CS170 Week 4 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

- Words of wisdom: You can't use two subscripts format on pointers to the array
- Words of wisdom: Write a helper function to return a certain point on the board
- Words of wisdom: You should be running dr memory or valgrind every time you compile
- Labs are now going to be cumulative
- Reminder: Don't forget to include the blue lines on the sample doxygen file heade

Functions Continued

Overloaded Functions:
Functions are differentiated by their parameters.
The compiler looks at the parameters to decide which function to use.
The biggest problem for programmers is dealing with ambiguities.
Passing constant pointers vs non constant pointers works the same way as it does with references.
When passing arrays in functions, use pointers rather than references.

Classes

Introduction to Object-Oriented Programming
(In comparison to Procedure Oriented in C):
Three ways to use C++: 1) As a better C, 2) Object oriented, 3) Generic (templates).

Procedural Programming:
sizeof(Student) = 22.
Methods we have used so far to prevent bad data and/or memory corruption aren't ideal, they can be considered a mask to the problem.
By putting a private keyword in the struct this disallows the user to access that data.
Once the data is private, you can selectively expose some of it.

Encapsulating Functions and Data:
Typically, data is private while functions which act on the data are public.
Functions will perform the data validation, this idea intends to prevent undefined behavior.
It is guaranteed that if you put an underscore at the end of a variable name, it will never become a keyword. Don't ever make a variable name with two leading underscores.
Typically, public comes before private because the user is usually only concerned with public data.

Anything you can do with a class you can do with a struct. Structs are the same thing as classes in C++. There is absolutely no difference between them as far as what they can do. C++ introduced classes to distinguish between old style C code.
We can now reference members of classes or structs using the scope resolution operator.
In CS170 when we see variables with underscores at the end, they will usually be part of a class, while without the underscore the variables are usually local. This is the convention Prof. Mead uses.
Functions have no bearing on sizes. So, with all of the private and public functions, the student struct's size will remain the same as it was before.

~15 min break~

Encapsulating Functions and Data Continued:
Everything between one access specifier (public or private) and another will follow the first.
(If you use the private keyword, everything is private until you use the public keyword again.)
The goal is to never have undefined values again.

Classes:
Helper functions will need to be under private sections.

Initializing Objects: The Constructor:
When you need to add functionality, you add a function-
You add a constructor, this is going to be an underlying theme for all of C++.
Class::Function ← when they have the same name means this is a constructor.
Example: (Student::Student(params))

Accessors and Mutators (Gettors and Settors):
We currently have no way to read data, we can only set.
No implementation will be allowed in the header files.
It's up to the designer to decide what the user is and isn't allowed to change.

Resource Management:
The student object should be responsible for everything. Clients make mistakes.
If “new” fails, it throws an exception.

Destroying Obejcts: The Destructor:
Student::Student → constructor
Student::~Student → destructor
The destructor must be public.
In advanced situations you sometimes will call the destructor yourself.
Objects will be destroyed in the reverse order which they were constructed.