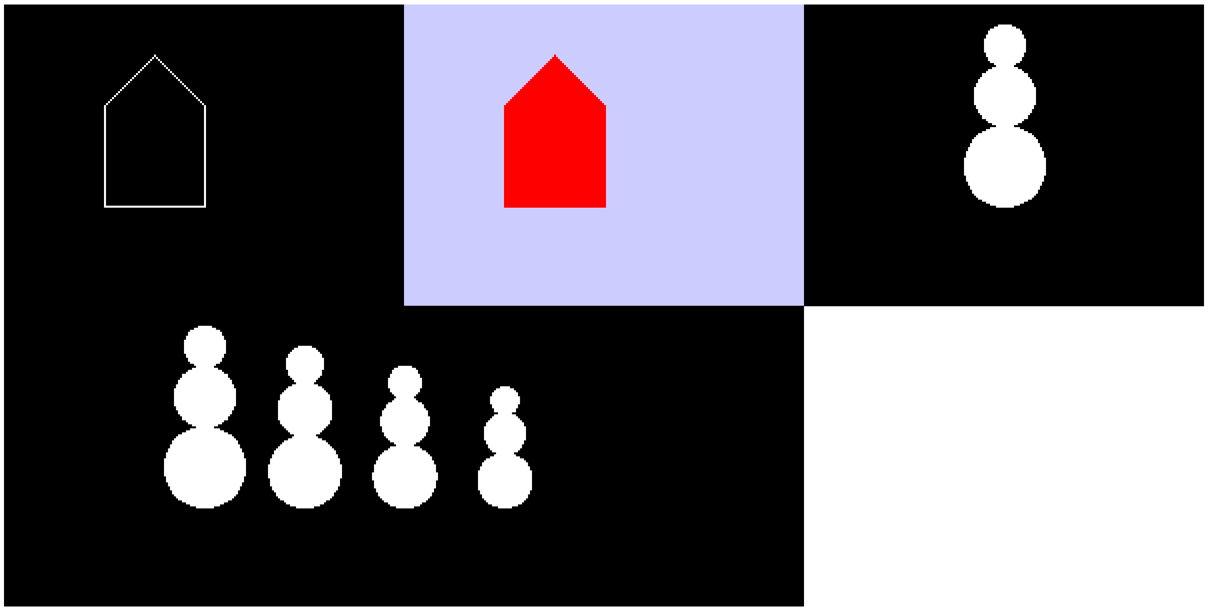
CS16, Spring 2010

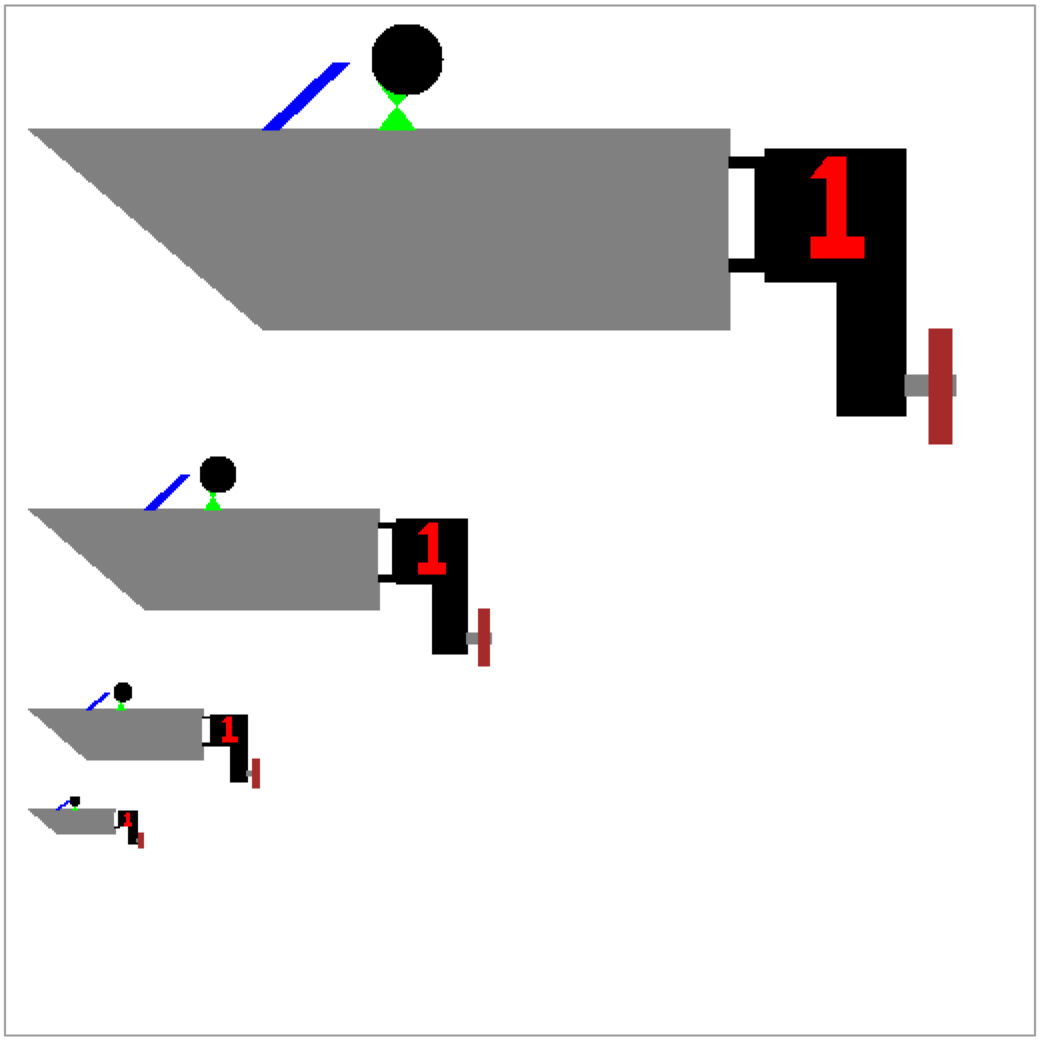
proj0: ("project 0") Simple graphics

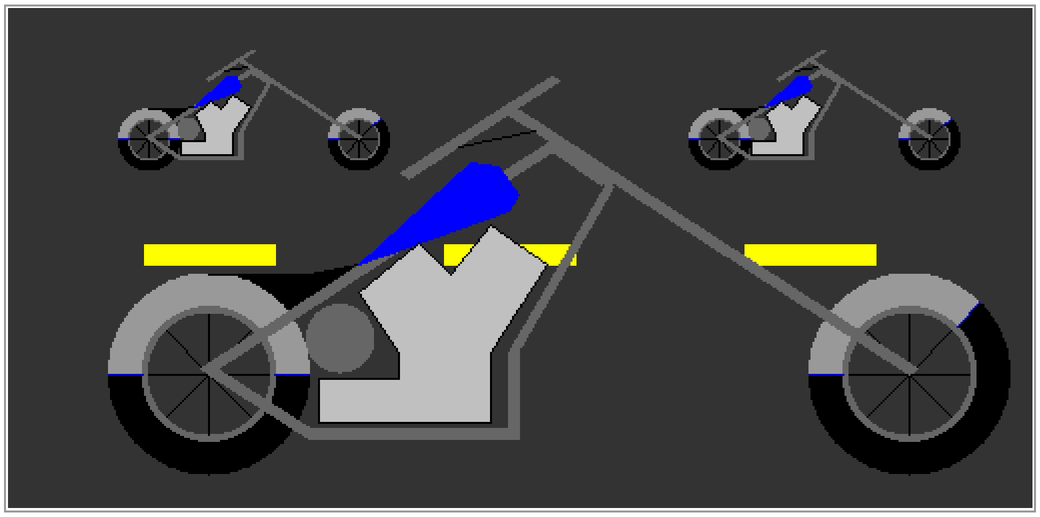
