour)mycolour=false;
        else mycolour=true;
        glutPostRedisplay();
}

void hanoimenu(int key)
{
    switch(key)
    {
    case HANOI_START:
        if(!start)
        {
            glutIdleFunc(idleFunction);
            start=true;
        }
        else
        {
            glutIdleFunc(update);
            start=false;
        }
        break;
    case HANOI_SOLVE:

        myinit();
```

```
        glutDisplayFunc(display);
        glutIdleFunc(update);
        glutKeyboardFunc(activekeyFunc);
        glutMouseFunc(NULL);
            break;
    case HANOI_LIGHT:
        if(mylight)
            {
                glDisable(GL_LIGHTING);
                glDisable(GL_LIGHT0);
                mylight=false;
        }
        else
        {
         glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0);
        glEnable(GL_LIGHT1);
        glEnable(GL_LIGHT2);
         mylight=true;
        }
        break;
    case HANOI_QUIT:
        glutIdleFunc(movepoints);
            glutKeyboardFunc(restart);
            glutDisplayFunc(gameover);
            myinit();
        break;
    case HANOI_ROTATE:
        if(myrotate)    myrotate=false;
        else    myrotate=true;
        break;
    case HANOI_FOG:
            if (glIsEnabled(GL_FOG))
      glDisable(GL_FOG);
    else {
      glEnable(GL_FOG);
      glFogi(GL_FOG_MODE, GL_EXP);
      glFogf(GL_FOG_DENSITY, 0.01);
    }

        break;
    }
}

/*Function to move the cube*/
void movecube(int sourcepeg,int destinationpeg)
{

    int cubeindex=mypeg[sourcepeg].gettop();//Index of cube to be moved
    int flag=0;                             //Flag to make sure only one move
for one call
    float midpoint=(mycube[cubeindex].coord[0]+mycube[cubeindex].coord
[9])/2.0;
```

```
    float destpointX=mypeg[destinationpeg].markpixel;//X co-ordinate of the
destination
    float destpointY=mypeg[destinationpeg].toppixel;//y co-ordinate of the
destination
    //To move the desired cube in upward direction
    if(mycube[cubeindex].coord[4]<200.0 && up==true)
    {
        for(int i=1;i<24;i+=3)
            mycube[cubeindex].coord[i]+= 0.5;
        flag=1;
    }
    //To move the desired cube towards right
    if(midpoint < destpointX && flag==0)
    {
        for(int i=0;i<24;i+=3)
            mycube[cubeindex].coord[i] += 0.5;
        midpoint+=0.5;
        flag=1;
    }
    //To move the desired cube towards left
    if(midpoint > destpointX && flag==0)
    {
        for(int i=0;i<24;i+=3)
            mycube[cubeindex].coord[i] -=0.5;
        midpoint-=.05;
        flag=1;
    }
    //To move the desired cube in downward direction
    if(mycube[cubeindex].coord[1] > destpointY && flag==0)
    {
        up=false;
        for(int i=1;i<24;i+=3)
            mycube[cubeindex].coord[i]-=0.5;
        flag=1;
        if(mycube[cubeindex].coord[1] <= destpointY)
            destinationreached=1;
    }
    glutPostRedisplay();
}
void idleFunction()
{
    int i,j,z;
    if(movetrack<solutionlength)
    {
        if(destinationreached==1)
        {
            i=mysolution[movetrack].source;
            j=mysolution[movetrack].destination;
            z=mypeg[i].pop();
            mypeg[j].push(z);
            destinationreached=0;
            movetrack++;
```

