4. C Programming

4.1. Compiling and running C programs

C is a **compiled** language. Compilation is a process that translates C program code in a C source file into a binary machine-code form of the program that the system knows how to execute (the computer doesn't understand high-level languages like C or Python or Java, instead it understands low-level machine code). If compilation succeeds (there are no syntax errors in your code), the compiler creates an executable file (the deafult name of which is `a.out`) that you can run from the comand line. For more info, refer to the basics of compiling and running C programs (http://www.cs.swarthmore.edu/~newhall/unixhelp/gccbasics.html).

We'll use the gnu compiler, `gcc`, to translate C to an executable form:

```
$ gcc firstprog.c
```

Then enter the path name of executable file on the command line to run it:

```
$ ./a.out
```

gcc supports command line options to include debug information in the executable file (-g) and to specify the name of the executable file (-o filename) rather than use the default "a.out". Let's try it out on one of the files you copied over:

```
$ gcc -g -o firstprog firstprog.c
$ ./firstprog
```

Along with the code you copied over, there's a Makefile. The Makefile contains rules for compiling executables from the .c source files. To execute these rules, type `make` command:

```
$ make        # this will compile all files as specified by the all: rule
```

`make` is a convenient way to compile without having to type in a long `gcc` command every time.

To clean up all the files that were compiled by `make`, you can run `make clean`:

```
$ make clean  # this removes all files generated by make (the can be rebuilt)
```

4.2. First C program: main, variables, printf

Let's open `firstprog.c` in an editor. You can use whatever editor you feel comfortable with, such as `code`, `vim` or `emacs`.

Look for examples of:

- C comments
- How to import a C library (`stdio.h`)
- The main function definition, function bodies (`{ }`), and C statements (end in `;`)
- Defining constants
- Declaring variables (note all are declared at the top of the main function)
- `printf` function (similar to Python `print` function with print formatting)
- Note that all the code is inside the body of a function!

Now let's compile and run the program:

```
$ make
$ ./firstprog
```

There is a suggestion for changing the code in a `TODO` comments. Let's try adding to do that together. When we change the code we need to save it in our editor, and then recompile the program before running it again:

```
# edit firstprog.c, then save and quit the editor before you run lines below
$ make              # recompile
$ ./firstprog       # run
```

Save and Recompile

If you have used Python you may be used to making changes in your editor, saving, and running it immediately. In C, there is one extra step between saving the file and running which is **recompiling** it. Every time you change your C program, you have to recompile it to build a new binary

4.2.1. printf formatted output

`printf` uses placeholders for specifying how a value should be printed (how its series of bytes be interpreted). See `firstprog.c` for examples. Here is a brief summary:

```
Specifying the type:
   %d:   int     (ex. -234)
   %f:   float or double (ex. -4.34)
   %g:   float or double
   %s:   string  (ex.  "hello there")
```

4.3. Functions program

Let’s next open the `functions.c` program and look at an example of a C function. I will use `code` in this example, but you can use any editor you would like.

```
$ code functions.c
```

And let’s look at examples of:

- Function Definition
- Parameters and local variable declarations
- Return type
- All function bodies are between (`{ }`)
- Function Call
- Function Prototype

Then compile and run to see what this program does:

```
$ make
$ ./functions
```

There are some `TODO` comments in `functions.c` with some suggestions for things to try out in this code. We encourage you to try them on your own.

4.3.1. more about C functions

The syntax for functions in C is similar to that in Python, except that C function definitions must define the **return type** of the function and **type and name of each parameter**. Here are two examples:

```c
/* sum: a function that computes the sum of two values
 * x, y: two int parameters (the values to add)
 * returns: an int value (the sum of its 2 parameter values)
 */
int sum(int x, int y) {
   int z;      // a local variable declaration

   z = x + y;  // an assignment statement
   return z;   // return the value of the expression z
}

/* A function that does not return a value has return type "void".
 * If a function takes no parameters, it should be given a "void" parameter.
 */
void blah(void) {
   printf("this function is called just for its side effects\n");
}
```

An example of calling these two functions from `main` looks like this:

```c
int main(void) {
   int p;              // local variable declaration

   p = sum(7, 12);     // call to function that returns an int value
   printf("%d\n", p);
   blah();             // call to void function (doesn't return a value)

   return 0;
}
```

5. Lab 1 Intro

Next, we'll take a look at the first lab assignment (https://www.cs.swarthmore.edu/~kwebb/cs31/f23/Labs/lab01). Note that all course assignments and lab practice (in-lab work, lab assignments, and homework assignments) will be posted to the Schedule (https://www.cs.swarthmore.edu/~kwebb/cs31/f23#classschedule) part of the course webpage.

6. Handy Resources

- CS Department Help page (https://www.cs.swarthmore.edu/help/basic_unix.html)

- Chapter 1 (https://diveintosystems.org/antora/diveintosystems/1.0/C_intro/index.html) on C programming in course textbook
- GitHub Org for CS31 (https://github.swarthmore.edu/CS31-F23)
- Using Git for CS31 (https://www.cs.swarthmore.edu/~kwebb//cs31/resources/labsetup.html)
- Git help page (https://www.cs.swarthmore.edu/git) and GtiHub one time setup steps (https://www.cs.swarthmore.edu/git/git-setup.php)
- Some Useful Unix Commands (https://www.cs.swarthmore.edu/~newhall/unixhelp/howto_unixCommands.html)
- vi (and vim) quick reference (https://www.cs.swarthmore.edu/~newhall/unixhelp/viquickref.pdf)
- Compiling and running C programs on our system (https://www.cs.swarthmore.edu/~newhall/unixhelp/gccbasics.html)
- C code style guide (https://www.cs.swarthmore.edu/~newhall/unixhelp/c_codestyle.html)
- C programing resources (https://www.cs.swarthmore.edu/~newhall/unixlinks.html#Clang)