Introduction

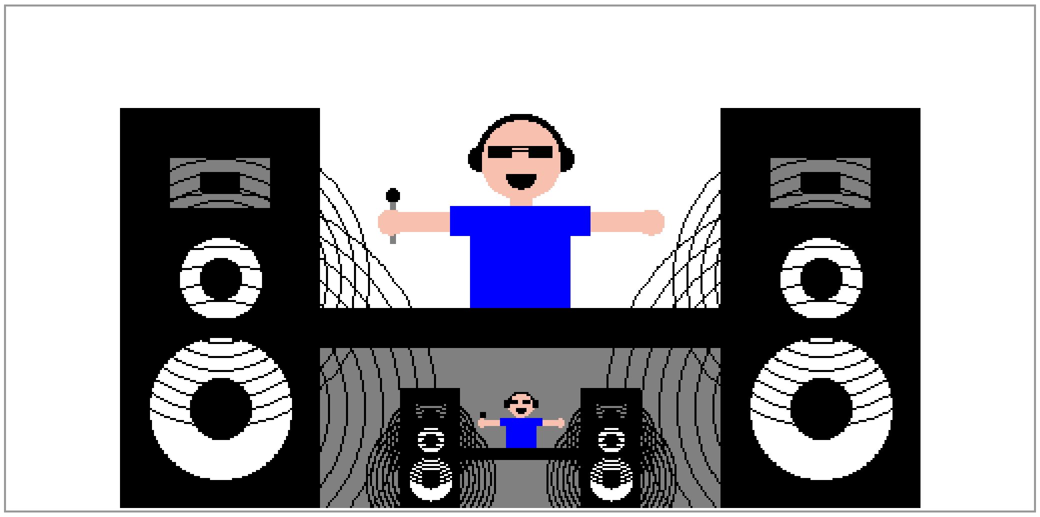
In this project, you'll learn how to write C code that can make drawings like these (made by the instructor):