```
            up=1;
        }
        if(movetrack>=solutionlength)
        {
        glutIdleFunc(movepoints);
            glutKeyboardFunc(restart);
            glutDisplayFunc(gameover);
        myinit();
        return;}
        movecube(mysolution[movetrack].source,mysolution
[movetrack].destination);
    }
    glutPostRedisplay();
}
void polygon(int a,int b,int c,int d,float draw[])
{
    glBegin(GL_POLYGON);
        glTexCoord2f(0.0,0.0);
        glVertex3f(draw[a*3],draw[a*3+1],draw[a*3+2]);
        glTexCoord2f(0.0,1.0);
        glVertex3f(draw[b*3],draw[b*3+1],draw[b*3+2]);
        glTexCoord2f(1.0,1.0);
        glVertex3f(draw[c*3],draw[c*3+1],draw[c*3+2]);
        glTexCoord2f(1.0,0.0);
        glVertex3f(draw[d*3],draw[d*3+1],draw[d*3+2]);
    glEnd();
}
void outline(int a,int b,int c,int d,float draw[])
{
    glLineWidth(1.0);
    glColor3f(1.0,1.0,1.0);
    glBegin(GL_LINE_LOOP);
        glVertex3f(draw[a*3],draw[a*3+1],draw[a*3+2]);
        glVertex3f(draw[b*3],draw[b*3+1],draw[b*3+2]);
        glVertex3f(draw[c*3],draw[c*3+1],draw[c*3+2]);
        glVertex3f(draw[d*3],draw[d*3+1],draw[d*3+2]);
    glEnd();
}
void colorcube(float draw[])
{
    polygon(0,3,2,1,draw);
    polygon(4,5,6,7,draw);
    polygon(2,3,7,6,draw);
    polygon(0,1,5,4,draw);
    polygon(1,2,6,5,draw);
    polygon(0,4,7,3,draw);
    /*outline(0,3,2,1,draw);
    outline(4,5,6,7,draw);
    outline(2,3,7,6,draw);
    outline(1,5,4,0,draw);
    outline(1,2,6,5,draw);
    outline(0,4,7,3,draw);*/
}
```

```
void display()
{
    if(mycolour)
        glClearColor(0.0,0.0,0.0,1.0);
    else
        glClearColor(1.0,1.0,1.0,1.0);

    glEnable(GL_TEXTURE_2D);
    static float i=0,flag=0;

    if(myrotate)
    {
        if(i<60.0 && flag==0)
            i+=0.05;
        else
        {
            flag=1;
            i-=0.05;
            if(i<-60.0) flag=0;
        }
    }
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glRotatef(10.0,1.0,0.0,0.0);
    glRotatef(i,0.0,1.0,0.0);
    glColor3fv(mypegcolor);
    colorcube(base);
    glColor3fv(mypegcolor);
    colorcube(peg1);
    glColor3fv(mypegcolor);
    colorcube(peg2);
    glColor3fv(mypegcolor);
    colorcube(peg3);
    glColor4fv(mycubecolor);
    colorcube(mycube[0].coord);
    glColor4fv(mycubecolor);
    colorcube(mycube[1].coord);
    glColor4fv(mycubecolor);
    colorcube(mycube[2].coord);
    glColor4fv(mycubecolor);
    colorcube(mycube[3].coord);
    glFlush();
    glutSwapBuffers();
}
void movepoints()
{
    static bool right=true;
    if(right)
    {
        point1[0][0]+=0.2;
        point1[1][0]+=0.2;
        point1[2][0]+=0.2;
        point2[0][0]-=0.2;
```

```
        point2[1][0]-=0.2;
        point2[2][0]-=0.2;
        if(point1[2][0]>100.0) right=false;
    }
    else
    {
        point1[0][0]-=0.2;
        point1[1][0]-=0.2;
        point1[2][0]-=0.2;
        point2[0][0]+=0.2;
        point2[1][0]+=0.2;
        point2[2][0]+=0.2;
    }
    if(point1[0][0]<-100.0) right=true;
    glutPostRedisplay();

}
void output(int x, int y,int z, char *string,int in=0)
{
  int len, i;
  glRasterPos3f(x,y,z);

  len = (int) strlen(string);
  if(in==1)
  {for (i = 0; i < len; i++)
    glutBitmapCharacter(font, string[i]);
  return;}

      for (i = 0; i < len; i++) {
          glutBitmapCharacter(font1, string[i]);
      };
}

void titledisplay()
{
    glEnable(GL_LIGHT0);
    glDisable(GL_LIGHTING);
    glDisable(GL_FOG);
    if(mycolour)
        glClearColor(0.6481,0.8196,0.0314,1.0);
    else
        glClearColor(0.0,0.478,0.804,1.0);
    glPointSize(5.0);

    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glColor3fv(mytitlecolor);
    glDisable(GL_TEXTURE_2D);
    /*glBegin(GL_POLYGON);
        glVertex3f(-249.0,-249.0,0.0);
        glVertex3f(-249.0,249.0,0.0);
        glVertex3f(249.0,249.0,0.0);
        glVertex3f(249.0,-249.0,0.0);
```

