CS170 Week 3 Lecture 1 Notes

Disclaimer: These notes are meant to be read in parallel with Professor Mead's online notes if you have missed class. Topics will not be covered extensively. Here, you will only see the minor details Prof. Mead spoke about which were not on his topic notes.

Pre-Class Mentions/Q&A

- ➔ First programming assignment will be given out today- you have two weeks to do it (it's harder than any assignments yet)
- ➔ Simple lab given out today (starting next week, labs will get harder and are cumulative)
- ➔ All labs this semester require Doxygen and full style guide
- ➔ Today we'll finish Simple I/O and begin Functions
- ➔ **Be on time** when returning to class from the break

Simple I/O Continued

Comparing printf with cout:
The lab this week will be using cout to format numbers for practice.
“cout” gives significant digits.
“cout” is much more obvious and intuitive while also very verbose.

Changing the default precision:
“cout” can be considered an instance of a struct.
“precision” can be considered a function inside of that struct.
“cout.precision” this is the syntax for accessing a function inside struct.
If you want unlimited precision, you have to use a library for that.
“precision” is only one of the more useful functions.
Setting precision once means you don't have to reset it again in the rest of the program (unlike with printf, you get the default precision unless you specify on every printf call).

Changing the field width:
Changing the width is only valid per use of “cout” (like printf above).

Using setf for More Control:
Overloaded functions- one setf takes one parameter, the other takes two parameters.
In C++ you can have multiple functions with the same name which take different parameters (types, numbers of parameters).
Don't mix printf and cout. Ever.
“<<” is an overloaded operator.
The compiler looks at the operands to decide what to print.
What is a signature? - The name of the function and the number and type/s of parameter/s.
In C every function has one signature.
Now, we have overloaded functions and the compiler checks the signature to decide which function to call. There is no limit to how many overloaded functions you can have.

Next week we will talk about classes and see how this all works.

Manipulators:
“std::setprecision(3)” can be used in the same line as “std::cout” and this can be more convenient than using “cout.precision” in it's own line.
Scope resolution operator works like any other operator and can go on it's own line.
“setw” sets width for only the following variable usage.
“fixed” forces more decimal places (like the printf default).

Overview of Input:
“>>” is the extraction operator.
For “cin”, & is not required for the compiler to see where things are going.
Overloading buffer = corrupting the program. Nothing happens usually.
Std library has a type called string which is dynamic and grows as needed.

~15 min break~

Functions:

References:
There is now another use for the “&”- “ri” and “i” are both the same thing as well as interchangeable.
“*pi” is a separate entity.
The power in references comes from referencing your de-referenced pointers. Maybe in CS280.

Reference Parameters:
We will see why these are convenient when we get to overloaded operators and other advanced topics.
All of these conveniences come with problems of their own. Watch out for ambiguity.
In CS120 we returned multiple values from functions using structs.
If using a non-const reference, you are promising that it won't be changed.
It's actually more expensive to pass a float by address than by value.
You only want to pass references or addresses of built in types if you are going to change them.
Why would we pass a const reference? - Efficiency/size. It's cheaper passing the address of something big than passing by value. Plain references will be changed, const references are usually big objects.
-*Write what you know and know what you write.*

Some Issues with References:
Skipped today because we won't have issues for the first assignment.

Default Parameters:
Much easier than the search and replace method.
Must be in order, as usual.
Must default parameters from right to left.
This happens in the prototype.
“a + b + c + d” operators are not sequence guarantee operators.
“(a) || (b && c)” order is guaranteed.
“<<” output operators are not sequence guarantee operators.
The order of their evaluation is unknown.
There are places where it doesn't make sense to have the compiler provide values for you.

Sometimes there are no universally accepted defaults.