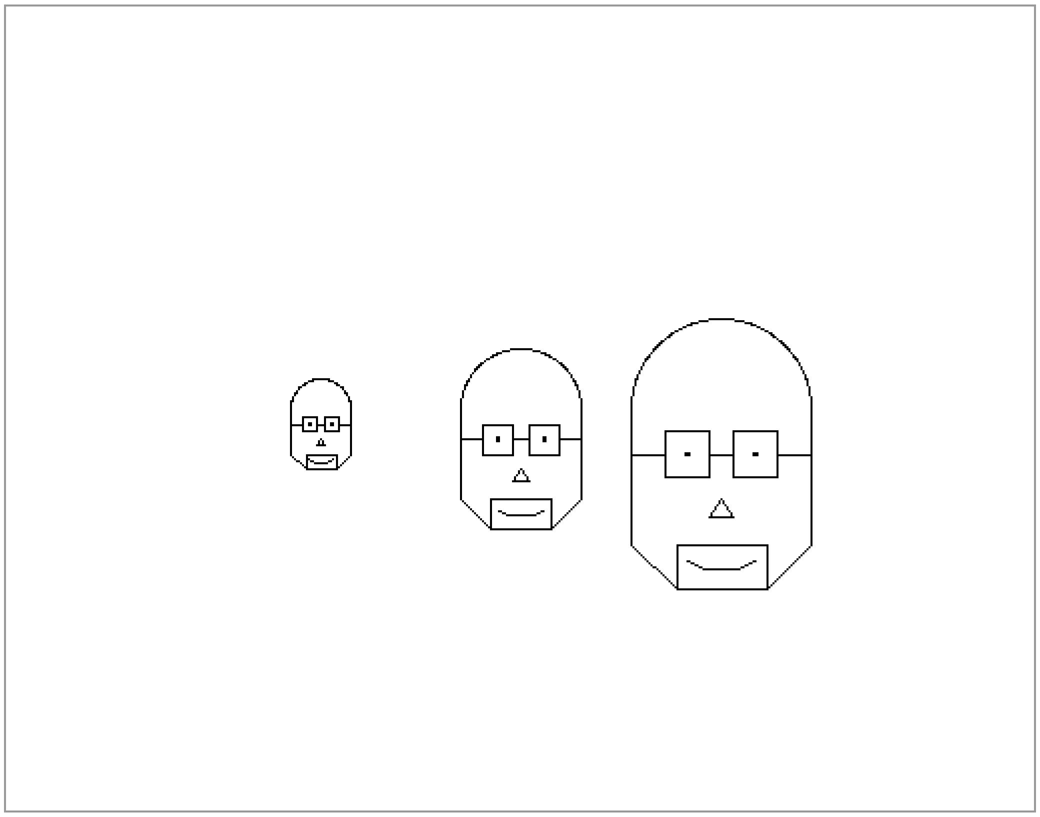


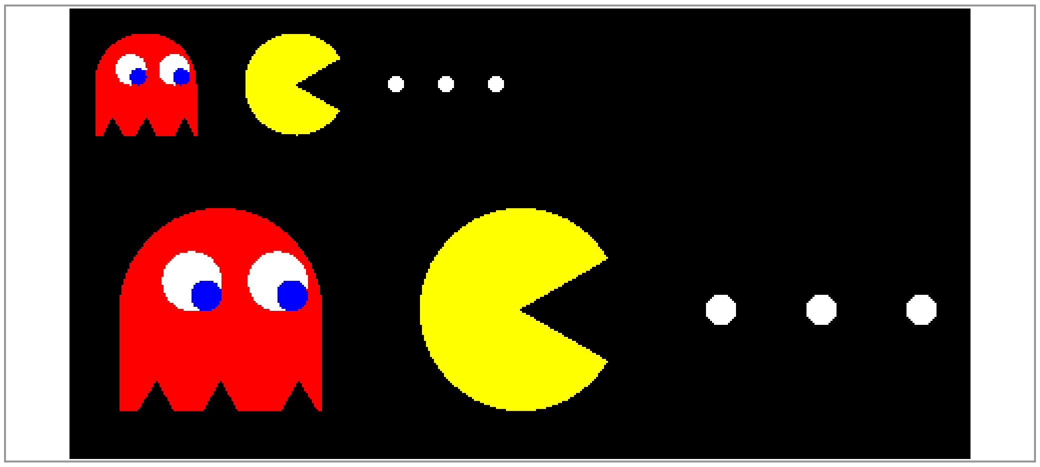
And drawings like these (the following are actual student drawings from CS16 Fall 2009):

