15-122: Principles of Imperative Computation

Recitation 1 — Josh Zimmerman, Nivedita Chopra

Administrivia and general advice

Welcome to 15-122 recitation! Take a moment to fill in the particulars for this section, so that you know which section you're in as well as your TA's name and email.

Section :

TA's Name :

TA's Email :

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

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

There's a lot of useful reference material about C0 at http://c0.typesafety.net/. If you have any questions about anything related to C0, it's an excellent first resource.

tl;dr:

- ASK ALL THE QUESTIONS! 
- Office Hours will be posted soon
- http://c0.typesafety.net/ is useful!

Basic syntax for C0 programs

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 C0 are:

- `int`: integers 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 some other languages: arrays have fixed size, so you need to know how long your array will be at the time you declare it. And you need to respect the array size whenever you use 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). `&&` and `||` (as well as other operators like +, -, etc.) are called *infix* operators, because they take two arguments and the operator is placed between the two arguments. The compiler mentions the word "infix operator" 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 C0:

```
1 if (condition) {
2    //do something if condition == true
3 }
4 else if (condition2) {
5    //do something if condition2 == true and condition == false
6 }
7 else {
8    //do something if condition == false and condition2 == false
9 }
```

Loops: There are two kinds of loops in C0— `while` loops and `for` loops.

- `while` loop : It takes a condition (something that evaluates to a boolean). The loop executes until the condition is false.
- `for` loop : It takes three statements separated by semicolons. Execute the first statement once at the beginning of the loop, loop until the second statement (a condition) is false, and execute the third statement at the end of each iteration.

`while` loop

```
1 int x = 0;
2 while (x < 5) {
3    printint(x);
4    print("\n");
5    x++;
6 }
```

`for` loop

```
1 for (int x = 0; x < 5; x++) {
2    printint(x);
3    print("\n");
4 }
```

These two examples do the same thing. Here, the `for` loop is preferred but there are cases (like binary search in an array, which we'll discuss later this semester) where `while` loops are cleaner.

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

```
1 int add (int x, int y) {
2    return x + y;
3 }
```

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 C0 too. C0 uses curly braces (i.e. { 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, we *highly* encourage you to always use braces, even for single-line statements.

Very Bad

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

Okay, but Risky

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

Good

```
1 if(x == 4) {
2    println("x is 4");
3 }
```

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!

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

```
$ cd private
$ mkdir 122
$ cd 122
$ cp /afs/andrew.cmu.edu/usr18/niveditc/public/badSyntax.c0 .
```

First, if you don't know what shell you're using, run the `shell` command (it should output either `csh` or `bash`). Now run the following command for easy access to tools you'll need to use for the course:

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

Then, use either `emacs` or `vim` and the C0 compiler (`cc0`) to correct the syntax errors in that file.

Checkpoint 0

Think you can fix those errors faster than your TA (or without a compiler)? Here's the code that's on the screen —

```
1 /* File created by Josh Zimmerman
2   Modified for use this semester by Nivedita Chopra. See the correct version in goodSyntax.c0
3  */
4
5 #use <conio>
6
7 def fib(i):
8    if(i == 0 or i == 1){
```

```
 9     return i;
10    }
11 return fib(i − 1) + fibi(i − 2)
12
13 int main():
14  for int i=0; i < 10; i++
15  printint(fib(i))
16  print(\n)
17  return 0
```

Compiling and Running the Program

Compile the code using the following command.

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

In this command, `cc0` refers to the C0 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 1

Why is there a 0 at the end of that output?

Checkpoint 2

If I compiled my file with the following command:

```
cc0 badSyntax.c0
```

What command should I now use to run the executable?