```
FINAL PROJECT
-------------

The final project for this class will utilize the shape abstract
base class that you built in assignment 8 and a main() function
provided by the instructor to navigate a shape (either a rhombus,
capsule, or lens) through an obstacle course from a provided starting
location (x1,y1,theta1) to a provided stopping location (x2,y2,theta2).
The obstacle course will be provided as input in the form of a binary
image in which black pixels (0) indicate obstacles. The output will
be the same image but with the starting, stopping, and various intermediate
positions of the shape written onto the image using grayscale values of 80
(just as in assignment 8).

The program will be run with three command line arguments as follows:

navigate obstacle.pgm output.pgm shape.txt

where:  navigate     is the name of the program
        obstacle.pgm is the name of the input binary image (obstacle course)
        shape.txt    is the input text image describing the input shape using
                     a format very similar to that in assignment 8, but with
                     3 additional nubmers for the stop location (x2,y2,theta2)
        output.pgm   is the name of the output image which shows the path taken
                     by the shape as it moves from the start location
                     (x1,y1,theta1) to the stop location (x2,y2,theta2).

All I/O reading and writing will be handled by the provided main. The only
additions to your shape objects required (beyond the requirements in the
writeup for assigment 8) are the following member functions for the
3 derived classes:

1) The capsule class will require the following public member function:

    void setParams(float width,height)

   where width specifies the length of the two line segments in the capsule,
   and where height specifies the diameter of its two arc segments.

2) The rhombus class will require the following public member function:

    void setParams(float width, float height, float corner_angle)

   where width and height specify the two different lengths of each side of
   the rhombus (width specifies the "first" side drawn starting from the
   center point of rotation, just  as described in assignment 8), while
   coner_angle specifies the angle of the first corner (beginning of the
   first line segment) in DEGREES.

3) The lens class will require the following public member function:

    void setParams(float radius, float arc_angle)

   where radius and arc_angle specify the radius of the two arcs making
   up the lens and the angle "swept out" by these two arcs (exactly as in
   assignment 8).
```

The following member function is required in the base class shape (this was optional but recommended in assignment 8):

```
  isValidLocation(unsigned char *image,int width,int height)
```

which checks to see if the current location of the shape object is valid (does not interesect any obstacle in the input image and does not go outside the image). Not that the required function in assignment 8 also included the location of the shape to test (x,y,theta) as arguments. The difference here is that the current values of (x,y,theta) in the object itself are to be used (this overloaded version of the function was recommended but not mandatory in assignment 8).

The following member function is also required in the base class shape (and was also required in assignment 8):

```
  drawOutline(unsigned char *image,int width,int height)
```

which draws the outline of the shape (using its current position) onto the image array using the grayscale value 80 for each pixel interesected by the outline.

The key function in this program is called computeDistances and has the following prototype (we have provided a simpler 2D version):

```
void computeDistances(
 unsigned char *obst,shape *s,int dest,int *dist,int xsize,int ysize,int zsize
);
```

where:

```
  obst  is the input binary obstacle course image (obstacles indicated in 0)

  shape is a pointer to the shape to be navigated through the obstacle course

  dest  is the 1D integer offset of the 3D grid point representing the
        destination location of the shape (the first two coordinates give
        integer values of the x,y values of the center of rotation and the
        third coordinate gives an integer representing a multipe of 10 degrees
        for the angle of rotation of the destination location). For example
        if the final location of the shape is x=10, y=30, and theta=60 degrees
        then the 3D indeces of the destination grid point are 10,30,6
        and the corresponding 1D offset (since we store the 3D array as a
        1D array) will be 10 + 30*xsize + 6*xsize*ysize.
  dist represents a 3D integer array (stored as a 1D array) storing the
        distances between each 3D grid point in the array to the final
        destination location of the shape. The distance value stored at each
        point represents the number of "adjacent 3D grid-point moves" needed to
        get from the given grid point to the final destination point.
        The main point of this fuction is to compute the values in this array
        using a queue data structure together with the isValidLocation()
        member function in your shape class.

  xsize is the width of the input 2D image (the obstacle course) as well as
```

```
        the width of the output 3D distance array computed by this function.

  ysize is the height of the input 2D image (the obstacle course) as well as
        the height of the output 3D distance array computed by this function.

  zsize is the depth of the output 3D distance array (this third dimension
        represents the various angles that a shape can be placed at in ten
        degree intervals, thus for this assignment zsize=36, which you will
        see specified in the provided main).
```

This function, starts from the destination location, and assuming it is a valid location, it sets the distance value to zero at this destination point. It then marches layer by layer (using a queue much as in homework #6) to the neighbors of this destination point and then the neighbors of the neighbors (and so on) checking if they are valid locations of your shape. If so, incremented distance values are stored at these locations (increments by one each time).

The primary job in your final project is to provide an optimized version of your isValidLocation() function so that the very large number of calls to this member function during the execution of the computeDistances() routine will not result in slow execution. In class, a look up table was recommended in which a list of pixels for the outline of the shape is recorded inside the object for each value of the rotation angle theta in ten degree increments. However, this is not a requirement. You may use any "clean" method you like to make this function run quickly. IN ORDER TO OBTAIN A PERFECT SCORE ON THE FINAL PROJECT, YOUR CODE MUST RUN IN 8 SECONDS OR LESS FOR EACH EXAMPLE (on a 2 GHz computer). Partial ccredit will be given

----
FORMAT

You will NOT be providing your own main() for the final project, but must instead use the main provided. Your code will be compiled and linked to a main stored on the computer used to test your code during the final exam. You must provide ONLY 3 files will EXACTLY the following names (all lower case letters including the first letter):

```
shape.h      : the header file for your shape class (and all derived classes)
shape.cpp    : the file which implements your non-inlined shape class functions
distance.cpp : the file which implements your computeDistances() functions
```

These three file MUST be located in the ROOT DIRECTORY of a USB pen drive (or a CD) that you bring with you to the final exam. These files also must be completely SELF-CONTAINED so that NO ADDITIONAL FILES (other than the provided main.cpp and pgmio.cpp) are need in order to compile your code. Failure to follow these guidelines exactly may result in a loss of points on your final project score.