# 24-780 B—Engineering Computation

Assigned: Wed. Oct 27, 2021
Due: Fri. Oct. 29, 2021, 11:59pm

## Problem Set 7: Shape Plus

*Before anything:*

Note the deadline for this assignment is only 2 days after the release date. I actually expect PS07 to be completed in 2 hours, *during lecture*. The extra time is added to account for remote students who miss out on the live lecture. Since the demo project is due on Tuesday, Nov. 2, I really want you to concentrate on finalizing your implementations.

*Now for the actual assignment:*

In PS06 you worked on an application to define, edit, and calculate properties of shapes. Quick aside, remember that you can carry on with your implementation or simply use my solution as your starting point. Now, we want to be able to use the same system, but we want to add specific types of shapes used in engineering rather than just weird, playful shapes. We will make use of object-oriented programming principles to facilitate our enhancements and set the stage for any future capabilities.

### Task 1: Prepare ViewManager code

Since we want ViewManager to be able display all kinds of Shapes, we need to prepare the way for polymorphism and particularly for virtual functions. The only changes we need are related to how we maintain our reference to the shape itself. In PS06, ViewManager class has a member variable called "theShape". We need to convert this to a pointer in ViewManager.h.

```
Shape theShape;
```

Becomes:

```
Shape *theShape;
```

No other changes will be required in ViewManager.h, but ViewManager.cpp will now have several errors since we have the wrong notation (using period instead of arrow). We can make a quick change in ViewManager.ccp using find/replace (replace "theShape." with "theShape->" ).

That's a start, but we also need to assure that there is an actual Shape object at the end of that pointer. Let's add the following dynamic instantiation to the constructor of ViewManager:

```
theShape = new Shape();
```

Amazingly, the changes so far have little to no effect on how everything works. We will need some changes to ViewManager::load() function, but that's later.

### Task 2: Prepare Shape class

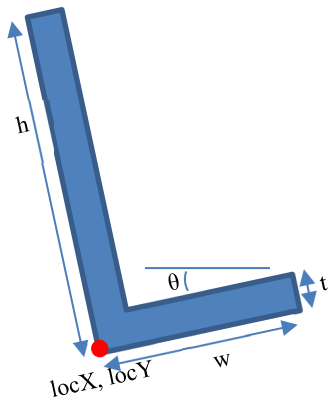
Since we want to create several types of shapes while keeping ViewManager mostly ignorant about it, we need to make use of virtual functions. In particular, the following functions need to be virtual so we can over-ride as needed;

readFile(), findNode(), writeFile(), and recalcSpline()

### Task 3: Develop new classes

We want at least two new classes: Lshape and Ishape. Each one will automatically populate nodes in the vector theGuides to define the shape

**Lshape** will be used to quickly define an L-shaped cross-section. The section can be defined with 6 parameters: locX, locY, height, width, thickness, and theta (in degrees).

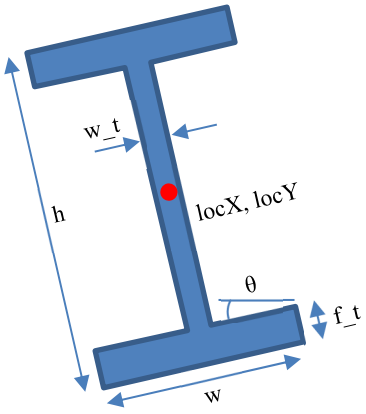



A typical input file (with "Lshape" in the name) will look like this:

```
locX: 12.1
locY: 2.1
height: 3.5
width: 2.0
thickness: 0.375
theta: 8.7
```

Note: You will not be able to use OpenGL transformations for the rotation because we want the rotation to be included in the shape properties calculations, not just for rotation to be a display thing.

**Ishape** will be used to quickly define an I-shaped cross-section. The section can be defined with 7 parameters: locX, locY, height, width, flangeT, webT, and theta (in degrees).




A typical input file (with "Ishape" in the name) will look like this:

```
locX: 12.1
locY: 2.1
height: 8.0
width: 6.0
flangeT: 0.375
webT: 0.25
theta: 8.7
```

Each of these classes will inherit from Shape, but will need to over-ride some virtual functions. We can use PS05 as a good example for how to implement readFile(). The code for the function writeFile() is obvious.

FYI: when you need to call a parent class's function from the over-ridden child-class function itself, just include the name of the parent class in the function call. For example, for our special shapes, we want to disallow the "finding" of points in *theSpline* vector. Thus, we need to over-ride the findNode() function so that no points could be added like this:

```
Node* findNode(double x, double y, double distance,
               bool searchSpline = false) {
      return Shape:: findNode(x, y, distance, false);
}
```

The function recalcSpline() is called whenever a Shape is loaded or changed. This is the place to put our code for generating nodes to define the special shape we want.

### Task 4: Finalize ViewManager functionality

Now we need to get ViewManager to work out the best way of adding our special shapes. Specifically, we need to make some changes in ViewManager::load() so that when we get a valid file (the is_open() returns true), we act accordingly, rather than just assuming every file is a basic Shape object. Since we are using pointers, we need to *delete* the old shape and then instantiate using *new*.

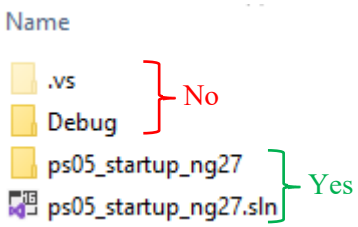
## Deliverables

9 files (zipped together):

ViewManager.h, ViewManager.cpp
Shape.h, Shape.cpp
Lshape.h, Lshape.cpp
Ishape.h, Ishape.cpp
ps07_shape_andrewID.cpp

Upload the zip file to the class Canvas page before the deadline (Friday, Oct. 29, 11:59pm).

Alternatively, if you are using Visual Studio, it may be easier to submit your entire solution rather than a collection of files. To do this, create a *zip file* of the whole project (the .sln file and the associated folder), being careful NOT to include the hidden folder called ".vs". This folder is used only to manage the IDE and is typically huge (100MB). Erasing or omitting it will just force Visual Studio to rebuild it when needed. The Debug folder should also be kept out of the zip file to avoid including executable files that some firewalls may disallow. *The name of the project should include your AndrewID*




## Learning Objectives

Use of classes and objects in C++.

Virtual functions

Adapting existing code for new objectives

Increasing understanding of graphical data representation.