Basic C Programming

CS 350: Computer Organization & Assembler Language Programming
Lab 0 (Not for turning in)

[Preliminary version while the `alpha.cs.iit.edu` machine gets set up]

A. Why?

- You'll be writing your programs for CS 350 in C and you'll definitely be writing programs for CS 351 in C.
- One of our later topics will be seeing how high-level programs in C are implemented as lower-level programs in machine code (the instructions that the hardware understands).

B. Outcomes

After this lab, you should be able to:

- Log into the `alpha.cs.iit.edu` machine and compile and run a simple C program.

C. Discussion

- C is a "lower-level" language than Java: its constructs more easily map to the data and operations found on typical hardware.
- The first "real" lab is Lab 1, not this one — as I'm writing this (on August 7), I don't know when you will all be getting accounts on `alpha.cs.iit.edu`, so I'm not sure if you'll be able to actually log in and run programs this first week of classes.
- As part of the zip file that makes up this lab, you should find `Lab00_sample.c`. Open it with a text editor and read through it. You'll find much of C is similar to Java, but there are some fairly large differences too.
- `#include <stdio.h>` tells the C compiler to read in the "header" file for the standard I/O library. (Program files generally have the extension `.c`. Header files contain prototypes and type definitions and have the extension `.h`.

- `double sqrt(double);` is the prototype for the square root function. In C, functions have to be defined (with their body) or declared (with just their prototype) before they can be used.
- `int main() { … return 0; }` is the declaration of the main program, which is what will be run when you execute the compiled version of this program. The integer returned is an error code passed back to the operating system: `0` means no error.
- `char ch = '!';` is a declaration with initialization for the variable `ch` of type character.
- `char *a_string = "This is a string";` is a declaration with initialization for a string variable `a_string` (forgive the terrible name). The star (asterisk) means that `a_string` is of type pointer-to-`char`: It stands for the memory address of a character (actually, the first character in a sequence of characters). [If you're thinking that this sounds similar to an array of characters, you're right; we'll see this later in the semester.]
- `char *empty_string = "";` is a declaration with initialization for a string variable `empty_string` which happens to contain no characters. (You can "print" an empty string; it just causes nothing to be printed out.)
- `printf("%s%s%c\n\n", a_string, empty_string, ch);` calls the built-in I/O routine `printf` (print formatted).
  - The first argument is a format string; it can contain actual characters to be printed (like the two newlines, `\n`) and it can contain format codes (which begin with `%`). The remaining arguments specify what values to print.
  - This format string contains three codes: The first `%s` indicates that `a_string` should be printed; the second `%s` indicates that `empty_string` should be printed; the `%c` indicates that the character `ch` should be printed.
- `int x = 17, y;` declares `x` and `y` and initializes `x` to `17`.

- `double d = sqrt(x);` declares and initializes `d` to the square root of `x`. Because of the prototype for `sqrt`, the compiler knows to convert the integer `17` to a double-precision floating point number `17.0`.
- `printf("The square root of %d = %f\n", x, d);` is another print statement; the `%d` is for the integer x; the `%f` is the format for floating-point numbers, so we use it for values of type `float` and `double`.
- `printf("Enter an integer >= 0 (and then return): ");` prints a prompt message.
- `scanf("%d", &y);` calls the standard input routine `scanf` (scan formatted). The format string `%d` says we should read in an integer; the argument `&y` says the integer should be assigned to `y`. We'll study the ampersand in detail later, but basically it says to get the memory address where `y` is stored.
- `printf("The square root of %d = %f\n", y, sqrt(y));` prints out `y` and the square root of `y`.
- The next collection of lines shows what might happen if we read in a value that's too large for the type of variable we're reading into.
  - `printf("Now enter an integer >= 2147483648: ");` prompts for a value too large to be represented as a 32-bit integer (the size we're likely to have with our current hardware).
  - `scanf("%d", &y);` reads that too-large integer into `y`.
  - `printf("The square root of %d = %f\n", y, sqrt(y));` calculates and prints out the square root of `y`
    - The actual bits stored in y won't stand for the value that was typed in, so we'll get some sort of wrong result.
    - The most likely wrong result happens if what got stored into `y` looks like a negative number: The `sqrt` routine will return a special value, Not-A-Number, which gets printed out as `nan`.
    - `if (y < 0) { printf("\"nan\" …" }` checks for this possibility and prints out an explanatory message.

- The remaining lines asks us to repeat the operation on the same value, but this time we read in the value as a "long" (probably 64-bit) integer.
  - `long int z;` declares `z` to be a variable of type long integer.
  - `printf("Try the same value (we'll read it as a long integer): ");` is a prompt.
  - `scanf("%ld", &z);` reads a long integer value into `z`. Note the format code `%ld`, which means long integer (compare to `%d` for regular integers).
  - `printf("The square root of %ld = %f\n", z, sqrt(z));` prints out z and its square root. Note again the `%ld` format for `z`.

D. Logging Into alpha and Compiling

- The `alpha` machine runs Linux; if you don't already know how to use Linux, it'll be good for you to do so. The `linux-account.pdf` file that's part of this lab will show you the basics of linux; the version attached refers to the old computer; substitute `alpha.cs.iit.edu` everywhere you see `dijkstra.cs.iit.edu`.
- Your Lab TAs will be able to show you how to log into `alpha.cs.iit.edu` from the lab machines.
- They'll also discuss possible ways to do this from your own personal machines. Basically, you'll need a way to run a secure shell session (ssh) in some sort of command-level terminal environment.
- For this lab, practice logging into the `alpha` machine and compiling and running the `Lab00_sample.c` program. Once you have a copy of the program in your current directory, the Linux command to compile the program is

  ```
  gcc -lm Lab00_sample.c
  ```

  "`gcc`" means "GNU [pronounced Guh-Noo] C compiler," the standard compiler for Linux environments. The dash `lm` says to include the math library (so you can use `sqrt`).
- If the compile succeeds, it produces an executable file named `a.out`. To run your program, execute that file with the command `./a.out`

