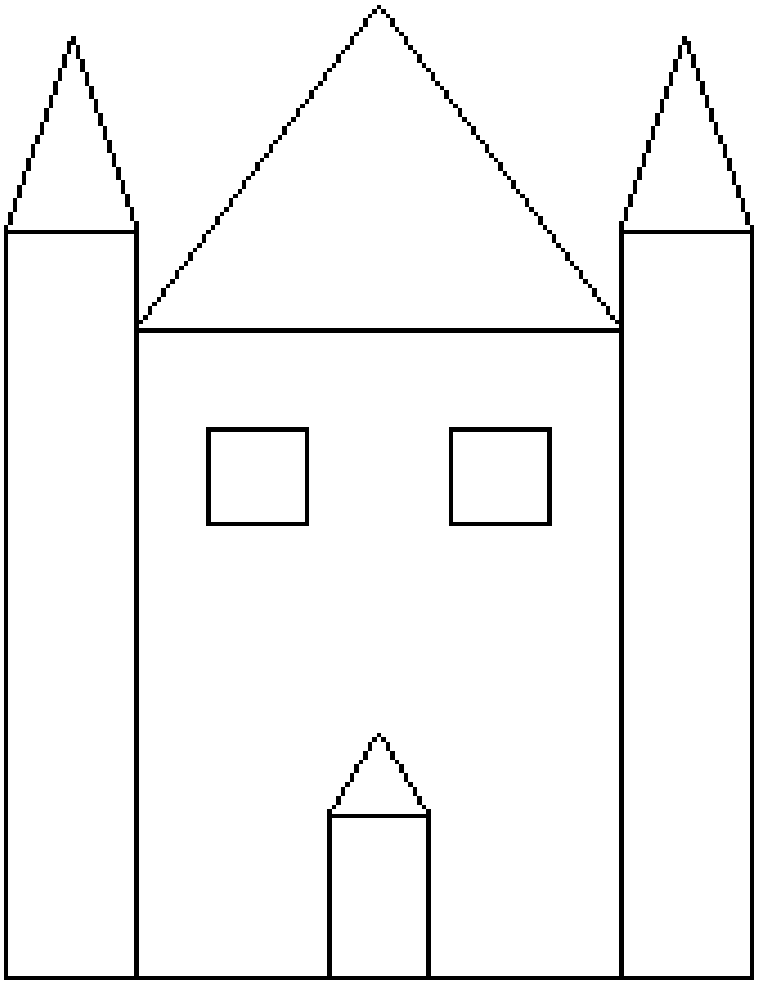
Graphics Examples

Here is the stylized picture of Poe's House of Usher, a drawing exercise from Chapter 7 of the CS106A textbook.



The point of this exercise as given in 106A was further practice with the graphics library and an exercise in top-down design and stepwise refinement. Look the picture, try to decompose the picture into its components, look for opportunities for unification and come up with a reasonable set of functions that can accomplish the job.

