Main

* Create list of class course
* Follow user or default CSV file path
* Parse text from the CSV file path
  + Vector/hash/tree setup
  + Open file
  + Loop until the end of the file is reached
    - If at least two strings are present
      * First string is the courseID
      * Second string is the courseName
      * For each additional string:
        + If string in course list

Increment prerequCount

Add string to prerequList

* + Return list

Vector

* Print course
  + Get user course input
  + Search for course
    - Loop through course list
      * If user course ID is course ID
        + Return course
    - Return empty course
  + Output course ID, name, and any prereques

Hash Table

* Setup
  + class HashTable
    - class bucket
      * courseID
      * key
      * next
      * If two strings are present
        + Call hashing for the first string
        + Add to class at the hash position within a temp list
        + Put first string to courseID
        + Put second string to courseName
        + For additional strings:

Increment prereqCount

Add string to prereqList

* + Search the list(String passed in as argument)
    - Create a new temp bucket for courses
    - Set temp bucket equal to bucket at hash location of the string
    - For each course in the table
      * If string matches courseID
        + Set the temp to current course
    - Return temp
  + Printing courses (String passed in as argument)
    - Create a new temp bucket
    - Set temp to hash of a the string
    - For each chained bucket
      * Output courseID, courseName, and prereq amount
      * Loop for every prereq course
        + Output prereq course

Tree

* Search for course
  + Create current node for searching
  + Set current to root node
  + Loop until course ID is found or until node points to null
    - IF a match is found, return that course
    - IF courseId is smaller than currentId
      * Follow the left branch of the tree
    - ELSE courseID will be larger than currentId
      * Follow right branch
* Print course
  + Search for target course
  + Loop until current course is null
    - IF current courseID equals target ID
      * Output course ID and name
      * For each prereq
        + Output prereq name and ID
    - IF current courseID less than target
      * Traverse left down the tree
    - ELSE
      * Traverse right

**Structure Analysis**

For vectors and an ideal hash table circumstance (no collisions), loading the data would be in constant time. In a hash table where there are some collisions, loading data could take as long as linear time while trees stay around logarithmic time. In terms of searching, vectors go to linear while hash tables remain with the same situation where perfect hash tables can search in constant time and linear for the worst of the hash tables. Trees can search from logarithmic to linear time, all depending on how balanced the tree is. When sorting and printing, trees and hash tables both complete in linear time while vectors perform around loglinear, largely dependent on what sort method is used.

Using vectors is almost like a double-edged sword. Loading the data with vectors is extremely efficient but would likely be without an order to it. To then print the vector in a sorted order would take the additional time of sorting the whole thing through a sorting algorithm then proceeding to print each element, which is obviously linear time for each element. Hash tables seems to have the benefit of consistency, where loading and searching depend on how close to optimal the hash table ended up. A rather balanced tree, like a binary search tree, can really speed up the process of searching over a hash table with loads of collisions, but they can frequently go back and forth depending on how close to optimal each data structure is.

For the exact situation we’ve mentioned with just a few classes, differences between the structures would be pretty minimal. That said, we have seen that once we scale the number of elements, it could become quite the difference in times, even for simply loading each element to the structure. For simplicity’s sake, it would likely be easiest to use vectors for the few classes we’ve been presented with. Scaling up, however, I would start recommending switching to hash tables. Trees could also be a viable option, but I have been having a difficult time imaging how exactly this would look as a tree in general and how balanced we could make it, so I am less likely to recommend the structure for this scenario.