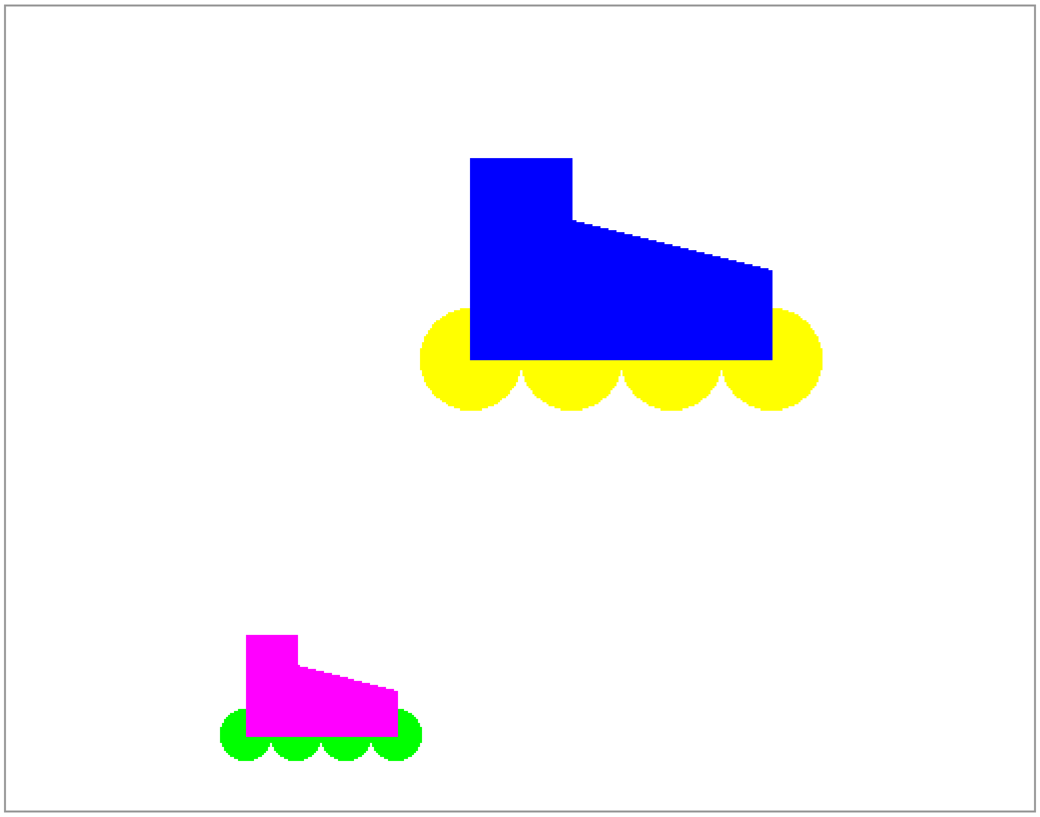

1

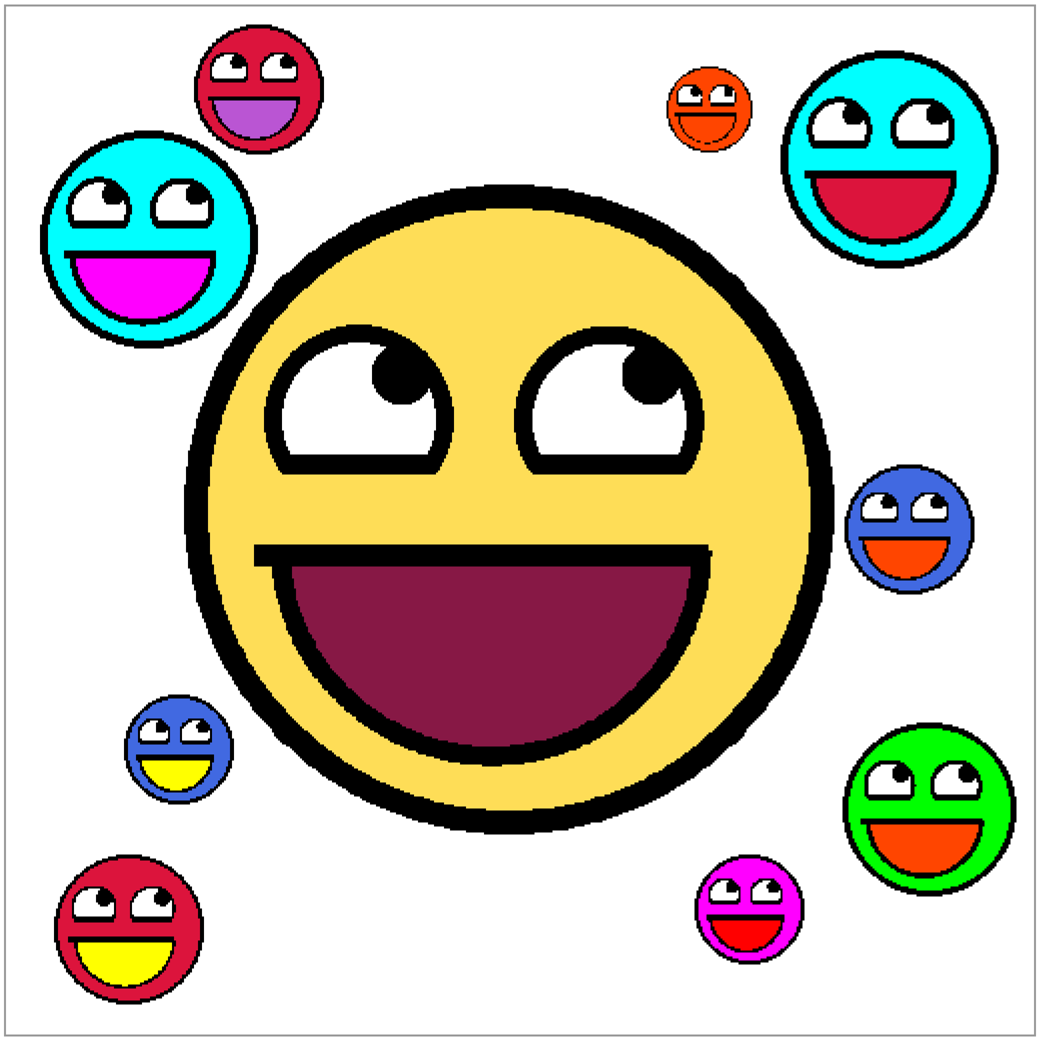












For more actual student drawings, visit: the drawings link on the course web page

What all of these drawings have in common is that they include some kind of parameterization—that is, the same drawing (e.g. the awesome face), can be drawing at:

- different locations on the screen
- with different sizes (in this case, the size is probably the radius of the face
- with different face and mouth colors.

This is essentially an exercise in designing functions. You need to:

- decide what you want to draw
- decide what will be parameterized:

  - at a minimum, this should be location in the drawing, and some aspect of size
  - you might include other parameters at your discretion.
- create a drawing function with those parameters
- invoke it several times in a main program to create your drawing

You'll be given some basic tools to draw in either black and white, or color:

- lines
- boxes (outline or filled)
- polygons (outline or filled)
- circles (outline or filled)

We can also discuss, as student interests dictate, how to extend those routine to do other interesting things like

- drawing stars with any number of points, in various styles
- drawing arcs and curves of various kinds

There's a program called ourDrawing.c—that's where you'll put your code for this project. All the other files—you pretty much shouldn't have to make any changes to those.

