15-122: Principles of Imperative Computation

Recitation 1 — Josh Zimmerman, Nivedita Chopra

Administrivia and general advice

Welcome to 15-122 section _____! I'm _____ and you can email me at _____. Before we get started, I want to give you some information that'll be useful to you, as well as some advice.

I have office hours, which I encourage you to go to if you're having any trouble with the material. The sooner you ask questions, the sooner I can help you. (Everyone else on course staff has office hours too. You can find those, and their contact information, on the course website.)

If at any point you don't understand something I say, *ask questions!* If I explained something in a way that made you confused, you're almost certainly not the only one, and I want to clarify what I said.

There's a lot of useful reference material about C_0 at http://c0.typesafety.net/. If you have any questions about anything related to C_0 , it's an excellent first resource.

tl;dr:

- Office hours: TBA
- ASK ALL THE QUESTIONS! 
- http://c0.typesafety.net/ is useful!

Basic syntax

Semicolons: statements are terminated by semicolons. What this means is that at the end of most lines, you'll need a semicolon. (exceptions are if statements, function definitions, `use` statements, and loops.)

Variables: variables must be explicitly declared and all variables have a type. Variables can never change type after they are declared. Some of the types in C_0 are:

- `int`: whole numbers x where $-2^{31} \leq x < 2^{31}$
- `bool`: Either `true` or `false`. Useful for conditionals, loops, and more.
- `string`: An ordered sequence of characters like "Hello!"
- `char`: A single character, like 'c'
- `t[]`: An array with elements of type `t`. Arrays are declared with `alloc_array`: `alloc_array(int, 10)` will make an array that can hold 10 `int`s. This is a big distinction from Python and other languages: arrays have fixed size, so you need to know how long your array will be at the time you declare it.

Conditionals: It's an error to put something that isn't a `bool` in a conditional. Note that `a || b` is true if either `a` or `b` are true (and false otherwise), and `a && b` is true if both `a` and `b` are true (and false otherwise). These are called *infix* operators. The compiler mentions them if you make a mistake with them, so it's good to be aware of this name for them.

Here's an example of `if` statements in C_0 :

```
if (condition) {
   //do something if condition == true
}
else if (condition2) {
   //do something if condition2 == true (and condition == false)
}
else {
   //do something if condition == false and condition2 == false
}
```

Loops: There are two kinds of loops in C_0 : while loops and for loops. While loops execute the loop until the condition they're given is false. For loops execute the first statement they're given once, loop until the second statement is false, and execute the last statement at the end of each iteration. These two examples do the same thing. Here, the for loop is preferable but there are cases (like binary search in an array, which we'll discuss later this semester) where while loops are cleaner.

While loop

```
int x = 0;
while (x < 5) {
   printint(x);
   print("\n");
   x++;
}
```

For loop

```
for (int x = 0; x < 5; x++) {
   printint(x);
   print("\n");
}
```

Function definition: This example defines a function called add that takes two ints as arguments and returns an int.

```
int add (int x, int y) {
   return x + y;
}
```

Comments: use // to start a single line comment and /* ...*/ for multi-line comments. It's good style to have a * at the beginning of each line in a multi-line comment.

Indentation and braces: Your code will still work if it's not indented well, but it's really bad style to indent poorly. Python's indentation rules are good and you should generally follow them in C_0 too. C_0 uses curly braces ({ and }) to denote the starts and ends of blocks, as seen above. For single-line blocks it's possible to omit the curly braces, but that can make debugging very difficult if you later add in another line to the block of code. For that reason, I *highly* encourage you to always use braces, even for single-line statements.

Bad

```
if (x == 4)
   println("x is 4");
```

Good

```
if(x == 4) {
   println("x is 4");
}
```

Another important note about indentation is that you should choose either tabs or spaces and stay consistent, since mixing styles makes your code unreadable if someone views your code with a different number of spaces per tab.

Fix syntax my!

Now, ssh in to unix.andrew.cmu.edu and run these commands (don't forget the "." at the end of the cp):

```
$ cd private
$ mkdir -p 122
$ cd 122
$ cp /afs/andrew.cmu.edu/usr5/jzimmerm/public/badSyntax.c0 .
```

If you're using the `csh` shell (which is the default, you should run the following command, which allows you easy access to tools you'll need to use for the course. (If you're using `bash`, you should first run the command `csh`, run `exit` after you're done, and then log out and log back in from the andrew machine.)

```
$ source /afs/andrew.cmu.edu/course/15/122/bin/setup-c0.csh
```

Then, use either `emacs` or `vim` (I personally prefer `vim`) and the C_0 compiler (`cc0`) to correct the syntax errors in that file.

Compile using the following command.

```
$ cc0 badSyntax.c0 -o badSyntax
```

In this command, `cc0` refers to the C_0 compiler. Next comes the name of the file in which we've written our program: `badSyntax.c0`. This file is called the *source code*. Then we can specify the file that we want to store the compiler's output in using the `-o` option. (This file is called an *executable*). Here this file is `badSyntax`. If the `-o` option is absent, the compiler's output is stored in `a.out`.

Then, when `cc0` no longer gives errors, run with the following command. (in general, to execute a file, you need to either put it in a "special" location like `/bin` or to specify a path to that file. In this case, the file is in the current working directory so we prepend `./`.

```
$ ./badSyntax
```

When you're done, your compiled version should output:

```
0
1
1
2
3
5
8
13
21
34
0
```

Checkpoint
If my first command was `cc0 badSyntax.c0` what command would I use to run the executable?