We won't go over this code in careful detail, mostly we are giving this to you as an example of the kind of code quality we are expecting in your assignments. Note the use of many #define-d constants that allow you to easily change the house proportions. Take a look at the decomposition into many small well-named functions that each do one task. Note how the functions are generalized through the use of parameters to be useful in a variety of situations. Read the comments that provide clues to the reader about how the parts fit together.

```
/* File: usher.c
 * -------------
 * Program to draw a representation of Poe's House of Usher.
 */
#include "genlib.h"
#include "graphics.h"

/*
 * Constants
 * ---------
 * The following constants control the sizes of elements in the display.
 */

#define HouseWidth 1.5
#define HouseHeight 2.0
#define HouseArch 1.0

#define TowerWidth 0.4
#define TowerHeight 2.3
#define TowerArch 0.6

#define TotalHouseHeight (HouseHeight + HouseArch)

#define DoorWidth 0.3
#define DoorHeight .5
#define DoorArch .25

#define WindowLevel 1.4
#define WindowSize 0.3

/* Function prototypes */

static void DrawBox(double x, double y, double width, double height);
static void DrawTriangle(double x, double y, double base, double height);
static void DrawPeakedBox(double x, double y,
                          double width, double height, double arch);
static void DrawWindow(double x, double y);
static void DrawDoor(double x, double y);
static void DrawTower(double x, double y);
static void DrawMainHouse(double x, double y);
static void DrawHouseOfUsher(double x, double y);


/* Main program calculates where to place lower left corner so that
 * entire house is centered within the graphics window.
 */
main()
{
    double  centerX, centerY;

    InitGraphics();
    centerX = GetWindowWidth()/2;
    centerY = GetWindowHeight()/2;
    DrawHouseOfUsher(centerX - HouseWidth/2 - TowerWidth,
                     centerY - TotalHouseHeight/2);
}
```

```
/*
 * Function: DrawHouseOfUsher
 * Usage: DrawHouseOfUsher(x, y);
 * ------------------------------
 * Draws the complete House of Usher with lower left corner at (x, y).
 * The left tower is drawn based at (x,y), the main house is drawn right
 * next to it, and the right tower is placed on the other side.
 */
static void DrawHouseOfUsher(double x, double y)
{
    DrawTower(x, y);
    DrawMainHouse(x + TowerWidth, y);
    DrawTower(x + TowerWidth + HouseWidth, y);
}




/*
 * Function: DrawMainHouse
 * Usage: DrawMainHouse(x, y);
 * ---------------------------
 * Draws the main part of the house, including the door and windows.  The
 * door is centered and the two windows are balanced on either side.  The
 * lower left corner of the house will be placed at (x, y).
 */
static void DrawMainHouse(double x, double y)
{
    double windowX;

    DrawPeakedBox(x, y, HouseWidth, HouseHeight, HouseArch);
    DrawDoor(x + (HouseWidth- DoorWidth)/2, y);
    windowX = x + HouseWidth/4 - WindowSize/2;
    DrawWindow(windowX, y +WindowLevel);
    windowX += HouseWidth/2;
    DrawWindow(windowX, y + WindowLevel);
}



/*
 * Function: DrawTower
 * Usage: DrawTower(x, y);
 * -----------------------
 * Draws a side tower with lower left corner at (x, y).  Uses
 * the TowerHeight, TowerWidth, and TowerArch constants to set sizes.
 */
static void DrawTower(double x, double y)
{
    DrawPeakedBox(x, y, TowerWidth, TowerHeight, TowerArch);
}
```

```
/*
 * Function: DrawDoor
 * Usage: DrawDoor(x, y);
 * ----------------------
 * Draws the door at (x, y).  Uses the DoorHeight, DoorWidth, and
 * DoorArch constants to set sizes of the various door components.
 */
static void DrawDoor(double x, double y)
{
   DrawPeakedBox(x, y, DoorWidth, DoorHeight, DoorArch);
}



/*
 * Function: DrawWindow
 * Usage: DrawWindow(x, y);
 * ------------------------
 * Draws a single window with the lower left corner at (x, y).  Uses
 * the WindowSize constant for the window width and height.
 */
static void DrawWindow(double x, double y)
{
    DrawBox(x, y, WindowSize, WindowSize);
}



/*
 * Function: DrawPeakedBox
 * Usage: DrawPeakedBox(x, y, width, height, arch);
 * ------------------------------------------------
 * Draws a rectangle with a triangular top.  The arguments are as
 * in DrawBox, with an additional arch parameter indicating the
 * height of the triangle.  This function is a common element in
 * several parts of the picture.
 */
static void DrawPeakedBox(double x, double y,
                          double width, double height, double arch)
{
     DrawBox(x, y, width, height);
     DrawTriangle(x, y + height, width, arch);
}
```

```
/*
 * Function: DrawBox
 * Usage: DrawBox(x, y, width, height)
 * -----------------------------------
 * This function draws a rectangle of the given width and height with
 * its lower left corner at (x, y).
 */

static void DrawBox(double x, double y, double width, double height)
{
    MovePen(x, y);
    DrawLine(width, 0);
    DrawLine(0, height);
    DrawLine(-width, 0);
    DrawLine(0, -height);
}




/*
 * Function: DrawTriangle
 * Usage: DrawTriangle(x, y, base, height)
 * ---------------------------------------
 * This function draws an isosceles triangle (i.e., having two equal sides)
 * with a horizontal base.  The coordinate of the left endpoint of the base
 * is (x, y), and the triangle has the indicated base length and height.
 * If height is positive, the triangle points upward.  If height is
 * negative, the triangle points downward.
 */
static void DrawTriangle(double x, double y, double base, double height)
{
     MovePen(x, y);
     DrawLine(base, 0);
     DrawLine(-base/2, height);
     DrawLine(-base/2, -height);
}
```