In ourDrawing.c, you'll use the functions provided in the other files to make your drawing.

What you draw is up to you—as we'll explain in lecture.

The code that does the basic underlying drawing of lines, boxes, etc. is provided for you, as well as code that copies your file to the web. However, what is good is that the vast majority of that code is code that you **already have the capacity to understand** at this point in your journey with the C programming language. There isn't time in a 10 week course to guide you the process of developing all that code, but we can go through some of it in lecture to explain how it works—and as time permits, we will.

Goals for this project

By the time you have completed this project, you should:

- Have some comfort with working with larger software projects
- Have an appreciation of how computing can be fun and expressive

Prior Skills/Knowledge Needed

Before starting this project, you should have completed the labs up through lab06. In particular, you should be comfortable working with arrays, structs, arrays of structs, and passing structs and arrays of structs to functions.

This is a pair programming project

Work with the same partner you had in lab07/lab08—or if he/she is not available for some reason, see your TA and/or instructor about getting a new pair partner assigned.

If you do work with a **new** partner, please complete a new version of W01, and post your new assignment to the forum on Gauchospace.

Step by Step Instructions

Step 0: Get together with your pair partner, and decide whose account you'll work in

If he/she is not available, and you are assigned a new partner, then complete a new version of worksheet W01 (html, pdf), including posting to the "Updated pair partners for proj0 forum" on Gauchospace.

Remember: don't share passwords. Instead, use scp or email to share files with each other at the end of each work session. (See previous labs for details.)

It is only necessary for one member of the pair to submit—but you are BOTH responsible for seeing that this HAS been done and that both of your names are in the ourDrawing.c file.

Step 1: Log on to CSIL, create ~/cs16/proj0 and copy the files for this project

Log on to CSIL, create `~/cs16/proj0`

The files for this project can be found here:

- http://www.cs.ucsb.edu/~pconrad/cs16/10S/projects/proj0/code

And here:

```
~pconrad/public_html/cs16/10S/projects/proj0/code/*
```

You can use the same techniques described in earlier labs to copy those into your ~/cs16/proj0 directory.

Step 2: Listen to the presentation in lecture.

The following material will be covered in lecture either before, or just as this project is being assigned.

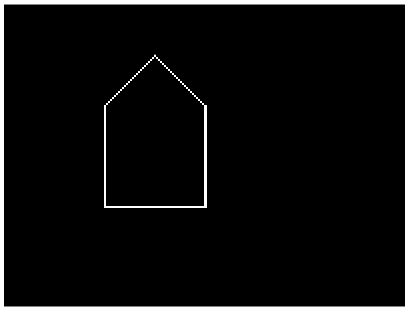
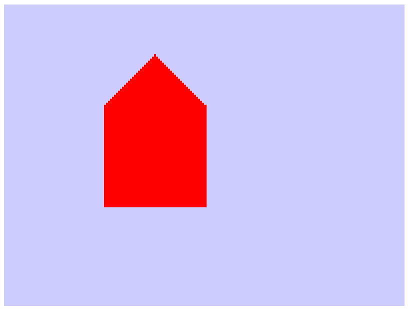
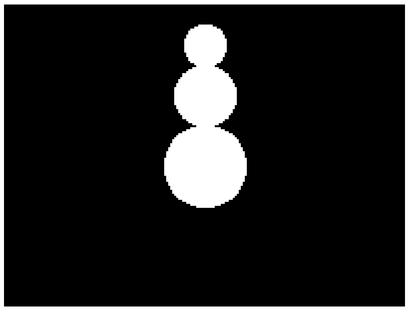
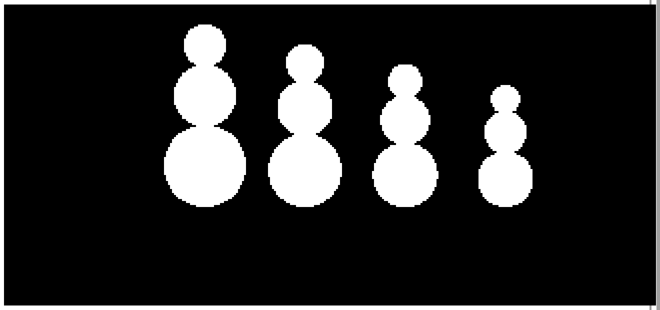
What you are given is a collection of code containing three libraries:

- **shapeFunctions.c:** a library of C functions for manipulating shapes, called shapeFunctions.c
  - many of these come from lab06
- **tdd.c:** a library of functions for test-driven development
  - these are all functions we've seen in previous labs
- **drawingFunctions.c**: a library of functions for drawing simple graphics
  - these mostly rely on concepts we've seen before
  - there are a couple of new items, which we'll briefly review
    - use of enum
    - use of the switch statement
    - writing files to disk (which is very similar to reading files)

You'll also see header files that support these C files:

- **shapes.h** contains struct definitions for struct Point, struct Circle, struct Polygon, etc.
- **drawing.h** contains struct definitions for making drawings
- **tdd.h** contains function prototypes for the tdd.c functions
- **shapeFunctions.h** contains function prototypes for the functions in shapeFunctions.c
- **drawingFunctions.h** contains function prototypes for the functions in drawingFunctions.c

You'll also see some programs that contain main programs that use these libraries:

file	what it does	sample image
testMain.c	contains only test cases, mostly for the functions in shapeFunctions.c	n/a
drawHouse.c	draws a black and white house, using simple lines	
drawFilledColorHouse.c	draws a filled house in color	
drawSnowman.c	draws a single snowman in black and white	
drawManySnowmen.c	shows how to make drawing a snowman be a function, so you can draw several snowmen of different sizes	

drawFlags.c	shows how to draw a simple tri-color flag (in this case the GermanFlag) using filledBoxes. Another example of abstracting a drawing into a function	
ourDrawing.c	draws whatever you want (at the moment, it's blank!)	

