**HƯỚNG DẪN THI ONLINE MÔN ĐỒ HỌA MÁY TÍNH**

**Cập nhật ngày 03/08/2021**

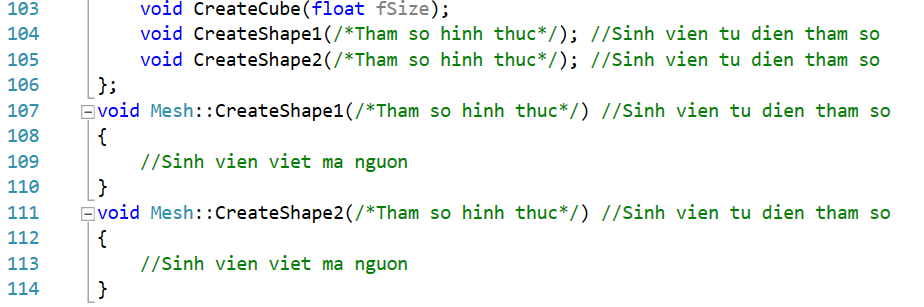
*Lưu ý: Hướng dẫn này chỉ áp dụng cho môn ĐHMT, các môn khác có thể quy định khác.*

* ***Sinh viên không cần ghi hình quá trình làm bài***
* ***Được sử dụng tài liệu giấy, Google.***
* ***KHÔNG cần phải chép mã nguồn ra giấy***
* ***Phòng thi ở trang exam.hcmut.edu.vn***

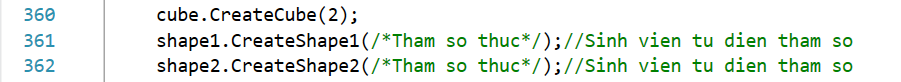
Sinh viên được cung cấp mã nguồn ban đầu (trong phần phụ lục của tập tin này), sau đó dựa trên đề bài để viết thêm vào những vị trí trong mã nguồn ban đầu. Sinh viên tạo project và thêm mã nguồn này vào project.

Cụ thể:

* Điền tham số hình thức, viết mã nguồn cho hàm *CreateShape1(…)* và hàm *CreateShape2(…)*



* Điền tham số thực cho lời gọi hàm *shape1.CreateShape1(…)* và *shape2.CreateShape2(…)*



**NỘP BÀI**

Sinh viên chuẩn bị các tập tin sau:

* + Mã nguồn đầy đủ của toàn bộ chương trình: Tên file là **ThiOnlineDHMT\_MSSV.cpp** (trong đó MSSV là mã số sinh viên). Lưu ý: với mã nguồn, chỉ nộp duy nhất một file này.
  + Chụp kết quả chạy của chương trình ScreenShot (2 hình ảnh: 1 hình ảnh cho Shape1 + 1 hình ảnh cho Shape2).
  + Sinh viên có thể nén các file trên rồi nộp 01 file duy nhất cũng được

Sinh viên sẽ nộp bài ở 2 địa chỉ sau:

* Địa chỉ 1: Được quy định ở trang web phòng thi trên exam.hcmut.edu.vn
* Địa chỉ 2: Nộp vào địa chỉ e-mail: [dhmt.bku@gmail.com](mailto:dhmt.bku@gmail.com) (đồng thời cc tới địa chỉ: [tgson@hcmut.edu.vn](mailto:tgson@hcmut.edu.vn)) .

**XỬ LÝ GIAN LẬN**

Sinh viên sẽ bị coi là gian lận nếu:

* + Có sự giống nhau bất thường giữa mã nguồn của các bài nộp. Trong trường hợp này, **TẤT CẢ** các bài nộp đều bị coi là gian lận. Do vậy sinh viên phải bảo vệ mã nguồn củamình.
  + **Sinh viên không được copy mã nguồn từ bất cứ nguồn nào.**
  + **Trong trường hợp bị phát hiện gian lận, sinh viên sẽ nhận điểm 0 cho điểm thi Cuối kỳ.**

**PHỤ LỤC: MÃ NGUỒN CỦA ThiOnlineDHMT.cpp**

// ThiOnlineDHMT.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include <Windows.h>

#include <math.h>

#include <iostream>

#include <glut.h>

using namespace std;

/////////////////////////////////////////////////////////////////////

#define PI 3.1415926

#define COLORNUM 14

float ColorArr[COLORNUM][3] = { { 1.0, 0.0, 0.0 },{ 0.0, 1.0, 0.0 },{ 0.0, 0.0, 1.0 },

