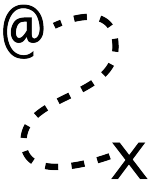
15-122: Principles of Imperative Computation, Spring 2013

Programming 0: Scavenger hunt



Due: Monday, September 2, 2013 by 22:00

Welcome to 15-122! This "zeroth" programming homework is designed as a low-stress opportunity for you to tour the course tools and workflow that we will use in 15-122.

The assignment takes the form of a scavenger hunt; you will add pieces of code to a file, `scavhunt.c0`, and then turn this file in through Autolab. There are two ways to do so. From the terminal on Andrew Linux (via cluster or ssh) type:

```
% handin hw0 scavhunt.c0
```

Your score will then be available on the Autolab website. Your submission can also be made through the web interface of Autolab. To do so, please create a zipped tarball (`tgz`), for example:

```
% tar -czvf hw0sol.tgz scavhunt.c0
```

Task 0 (0 pt) *Make sure you turn in your academic integrity form! If not, you'll temporarily get -500 points on this task, which will pretty much destroy your grade in the course.*

1 Obtaining the handout code

Task 1 (1 pt) *Obtain the handout file* `scavhunt.c0`*, containing a function* `greet`*.*

From the course web page, you will need to download the handout code, a gzipped tarball file named `hw0-handout.tgz`, and unpack it. **IMPORTANT:** if you are using Andrew AFS, make sure that you only put code for 15-122 into your `private` directory or into another directory with appropriate AFS permissions set. Failing to do this will cause you to run afoul of the course's academic integrity policy.

```
% cd $HOME/private/15122
% tar xzvf 15122-prog0.tgz
x hw0-handout/
x hw0-handout/scavhunt-main.c0
x hw0-handout/scavhunt.c0
% cd hw0-handout
% ls -la
drwxr-xr-x   4 rjsimmon  staff  136 Jan  7 12:13 .
drwxr-xr-x  16 rjsimmon  staff  544 Jan  7 12:13 ..
-rw-r--r--   1 rjsimmon  staff  181 Jan  5 16:50 scavhunt-main.c0
-rw-r--r--   1 rjsimmon  staff  216 Jan  5 14:55 scavhunt.c0
```

You can't use `scavhunt-main.c0` yet, but you can test the `greet` function contained in `scavhunt.c0` using Coin.

```
$ coin -d scavhunt.c0
C0 interpreter (coin) 0.3.2 'Nickel' (r255, Thu Jan  3 14:12:00 EST 2013)
Type '#help' for help or '#quit' to exit.
--> greet("Hello", "world");
"Hello, world!" (string)
--> println(greet("Hello", "world"));
Hello, world!
(void)
--> print(greet("Hi", "friend"));
(void)
--> print("\n");
Hi, friend!
(void)
--> #quit
```

This demonstrates something important (and potentially confusing) about C0 programs. When you use the `print` function to print strings, the output is *buffered* and does not generally get printed out until a newline is printed. The escape sequence `\n` represents a newline, so printing the string `"\n"` empties the buffer, printing `Hi, friend!`.

2 Using the C0 tutorial

Task 2 (1 pt) *The C0 tutorial's page on "Statements" contains the code for a C0 file named* `fact.c0`*. Copy all the contents of this file (it's just one function) into* `scavhunt.c0`*.*

The C0 tutorial at `http://c0.typesafety.net/` will help with early assignments. The point about buffered output above is explained on the page "Debugging C0 Programs" in the C0 tutorial.

3 Viewing images from AFS

Task 3 (1 pt) *Copy the contents of* `/afs/andrew/usr/rjsimmon/public/snippet.png` *(it's just one function) into* `scavhunt.c0`*.*

For the next two programming assignments, it will be very helpful for you to already know view how to images that live on AFS on your own computer. The PNG file above, which is publicly readable when you're logged into a Linux cluster machine or connected with SSH to `linux.andrew.cmu.edu`, can be viewed using a program like `display`, `gpicview`, `qiv`, `eog`, or `gthumb`. To use one of these programs, you will either need to be on a Linux cluster or you will need to use `ssh -X` to log on to `linux.andrew.cmu.edu` with X11 forwarding. You can also copy the file by transferring it from AFS to your computer with the `scp` command-line program or with a program like WinSCP and viewing the image with whatever built-in

image-viewing software your operating system uses. Play around and find a method you like.

Once you have added the function from `snippet.png`, you can use the `cc0` compiler and the provided `scavhunt-main.c0` program to compile and run your scavenger hunt code.

```
% cc0 -d -o scavhunt scavhunt.c0 scavhunt-main.c0
% ./scavhunt
```

The `-d`, which also appeared in the call to Coin, makes sure contracts are checked. The arguments `-o scavhunt` tells the compiler to produce an executable file named `scavhunt`. You could omit this argument and the executable file would be named `a.out`.

```
% cc0 -d scavhunt.c0 scavhunt-main.c0
% ./a.out
```

4 Updates, clarifications and questions on Piazza

Task 4 (1 pt) *Modify the function from* `snippet.png` *as described in the "Welcome to 15-122!" post on 15-122 Piazza.*

Piazza can be found at `http://piazza.com/`. Please read through the whole "Welcome to 15-122!" post, as it explains some of the guidelines for how we will be using Piazza.

5 Getting feedback from Autolab

Task 5 (1 pt) *Add a function (that takes no arguments and returns a string) to* `scavhunt.c0`*. It does not matter what string this function returns.*

To figure out the name of this function, you will need to look at Autolab's output. The autograder we use for 15-122 doesn't give a ton of feedback, but it does give some feedback when tests fail. First, it says why the test failed (for instance, `File did not compile`), and then it gives a hint. These hints aren't perfect, and they can sometimes be unintentionally misleading. (Usually, this is because of a mistake that we didn't anticipate, or because an error in an earlier part of the assignment triggered another error later on.)

Also, the hints almost always assume that the test compiled, so if the test reports that the file did not compile, you should look to the first part of the autograder's input to see the output from the C0 compiler and get an idea of what went wrong, rather than paying attention to the hint. You'll need to do this to learn the name of the mystery function needed to complete this assignment. *Don't make a habit of doing things this way.* Autolab is not your compiler, and you should usually test your code before you hand in your work.

There are two other critical features in Autolab. First, the *Gradebook* link lets you see your grades and to see the number of late days you have left (remember that your three late days only apply to programming assignments). Second, you can view the code you handed in with Autolab; when there's a manually-graded piece of a programming assignment you'll get comments from TAs on Autolab.