We'll also go over what happens when you type "make". This week there is a Makefile that controls what happens.

You'll need to understand just two things:

- typing "make"
- typing "make clean"

How to use the code:

To start off, type "make" (all by itself) at the Unix command prompt.
When you do you'll see output indicating that

- several C files are being compiled into programs
- some of these files are being run to produce images with .pbm extensions
- some commands are being run to convert those .pbm files into .gif files
- those .gif files are being copied to a web page
- some test code is being run.

Here's an example:

```
-bash-3.2$ make
gcc -c testMain.c
gcc -c drawingFunctions.c
gcc -c shapeFunctions.c
gcc -c tdd.c
gcc -lm -Wall -g  drawingFunctions.o shapeFunctions.o tdd.o testMain.o -o
testMain
gcc -c drawHouse.c
...
```

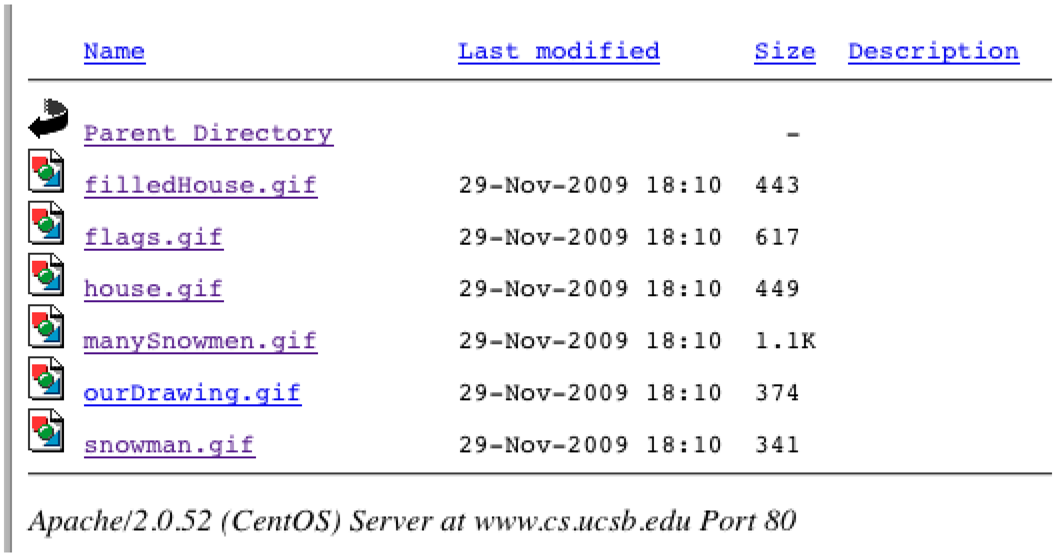
(many lines of output deleted... full transcript available in make.output.txt)

```
All tests passed!
-bash-3.2$.
```

The key line of output to look for is this one—except it will have your username, not jsmith—unless your username happens to be jsmith:

```
Visit http://www.cs.ucsb.edu/~jsmith/cs16/proj0 to see your pics
```

Go to that website, and you'll see something like this:



```
     Name                     Last modified       Size  Description
   ----------------------------------------------------------------
     Parent Directory                                -
     filledHouse.gif          29-Nov-2009 18:10   443
     flags.gif                29-Nov-2009 18:10   617
     house.gif                29-Nov-2009 18:10   449
     manySnowmen.gif          29-Nov-2009 18:10   1.1K
     ourDrawing.gif           29-Nov-2009 18:10   374
     snowman.gif              29-Nov-2009 18:10   341
   ----------------------------------------------------------------
   Apache/2.0.52 (CentOS) Server at www.cs.ucsb.edu Port 80
```

The drawings in that directory are the ones created by the code in the files below:

GIF file	C file
filledHouse.gif	drawFilledColorHouse.c
flags.gif	drawFlags.c

house.gif	drawHouse.c
manySnowmen.gif	drawManySnowmen.c
ourDrawing.gif *This is where your drawing will appear!*	ourDrawing.c *This is where you put your code!*
snowman.gif	drawSnowman.gif

The "make clean" command:

If your files get messed up and you want to start over, the "make clean" command will get rid of all the extra files in your current directory so that you can type "make" again from scratch.

What you need to do:

1. Look at the code in the C files shown above for examples of how to make drawings.
   - Compare the code to the pictures
   - Notice how the code draws using lines, polygons and circles
2. Together with your lab partner, decide what you want to draw
   - It should be something you can draw using lines, polygons, boxes and circles.
   - Plan out your drawing on paper.
3. Register your drawing on Gauchospace (see further instructions below in Step 5)
4. Work on the code for your drawing (see further instructions below in Step 6)

Step 5: Registering your drawing on Gauchospace

You need to log into Gauchospace and go to the discussion form near the top of the main page labeled 'register your proj0 drawings here'.

