C track: assignment 1

Goals

In this assignment you will learn the basics of compiling programs and the most fundamental aspects of the C language.

Prerequisites

- The Unix environment

  You are expected to have an account on the CS cluster and to understand the basics of using Unix, the filesystem, logging in and out, etc. You should also be familiar with the "`man`" command to access on-line manual pages. Note that all the C library functions have man pages which describe their use in great detail.

- Text editing

  You are free to use whatever text editor you like. Good ones include `emacs`, `vi`, `nedit`, or `pico`. I recommend learning `emacs` because:

  - It's incredibly powerful.
  - It provides syntax coloring for many programming languages, including C.
  - It allows you to access a lot of documentation through the Info system.

  However, teaching `emacs` is beyond the scope of this course. You can type "<control>-h t" in `emacs` to bring up the emacs tutorial. If you are really pressed for time, `pico` or `nedit` have a much shorter learning curve, but are also much less powerful.

Getting set up on the CS cluster

DO NOT SKIP THIS SECTION!

Many of you will want to develop and run your programs on your own computer,

and you are welcome to do so, but the official environment for developing your programs is on the CS cluster. This consists of (a) the computers in Annenberg 104 which run Linux, and (b) any remote login machines available for the cluster. [Currently, there are no remote login machines available on the cluster, but we're working on it.] They all run the same software. You need to do a few small things to get set up to use an up-to-date programming environment on the CS cluster.

The first rule is as follows:

Do not, under any circumstances, develop or run your programs on the CS cluster machine called `login.cs.caltech.edu` or `login.cms.caltech.edu`! This computer is not intended to be used for program development at all.

Instead, if you want to develop and test your code remotely, use a remote login machine, if there is one. It will work just like all the other CS cluster computers in the lab.

The software for the CS 11 C track lives in the directory `/cs/courses/cs11/install`. In order to make this software available to you, you need to do these simple steps:

1. Log in to a CS cluster computer.
2. Execute the following line in a terminal (where `%` represents the terminal prompt):

   ```
   % cp /cs/courses/cs11/setup/bashrc-cs11 ~/.bashrc-cs11
   ```

3. If you do not have a file in your home directory called `.bashrc` (use `ls -a` to check for files beginning with a dot, which are normally hidden), execute the following line in the terminal:

   ```
   % cp ~setup/general.bashrc ~/.bashrc
   ```

4. At the end of your `.bashrc` file, add the following line:

   ```
   source ~/.bashrc-cs11
   ```

Now log out and log back in again. You're all set!

Language concepts covered this week

- basic input/output using `printf` and `scanf`
- conditionals (`if` statements)

- loops (`for` statements)
- numeric types and conversions (`int` to `double`)
- C strings
- preprocessor directives (`#include`)
- using standard libraries
- the `main()` function
- compiling code using `gcc`

Suggested Reading

- Darnell and Margolis, chapter 3.
- K&R, chapter 1 (all) and chapter 7 (pp. 153-159). Some of the material on `scanf` presupposes an understanding of pointers. Since you haven't seen pointers yet, I've included an "aside on scanf" below to help you out.
- If you don't understand the C compilation process, take a moment and read this page now.

Programs to write

- `hello1`

  Write a program called `hello1` to print "hello, world!" to the terminal. Use the `printf` function to do the printing. Make sure there is a newline at the end of the message. Compile it using the command:

  ```
  % gcc -Wall -Wstrict-prototypes -ansi -pedantic hello1.c -o hello1
  ```

  ("%" is the unix shell prompt.) `gcc` is the GNU C Compiler, whose job it is to convert the file `hello1.c` (called source code; this is the file you create) into a binary executable. `hello1` is the name of the binary program; the `-o` option tells the compiler that the next argument is the name of the output file. Don't worry about the `-Wall`; it turns on compiler warnings so that the compiler will warn you about anything it considers dubious but which is still legal. It's a good habit to use `-Wall` whenever you use gcc. Note that although almost all C compilers have some option for enabling or suppressing warnings, there is no standard command-line option for these, so `-Wall` will only work for gcc. Make sure that your `main` function returns 0 or the compiler will issue a warning. The options `-Wstrict-prototypes -ansi -pedantic` ensure that your program is ANSI-compliant and that your prototypes have been declared correctly. Don't worry about this for now, but do use it; it will make your life much easier later on.