{ 1.0, 1.0, 0.0 },{ 1.0, 0.0, 1.0 },{ 0.0, 1.0, 1.0 },

{ 0.3, 0.3, 0.3 },{ 0.5, 0.5, 0.5 },{ 0.9, 0.9, 0.9 },

{ 1.0, 0.5, 0.5 },{ 0.5, 1.0, 0.5 },{ 0.5, 0.5, 1.0 },

{ 0.0, 0.0, 0.0 },{ 0.7, 0.7, 0.7 } };

class Point3

{

public:

float x, y, z;

void set(float dx, float dy, float dz)

{

x = dx; y = dy; z = dz;

}

void set(Point3& p)

{

x = p.x; y = p.y; z = p.z;

}

Point3() { x = y = z = 0; }

Point3(float dx, float dy, float dz)

{

x = dx; y = dy; z = dz;

}

};

class VertexID

{

public:

int vertIndex;

int colorIndex;

};

class Face

{

public:

int nVerts;

VertexID\* vert;

Face()

{

nVerts = 0;

vert = NULL;

}

~Face()

{

if (vert != NULL)

{

delete[] vert;

vert = NULL;

}

nVerts = 0;

}

};

class Mesh

{

public:

int numVerts;

Point3\* pt;

int numFaces;

Face\* face;

public:

Mesh()

{

numVerts = 0;

pt = NULL;

numFaces = 0;

face = NULL;

}

~Mesh()

{

if (pt != NULL)

{

delete[] pt;

}

if (face != NULL)

{

delete[] face;

}

numVerts = 0;

numFaces = 0;

}

void DrawWireframe();

void DrawColor();

void DrawPoint();

void CreateCube(float fSize);

void CreateShape1(/\*Tham so hinh thuc\*/); //Sinh vien tu dien tham so

void CreateShape2(/\*Tham so hinh thuc\*/); //Sinh vien tu dien tham so

};

void Mesh::CreateShape1(/\*Tham so hinh thuc\*/) //Sinh vien tu dien tham so

{

//Sinh vien viet ma nguon

}

void Mesh::CreateShape2(/\*Tham so hinh thuc\*/) //Sinh vien tu dien tham so

{

//Sinh vien viet ma nguon

}

void Mesh::CreateCube(float fSize)

{

int i;

numVerts = 8;

pt = new Point3[numVerts];

pt[0].set(-fSize, fSize, fSize);

pt[1].set(fSize, fSize, fSize);

pt[2].set(fSize, fSize, -fSize);

pt[3].set(-fSize, fSize, -fSize);

pt[4].set(-fSize, -fSize, fSize);

pt[5].set(fSize, -fSize, fSize);

pt[6].set(fSize, -fSize, -fSize);

pt[7].set(-fSize, -fSize, -fSize);

numFaces = 6;

face = new Face[numFaces];

//Left face

face[0].nVerts = 4;

face[0].vert = new VertexID[face[0].nVerts];

face[0].vert[0].vertIndex = 1;

face[0].vert[1].vertIndex = 5;

face[0].vert[2].vertIndex = 6;

face[0].vert[3].vertIndex = 2;

//Right face

face[1].nVerts = 4;

face[1].vert = new VertexID[face[1].nVerts];

face[1].vert[0].vertIndex = 0;

face[1].vert[1].vertIndex = 3;

face[1].vert[2].vertIndex = 7;

face[1].vert[3].vertIndex = 4;

//top face

face[2].nVerts = 4;

face[2].vert = new VertexID[face[2].nVerts];

face[2].vert[0].vertIndex = 0;

face[2].vert[1].vertIndex = 1;

face[2].vert[2].vertIndex = 2;

face[2].vert[3].vertIndex = 3;

//bottom face

face[3].nVerts = 4;

face[3].vert = new VertexID[face[3].nVerts];

face[3].vert[0].vertIndex = 7;

face[3].vert[1].vertIndex = 6;

face[3].vert[2].vertIndex = 5;

face[3].vert[3].vertIndex = 4;

//near face

face[4].nVerts = 4;

face[4].vert = new VertexID[face[4].nVerts];

face[4].vert[0].vertIndex = 4;

face[4].vert[1].vertIndex = 5;

face[4].vert[2].vertIndex = 1;

face[4].vert[3].vertIndex = 0;

//Far face

face[5].nVerts = 4;

face[5].vert = new VertexID[face[5].nVerts];

face[5].vert[0].vertIndex = 3;

face[5].vert[1].vertIndex = 2;

face[5].vert[2].vertIndex = 6;

face[5].vert[3].vertIndex = 7;

}

void Mesh::DrawWireframe()

{

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE);

for (int f = 0; f < numFaces; f++)