Look to see if anyone else has already registered the same item that you want to draw. You need to draw something that has not already been chosen in order to get full credit.

You may make your item unique by adding a unique "twist"—e.g. if they are drawing a "car", you can't just draw a "car", but you can draw:

- a car with a banana for a hood ornament
- a police car with red and blue lights
- a car with its hood up and steam rising from the engine.

Step 6: Working on your drawing

To make your drawing, you put code into the file ourDrawing.c.

The first step is to set up the function call to initDrawing with the correct parameters:

initDrawing(&d, *drawing_type*, *width*, *height*, *background_color*);

The choices you'll need to make here are:

1. A black and white, or color drawing?

   If you choose black and white, then you'll use DRAWINGTYPE_BW as your drawing_type, and either BW_WHITE or BW_BLACK as the background color.

   If you choose color, you'll use DRAWINGTYPE_COLOR as your drawing_type, and either COLOR_WHITE, COLOR_BLACK, or one of the other choices in drawing.h such as COLOR_BLUE, COLOR_YELLOW, etc. as your background color.

   You can also use any six digit hexadecimal web color code, preceded by 0x, for example: 0xBDB76B is a shade of khaki.

2. The size of your drawing?

   The maximum width and height are currently 512 and 512, defined in drawing.h. You may experiment with modifying the values in drawing.h if you want to make a larger drawing—but keep in mind that you may run into disk space issues if the drawing gets too big.

Q: There are two functions, `drawShape1()` and `drawShape2()`. Does that mean we need to have two shapes in our drawing?

A: Not necessarily—It is ok to have just one shape if it is "reasonably complex".

As an example: If your one shape is just a bowling ball—i.e. a circle with three little circles inside for the holes—you should probably make a second shape.

But if your shape is "the Santa Barbara Mission", complete with towers and colums, then you only need one shape.

However, if you have a complex shape, you should break it down into smaller functions. For example, if you draw a skateboard you might have a `drawSkateboard()` function to draw the board, and that function might call `drawWheel()` multiple times to draw the wheels (e.g. twice if it is a "side view").

Finally, realize that `drawShape1()` and `drawShape2()` are just placeholder names—you should rename them to whatever you are actually drawing.

If you aren't sure what is complex enough to be "reasonably complex", ask your instructor or TA.

The next step is to add code into one of the two functions called drawShape1 and drawShape2. In each case, you should rename the function to a more sensible name such as:

- drawCar
- drawAppleTree
- drawSkateboard
- etc.

Decide what parameters the drawing functions will take.

At a minimum, it should take a 'reference point', and at least one dimension—a width, height, or radius—something that allows you to draw your picture at different scales.

- The reference point is some point that everything else in the drawing is calculated from.
    - It could be the center of the drawing, or the upper left corner, or the lower right corner, or the center bottom, for example
    - What you choose as your reference point depends on the nature of what you are drawing
    - For an ice cream cone, it might be the bottom of the cone.
    - For a sword, it might be the place where the handle meets the blade.

- You should allow the user to specify at least one dimension for the picture so that you can draw it at different sizes.
  - Depending on the nature of the drawing, this might be height in pixels (e.g. for Storke Tower), or radius in pixels (e.g. for a Pizza), or width in pixels (for a flag).
  - For some things, it may be more sensible to calculate additional dimensions from those already given, e.g.
    - calculate width automatically from the height (e.g. the width of Storke Tower might be 1/8 or 1/10 of the height,
    - calculate height automatically from width: (e.g. the height of the US flag is the width/1.9)
    - the length of the minute hand on a clock is 90% of the radius
  - For other things, it may be useful to specify both width and height
    - That allows you to make, for example, tall skinny houses and short fat houses.

What if my drawing has hard coded points?

That's ok as a starting point.

Hard coded points can easily be converted into a formula with a little extra work—once you get the hang of it, it isn't that tough.

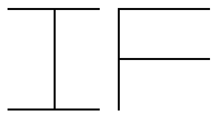
A drawing with hard coded points is also ok for partial credit.

What if I don't *want* my drawing to have "two" of something on it at different sizes?
Or what if my converting my complex drawing from hard coded points is going to be a major pain!

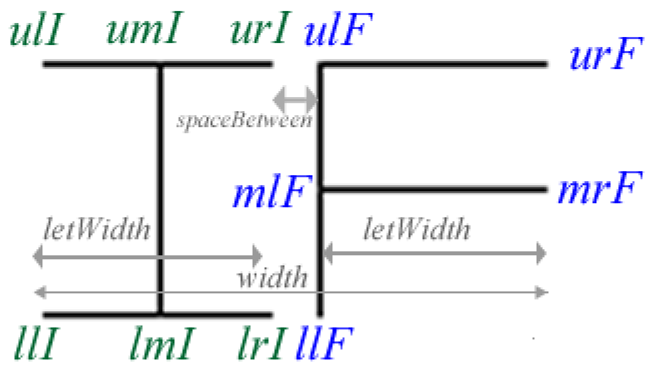
If you already have a drawing that you are very proud of, and you don't want to mess up the artistry, but you still want full credit, here's a work around for you.

This is also a way out if you are not feeling very creative and just want to get this assignment done.

