CS/ENGR 101: Gameboy Session
Fall 2006 – Day 4

This handout is a C programming lab exercise that introduces functions in the C programming language.

For this lab exercise, use the VisualHam project from the last session. To do this, click on the File menu and choose Open Project. In the open project dialog box, browse to your project using the I: drive letter (rather than through the MyDocuments network share name).

Functions
A function is a subprogram. They are used to encapsulate computational ideas so that we don’t have to keep repeating code. When a function is used, we say it is called, and the subprogram that uses the function the caller. When a function is called, the caller can pass data to it called the arguments, and the program jumps to the function code and executes it. When the function is finished, it can return a result, and the program jumps back to where the function was called and resumes executing.

As we saw with library routines, a function call is just the name of the function and a comma-separated list of arguments you want to pass to the function in parentheses. An argument can be a constant, an expression, or a variable. If the function returns a result, you need to assign it to a variable to save it. An example of a function that does not return a value is `ham_PutLine`, which we called using:

```
ham_PutLine (x1, y1, x2, y2, COLOR_RED);
/* arguments are 4 variables and 1 constant */
```

An example of a function that does return a value is `rand`, which we called using:

```
red = rand() % 256;
/* rand is called with no arguments */
/* Its result mod 256 is stored into a variable */
```

Users can define their own functions when they want to be able to do things more than once with different data. A function definition looks like a small program. The header defines the type of the returned result (void if no result), the name of the function, and the types and names of the arguments. For example, clearing the Gameboy screen might be something that needs to be done in more than one place in a program. And we might want to use a different background color each time. We can define a function that does this as follows:
/* Function header syntax: */
<return type> <name> (<type> <name>, ...) */

/* ClearScreen does not return a value and has 1 argument */
void ClearScreen (int color)
{
    int count; /* declare local variables */
    for (count = 0; count < 160; count++)
        ham_PutLine (0, count, 239, count, color);
}

For this class, function definitions appear before the main program definition.

With this function, we can replace the screen-clearing looping code with a call to the function. Here are some examples:

    ClearScreen (COLOR_BLACK);
    ClearScreen (bgcolor);

The first call will cause the screen to be cleared with a black background. The second call will cause the screen to be cleared with whatever color the variable bgcolor holds.

Exercise
For today’s exercise, modify your program from last session to do the following:

1. Add the ClearScreen function above to your program and replace the screen-clearing code in the main program with calls to ClearScreen.
2. Write a RandomColor function that has no arguments and returns a random color value. Basically, this will encapsulate the idea of getting random values for red, green, and blue variables, and using the RGB macro to create a color value. Replace the random color code in the main program with calls to RandomColor.
3. Instead of choosing a random background color, cycle through 4 or 8 different colors that are changed each time the A button is pressed. The idea would be to add a variable that indicates which color is currently being used so you know which color comes next. When the button is pressed, update this variable and the background color variable (bgcolor) to the appropriate next value.