

Comments Regarding Lab Assignment 2

Contents

1	First: A Note about the Labs	2
2	Semicolons: When to use them, when not to use them	3
3	if-else Structure Syntax	4
4	Operators	5
5	Finding the Maximum/Minimum	6
5.1	Method 1	7
5.2	Method 2	8

1 First: A Note about the Labs

- I noticed that in labs, most people tried to “just write the programs” required
 - Since you are new to this, of course there were some problems. . .
 - Concerning the programming part, most people just had simple syntax errors
 - Learning the syntax just takes time, so that’s expected
 - However some people were writing code and weren’t entirely sure what was happening
 - This is *also expected*, but. . .
-
- Unlike physics and chemistry labs, there’s no worry of destroying expensive resources or equipment in these labs
 - (*Unless of course you get so aggravated you decide to put your fist/foot through the computer. . . computers can often make that happen*)
 - With these CS labs, you can
 - Create and test as much code as you like
 - Compile and run a program as many times as you like
 - In short, you can try out whatever you’d like and test it
-
- So, *if you’re having problems, don’t just keep working on the same thing*
 - Instead. . .
 1. Create a new project
 2. Play around with *only* the stuff you’re having difficulty with
 3. Test it, *many times*
 4. Make sure what you’ve written does what you expect it to, and you understand what’s going on
 5. Then incorporate what you’ve learned into the original program
 - ***In these labs, you are free to tinker as much as you’d like!***

2 Semicolons: When to use them, when not to use them

- Many had some issues with semicolons... where to use them and where not to
- This explained in the book, but since many of you still didn't have the book on Wednesday, the confusion is understandable.
- So I thought I'd *quickly* explain when/where to use semicolons

- Basically, semicolons are required at the end of every line of C++ code
- *Except the following...*

1. At the end of C++ *preprocessor* statements. These are things like

```
#include <file>
```

2. At the end of C++ control or looping definitions, such as after an `if (<condition>)` or `else`
3. After opening/closing curly braces (most of the time)
4. After function headers, such as `int main()`

- Remember, however, that C++ errors regarding missing semicolons are usually fairly easy to interpret
- So if in doubt, *try compiling*
- If you get an error about missing semicolons, you know you need one
- The book goes into some detail about this, but you'll learn it with time also

3 if-else Structure Syntax

- Everyone seemed to get the syntax for `if`, `if-else`, and `if-else-if` structures figured out by the end of the lab
- The only thing I'll comment on is the following...
- After the `if`, the condition *needs to be in parenthesis*

- So, the following is incorrect

```
if x <= y
{
    ...
}
```

- As is the following

```
if (x <= y) && (x <= z)
{
    ...
}
```

- Parenthesis are required around the *entire* condition
- So the following are correct

```
if (x <= y)
{
    ...
}

if ((x <= y) && (x <= z))
{
    ...
}
```

- Now that you have the book, this should become clear

4 Operators

- There was some confusion about operators, such as `<`, `<=`, and `==`
- Just remember that *operators are functions!*
- Many C++ operators are *binary*, in that they require two arguments
- For example, `x <= y`, `x == y`
- C++ allows one to chain multiple statements like this together
- For example. . .

`w == x == y == z`

- However, the `==` operator is *left-associative* in C++
- That is, C++ treats the above statement as the following

`((w == x) == y) == z`

- Now, *does anyone see any problems?*

- Suppose `w, x, y, z` are integers
- C++ will evaluate the left-most equality `(w == x)` first
- Suppose that statement evaluates to `true`
- Then the remaining statement is. . .

`((true == y) == z)`

- At this point, we're testing the equality between a boolean and an integer
- As we saw Wednesday, this is *valid* (an instance of automatic casting)
- And C++ reports no warning/errors (in some cases it might)

- To correct this, we have to break the statement into multiple equality tests, and *conjoin* them, using the boolean `&&` operator. . .

`(w == x) && (x == y) && (y == z)`

5 Finding the Maximum/Minimum

- Many of you were having difficulty finding the minimum/maximum of 3 numbers.
 - Some had an initial set of code that worked in most cases. . .
 - But would fail if there were any duplicates in the 3 input numbers
-
- Now, I *know* that if I were to give any one of you a set of numbers, you could easily figure out the maximum & minimum
 - So why the difficulty in getting a machine to do it?
 - Well, partly just working with the language, which you're still learning
 - But before anyone should write anything, one needs to look at
 1. *Can I do this task in general, without a computer?*
 2. *If so, **How do I do it?***
-
- I want to go over two common methods for calculating the min/max of a set of numbers
 - Most of you did something like one of these. . .
-
- But first, note the following
 - Given any 3 numbers x , y , and z , the maximum is defined simply as *one* of the largest number in the set.
 - So if x , y , and z are all equal, the maximum is any one of them

5.1 Method 1

- The first method is probably the most intuitive
- And in general, it's the method used for a *list* of numbers

- Suppose I give you a list of numbers, and you are to figure out the maximum
- You don't have to look at all the numbers at once, but rather just one at a time.
- You start with the first number, and (for now) make that number the maximum.
- You then look at the next number...
 - *IF* the next number is greater than the current maximum, you set the maximum to the new number, and move onto the next number
 - *OTHERWISE* the maximum remains unchanged, and you move onto the next number

- So in our example with 3 numbers, the following C++ code would accomplish the above algorithm

```
// Initially, set the maximum to the first number
maximum = number1;

// If the second number is greater than the maximum, it becomes the
// new maximum
if (number2 > maximum)
{ // In this case, number2 becomes the new maximum (for now)
  maximum = number2;
}

// If the third number is greater than the maximum, it becomes the
// new maximum
if (number3 > maximum)
{ // In this case, number3 becomes the new maximum
  maximum = number3
}

// Once we reach here, maximum contains the value of the largest
// of the three numbers.
```

- The minimum can be calculated in a similar fashion (replace all > above with <, and that will do it)

5.2 Method 2

- The second method uses a sort of *logical* approach
 - Note that for a general list of numbers (of arbitrary length), this approach would not work
 - But it has a certain intuitive appeal when we have a *fixed* number of integers.
-
- Notice the following. . .
 - Given any set of 3 numbers, the maximum is *one of the numbers*
 - So, for three numbers x , y , and z , there are only 3 cases
 1. x is the maximum. In this case, x is greater than *or equal to* y and z .
 2. y is the maximum. In this case, y is greater than *or equal to* x and z .
 3. z is the maximum. In this case, z is greater than *or equal to* x and y .
 - Notice that if any of the values are repeated, *it doesn't matter!*
 - The following C++ code exploits this (note that for brevity, I'm using x , y and z instead of the more appropriate `number1`, etc)

```
// Case 1. X is the maximum since it's >= y AND z
if (x >= y && x >= z)
{
    maximum = x;
}

// Case 2. Y is the maximum since it's >= x AND z
else if (y >= x && y >= z)
{
    maximum = y;
}

// Case 3. Z is the maximum since it's >= x AND y
else if (z >= x && z >= y)
{
    maximum = z;
}

// Once we reach here, maximum is contains the largest value of
// x, y, and z
```