Make a drawing function to draw a drawing consisting of just the initials of your first name, and that of your pair partner. For example, if Ian Smith were to partner with Frieda Jones, their drawing might be IF, and might have points like this:



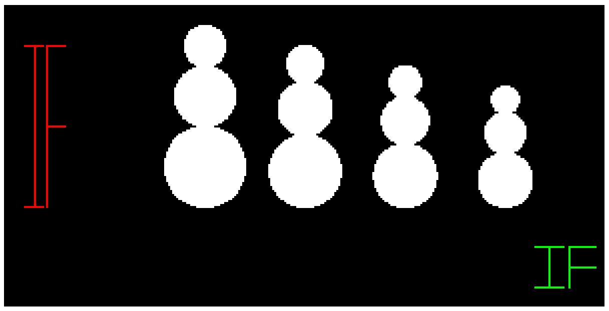
We can label these points like this:



And then draw this with code like the following:

```
void drawIF(struct Drawing *d,
            struct Point ul, // upper left, or center bottom, or whatever
            double width,
            double height,
            int color)
{
  double letWidth = width * 0.45; // width of each letter
  double spaceBetween = width * 0.1; // space between letters
  struct Point ulI, umI, urI, llI, lmI, lrI; // I upper left, middle, right,
etc.
  struct Point ulF, mlF, llF, urF, mrF;// F upper, middle, and lower left,
etc.


  // set points of I across the top
  initPoint(&ulI, ul.x, ul.y);
  initPoint(&umI, ul.x + letWidth/2.0, ul.y);
  initPoint(&urI, ul.x + letWidth, ul.y);



  // set points of I across the bottom
  initPoint(&llI, ul.x, ul.y + height);
  initPoint(&lmI, ul.x + letWidth/2.0, ul.y + height);
  initPoint(&lrI, ul.x + letWidth, ul.y + height);



  // draw the I
  drawLine(d, ulI, urI, color); // across the top
  drawLine(d, llI, lrI, color); // across the bottom
  drawLine(d, umI, lmI, color); // down the middle


  // set points of F down the left hand side, top to bottom
  initPoint(&ulF, ul.x + letWidth + spaceBetween, ul.y);

etc...

}
```

With this in place, you can now "tag" your drawing with two tags of different sizes and colors, to create an effect something like this:



With function calls like these:

```
drawIF(&d,makePoint(10,20),20,80, COLOR_RED);

drawIF(&d,makePoint(265,120),30,20, COLOR_GREEN)
```

If you do this, then even if your main drawing is hard coded and can't be scaled, you'll get full credit for having created at least one drawing that can be scaled and relocated (i.e. your tag), and you wont have to mess up the artistic integrity of your main drawing.

Why having a drawing in two places at different sizes is so important

The reason it is so important to have a drawing element that appears at different places and with different sizes is that this shows you really understand how to apply the idea of **abstraction**—which is one of the central ideas in Computer Science.

Finishing up

You are ready to move on to scripting and submitting when:

- Your drawing appears on your web page at **http://www.cs.ucsb.edu/~yourusername/cs16/proj0/ourDrawing.gif**
- Your drawing has at least one element that
    - appears at two different places
    - at two different sizes,
    - as a result of two calls to the same function with different parameters

Step 7: Script and submit

To script and submit, create a script proj0.txt in which you:

- cd into your ~/cs16/proj0 directory and type pwd to show you are there
- type **make clean**
- type **make**
- type **make clean** again
- exit the script

Check the web site one last time to make sure that your drawing appears there under the name **http://www.cs.ucsb.edu/~yourusername/cs16/proj0/ourDrawing.gif**.

Then, submit the contents of your proj0 directory via:

```
turnin proj0@cs16 proj0
```

Evaluation and Grading (300 pts total—but counts double, i.e. 600pts)

* Note that since Gauchospace currently has a maximum point value of 300, we'll compute a grade out of 300 points, and then enter it in Gauchospace twice.

- (30 pts) Registering drawing on Gauchospace so that it does not duplicate another drawing.
- (30 pts) Scripting and submitting according to instructions
- (40 pts) Submitting on time (note: after midnight 06/01, NO CREDIT FOR THIS PROJECT)
- (100 pts) Drawing something interesting with code.
    - 50 pts is at risk here for style issues, e.g. choice of variables names, commenting, etc.
    - 50 pts is for code that works—i.e. that draws something.
- (50 pts) Drawing appears on web site (this should come for free)
- (50 pts) Code and drawing contains at least one item that appears at different positions and sizes.

Extra Credit Opportunity

We will have a competition for the best pictures—there will be prizes for the most original, most artistic, and the best code. Some of these will be judged by the students in the class, others by the course teaching staff (instructor and TAs.) The number of points to be awarded in each category will be determined later—but what is certain is that to be eligible for the extra credit, you MUST submit your project on time, i.e. no later than midnight Wednesday 05/26.

Due Date

- Midnight Wednesday 05/26—you must meet this deadline to be eligible for the extra credit competition.
- Accepted until midnight Tuesday 06/01 with -40 deduction.
- NOT ACCEPTED after midnight Tuesday 06/01
    - If you aren't done by 11pm 06/01, go ahead and SUBMIT WHAT YOU HAVE—even if it isn't perfect yet.
    - Getting some points (partial credit) is better than getting a zero.

Advanced stuff:

For students that really want to go "beyond the call of duty": here are some extra things you may like to experiment with.

- Recursive Fill: recFill.c

    - Note: if you work with recFill, you may need to increase the available stack space by typing:
      `ulimit -s 1000000` at the shell prompt before running your program.
- Color gradients: colorFunc.c, gradientExample.c, testColor.c, example image

---