```
    glEnd();*/
    glColor3f(1.0,1.0,1.0);
    glBegin(GL_POINTS);
        glVertex3fv(point1[0]);
        glVertex3fv(point1[1]);
        glVertex3fv(point1[2]);
        glVertex3fv(point2[0]);
        glVertex3fv(point2[1]);
        glVertex3fv(point2[2]);
    glEnd();
    glColor3f(1.0,1.0,1.0);
    output(-60.0,230.0,100.0, "PROJECT TITLE");
    output(-80.0, 150.0,100.0, message,1);
    output(-65.0, 0.0,100, "PROJECT BY");
    output(-130.0,-25,100,"  BABLU KUMAR  -  1CE14CS019");

    output(-120.0,-170,100.0,"PROJECT GUIDE :  MANJULA M");
    output(-90.0,-195,100.0,"DEPARTMENT OF CSE");
    output(-120.0,-220,100.0,   "CITY ENGINEERING COLLEGE ");
    output(-145.0,-90.0,100.0,"PRESS ANY KEY TO VIEW THE PROJECT");
    glutIdleFunc(movepoints);
    glFlush();
    glutSwapBuffers();
}

void activekeyFunc(unsigned char key,int x,int y)
{
    if(key=='r' || key=='R')
        if(myrotate)    myrotate=false;
        else    myrotate=true;
    if(key=='s' || key=='S')
        if(!start)
        {
            glutIdleFunc(idleFunction);
            start=true;
        }
        else
        {
            glutIdleFunc(update);
            start=false;
        }
        if(key=='l' || key=='L')

            if(mylight)
            {
                glDisable(GL_LIGHTING);
                glDisable(GL_LIGHT0);
                mylight=false;
        }
        else
        {
         glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0);
```

```
        glEnable(GL_LIGHT1);
        glEnable(GL_LIGHT2);
         mylight=true;
        }

        if(key=='f' || key=='F')
          if (glIsEnabled(GL_FOG))
      glDisable(GL_FOG);
    else {
      glEnable(GL_FOG);
      glFogi(GL_FOG_MODE, GL_EXP);
      glFogf(GL_FOG_DENSITY, .01);
    }
    if(key=='q' || key=='Q')
    {
        glutIdleFunc(movepoints);
            glutKeyboardFunc(restart);
            glutDisplayFunc(gameover);
            myinit();
    }


}
void menudisplay()
{

    glEnable(GL_LIGHT0);
    glDisable(GL_LIGHTING);
    glDisable(GL_FOG);
    if(mycolour)
        glClearColor(1.0,0.6314,0.1524,1.0);
    else
        glClearColor(0.0,0.3686,0.8476,1.0);
    glPointSize(5.0);
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glColor3fv(mymenucolor);
    glDisable(GL_TEXTURE_2D);
    /*glBegin(GL_POLYGON);
        glVertex3f(-249.0,-249.0,0.0);
        glVertex3f(-249.0,249.0,0.0);
        glVertex3f(249.0,249.0,0.0);
        glVertex3f(249.0,-249.0,0.0);
    glEnd();*/
    glColor3f(1.0,1.0,1.0);
    glBegin(GL_POINTS);
        glVertex3fv(point1[0]);
        glVertex3fv(point1[1]);
        glVertex3fv(point1[2]);
        glVertex3fv(point2[0]);
        glVertex3fv(point2[1]);
        glVertex3fv(point2[2]);
    glEnd();
```

```
    glColor3f(1.0,1.0,1.0);
    output(-120.0, 150.0,100.0, "SELECT AN OPTION FROM MENU",1);
    output(-205.0, 0.0,100, "PRESS 'S' OR 's' TO SEE THE SOLUTION OF TOWER OF
HANOI");
    output(-205.0, -40.0,100, "[INSTRUCTIONS: Press 'S' or 's' to start/pause,
press R or r to rotate]");
    output(-205.0,-100,100,"PRESS 'P' OR 'p' TO SOLVE THE TOWER OF HANOI
PUZZLE");
    output(-205.0, -130.0,100, "[INSTRUCTIONS: Use mouse to drag and drop]");
    output(-220.0,-235.0,100.0,"F1-Full Screen        F2-Exit        F3-
Solve          F4-Quit       F5-Change Background");
    glutIdleFunc(movepoints);
    glFlush();
    glutSwapBuffers();
}
void update()
{
    glutPostRedisplay();
}
void gameover()
{