{

glBegin(GL\_POLYGON);

for (int v = 0; v < face[f].nVerts; v++)

{

int iv = face[f].vert[v].vertIndex;

glVertex3f(pt[iv].x, pt[iv].y, pt[iv].z);

}

glEnd();

}

}

void Mesh::DrawColor()

{

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL);

for (int f = 0; f < numFaces; f++)

{

glBegin(GL\_POLYGON);

for (int v = 0; v < face[f].nVerts; v++)

{

int iv = face[f].vert[v].vertIndex;

int ic = face[f].vert[v].colorIndex;

ic = f % COLORNUM;

glColor3f(ColorArr[ic][0], ColorArr[ic][1], ColorArr[ic][2]);

glVertex3f(pt[iv].x, pt[iv].y, pt[iv].z);

}

glEnd();

}

}

void Mesh::DrawPoint()

{

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_POINT);

glPointSize(3);

glColor3f(0, 0, 0);

for (int f = 0; f < numVerts; f++)

{

glBegin(GL\_POINTS);

glVertex3f(pt[f].x, pt[f].y, pt[f].z);

glEnd();

}

}

//////////////////////////////////////////////////////////////////////

int screenWidth = 1000;

int screenHeight = 500;

float angle = 0;

int nChoice = 0;

Mesh cube;

Mesh shape1;

Mesh shape2;

void drawAxis()

{

glColor3f(0, 0, 1);

glBegin(GL\_LINES);

glColor3f(1, 0, 0);

glVertex3f(0, 0, 0);//x - red

glVertex3f(4, 0, 0);

glColor3f(0, 1, 0);

glVertex3f(0, 0, 0);//y - green

glVertex3f(0, 4, 0);

glColor3f(0, 0, 1);

glVertex3f(0, 0, 0);//z - blue

glVertex3f(0, 0, 4);

glEnd();

}

void myDisplay()

{

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

gluLookAt(4.5, 4, 4, 0, 0, 0, 0, 1, 0);

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

////////////////////////////////////////////////////

glViewport(0, 0, screenWidth / 2, screenHeight);

glPushMatrix();

drawAxis();

glRotatef(angle, 0, 1, 0);

glColor3f(0, 0, 0);

if (nChoice == 0)

cube.DrawWireframe();

else if (nChoice == 1)

shape1.DrawWireframe();

else if (nChoice == 2)

shape2.DrawWireframe();

/////////////////////////////////////////////////////////////

glViewport(screenWidth / 2, 0, screenWidth / 2, screenHeight);

glPopMatrix();

drawAxis();

glRotatef(angle, 0, 1, 0);

if (nChoice == 0)

cube.DrawColor();

else if (nChoice == 1)

shape1.DrawColor();

else if (nChoice == 2)

shape2.DrawColor();

glFlush();

glutSwapBuffers();

}

void mySpecialFunc(int key, int x, int y)

{

if (key == GLUT\_KEY\_LEFT)

angle = angle + 5;

else if (key == GLUT\_KEY\_RIGHT)

angle = angle - 5;

glutPostRedisplay();

}

void myKeyboard(unsigned char key, int x, int y)

{

switch (key)

{

case '0':

nChoice = 0;

break;

case '1':

nChoice = 1;

break;

case '2':

nChoice = 2;

break;

}

glutPostRedisplay();

}

void myInit()

{

float fHalfSize = 5;

glClearColor(1.0f, 1.0f, 1.0f, 1.0f);

glFrontFace(GL\_CCW);

glEnable(GL\_DEPTH\_TEST);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-fHalfSize, fHalfSize, -fHalfSize, fHalfSize, -1000, 1000);

}

int main(int argc, \_TCHAR\* argv[])

{

cout << "Press -> or <- to rotate" << endl;

cout << "0. Cube" << endl;

cout << "1. Shape 1" << endl;

cout << "2. Shape 2" << endl;

glutInit(&argc, (char\*\*)argv); //initialize the tool kit

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);//set the display mode

glutInitWindowSize(screenWidth, screenHeight); //set window size

glutInitWindowPosition(100, 100); // set window position on screen

glutCreateWindow("Thi Online"); // open the screen window

myInit();

glutKeyboardFunc(myKeyboard);

glutDisplayFunc(myDisplay);

glutSpecialFunc(mySpecialFunc);

cube.CreateCube(2);

shape1.CreateShape1(/\*Tham so thuc\*/);//Sinh vien tu dien tham so

shape2.CreateShape2(/\*Tham so thuc\*/);//Sinh vien tu dien tham so

glutMainLoop();

return 0;

}