First – The Field of Conflict... (the penultimate battle between good and evil...)

Exam 2 is an on line exam. You have the entire class time to do the exam. If you finish early, you may leave or stay and work. If you do not finish by 1:20, you will not receive extra time.

The exam consists of four problems in which you write code. You need to create the folders and files required. You must submit them to the handin folder created for you in the course directory.

The problems focus on the fundamentals of programming with Processing that were presented in homeworks 1- 11 .

You may access the Processing API. You MAY NOT access any other material.

We will, where possible help you with syntax errors but we will not help you with logic problems.

Non-working code can receive partial credit but it must compile.

Second – The Rules of Engagement...

- **You will write four separate programs for this exam – one program for each question.**
- **Each program must be in a separate folder that has the same name as the .pde file inside it.**
- **You will put all four folders into a fifth outer folder named with your Andrew id.**
- **You will put the folder named with your Andrew ID into your handin folder on the server. Use the Exam2 folder.**
- **You will zip the folder named with your Andrew ID and mail it as an attachment using the subject line:**
  Exam 2
  to Jim at:
  jr2u@andrew.cmu.edu
- **You should mail a copy to yourself as backup security.**

There is partial credit for all four question – do not erase non-working code.

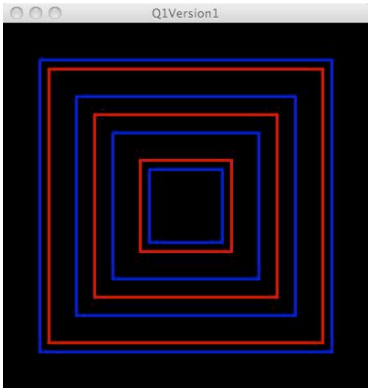
Question 1 - 10 points

Copy the following code into a processing program:

```
color c1, c2;

void setup( )
{
  size( 400, 400 );
  c1 = color( 200, 0, 0 );  // red
  c2 = color( 0, 0, 200 );  // blue
  strokeWeight( 3 );
  noFill( );
  rectMode( CENTER );
}

void draw( )
{
   background( 0 );
   design( 100, c1 );  // red  rects with 100 pixel separation
   design( 80, c2 );   // blue rects with  80 pixel separation
   noLoop( );
}
```

Define the function `design( float, color)`
where the arguments are:
argument #1 width and height of the innermost rectangle. It is also the amount added to the width and height of each subsequent rectangle drawn.
argument #2 is the color of the stroke making the rectangle.
Rectangles are drawn as long as they are visible in the window.
The fill value, strokeWeight value, and rectMode are set correctly in the setup( ) function.

You will need to use some form of loop to do this. Hardwiring the number of rectangles will not receive full credit.
Expected output:



Question 2 - 10 points

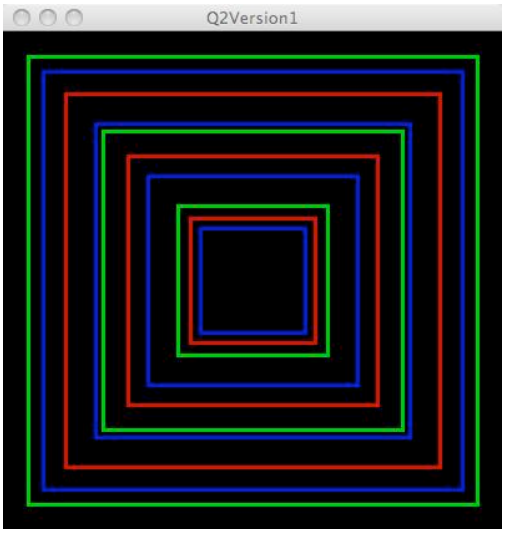
Open a new, empty Processing program.

- Copy the following code into the program:

```
int   [ ] spacingList =  { 100, 84, 120 };
color [ ] colorList   =  {  color( 200, 0, 0 ),
                            color( 0, 0, 200 ),
                            color( 0, 200, 0 ) };

void setup( )
{
  size( 400, 400 );
  strokeWeight( 3 );
  noFill( );
  rectMode( CENTER );
}

void draw( )
{
   background( 0 );
   drawAllDesigns( );
   noLoop( );
}
```

- Copy your `design( )` function from question 1 into the program.
- Define the function, `drawAllDesigns( )` so that it traverses the arrays and draws designs using the data in the arrays.
- Your `drawAllDesigns( )` must use the `design( )` function you defined in question 1 to receive full credit for this problem. If you were not able to get question one working, you can put code to draw the rectangles inside the loop that is traversing the arrays for partial credit.
- You MAY NOT hardwire the loop to work with only three-element arrays.
- Your code must work with parallel arrays of any length.