    glEnable(GL_LIGHT0);
    glDisable(GL_LIGHTING);
    glDisable(GL_FOG);

    if(mycolour)
        glClearColor(0.09412,0.0,0.3255,1.0);
    else
        glClearColor(0.9059,0.0,0.6745,1.0);
    glPointSize(5.0);
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glColor3f(0.09412,0.0,0.3255);
    glDisable(GL_TEXTURE_2D);
    /*glBegin(GL_POLYGON);
        glVertex3f(-249.0,-249.0,0.0);
        glVertex3f(-249.0,249.0,0.0);
        glVertex3f(249.0,249.0,0.0);
        glVertex3f(249.0,-249.0,0.0);
    glEnd();*/
    glColor3f(1.0,1.0,1.0);
    glBegin(GL_POINTS);
        glVertex3fv(point1[0]);
        glVertex3fv(point1[1]);
        glVertex3fv(point1[2]);
        glVertex3fv(point2[0]);
        glVertex3fv(point2[1]);
        glVertex3fv(point2[2]);
    glEnd();
    glColor3f(1.0,1.0,1.0);
    if(won==false)
        output(-60.0, 150.0,100.0, "GAME OVER",1);
```

```
    else
        output(-50.0, 150.0,100.0, "YOU WON",1);
    output(-205.0, 0.0,100, "PRESS 'R' OR 'r' TO RESTART THE PROJECT");
    output(-205.0, -25.0,100, "PRESS 'Q' OR 'q TO QUIT THE PROJECT");
    glutIdleFunc(movepoints);
    glFlush();
    glutSwapBuffers();
}
void keyFunc(unsigned char,int,int);
void myinit();
void restart(unsigned char key,int x,int y)
{
    if(key=='q' || key=='Q') exit(0);
    else if(key=='r' || key=='R')
    {
        glutDisplayFunc(titledisplay);
        glutKeyboardFunc(keyFunc);
        glutIdleFunc(movepoints);
        myinit();
    }
}
void movedisk(int sourcepeg,int destinationpeg)
{

    if(mypeg[sourcepeg].top==-1)
        {

            return;
        }
    int cubeindex=mypeg[sourcepeg].gettop();//Index of cube to be moved
    if(mypeg[destinationpeg].top >= 0)
        if(cubeindex < mypeg[destinationpeg].stack[mypeg[destinationpeg].top])
        {
            won=false;
            glutIdleFunc(movepoints);
            glutKeyboardFunc(restart);
            glutDisplayFunc(gameover);
            myinit();
        }


    float midpoint=(mycube[cubeindex].coord[0]+mycube[cubeindex].coord
[9])/2.0;

    float destpointX=mypeg[destinationpeg].markpixel;//X co-ordinate of the
destination
    float destpointY=mypeg[destinationpeg].toppixel;//y co-ordinate of the
destination
    //To move the desired cube in upward direction

    //To move the desired cube towards right
    while(midpoint < destpointX)
    {
```

```
    for(int i=0;i<24;i+=3)
        mycube[cubeindex].coord[i] += 0.5;
    midpoint+=0.5;
    glFlush();
}
//To move the desired cube towards left
while(midpoint > destpointX)
{
    for(int i=0;i<24;i+=3)
        mycube[cubeindex].coord[i] -=0.5;
    midpoint-=.5;
    glFlush();
}
//To move the desired cube in downward direction
while(mycube[cubeindex].coord[1] > destpointY)
{
    up=false;
    for(int i=1;i<24;i+=3)
        mycube[cubeindex].coord[i]-=0.5;

    if(mycube[cubeindex].coord[1] <= destpointY)
        destinationreached=1;
    glFlush();
}
while(mycube[cubeindex].coord[1] < destpointY)
{
    up=false;
    for(int i=1;i<24;i+=3)
        mycube[cubeindex].coord[i]+=0.5;

    if(mycube[cubeindex].coord[1] >= destpointY)
        destinationreached=1;
    glFlush();
}
int i,j,z,count=0;


i=sourcepeg;
j=destinationpeg;
z=mypeg[i].pop();
mypeg[j].push(z);



for(i=0;i<4;i++)
    if(i==mypeg[2].stack[i])
        count++;
if(count==4)
{
    won=true;
    glutIdleFunc(movepoints);
        glutKeyboardFunc(restart);
        glutDisplayFunc(gameover);
```

