Com Sci 31 Scribe Notes for 10/11/13

by Alex Peters and Kimberly Lui

Arithmetic Operators & Precedence

Take, for example, `int x = 3 * 4 + 5 % 2;`. To solve this, we must have knowledge of the precedence of arithmetic operators. The order from highest precedence to lowest is as follows:

1. `()`
2. `* / %`
3. `+ -`

Associativity of the operators is from left to right. Therefore, `int x = 3 * 4 + 5 % 2 = 12 + 1 = 13`. In the same way, `int x = 3 * (4 + 5) % 2 = 3 * 9 % 2 = 27 % 2 = 1`.

Variable Assignment

Let `int x = 5` and `int y = 6`. For the value of `x` to increase by 1 to reach the value of `y`, there are four possible ways to command such addition:

- `++x`
- `x = x + 1`
- `x += 1`
- `x++`

Of these four options, `x = x + 1` and `x+=1` are the same in that they perform the same action. However, `x++` and `++x` work differently. For example, if one were to say that `inti = 1` and then declare `int j = i++`, only i would take on the value of 2 while the value of `j` would be 1. In this case, the increment only occurred after `j` took on the value of `i`. On the other hand, if `int = 1` and `int j = ++i`, both `i` and `j` would take on the value of 2; `j` first took on the value of `i`, and then the increment occurred, thus affecting both variables.

Say that we had `int x, y, z` and then declared `x = y = z = 10`. To break this declaration into components, one would not use:

```
x = 10;
y = 10;
z = 10;
```

To get the correct components for `x = y = z = 10`, one would instead write:

```
z = 10;
y = z;
x = y;
```

Just because `x = y = z = 10`, it does not mean that all of the values are equal to 10. Take, for example:

```
int x;
bool y;
int z;
x = y = z = 10;
```

In such a case, not all of the variables are equal to 10.

Comments

When adding a single-line comment to your code, it is sufficient to simply add a `//` before your comment. Multiple-line comments, however, require `/*` to be added before the comment and `*/` at the end. Comments can be used to help one better understand code. It is important to write effective comments, as in anytime that there are areas where there might be a source of confusion. On the flip side, do not write excessive comments.

True/False

If one were to write `x = (y == z);` there are two possible responses one might receive. If the value of `y` were indeed equal to the value of `z`, the statement would be true, and the number 1 would be returned, meaning`x = 1`. However, if `y`and `z` held different values, the statement would be false, and the number 0 would be returned, or `x = 0`.

==is not the same thing as =. Writing `x = (y = z)` would be like assigning the value of `z` to `y`. Thus, the statement would be regarded as true, and `x = 1`. The reality is that, 99.99% of the time, one will receive `x = 1` in such a case.

`short` vs. `int`

When assigning a value to a variable, one could use, for example, `short x = 1000;` or `int x = 1000;`. The difference between `short` and `int` is that `short` stores less memory—16 bits, or 2 bytes. Generally, one bit is reserved for the sign of the stored value, while the remaining 15 bits are for the numbers. `int`, on the other hand, stores up to 32 bits, or 4 bytes. As with `short`, one of the bits is reserved for the sign. The values that can be held with `short` are -32,768 to 32,767. Note that 32,767 is just 2^{15} - 1. `int` can hold values from -2,147,483,648 to 2,147,483,647, or 2^{31} - 1.

Integer Overflow

If one were to type `short x = 10000 * 10000;` a response might be given, but certainly not the one you would expect. 10000 * 10000 yields a value that is clearly too large for `short` to hold. This is an example of integer overflow. You cannot know what response to expect in such a case. Just an example of a value that might be returned to you is `-18543`, which is clearly not correct.

Conditional Statements

Say that you were deciding on whether or not to go to the beach. If it was sunny, you might go; otherwise, you would stay home and read a book. You could write this as:

```
if (sunny)
go to the beach
else
stay home and read a book
```

The same method can be used for conditional statements in programming. For example, you could write:

```
if (1 == 2)
cout<< "A";
else
cout<< "B";
```

In this case, the given response would be `B`. If you were to forget the `else` in the statement, `B` would still be displayed, but not because the statement was false. In this case, regardless of whether or not the `if` statement was true, `B` would be displayed. For example, writing:

```
if (1 < 2)
cout<< "A";
cout<< "B";
```

would return both `A` and `B`. `cout<< “B”;` is its own separate statement, as only the line that comes immediately after the `if` statement is included with it. If one wanted to group lines together in the `if` statement, simply place `{}` around the lines.

Login Program

Say that we are creating a login program. A simple one might look like this:

```
int password;
cin>> password;
if (password==1234)
cout<< “The secret word is Boelter!”;
else
cout<< “Incorrect Password!”;
```

Were one to accidentally write `if (password = 1234)` instead of `if (password==1234)`, the statement becomes true, no matter what user input is given, and the secret word will be displayed. Similarly, if one wrote `if (1)`, the statement is true no matter what, and the secret word is displayed. On the flip side, `if (0)` would be regarded as a false statement, and `Incorrect Password!` would be displayed. Actually, any non-zero values in the `if` statement would be regarded as true statements.

Strings

To be able to use strings, one must first write `#include <string>`. After that, an example of what using strings might look like is:

```
stringstr;
str = “”;
str = “cs31”;
if (str==”cs31”)
cout<< “You are in the right class”;
```

When assigning “” to a string, the string becomes what is called an empty string, or a string that contains no characters.

Casting

If one were to use `double x;` and then declare `x = 5 / 2;` with the intention of receiving a response containing decimals, the answer would not be the one he or she was aiming for. Regardless of declaring x as a double, both 5 and 2 are integers, and therefore, the program will store and display integers. Thus, `x = 2`. Examples of declarations that would work in achieving answers that aren’t integers are `x = 5.0 / 2;`,`x = 1.0 * 5 / 2;`, `x = double (5) / 2;`, `x = double (5) / double (2);`, `x = 5 / double (2);`, and `x = 5.0 / 2.0;`. In some compilers, `x = (double) 5 / 2;` could possibly work; however, `x = double (5 / 2);` will not work.