Lab 4: *You'll need to know how to call library functions, and how to write and call your functions. For a head-start, you can use the file Lab4Template.cpp, supplied with this lab. It contains a framework including two useful functions that you can call from main( ).*

Ask the user for three things:

1. Ask the user if they want to draw either a triangle or a sine curve. <u>You have to provide code to do both.</u>

2. For the triangle, specify the size (in max row height to be printed).

3. For the triangle, specify the character to use.

4. For the sin curve, plot only the positive values from 0 to Pi radians (or 0 to 180 degrees).

If the user specifies a triangle of size 6 using '@', then output:

```
@
@@
@@@
@@@@
@@@@@
@@@@@@
@@@@@
@@@@
@@@
@@
@
```

note that the triangle is 11 rows in size, but its center uses 6 characters.

A suggestion… You might find this function helpful for drawing a triangle. You should call this function with an increasing number from 1 to size, followed by a decreasing number (size-1) to 1 to get a perfect triangle. That is, place this function *<u>call</u>* in main within a loop that goes from 1 to size, then in a second loop that counts down from (size-1) to 1. This function is supplied in the Lab4Template.cpp file.

```
// this function is called from main e.g. drawLine( 10, '$' );
void drawLine (int lineLength, char displayChar)
   {
   for (int x=1;  x <= lineLength;  x=x+1)
      {
      cout << displayChar;
      }
   cout << endl;
   }
```

For the sine curve, you might find this function useful for drawing a sine curve from 1 to 180 degrees:

```
// this function places an asterisk '*' at a specific column on
// the display
void placePoint ( int lineLength )
   {
   for (int x=1;  x < lineLength;  x=x+1)
      {
      cout << " "; // print blanks from 1 to one less than the desired point
```

```
    }
cout << "*" << endl; // then print an asterisk and skip to next line
}
```

Some detailed help on the sin curve:

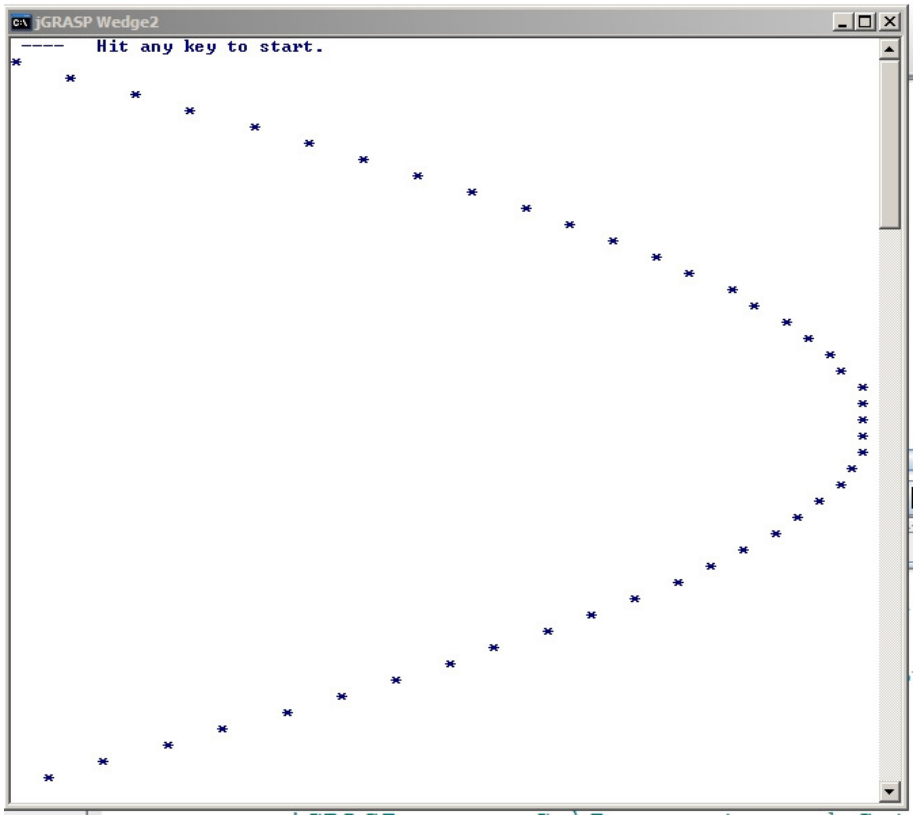
1. Create a "for" loop that goes from 0 to 180, representing degrees. You might want to increment by 4 rather than 1, so that it gives a more reasonable number of points to plot. i.e. for (int t=0; t<180; t=t+4) will give you 45 points: 0, 4, 8, 12, …. 180.

2. Convert each value of the loop variable to radians by multiplying by Pi and dividing by 180. Make sure the number that holds the radian value is a double. If your loop increments from 0 to 180 by 4, then you should get 45 radian values from 0 to Pi. These are thus very plottable.

3. Find the sin of each of those values. The 45 values should range nicely from 0 to 1.0 and then back to 0;

4. In order to plot these, we have to NORMALIZE the decimal sin values to integers in the range 0 to 80, because there are 80 spaces in a displayable line on our output console. Simply multiply each sin value by 80 and cast to an integer. i.e. if plotPoint is an integer variable: plotPoint = (int) (80.0 * sinValue);
Your 45 sine values (ranging from 0 to 1) will be converted to integer values in the range 0 to 80. The term for this exercise is providing a GAIN of 80 to each point.

5. Call the placePoint( ) function with the 45 plotPoints. The result:



If you're feeling REALLY adventurous (this is optional), plot from 0 to 360 instead of 0 to 180, incrementing each point by 8 instead of 4. Then normalize to 40 instead of 80 (that is, use a gain of 40 instead of 80), but then add 40 to each point. This is called adding a BIAS of 40 to each point. The result is a shift of the entire curve to the right, and the result looks like this: