15-122: Principles of Imperative Computation

Recitation Week 1

Rob Simmons, Josh Zimmerman

Basic syntax for C0 programs

Semicolons: statements are terminated by semicolons. At the end of most lines, you'll need a semicolon. (Exceptions are `if` statements, function definitions, `use` statements, and loops.)

Types: 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.
  - `a || b` is true if either `a` or `b` are true
  - `a && b` is true if either `a` and `b` are true
  - `!a` is true if `a` is false
  - `x < y`, `x <= y`, `x > y`, `x >= y` all compare two integers `x` and `y` and return a `bool`
  - `x == y` and `x != y` compare most C0 types and return a `bool`. You can't compare strings this way, though, and it's usually bad style if you're writing code like `e == true`.
- `string`: An ordered sequence of characters like "Hello!"
- `char`: A single character, like `'c'`, `'z'`, `'F'`, or `'?'`.

Boolean operators: Both `&&` and `||` are *short-circuiting infix operators*.

They are *infix* (like other operators +, -, %, etc.) 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.

They are *short-circuiting* because, if the expression to the left of `&&` evaluates to `false` or if the expression to the left of `||` evaluates to `true`, then the expression to the right will never get executed. This means that, even though evaluating the expression `y/x == 0` will cause an error if `x` is zero, evaluating the expression `x == 0 || y/x == 0` can never cause an error.

Locals: locals (confusingly also called "local variables," "assignable variables," "assignables," or "variables") are explicitly declared along with their type. Locals can never change type after they are declared.

```
1 int x = 5;          // x is initialized to 5
2 int y;              // We can't use y until we assign to it!
3 string str = "hello";
4 y = x + 4;          // y is now equal to 9
5 x = x + 1;          // x is now equal to 6
6 x = "world";        // ERROR! string and int are different types!
```

Conditionals: These are one way we use `bool` values. 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 }
```