Your first programming assignment is designed to familiarize you with the OpenGL API, including two-dimensional geometric primitives, keyboard and mouse controls, colors, and animation. Linked to the course website (http://www.cs.siue.edu/wwhite/CS482/Syllabus.htm) is a simple program containing the following features (illustrated in the figures below):

- Opening with an empty black window, the application begins by asking the user to specify a dozen 2D vertices via mouse clicks, with each mouse click resulting in the display of a red vertex.
- Once all twelve vertices are created, the application switches to keyboard mode, in which the pressing of any single numerical digit results in the vertices being used to create an OpenGL 2D geometric primitive:
- In addition, when the user presses the ‘A’ key, the application begins animating by having the vertices take turns spinning twice in a small circular pattern before returning to their original positions.

Noteworthy aspects of the code include:

- A GLfloatPoint class to deal with 2D vertices, with some unused functionality that will be useful in your assignment.
- Global variables, often considered taboo in non-graphical programming, prove very useful when the alternative is to repeatedly pass large data structures or dynamic state variables.
- Separate functions to deal with updating positions of animated objects (TimerFunction) and displaying objects in their current positions (Display).

Your assignment is somewhat more complicated than the sample program. Your program shall allow the user to generate an unlimited number of 2D vertices, including the deletion of vertices by means of mouse-clicking on any existing vertex. At any point during the execution of the program, the user will be able to render the convex hull of the current set of vertices by means of the following algorithm:

1. Let \( v_0 \) be the lowest vertex in the set of vertices \( V \).
2. Sort the remaining vertices \(<v_1, v_2, \ldots, v_{n-1}>\) by polar angle in counter-clockwise order around \( v_0 \) (if more than one vertex has the same angle, remove all but the one farthest from \( v_0 \)).
3. Push \( v_0, v_1, \) and \( v_2 \) onto a stack \( S \).
4. Loop through the remaining \( n-3 \) vertices \( v_3, \ldots, v_{n-1} \) as follows
   a. While the angle between \( S \)’s next-to-top value, \( S \)’s top value, and \( v_i \) makes a non-left turn,
      i. Pop the top element of \( S \) off
   b. Once that while-loop is finished, push \( v_i \) onto \( S \)
5. The resulting stack \( S \) contains the vertices that comprise the convex hull, in order.
An executable version of this program is provided. Note the required functionality, illustrated below:

- Unlike the sample program, there is no limit to the number of vertices that the user can generate.
- When the user clicks on an existing vertex, that vertex is removed from the linked list of vertices and no longer rendered.
- When the user clicks the 'R' key, the linked list is “reset” by having all current vertices removed from the linked list.
- The user may click the 'V' key throughout the execution of the program to toggle the visibility of the vertices on and off.

- To demonstrate the successful calculation of the lowest vertex in the set of current vertices, the user may press the ‘L’ key, generating a set of lines from the lowest vertex to each of the other vertices on display.
- The GLfloatPoint functionality for determining whether one vertex is lower than another (provided with the sample program) may be used for this purpose.

- To demonstrate the successful calculation of the convex hull of the set of vertices, the user may press the ‘H’ key, generating a line loop traversing the vertices comprising the stack created by the above algorithm.
- The GLfloatPoint functionality (i.e., left-turn determination, polar angle determination) and the Stack subclass of the LinkedList class, provided with the sample program, supports the implementation of this algorithm.

- Animation will be activated by the 'A' key, just as in the sample program, but the animation will affect all vertices simultaneously, with each vertex having its own randomly generated rotation increment and rotation radius.
- Global linked lists for each vertex’s current rotation, rotation center, rotation radius, and rotation increment are needed to implement this animation functionality.
- A triangle fan will be used to render the convex hull in animation mode, with the convex hull being adjusted dynamically.

Zip-compress a folder containing only the relevant .cpp and .h files, and copy it to your dropbox by **Tuesday, September 6, 2016, at 12:00 Noon**. Late assignments are not accepted without verifiable medical documentation. You must write your **own** code, and no one but the instructor may see your code.