**Project One Pseudocode**

**SNHU: CS-300**

**Nathan Book**

**2/17/24**

**Vector Pseudocode:**

Define a structure to store course objects

Define a class containing all relative data members

Create a vector to store course objects

Open the course file

For loop to iterate over file

While the end of the file is not reached

Partition or “parse” the line into courseNumber, title, and prerequisites

Check for the first two parameters, courseNumber and title

Check if line contains any prerequisites after the first two

Create an object with partitioned data

Add object to vector

Close the info file

Prompt user to input course number

For loop over vector to search for matching course number

If course number is found, print out course info (title and prerequisites)

Else if course number is not found, print “Error: Course Not Found”

Return to prompt to obtain user input

**Hash Table Pseudocode:**

Define a structure to hold variable declarations

Define a class that contains the data members and methods for the hash table

Create a default class constructor

Create a class constructor to be called to resize hash table

Create a class to free storage in hash table

Create class to calculate the hash value of any given key

Create a class that inserts data into the hash table

Create the key for data

Declare previous node variable

If previous node is equal to null

Assign node to key position

Else if previous node equals max table size

Assign old node key to max, set key, set old node, and point old node to next node

Else

Find the next open node

Create a class to print all data elements

For loop iterating over file

Create a class constructor to remove a node from hash table

Create a class constructor to search hash table

Declare search variables

If no is not equal to null and key node is not max table size and node ID equals 0

Return node at data

If no is equal to null or node at key value is equal to max table size

Return data

While node is not equal to null

If key node is not equal to max table size and data node equals 0

Return data node

Node equals next node

Return data

Open data file

For loop to read over data file

Store data in table using declared data elements

Close file

Use class constructors in main to desired result

**Tree Data Pseudocode:**

Declare variables

Structure to hold data

Structure to hold tree info

Define a class to contain data members and methods

Private

Public

Default constructor()

Destructor()

Define a class to traverse binary tree in order

Define a remove node private method

If node equals null

Return node

If called element is less than zero

Remove left node

If called element is greater than zero

Remove right node

Else

If left node equals null and right node equals null

Delete node

Node is null

Else if left node does not equal null and right node equals null

Swap left node with current node

Delete current node

Else if left node is null and right node is not null

Swap right node with current node

Delete current node

Else

Find the leftmost child of rightmost subtree, successor

Return node

Define constructor to insert data into tree

If root equals null

Root equals new node

Else

Call add node from this bid insert

Define constructor to remove node given data

Define constructor to search binary tree

Declare current node variable root

While current is not null

If current equals 0

Return current

Else If current is less than zero

Current equals left node

else current is greater than zero

Current equals right node

Return data

Define constructor to display tree information

Output ID, nth parameters

Define constructor to load file

Parse over entire file

Try

For loop iterating over entire file

Create data structure and add elements to tree

**Menu Pseudocode:**

Main()

Process command line arguments

Define a vector to hold course information

Declare choice equals zero

While choice is not nine

Output Menu options

Input choice

Switch (choice)

Case 1:

Load course information into vector in alphanumeric order

Case 2:

Print course list in alphanumeric order

Case 3:

Print a selected course and all its prerequisites

Case 4:

Exit out of program

Output exiting statement

Return 0

**Evaluation:**

1. Vector worst-case

Reading a file: O(N)

Creating course objects: O(N log N)

1. Hash Table worse-case

Reading a file: O(N)

Creating course objects: O(N)

1. Tree worse-case

Reading a file: O(N)

Creating course objects: O(log N)

**Recommendation:**

There is no other way to input all the data from a file into a storage structure other linearly iterating over each line in the file. So, each sorting method will have a worse case O(N) when reading the file. Each one of the vector, hash, and tree creating course objects runtimes is considered *fast*. Considering the requirements and functionality given by the ABCU advisors, the vector sorting method will be the best choice. This is primarily due to the partitioning that will be required to separate the course info from the prerequisites. While also being able to print all of the above if requested.