CSI 370 Computer Architecture Report
Data Structures and Algorithms with Advent of Code

Mason Lee
Champlain College
mason.lee@mymail.champlain.edu

What?
The project I am planning on doing is implementing different data structures in Assembly. While I think that on its own it has the potential to have enough substance for this project, to make it a more unique project, and give it a cool spin, I plan to use the data structures I implement in Assembly to solve specific data structure focused puzzles from previous years of Advent of Code. Advent of Code is an annual programming challenge event that takes place in December, containing a series of daily programming puzzles. The difficulty of the problems vary a lot from day to day, typically starting easier and getting progressively harder. One thing that many days have in common is that the "correct" solution can involve some sort of data structure. For instance, there may be a day when a really good way to solve the puzzle is by using a stack, a queue, or maybe a tree of some sort.

For a better understanding of the type of puzzle I am planning on solving using the data structures I create, here are some puzzles I am thinking about attempting to solve:

- 2021 Day 6 Lanternfish
- 2021 Day 10 Syntax Scoring
- 2022 Day 5 Supply Stacks

The data structures I am planning on implementing are:

- Multi-dimensional arrays
- Linked lists
- Stacks
- Queues
- Trees (This one might turn into a stretch goal given time constraints)

Why?
There are a couple of reasons I want to do this project specifically. First, I want to focus on a software-related project because I like the idea of working with software better than hardware. One of the things that draws me to computer science is the ability to create clean solutions to seemingly complex or tedious problems, which is frequently done using data structures. To have some sort of unique spin on the project of creating data structures, I wanted to apply it to something fun. Advent of Code is something that I do every year as a fun side project and I think it is a great way to practice coding skills as well as learn how to solve different types of problems. Another reason I started with the idea of data structures is that I wanted to learn how I can use dynamic memory allocation in low-level programming.

How?
While I want to keep the project almost entirely in Assembly, I plan on using a little bit of C++ to help with certain tasks. For each puzzle you get a huge text input that is specific to each user. I plan to keep this in a text file and use C++ to get the data from the input file, which will then be transferred over to the Assembly side where it can be loaded into different data structures. Of course, the bulk of this project will be implementing different data structures in Assembly. My current plan for this will be using C's malloc and free functions to point to different locations in memory to create a data structure. Malloc is a function that can dynamically allocate memory by passing in the number of bytes to allocate as a parameter, and free is a function that deallocates memory given a memory location. Member functions of the data structures I create will also be implemented in Assembly by passing the data structure in as the first parameter much like C++ does under the hood with the "this" keyword.

Challenges?
Some of the challenges in this project will most likely be related to actually creating data structures in Assembly. While I'm sure it will be challenging, I think it will be an eye-opening experience to see how much goes on in the background when using dynamic memory allocation rather than static memory allocation. This could be useful when programming in a high-level language when deciding whether something should be implemented using static memory or dynamic memory. Another potential challenge might be formatting the data. Oftentimes the text input that you get is in a very human-readable format, but not at all in a very machine-readable format which makes it difficult to parse programmatically. This is something that may end up needing to be done with C++ while the algorithmic programming is done in Assembly.