Declan Ferguson

C++ 211

May 6, 2024

Final Project

Pointers were probably the most interesting thing to learn about for me. I learned about how they save an address to point toward something in memory that would usually be out of scope. I have heard the terms thrown around and had even used them but didn't really understand what they were. The most difficult part was being able to grasp what they were their operators. I would mix up when to use the dot operator and arrow operator. I overcame this by rereading the book and continuing practice problems.

Here is a linked list problem from Zylabs. I create the InsertAtFront() member function. It is also in the Github Repository as Pointers.cpp and InventoryNode.h.
https://learn.zybooks.com/zybook/SKAGITCS211DupeSpring2024/chapter/2/section/21

Recursion was also hard for me to grasp at first. Recursion is the process of dividing a problem up with a function that calls itself until its base case is reached. My biggest challenge with the recursion problems was learning how to properly structure them. I would often forget to create the base case first, or even at all. This led me to a lot of confusion. Once I was able to understand the importance of the base case allowing the recursive function to end, much like a normal loop the problems became much easier for me. I also really struggled with permutations, but a lot of that I feel was on Zylab's end with their unit tests.

Here is a Zylabs problem where I had to write a recursive function for a linked list. You can also find it in the GitHub repository. The files are Recursion.cpp, IntNode.cpp, and IntNode.h.

https://learn.zybooks.com/zybook/SKAGITCS211DupeSpring2024/chapter/3/section/15

Arrays were probably the easiest subject for me this quarter because of their similarity with vectors. Because of that, I struggled with the differences between the two containers. For one when arrays are used in a function, the parameter is passed as a pointer, rather than the actual whole container. The compiler does this to save on memory as arrays can get rather large. This means that I would have to also keep a fixed number of elements to run the array through loops or have the user create an extra parameter for the number of elements. Also conceptualizing arrays as a grid was rather hard at first. Drawing them out helped me through that problem though.

In this link, I created a basic script that finds the dot product of two matrices and outputs it. It is also in the GitHub Repository under arrays.cpp.

https://learn.zybooks.com/zybook/SKAGITCS211DupeSpring2024/chapter/4/section/8

I learned about the importance of abstraction for writing more complex code. It can help keep code more readable. It can also be detrimental as well and abstraction can end up being overly complicated. A hard thing for me to understand is when it is useful to implement abstraction into code. A way I overcame that is through seeking more resources for help and explanations of abstraction.

The code is from the stack project that was assigned. It is in the GitHub repository under Stack-DeclanFerguson.cpp.

Inheritance is one of the main principles of object-oriented programming. It describes the process of derived classes. The child class "inherits" all its protected, or public member functions. The thing I struggled with learning about inheritance is understanding the hierarchy between parents and their derived class. I overcame this problem through the real-world analogies that the book used. For example, a circle is a shape. Its class would have a radius, while a square class would have a side length. The shape class would have neither of these things but would rather have something that both derived classes have like a center point variable.

The problem is also in the git hub under these files: OfferedCourse (1).h, OfferedCourse (1).cpp, Inheritancemain (1).cpp, Course (1).h, Course (1).cpp.

https://learn.zybooks.com/zybook/SKAGITCS211DupeSpring2024/chapter/6/section/13

Polymorphism refers to the process of making code whose behavior changes depending on certain factors. Overloaded functions and a container that can contain both a class and its derived child are both examples of polymorphism. The hardest part of polymorphism for me was understanding the virtual keyword and how to use it. I would have trouble knowing when it was a good idea to create virtual functions or rather just make a new one. I overcame this through a lot of trial and error.

https://learn.zybooks.com/zybook/SKAGITCS211DupeSpring2024/chapter/6/section/14

In total, I was introduced to a lot of new programming concepts this quarter. I thought that a lot of the concepts were more interesting. Over spring break, I tried to mess around making a project but struggled because I didn't know how pointers worked. After having learned about the principles of Object-Oriented programming I am excited to try to make some things on my own. I heard a lot about OOP principles before, but I never really knew what they were until after this class. I understand the benefits of creating modular code. The fact that it can be reused and how I could make programs more readable through abstraction. I practiced recursion problems which at first seemed very difficult. Once I was able to understand its similarity to loops and how it solves problems by dividing it up into smaller sections it made a lot more sense. This will mark the end of my computer science classes at Skagit, but I look forward to more after that.