- `hello2`

  Modify `hello1.c` so that the program (now called `hello2`, corresponding to the source code file `hello2.c`) prints a prompt string (*e.g.* "Enter your name: "), after which you enter your name and it prints "hello, <your name>!" to the terminal (with your name substituted for "<your name>", of course). **Make sure your program prints a prompt string before reading the input** (a lot of people forget to do this). The name you enter should be a single word only (say, your first name). Use the library function `scanf` to read the string from standard input (also known as `stdin`) and print to standard output (also known as `stdout`). `stdin` and `stdout` both represent input and output from the terminal (as opposed to, say, a file).

  ASIDE ON SCANF: `scanf` should be invoked as follows:

  ```
  char s[100];  /* N.B. strings are arrays of chars in C */
  /* ... maybe some intervening code ... */
  scanf("%99s", s);
  ```

  When run, the program will pause while executing the `scanf` until you enter a string and hit return, or until 99 characters have been entered (whichever comes first). [**NOTE**: the reason you can't enter 100 characters with this code will be explained in a later lecture.] Then it will continue. The character array `s` will then contain the string you entered, and can be passed to `printf`. I will discuss this in more detail in later lectures. Note that `scanf` used as above will also ignore anything past the first whitespace character (space or tab). There are ways to get around this, but they aren't important now (so don't worry about it). That's why your name should just be a single word.

- `hello3`

  Modify `hello2` so that the program prints a prompt, you enter your name, the computer generates a single random number `n` between 1 and 10 and prints that many messages. The format of each message that you will print is: "<n>: hello, <your name>!" or: "<n>: hi there, <your name>!", where <n> is the random number you generated and <your name> is, well, your name. Print the first message when <n> (**not** the loop index variable) is even and the second message when <n> is odd. **All the messages for a single run of the program will thus be identical.**

  I repeat: **All the messages for a single run of the program are identical.** Pay attention to this! Every term, about half the students print a different message depending on whether the loop index variable is odd or even instead of n, or don't print the number n at all. If you do this, you'll have to redo the program and you'll lose marks.

By "loop index variable" I mean *e.g.* `i` in: `for (i = 0; i < n; i++) ....`

Use the `rand` library function to generate the random numbers; this function is found in the `stdlib.h` header file, so be sure to add:

```
#include <stdlib.h>
```

to the top of your program source code. You will also need to "seed" the random number generator to get it started; the best way to do this is to include this line:

```
srand(time(0));
```

at the beginning of the program. `srand` is also in `stdlib.h`, but `time` is in the `time.h` header file, so you should `#include` that as well. Be aware that there is not always 100% standardization on the locations of functions in header files among different operating systems. You should also look at the man pages for `rand` and `srand`, which will contain useful information. To do this, type: `man 3 rand` at the unix prompt (it's in section 3 of the online manuals; just `man rand` will not work, because there is another `rand` that has a man page).

A slightly tricky part of this task is converting from the return value of `rand()`, which returns an int between 0 and `RAND_MAX` (which is a large integer constant defined in `<stdlib.h>`) into a random number between 1 and 10. There's more than one way to do this.

ASIDE ON NUMERIC CONVERSIONS: You can convert an integer to a real number (double or float) as follows:

```
int i = 10;
double d = (double)i;  /* Convert int to double. */
float f = (float)f;    /* Convert int to float.  */
```

Recall that doubles are double-precision while floats are single-precision. Similarly, you can convert real numbers (doubles or floats) to integers as follows:

```
double d = 12.3;
int i = (int)d;    /* Convert double to int. */
```

In this case, the conversion to integers throws away everything to the right of the decimal point (the fractional part). I'll talk about this more in lecture 2.

To hand in

The `hello1.c`, `hello2.c`, and `hello3.c` programs. Before you hand them in, make sure that you run the style checking program on all of them to catch obvious style mistakes. See the style guide for more information on this.
