Homework 5:

We discussed how to implement edge lists (for meshes) and primitive lists (for solids) such that vertices are not duplicated. Implement these in C++ in such a way that geometry and topology are separate, and such that we can use this for the next homework, which will render an arbitrary number of lines (for meshes) or polygons (for rendering solids).

1) Class MeshBuffer should have, at the very least:
    a. a method to add a "line" or "edge" to the mesh, such that the vertices are not duplicated
    b. a method to modify the position of a particular vertex, without disrupting the topology of the mesh (it should be clear that this involves "finding" the vertex in whatever underlying data structure you choose)
    c. methods to retrieve the entire mesh in a data format that is consistent with it being passed directly into OpenGL
2) Class TriangleBuffer should have, at the very least:
    a. a method to add a "triangle" to the solid, such that the vertices are not duplicated
    b. a method to modify the position of a particular vertex, without disrupting the topology of the solid (it should be clear that this involves "finding" the vertex in whatever underlying data structure you choose)
    c. a method to calculate the normal to that triangle (and save it)
    d. any vertex modifications should trigger recalculation of normals for each triangle in which that vertex plays a part
    e. methods to retrieve the entire solid in a data format that is consistent with it being passed directly into OpenGL (vertices and normals)

Use Angel's vec.h tools as your representation of vertices and normals in 3D homogenous coordinates.