```
            myinit();
    }
    glutPostRedisplay();
}

void playkey(unsigned char key,int x,int y)
{

    if(key=='r' || key=='R')
    {
            if(myrotate)    myrotate=false;
        else    myrotate=true;
        return;
    }
    if(key=='l' || key=='L')
    {
            if(mylight)
            {
                glDisable(GL_LIGHTING);
                glDisable(GL_LIGHT0);
                mylight=false;
        }
        else
        {
         glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0);
        glEnable(GL_LIGHT1);
        glEnable(GL_LIGHT2);
         mylight=true;
        }
            return;
    }
        if(key=='f' || key=='F')
          {if (glIsEnabled(GL_FOG))
      glDisable(GL_FOG);
    else {
      glEnable(GL_FOG);
      glFogi(GL_FOG_MODE, GL_EXP);
      glFogf(GL_FOG_DENSITY, .01);
    }return;}
        if(activesource)
        {
            switch(key)
            {
            case 'a':case 'A':
                mysource=0;
                break;
            case 'b':case 'B':
                mysource=1;
                break;
            case 'c':case 'C':
                mysource=2;
            }
```

```
            activesource=false;
        }
        else
        {
            activesource=true;
            switch(key)
            {
            case 'a':case 'A':
                mydest=0;
                break;
            case 'b':case 'B':
                mydest=1;
                break;
            case 'c':case 'C':
                mydest=2;
            }
            if(mysource==mydest) return;
            movedisk(mysource,mydest);
        }

    if(key=='q' || key=='Q')
    {
        glutIdleFunc(movepoints);
            glutKeyboardFunc(restart);
            glutDisplayFunc(gameover);
            myinit();
    }
}
void playmouse(int btn,int state,int x,int y)
{
    //if(state==GLUT_DOWN)
    //{
    if(btn==GLUT_RIGHT_BUTTON)return;
        if(activesource)
        {
            if(x>370 && x<590) mysource=0;
            if(x>591 && x<750) mysource=1;
            if(x>751 && x<985) mysource=2;
            activesource=false;
        }
        else
        {
            if(x>370 && x<590) mydest=0;
            if(x>591 && x<750) mydest=1;
            if(x>751 && x<985) mydest=2;
            activesource=true;
            if(mysource==mydest) return;
            movedisk(mysource,mydest);
        }
    //}
}
void menukeyfunc(unsigned char key,int x,int y)
{
```

```
    if(key=='s' || key=='S')
    {
        myinit();
        glutDisplayFunc(display);
        glutIdleFunc(update);
        glutKeyboardFunc(activekeyFunc);
        glutMouseFunc(NULL);
    }
    if(key=='p' || key=='P')
    {
        glutSetCursor(GLUT_CURSOR_INFO);
        myrotate=false;
        glutDisplayFunc(display);
        glutIdleFunc(update);
        glutKeyboardFunc(playkey);
        glutMouseFunc(playmouse);
    }
}

void keyFunc(unsigned char key,int x,int y)
{

//  if(key=='e' || key=='E')
    //{
        glutDisplayFunc(menudisplay);
        glutIdleFunc(movepoints);
        glutKeyboardFunc(menukeyfunc);
    //}

}

void myreshape(int w,int h)
{
    glViewport(0,0,w,h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if(w<=h)
        glOrtho(-250.0,250.0,-250.0*(GLfloat)h/(GLfloat)w,
250*(GLfloat)h/(GLfloat)w,-350.0,350.0);
    else
        glOrtho(-250.0*(GLfloat)w/(GLfloat)h,
250.0*(GLfloat)w/(GLfloat)h, -250.0,250.0,-350.0,350.0);
    glMatrixMode(GL_MODELVIEW);
}
void myinit()
{
    mypeg[0].initialize(peg1);
    mypeg[1].initialize(peg2);
    mypeg[2].initialize(peg3);
    mypeg[0].setmark(-110.0);
    mypeg[1].setmark(0.0);
    mypeg[2].setmark(110.0);
    mycube[0].initialize(cube1);
```