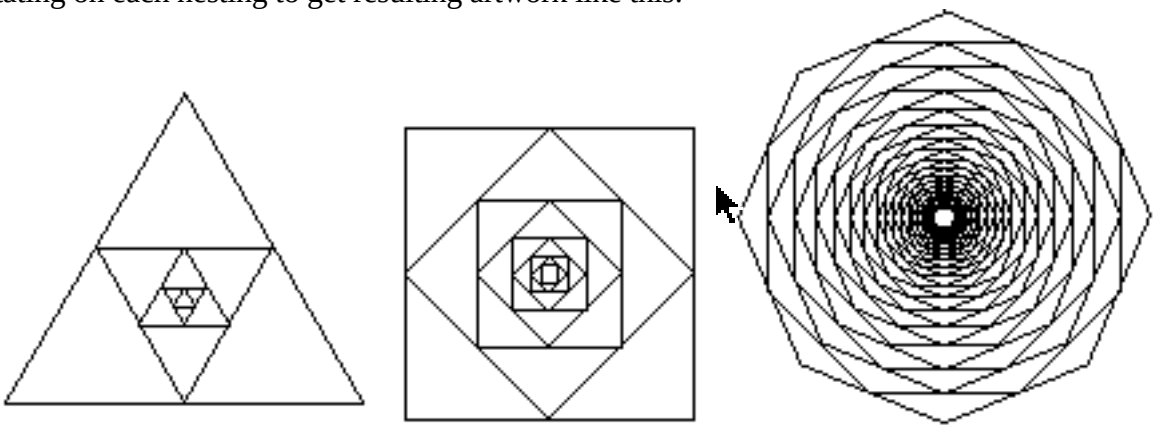
The N-Gon: an exercise in generalization

To construct shapes from the graphics primitives, you might create utility functions: one to draw a rectangle, another for triangles, one for a diamond, and so on. When you think about, you might want to unify these into a more general shape-drawing function that can draw a shape with any number of equilaterial sides. Given a radius and applying some simple geometry, we can create a function to draw an polygon with n equal sides. Now notice a diamond is nothing more than a square on its side (more precisely rotated 45 degrees). By adding a starting angle of rotation, we can further generalize this idea to handle such shapes. Demonstrating generalization to the extreme, a function that can draw any equilateral polygon with any starting angle of rotation:

```
/*
 * Function: DrawNGon
 * Usage: DrawNGon(cx, cy, numSides, radius, startAngle);
 * --------------------------
 * Draws a n-sided equilateral polygon centered around the point (cx, cy)
 * and of the size such that it would be exactly inscribed in a circle
 * of the specified radius.  The starting angle determines where the n-gon
 * begins. (i.e. a "corner" is placed at exactly that angle)  From there,
 * the function will draw segments of the polygon in counter-clockwise
 * direction until all sides have been drawn.
 */
static void DrawNGon(double centerX, double centerY, int numSides,
                     double radius, double startAngleInRads)
{
   double angle, angleIncr;
   int i;

   angleIncr = (2*PI)/numSides;
   angle = startAngleInRads;
   MovePen(centerX + radius*sin(angle),centerY + radius*cos(angle));
   for (i = 0; i < numSides; i++) {
        DrawChord(radius, angle, angle + angleIncr);
        angle += angleIncr;
   }
}


/*
 * Function: DrawChord
 * Usage: DrawChord radius, startAngle, stopAngle);
 * --------------------------
 * Helper function for DrawNGon.  Given the pen is position at angle
 * startAngle on a circle of specified radius, this function will draw a
 * chord from that position to the  position of the stopAngle on the same
 * circle, therefore drawing a chord on the circle from start to stop.
 */
static void DrawChord(double radius,  double start, double stop)
{
     double startX, stopX, startY, stopY;

     startX = radius*sin(start);
     stopX  = radius*sin(stop);
     startY = radius*cos(start);
     stopY  = radius*cos(stop);
     DrawLine(stopX - startX, stopY - startY);
}
```

With our general shape routine in place, we can construct any number of nice wrapper functions that provide a simpler interface for the common shapes. Functions like DrawDiamond or DrawTriangle would be simple covers that pass through the correct arguments to the multi-purpose DrawNgon function. Therefore, a client's use of our functions would be cleaner and more readable.

Lastly, just for fun, we can create a function to draw spriograph-like nested polygons by constructing a function that repeatedly calls DrawNgon in a loop while slightly shrinking and rotating on each nesting to get resulting artwork like this:



n = 3 *n = 4* *n = 8*

```
/*
 * Function: DrawNestedNGon
 * Usage: DrawNestedNGon(cx, cy, numSides, radius);
 * ------------------------
 * Draws a set of nested n-sided equilateral polygons centered around
 * (cx, cy).  The outermost polygon is of size according to radius and
 * has starting angle of 0.  The nextmost polygon is rotated half the distance
 * of one side and its radius decreased so that its vertices hit the
 * halfway points of the outer polygon edges.  It continues nesting like
 * this until it gets to the center and the radius becomes too small to
 * draw any more shapes.
 */
static void DrawNestedNGon(double cx, double cy, int numSides, double radius)
{
     double angle, angleIncr;

     angle = 0.0;
     angleIncr = PI/numSides;
     while (radius > 0.05) {
         DrawNGon(cx,cy,numSides,radius,angle);
         radius = cos(angleIncr)*radius;
         angle += angleIncr;
     }
}
```

How many IBM repairmen does it take to change a flat tire? Five. One to jack up the car, and four to swap tires until they find the one that is flat.