```
#include <cstdlib>
#include <GL/glut.h>
#include <vector>
#include <fstream>

//some methods you need to modify:
void renderScene(void);
void mouseMove(int x, int y);
void readMesh(const char filename[100]);

void computeMassCenter(float massCenter[3]);
void computeNormal(int faceID, float norm[3]);

//some methods you want to look at:
void reshapeScene(GLsizei width, GLsizei height);
void initGL();
void mouseClick(int button , int state, int x, int y);
void keyBoard(unsigned char key, int x, int y);

float ObjTranslation[3]={0,0,0};
float EyeTranslation[3]={0,0,-5}; //default eye position

int win_width, win_height;
int gButton;
float whratio;
int mouseX, mouseY;

// Design whatever data structure you like as the point list
std::vector<float> ptlist[3];
std::vector<int> facelist[3];

GLfloat rot_x = 0.f, rot_y = 0.f;

////////////////////////////////
void usage(void)
{
   printf("This program is the first homework from XXX \r\n");
   printf("The input is a .m triangle mesh. \r\n");
   printf(" f - flat shading\r\n");
   printf(" s - smooth shading\r\n");
   printf(" w - wireframe\r\n");
   printf(" key ESC - exit\r\n");
}

int main(int argc, char **argv) {
        //1. The first thing you must do is call the function glutInit. It initializes GLUT library.
        glutInit(&argc, argv);

        //2. Define the display mode using the function glutInitDisplayMode(unsigned int mode)
        glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH );


        //3. Define the window
        //3.1. Establish the window's position, i.e. its top left corner.
        glutInitWindowPosition(100,100);
```

```
	//3.2. Choose the window size, using the function glutInitWindowSize(int width, int height).
	int width = 640;
	int height = 480;
	whratio = (float)width /(float)height;
	glutInitWindowSize(width, height);


	//3.3 After these settings, we can create the window with a title
	glutCreateWindow("XXX's Homework 1");

	//4. Assign a function responsible for the rendering (register a callback)
	glutDisplayFunc(renderScene);

	//5. Assign a reshape function for the "resize" event received by the window.
	glutReshapeFunc(reshapeScene);

	//6. Assign functions for mouse and keyboard events respectively
	glutMouseFunc(mouseClick);
	glutMotionFunc(mouseMove);
	glutKeyboardFunc(keyBoard);

	//7. Some GL initialization
	initGL();

	//8. Load the Mesh,
	//read the mesh , and store them into the vertex and face lists,  argv[1] is the filename

	readMesh("torus.m");

	//9. Finally, get in the application event processing loop
	glutMainLoop();
	return 1;
}


/* Called when a "resize" event is received by the window. */
void reshapeScene(int width, int height)
{
	win_width=width;
	win_height=height;

	GLsizei mwidth = (width<height*whratio)?width:(height*whratio);

	glViewport(0,0,mwidth, (float)mwidth/whratio);

	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();

	gluPerspective( 40.0, /* field of view in degrees */
		whratio, /* aspect ratio */
		1.0, /* Z near */
		100.0 /* Z far */);

	glMatrixMode(GL_MODELVIEW);
```

```
        glutPostRedisplay();
}

void initGL(){

  GLfloat lightOneColor[] = {1, 1, 1, 1};
  GLfloat globalAmb[] = {.1, .1, .1, 1};
  GLfloat lightOnePosition[] = {1, 1, 1, 0.0};

  glEnable(GL_CULL_FACE);
  glFrontFace(GL_CCW);
  glEnable(GL_DEPTH_TEST);
  glClearColor(0,0,0,0);
  glShadeModel(GL_SMOOTH);

  glEnable(GL_LIGHT1);
  glEnable(GL_LIGHTING);
  glEnable(GL_NORMALIZE);
  glEnable(GL_COLOR_MATERIAL);

  glLightfv(GL_LIGHT1, GL_DIFFUSE, lightOneColor);
  glLightModelfv(GL_LIGHT_MODEL_AMBIENT, globalAmb);
  glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);

  glLightfv(GL_LIGHT1, GL_POSITION, lightOnePosition);
}

void  mouseClick(int button , int state, int x, int y)
{
  if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
  {
        gButton = GLUT_LEFT_BUTTON;
        mouseX = x;
        mouseY = y;
  }

  if (button == GLUT_MIDDLE_BUTTON && state == GLUT_DOWN) {
     gButton = GLUT_MIDDLE_BUTTON;
        mouseX = x;
     mouseY = y;
   }

  if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) {
     gButton = GLUT_RIGHT_BUTTON;
        mouseX = x;
     mouseY = y;
   }
  return ;
}

void mouseMove(int x, int y)
{
  /* rotation*/
  if (gButton == GLUT_LEFT_BUTTON )
  {
        ////////////added////////////////
```

```
            rot_y += (x - mouseX);
            rot_x += (y - mouseY);
            ////////////////////////////////
            glutPostRedisplay();
 }

 /*xy translation */
 if (gButton == GLUT_MIDDLE_BUTTON)
 {
             ////////////added///////////////
             GLfloat scale_y = 10.f / win_height;
             GLfloat scale_x = 10.f / win_width;
