CIS 330: Project #2D
Assigned: April 30th, 2018
Due May 5th, 2018
(which means submitted by 6am on May 6th, 2018)
Worth 4% of your grade

Please read this entire prompt!

Assignment: You will implement subtypes with C, building off your work from project 2C.

1) Make a union called ShapeUnion with the three types (Circle, Rectangle, Triangle).
2) Make a struct called FunctionTable that has pointers to functions.
3) Make an enum called ShapeType that identifies the three types.
4) Make a struct called Shape that has a ShapeUnion, a ShapeType, and a FunctionTable.
5) Modify your 9 functions to deal with Shapes.
6) Integrate with the new driver program (driver_2D.c). Test that it produces the correct output.

The driver program further specifies what the interface should be. Before you ask questions, make sure you've looked at the driver program. Although this prompt could be even more detailed, it is much more detailed than what you'll get in the real world. Reading code and inferring details yourself is a critical skill to develop.

== 1. ShapeUnion ==

A union of Rectangle, Circle, and Triangle.

== 2. FunctionTable ==

This struct has two data members. Both data members are pointers to function. The first data member should be named "GetArea". The second data member should be named "GetBoundingBox".

== 3. ShapeType ==

An enum that identifies whether you have you a rectangle, circle, or triangle.

== 4. Shape ==

Contains a ShapeUnion, ShapeType, and a FunctionTable. The FunctionTable must have name "ft".

== 5. Modify your 9 functions ==

Each of your 9 functions takes in a Rectangle, Circle, or Triangle. Modify the pointer type to Shape instead of Rectangle, Circle, or Triangle. Further modify the function to carry out the desired operation (initialize, get area, get bounding box).

For example, "double GetRectangleArea(Rectangle *)" will become "double GetRectangleArea(struct Shape *)". The function will then operate on the Shape, knowing that the shape is a rectangle. You will need to make changes to your prototypes.h and to your .c file that implements the functions.

Note that initialize will become much bigger. If you are implementing InitializeRectangle, then you will need to initialize the "Rectangle" part of the ShapeUnion in the Shape, but you will also need to initialize the Shape itself. This includes setting functions in the FunctionTable.

== 6. Integrate with driver ==

A new driver program has been posted. It needs to: compile, run, and produce the correct output for you to complete the assignment.

== 7. ADMIRE!! ==

Look at what you've done. You have now done subtyping/supertyping. The driver program operates on Shapes without knowing their details.

== IMPORTANT ==

FunctionTable and Shape have a "circular dependency". This means that, when you define FunctionTable, you want to tell the compiler about Shapes. So that would make you want to define Shapes first. Unfortunately, when you tell the compiler about Shapes, you want it to know about FunctionTables! So that means putting either first is problematic. As it turns out, you have to put FunctionTable first. The compiler needs to know the size of the Shape struct, and it can't know that without knowing the size of the FunctionTable struct.

Your code should be:

```
struct Shape;  /* tells the compiler that a struct named Shape will be defined later */

typedef struct
{
  /* add function pointers.  Refer to Shape as "struct Shape" */
} FunctionTable;
```

```
struct Shape
{
   /* add data members for Shape */
};
```

Also: note that if you do

```
typedef struct
{
} Shape;
```

then it will work, but gcc will issue some annoying warnings about your function prototypes not matching.

== What to modify ==

You will need to modify my_struct.h, my_struct.c, and prototypes.h. You should **not** modify driver_2D.c. If you modify this latter file, you will have points deducted. The grader program (grader.sh) will reveal whether it has been modified.

== Success ==

Your executable will be named "project_2D". Run your program as:

```
./project_2D > my_output
```

and call:

```
diff my_output driver_output
```

If diff returns no differences, then your program produces the correct output.

== What to turn in ==

Before you submit, make sure to test your code on ix-dev using the provided grader program script (grader.sh) within your project directory (on ix-dev).

Your actual source code (my_struct.h, my_struct.c, and prototypes.h) will still be graded for good programming practices. The project will be graded on ix-dev.

Submit to Canvas a tarball named 2D_turnin.tar with the following files:

```
tar -cvf 2D_turnin.tar my_struct.h my_struct.c prototypes.h my_output
```