```
    mycube[1].initialize(cube2);
    mycube[2].initialize(cube3);
    mycube[3].initialize(cube4);
    mypeg[0].stack[0]=0;
    mypeg[0].stack[1]=1;
    mypeg[0].stack[2]=2;
    mypeg[0].stack[3]=3;
    mypeg[0].top=3;
    mypeg[0].toppixel=-170.0+35.0*4.0;
    glClearColor(1.0,1.0,1.0,1.0);

        movetrack=0;
        destinationreached=0;
        mypeg[1].toppixel=-170.0;
        mypeg[2].toppixel=-170.0;
        mypeg[2].stack[0]=0;
        mypeg[2].stack[1]=0;
        mypeg[2].stack[2]=0;
        mypeg[2].stack[3]=0;
        mypeg[1].top=-1;
        mypeg[2].top=-1;

        //glutSetCursor(GLUT_CURSOR_NONE);
        glutSetCursor(GLUT_CURSOR_LEFT_ARROW);
        up=true;
        activesource=true;
}

/* FUNCTION TO GET THE MOVES */

void towers(int num, int frompeg, int topeg, int auxpeg)
{
    if (num == 1)
    {
        mysolution[solutionlength].source=frompeg;
        mysolution[solutionlength++].destination=topeg;
        return;
    }
    towers(num - 1, frompeg, auxpeg, topeg);
    mysolution[solutionlength].source=frompeg;
    mysolution[solutionlength++].destination=topeg;
    towers(num - 1, auxpeg, topeg, frompeg);
}
int main(int argc,char **argv)
{
    GLubyte image[64][64][3];
    int i,j,c;
    for(i=0;i<64;i++)
        for(j=0;j<64;j++)
        {
            c=((((i & 0x0)==0)^(j & 0x8)==0))*255;
            image[i][j][0]=(GLubyte)c;
            image[i][j][1]=(GLubyte)c;
```

```
            image[i][j][2]=(GLubyte)c;
        }

    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
    glutInitWindowSize(1350,700);
    glutCreateWindow("hanoi");
    glutReshapeFunc(myreshape);
    glutDisplayFunc(titledisplay);
    glutKeyboardFunc(keyFunc);
    glutIdleFunc(movepoints);
    //glutIdleFunc(idleFunction);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_TEXTURE_2D);
    glTexImage2D(GL_TEXTURE_2D,0,3,64,64,0,GL_RGB,GL_UNSIGNED_BYTE,image);
    glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_REPEAT);
    glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_REPEAT);
    glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
    glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
      glutCreateMenu(hanoimenu);
  glutAddMenuEntry("Solve", HANOI_SOLVE);
      glutAddMenuEntry("(S)tart/Stop", HANOI_START);

  glutAddMenuEntry("(R)otate On/Off", HANOI_ROTATE);
  glutAddMenuEntry("(L)ight On/Off", HANOI_LIGHT);
  glutAddMenuEntry("(F)og", HANOI_FOG);
  glutAddMenuEntry("(Q)uit", HANOI_QUIT);
  glutAttachMenu(GLUT_RIGHT_BUTTON);
  glLightfv(GL_LIGHT1, GL_POSITION, lightOnePosition);
  glLightfv(GL_LIGHT1, GL_DIFFUSE, lightOneColor);
  glLightf(GL_LIGHT1, GL_SPOT_CUTOFF, 10);
  glLightfv(GL_LIGHT1, GL_SPOT_DIRECTION, lightOneDirection);
  glEnable(GL_LIGHT1);

  glLightfv(GL_LIGHT2, GL_POSITION, lightTwoPosition);
  glLightfv(GL_LIGHT2, GL_DIFFUSE, lightTwoColor);
  glLightf(GL_LIGHT2,GL_LINEAR_ATTENUATION,.005);
  glLightf(GL_LIGHT2, GL_SPOT_CUTOFF, 10);
  glLightfv(GL_LIGHT2, GL_SPOT_DIRECTION, lightTwoDirection);
  glEnable(GL_LIGHT2);

  glLightfv(GL_LIGHT0, GL_DIFFUSE, lightZeroColor);
  glEnable(GL_LIGHT0);

  glEnable(GL_LIGHTING);

  glutSpecialFunc(special);
    towers(4,0,2,1);
    myinit();
    glutFullScreen();
    glutMainLoop();
}
```