             ObjTranslation[0] += scale_x * ( x - mouseX  );
             ObjTranslation[1] += scale_y * ( mouseY - y);
   glutPostRedisplay();
 }

 /* zoom in and out */
 if (gButton == GLUT_RIGHT_BUTTON )
 {
            double scale = 10./win_height;
             ObjTranslation[2] -= scale*(mouseY-y);
             glutPostRedisplay();
 }

 mouseX = x;
 mouseY = y;
}


void keyBoard(unsigned char key, int x, int y)
{
            switch( key )
            {
            case 'f':
                        //Flat Shading
                        glPolygonMode(GL_FRONT, GL_FILL);
                        break;
            case 's':
  //Smooth Shading
                        glPolygonMode(GL_FRONT, GL_FILL);
                        break;
            case 'w':
              //Wireframe mode
                        glPolygonMode(GL_FRONT, GL_LINE);
                        break;
            case '?':
                        //usage
                        usage();
                        break;
            case 27:
                        exit(0);
                        break;
            }
 glutPostRedisplay();
}
```

```
/* setup the object, transform from the world to the object coordinate system */
void setupObject(void)
{
	glTranslatef(ObjTranslation[0],ObjTranslation[1],ObjTranslation[2]);
	//you can also do them via a Matrix
	//double transMatrix[16];
	//glMultMatrixd(( GLdouble *)  transMatrix );
	glRotatef(rot_x, 1.0, 0.0, 0.0);
	glRotatef(rot_y, 0.0, 1.0, 0.0);
}

//Setup the eye position
void setupEye(void)
{
 //you may want to adjust your eye according to the object
 float transMatrix[16]={1,0,0,0,0,1,0,0,0,0,1,0,EyeTranslation[0], EyeTranslation[1], EyeTranslation[2],1};
 glMultMatrixf(( GLfloat *)  transMatrix );
 //the above is the same as: //glTranslatef( EyeTranslation[0], EyeTranslation[1], EyeTranslation[2]);
}

void renderScene(void)
{
	/* clear frame buffer */
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glLoadIdentity();

	setupEye();
	setupObject();

	//Render the mesh (you may need to change them, according to your mesh data structure)
	glBegin(GL_TRIANGLES);
	for (int i=0;i<facelist->size();++i)
	{
		int vertID[3];
		float norm[3];
		computeNormal(i,norm);
		glNormal3f(norm[0], norm[1], norm[2]);
		for (int j=0;j<3;++j)
		{
			vertID[j] = facelist[j][i];
			float x = ptlist[0][vertID[j]];
			float y = ptlist[1][vertID[j]];
			float z = ptlist[2][vertID[j]];
			glVertex3f(x,y,z);
		}
	}
	glEnd();

	glutSwapBuffers();
}
```

```
void readMesh(const char filename[100]){
	FILE * fp = fopen( filename, "r" );
	if( !fp ){
		printf("Can't open file %s\n", filename );
		return;
	}

	char line[1024];
	int id;
	while( !feof(fp)  ){
		fgets( line, 1024, fp );
		if( !strlen( line ) ) continue;
		char * str = strtok( line, " \r\n");
		if (!str) continue;
		if( !strcmp(str, "Vertex" ) ) {
			str = strtok(NULL," \r\n{");
			id = atoi( str );
			for( int i = 0 ; i < 3; i ++ ){
				str = strtok(NULL," \r\n{");
				ptlist[i].push_back(atof( str ));
			}

			str = strtok( NULL, "\r\n");
			if(!str)   continue;
			std::string s(str);
			int sp = s.find("{");
			int ep = s.find("}"); //if we want to store attributes, use the index sp and ep
			continue;
		}

		if( !strcmp(str,"Face") ) {
			str = strtok(NULL, " \r\n");
			if (!str) continue;
			id = atoi( str );
			int vid[3];
			for( int i = 0; i < 3; i ++ )  {
				str = strtok(NULL," \r\n{");
				vid[i] = atoi(str);
				facelist[i].push_back(vid[i]-1);
			}
			if (!str)  continue;
			str = strtok( NULL, "\r\n");
			if( !str || strlen( str ) == 0 ) continue;
			std::string s(str);
			int sp = s.find("{");
			int ep = s.find("}"); //for attributes
			continue;
		}
	}
	fclose(fp);

	//Step 4
	float mass[3];
	computeMassCenter(mass);
}
```

```
void computeMassCenter(float massCenter[3]){
	massCenter[0] = massCenter[1]=massCenter[2] =0;
	for (int i=0;i<ptlist[0].size();++i){
		massCenter[0]+=ptlist[0][i];
		massCenter[1]+=ptlist[1][i];
		massCenter[2]+=ptlist[2][i];
	}
	massCenter[0]/=ptlist[0].size();
	massCenter[1]/=ptlist[1].size();
	massCenter[2]/=ptlist[2].size();
	for (int i=0;i<ptlist[0].size();++i) {
		ptlist[0][i]-=massCenter[0];
		ptlist[1][i]-=massCenter[1];
		ptlist[2][i]-=massCenter[2];
	}
}

void computeNormal(int faceID, float norm[3]){
	int vid[3];
	vid[1] = facelist[1][faceID];
	vid[2] = facelist[2][faceID];
	float x[3],y[3],z[3];
	for (int i=0;i<3;++i){
		vid[i] = facelist[i][faceID];
		x[i]=ptlist[0][vid[i]];
		y[i]=ptlist[1][vid[i]];
		z[i]=ptlist[2][vid[i]];
	}
	float p1[3], p2[3];
	p1[0]=x[1]-x[0];p1[1]=y[1]-y[0];p1[2]=z[1]-z[0];
	p2[0]=x[2]-x[1];p2[1]=y[2]-y[1];p2[2]=z[2]-z[1];
	norm[0]= p1[1]*p2[2]-p1[2]*p2[1];
	norm[1]= p1[2]*p2[0]-p1[0]*p2[2];
	norm[2]= p1[0]*p2[1]-p1[1]*p2[0];
}
```