Shapes

Lets first start by showcasing the needed components in order to construct a shape.

Points

```
Point makePoint(int x, int y);
```

Through the `makePoint` function we can create a point structure. This struct contains a x and a y value. You can consider this "point" to represent a pixel within the Pixel array. You can also manually construct a point through the following method:

```
typedef struct {
  int x;
  int y;
} Point;
```

A points values can be set through manually changing the values encapsulated within the point structure. An example of this being:

```
Point p;
p.x = 2;
p.y = 3;
```

Points can also be created using the alias `Vec2`, this may help with readability especially when using the variable in cases where the name point may not be appropriate.

The other type of point is the `Point3`. The a point 3 is not currently used within the implementation, however it will be extremely useful in creating 3d programs. These can also be refrenced through the name `Vec3`. Values can be set in the same manner as the `Point`, however, with the addition of a new `z`.

```
Point3 p;
p.x = 1;
p.y = 2;
p.z = 3;
```

Lines

Lines can be useful in creating practically any visual application. A line can be created through the following method:

```
Line makeLine(Point start, Point end);
```

Additionally we the `makeLineExplicit` function will allow us to create a line without needing to first create points.

```
Line makeLineExplicit(int x, int y, int x1, int y1);
```

The start and end poisitions both being created through the aforementioned line points. A line is only defined through two points since the points inbetween can be assumed. This is also helpful in allowing us to implement out Bresenham line algorithm.

The line structure can be seen below.

```
typedef struct {
  Point start;
  Point end;
} Line;
```

Lines can then be implemented within our pixel array using the `addLine` function. This function implements Bresenham line algorithm in order to calculate where each pixel should be placed. The add line function takes the following parameters:

```
void addLine(cell c, Line l, Pixel pix);
```

Triangles

Triangles operate very similarly to lines. Triangles are infact made up of 3 lines. A triangle may be created through the following:

```
Triangle makeTriangle(Point p1, Point p2, Point p3);
```

A triangle can also be created manually, the structure of which is the following:

```
typedef struct {
  Point p1;
  Point p2;
  Point p3;
} Triangle;
```

We can use similar functions to contruct our triangle points as well.

```
Triangle makeTriangle(Point p1, Point p2, Point p3);
Triangle makeTriangleExplicit(int pone1, int pone2,
                              int ptwo1, int ptwo2, int pthree1, int pthree2);
```

I would stray away from using the explicit function because it is ironically more verbose than using:

```
makeTriangle(makePoint(1, 2), makePoint(3, 3), makePoint(5, 10));
```

Rectangle

A rectangle is implemented in a similar way to the triangle. However we construct this through using `p1 -> p2`, `p2 -> p3`, `p3 -> p4`, `p4 -> p1`. An example of this can be seen below:

```
#    #

#    #
```

We can see that we have four points within the image above. We can understand `p1: (0,0)`, `p2: (5, 0)`, `p3: (5, 5)`, and `p4: (0, 5)`.

Now that we understand how the abstraction is being created we can introduce the usage, which can be seen below.

```
void addRectangle4(cell c, Rectangle4 s, Pixel pix);
void addRectangle(cell c, Rectangle s, Pixel pix);
```

As you can see we have two types of rectangles. A rectangle 4 is created through the contructor `makeRectange4`, and a 2 pointed rectangle can be created using `makeRectangle`. The usage of which can be seen below.

```
Rectangle makeRectangle(Point p1, Point p2);
Rectangle4 makeRectangle4(Point p1, Point p2, Point p3, Point p4);
```

A rectangle only contains 2 points, making it more similar to a line structure than a rectangle. This is because the other points can be derived from the original 2 points. However, if you wish to make a scewed version of a rectangle I have also implemented a 4 point variation. In fact the basis of the rectangle is the rectangle function. The two other points are calculated and applyed to make a rectangle4.

The implementations for adding your created rectangle can be seen below.

```
void addRectangle(cell c, Rectangle s, Pixel pix);
void addRectangle4(cell c, Rectangle4 s, Pixel pix);
```

There is also another variation of this method that allows you to fill the area within. These can be seen below.

```
void addRectangleFilled(cell c, Rectangle s, Pixel pix);
void addRectangle4Filled(cell c, Rectangle4 s, Pixel pix);
```