Patrick Fischer

Senior Seminar

Academic Reflection

I have had many interesting courses throughout my time at Lipscomb. During my freshman year, I took Dr. Nordstrom's Introduction to Computer Science as an elective. I was fairly new to programming at the time, and Dr. Nordstrom's course was about an interesting Java-based language called Processing. This language allowed a user to define various shapes or objects and display them in a window. The course was much more interesting than other courses that I was taking at the time, and I decided to take the next course: Dr. Arisoa's Introduction to Computer Programming. I was more prepared for this course than Dr. Nordstrom's, and I quickly found out that I enjoyed the content of Dr. Arisoa's class. Each assignment was like a puzzle, and the course convinced me to fill half of my next semester's schedule with computer science courses.

During my first semester of my sophomore year I took Dr. Borera's Object-Oriented Programming course. This course used Java, so I was somewhat familiar with it after learning C# and Processing. Like Dr. Arisoa's course, every assignment for this course was like a puzzle. We had many interesting projects in this class, and one of my favorites was a project called Purple America. For this project, we were given a number of files containing x and y coordinates. When plotted using a Java library these coordinates displayed a map of the United States. We were also provided with presidential election data for every county in the United States for a number of years. We used this data to color every county in the United States according to how they voted. It was quite rewarding to see a program do something besides output data to the terminal, and this course was one that convinced me to declare a computer science major.

During the second semester of my sophomore year I took Dr. Nix's Data Structures and Algorithms course. This course was markedly different from Dr. Arisoa's and Dr. Borera's courses. Instead of giving multiple homework assignments per week, Dr. Nix assigned a small number of large projects. One of the first projects was to build a linked list using object-oriented programming. This project built on various concepts that were taught in Dr. Borera's class, and the assignment was useful in many later projects and courses. The next project assigned by Dr. Nix was to build a binary search tree using the linked list. This was an interesting project. The tree stored nodes containing data, and each node also contained a left and right node represented by a linked list. Inserting data into the tree required the program to traverse the tree until it found a node that met a specified condition. We later used this project to create a self-balancing binary search tree called an AVL tree. At the time, I did not appreciate how useful these trees were, but that changed by the time I took compiler construction.

During my next semester, I took another course with Dr. Arisoa: Design and Analysis of Algorithms. I expected this course to be fairly similar to Dr. Nix's data structures course, but it was quite different. Instead of implementing various algorithms Dr. Arisoa taught us about the theories underpinning each one. We used a variety of mathematical tools and proofs to determine how algorithms worked, how efficient they were, and to prove that the algorithm solved the problem it was created for. Many of the principles taught in this class have been quite useful in my other courses for analyzing runtimes of algorithms and for proving that an algorithm will work.

During my second semester of my junior year I had the wonderful opportunity of taking Dr. Fortune's Fundamentals of Automata and Formal Language Theory course. This course was much different from any of the other computer science courses that I have taken. Instead of focusing on algorithms this course focused on regular languages, context-free grammars, and finite automata. Regular languages are used in many programming languages as tools to efficiently search through strings, and context-free grammars appear in many tools designed for compilers. The most interesting part of the course was the material taught about finite automata. At the most basic level, a deterministic finite automaton can be thought of as a computer with a very limited amount of memory. It takes as input a set of symbols, and it uses this input to move through a series of states until it reaches the end of the input. The automaton then returns if the state is an accept state. This has a large number of uses for compilers and regular languages, as the automaton can be converted to a regular expression using certain algorithms. The semester-long project for this course was to design a program that modeled a deterministic finite automaton, as well as performing various mathematical operations on the automaton. This was one of the more difficult projects that I have worked on, but it helped me gain a better understanding of finite automata.

I took one of the more interesting courses of my college career during the first semester of my senior year: Dr. Crawley's Compiler Construction. This course was mainly about implementing a compiler through various projects throughout the semester. This course required knowledge from many of my previous courses. It required an understanding of regular languages and regular expressions, a working knowledge of linked lists and tree data structures, and knowledge of context-free grammars and context-free languages. The project itself was fairly difficult, as I was unfamiliar with most of the tools that were required to build various portions of the compiler. The first task was to build a scanner using a tool called Flex. This tool would take a file as input and scan through the text that it contained. When the scanner found a string of text that matched a rule specified by a regular expression it would return the string to the parser. The parser would then analyze the string against another set of rules specified by context-free grammars. When a match was found, it would request another string from the scanner and repeat the same process. When the parser finished this process, it would store the results in a tree data structure. After scanning and parsing the entire input program another program used the results from the parser to generate assembly code. This portion of the project uncovered a large number of bugs in my scanner and parser, and it required long hours of debugging to fix all of them. Eventually, my project was finished and ready to be turned in. This was the largest project that I have worked on, and it was incredibly rewarding to see the finished result.

My final semester is halfway over, and I am looking forward to my upcoming graduation. I have had a number of wonderful opportunities at Lipscomb, and I cannot wait to see what the future holds.