Expected output:



Question 3 - 15 points

Open a new, empty Processing program.
- Copy the following code into the program:

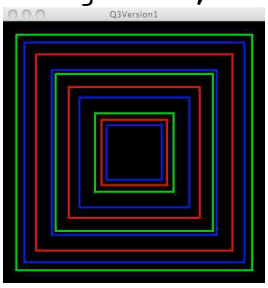
```
int   [ ] spacingList1 = { 100,  84, 120 };
int   [ ] spacingList2 = { 120, 140, 170 };
int   [ ] spacingList3=  { 130, 170, 190 };
color [ ] colorList   =  { color( 200, 0, 0 ),
                           color( 0, 0, 200 ),
                           color( 0, 200, 0 ) };

void setup( )
{
  size( 400, 400 );
  strokeWeight( 3 );
  noFill( );
  rectMode( CENTER );
}

void draw( )
{
   background( 0 );
   drawAllDesigns( spacingList1, colorList );
   //drawAllDesigns( spacingList2, colorList );
   //drawAllDesigns( spacingList3, colorList );
   noLoop( );
}
```

- Copy your `design( )` function from question 1 into the program.
- Copy your `drawAllDesigns( )` from question 2 into the program.
- Modify the signature of your `drawAllDesigns( )` function to accept arguments so it can traverse different arrays to draw the design.
- You may not write three different functions do to this. One definition of `drawAllDesigns( )` must be able to traverse any one of the three `spacingList` arrays.

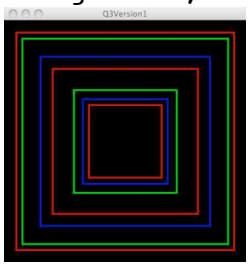
Expected outputs are on the following page:

The following versions of the `draw( )` function produce the following outputs:

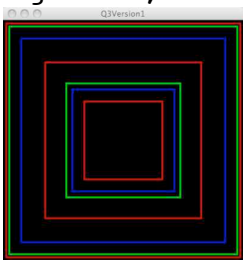
```
drawAllDesigns( spacingList1, colorList );
//drawAllDesigns( spacingList2, colorList );
//drawAllDesigns( spacingList3, colorList );
```




```
//drawAllDesigns( spacingList1, colorList );
drawAllDesigns( spacingList2, colorList );
//drawAllDesigns( spacingList3, colorList );
```




```
//drawAllDesigns( spacingList1, colorList );
//drawAllDesigns( spacingList2, colorList );
drawAllDesigns( spacingList3, colorList );
```



// Question 4 - 15 points

// Copy the following code into a new program:

```
int x, y, length;
int rotatingX, rotatingY;
int diam;
color ul, ur, ll, lr;

void setup( )
{
   size( 400, 400 );
   x = width/2;
   y = height/2;
   length = width/5;
   diam = 33;
   smooth( );
        // red for upper left quadrant
   ul = color( 200, 0, 0 );
        // green for upper right quadrant
   ur = color( 0, 200, 0 );
        // yellow for lower left quadrant
   ll = color( 200, 200, 0);
        // blue for lower right quadrant
   lr = color( 0, 0, 200);
}

void draw( )
{
   background( 0 );
   rotatingX =
       x + int( cos(radians(frameCount*2) ) * length );
   rotatingY =
       y + int( sin(radians(frameCount*2) ) * length );
   coloredBackground( rotatingX, rotatingY);
   stroke( 255 );
   line( x, y, rotatingX, rotatingY);
   fill( 255 );
   noStroke( );
   ellipse( width/2, height/2, 10, 10 );
   ellipse( rotatingX, rotatingY, 20, 20 );
}
void coloredBackground( int rotx, int roty )
{

}
```

Finish the definition of `coloredBackground( int rotx, int roty )`.

The arguments are the location of the rotating circle. The function must use the argument values to determine which quadrant the circle is in and fill that quadrant with the corresponding quadrant color.

Sample output – the four images below are taken at different times during the run of the program: