CS/ENGR 101: Gameboy Session
Fall 2006 – Day 2

This handout a C programming lab exercise that introduces library function calls, variables, assignment, expressions, and repetition constructs in the C programming language.

For this lab exercise, create a new VisualHam project. Recall the steps for doing this are:

1. Click on the File menu and choose New, then Project. In the new project dialog box, select [C] Empty, type in a project name (without spaces, e.g., day2), and specify a place on your network drive space for a location (e.g., I:\cs101\day2). Click OK.

2. Double-click on the main.c entry in the left panel to bring it up in the main edit window.

Library function calls
As noted before, there a number of computations we might like a program to do that we would not want to write ourselves or that have already been written that we want to reuse. The C programming language comes with a number of libraries that are collections of these computations. Each computation is put into a function and given a name. Functions in C may return a result (that is, the call evaluates to the function's return value) or they may be used to have some action done like the Gameboy graphics functions ham_Init and ham_PutLine.

To tell the compiler we want to use such a function, we must include its library header file. For example, we include mygba.h so we can use the Gameboy graphics functions. Another useful library is the “standard library” in stdlib.h. Once the library header file has been included, we can use any function in the library. For example, the srand and rand functions from the standard library. These functions provide an interface to a pseudo-random number generator. A pseudo-random number generator produces a sequence of numbers that appears to be random. A generator requires an initial value called a seed to start the generation process. Thereafter, each random number produced is used in the computation of the next random number. Using the same seed produces the same sequence, while different seeds produce different sequences. The function srand is used to seed the generator by calling it with one argument, the seed. The function rand returns the next random number and is called with no arguments. The numbers returned are integers between 0 and a system-defined constant RAND_MAX.

Variables, assignment, and expressions
So far, we have only used literal values like 0 and 239, and defined constants like COLOR_RED. Usually, we want to compute a value and save it for use in our program. To do this we need to declare a variable and assign it a value. A C variable declaration consists of two parts, a type and a name, and causes the compiler to create a memory location with the specified name to hold a value. The type of a variable specifies the kind of values the variable can hold, such as int for integers or double for real numbers. The C syntax for declaring variables is as follows:
int x1;       /* one variable: <type> <name>; */
int y1;
int x2, y2;   /* list of variables: <type> <comma-separated names>; */

To store a value into a variable, a value is assigned to the variable using an assignment statement. The value may be a literal, a variable or constant, a function call, or an expression. The C syntax for assignment is as follows:

/* syntax: <name> = <value>; */

x1 = 0;      /* assign a literal */
y1 = x1;     /* assign a variable */
x2 = rand(); /* assign a function call */
y2 = y2+1;   /* assign an expression */

Expressions are computations that combine values using various operators. The arithmetic operators are addition (+), subtraction (−), multiplication (∗), division (/), and remainder (%). They have standard mathematical precedence and associativity that may be overridden using parentheses.

Note that the variable being assigned may be used in the value expression. Since this is common in programming, C has special operators that combine computation and assignment. For any operator <op>, there is a operator <op>=. A statement <variable> <op>= <value>; is equivalent to the statement <variable> = <variable> <op> <value>;. Some examples are shown below:

x1 += 1;     /* Same as: x1 = x1 + 1; */
y2 *= 2;     /* Same as: y2 = y2 * 2; */

Since adding and subtracting 1 is extremely common in programming, C has unary operators ++ and -- that increment by 1 and decrement by 1, respectively. Some examples are shown below:

x2++;       /* Same as: x2 = x2 + 1; */
y1--;       /* Same as: y1 = y1 - 1; */

**Repetition constructs**

There are several repetition constructs in C. We will look only at two. The first repetition construct is the while statement. A while loop is used when you don't know the exact number of times a loop body is executed. Instead the loop body repeats as long as a condition is true. A condition is an expression that returns a true or false value. Typically, the condition depends on one variable's value that we call the loop-control variable. However, for the Gameboy main program, we want the loop body to loop forever, so there is a constant value TRUE used as the loop condition. Since this constant never changes value, the loop runs forever. The C syntax for a while loop is as follows:
while (TRUE)  /* Header syntax: while (<condition>) */
{
   /* curly braces surround the loop body */
}

The second repetition construct is the for loop, used when you know how many times you want to repeat. It also is called a counter-controlled loop. The loop-control variable is a counter that is incremented each time the loop body is executed. The loop repeats as long the counter is less than the number of times the body is to be repeated. The C syntax for a for loop is as follows:

```c
int counter;  /* Declare the counter variable */

/* Header syntax: for (<init counter>; <condition>; <increment>) */
for (counter = 0; counter < 10; counter++)
{
   /* curly braces surround the loop body */
   /* loop body is repeated 10 times */
}
```

Exercise

For today’s exercise, we will write a "screen saver" program that draws lines in random places. Here are some ideas to consider.

1. Write code that will set the background to one color. If we use, `ham_PutLine`, how many lines will we need to draw? What will be the coordinates of their endpoints.
2. Compute random coordinates using the `rand()` function for two points on the screen, then draw a line of a different color between them.
3. The lines are drawn very quickly. Put a delay between writing each line by using a for loop that just counts to 100000.
4. The macro `RGB(r,g,b)` can be used like a function call to create color codes from specifying the red, green, and blue, components. Color components are values between 0 and 255. Choose random values for red, green, and blue to draw lines of a random color.
5. Eventually, the screen fills up. Modify your program so that it draws 50 lines, then clears the screen and starts over again by using a for loop to count the 50 lines, followed by the code to clear the screen. Both actions should go in the main while loop.