**Project One**

**Vector Pseudocode:**

**File Management:**

Declare vector structure:

string Course Number

String name

Vector<string> prerequisites

Declare file name to be opened

Open (filename.txt)

If (file == NULL)

Print out file not found

End program

While (in the file)

Read each line

Saving each variable in the relevant structure type

//Error checking

For all courses

If (course name and prerequisites are empty)

Return error

Int count

For all prerequisites

For all courses

If (prerequisites is equal to course name)

Count++

++i

repeat

If count = 0

Return with error

**Create and store objects:**

Initialize vector structure

For Each course

Get input from user to store data in structure

Store course Number

Store course name

Ask how many prerequisites there are

Store prerequisites

Write to file with the new structure and formatting

Ask if they would like to add an additional course

If no, break

**Search and print out Specifics:**

Declare search variable

Prompt user for the course they want info on

Set search variable equal to the course the user wants

Print Out course information

For all courses

If course is same as search variable

Print out course info

For each prerequisite

Print out prerequisite info

if course is not found

output “course is not found”

**Menu:**

While key is not 9

Print menu

If key = 1

LoadDataStructure()

Else If key = 2

PrintCourseList()

Else If key = 3

PrintCourse()

Else If key = 9

Return()

Repeat

**Print In alphabetical order:**

selectionSort(){

For all courses[i] {

For all courses[j] {

If course[j] < course[indexsmallest]

indexsmallest = I }

Temp = courses[i]

Courses[i] = course[indexsmallest]

Courses[indexsmallest] = temp

Repeat }

}

SelectionSort()

For all courses

Print course

**Hash Table Pseudocode:**

**File Management:**

ReadFile()

For loop to read each line in the file

Create data structure for the course information

Store each piece of data into the appropriate place

Make sure data is being read correctly(no errors)

Check to see if the prerequisite exists in the file

If prerequisite equals any course

Insert the course in the hash table

Else

Print out that there are errors in the file.

**Create and store objects:**

createClass()

for all courses in the file

Create key for given class

Retrieve a node using said key

If Node is empty

Set node equal to this key position.

Else

If node key is equal to UINT\_MAX

oldNode->key = key

oldNode->bid = bid

oldNode->next = NULL

else

find the next open node

set node equal to the next open node

**Search and print out Specifics:**

SearchAndPrint()

For all courses in the file

Create key for the given course

Set node equal to key

If node is not NULL

Print out the course node

If node is not found

Error

While node is not equal to NULL

If current node matches key

Print out the course not

Increment node

**Menu:**

While key is not 9

Print menu

If key = 1

LoadDataStructure()

Else If key = 2

PrintCourseList()

Else If key = 3

PrintCourse()

Else If key = 9

Return()

Repeat

**Print In alphabetical order:**

Copy Hash table to an array

Sort array with selectionsort Function above

For all elements in the array

Print course

**Tree Pseudocode:**

**File Management:**

ReadFile()

For loop to read each line in the file

Create data structure for the course information

Store each piece of data into the appropriate place

Make sure data is being read correctly(no errors)

Check to see if the prerequisite exists in the file

If prerequisite equals any course

Insert the course in the data tree.

Else

Print out that there are errors in the file.

**Create and store objects:**

createClass()

for all courses in the file

if root is null

set root equal to current course

else call AddNode()

addNode()

if course number is smaller than current node

if there is no left node

add course to left

else recurse down the left side

call addNode with new values

else if course is smaller than current node

if there is no right node

add course to right

else recurse down the right side

call addNode with new values

**Search and print out Specifics:**

SearchAndPrint()

Set current course equal to root node

While current course is not null

If current course equals the class that is being searched

Print out course information

If the course being searched is less than the current node

Traverse left

If the course being searched is greater than the current node

Traverse right

Repeat

**Menu:**

While key is not 9

Print menu

If key = 1

LoadDataStructure()

Else If key = 2

PrintCourseList()

Else If key = 3

PrintCourse()

Else If key = 9

Return()

Repeat

**Print In alphabetical order:**

TreePrintInOrder()

If node is null

Return

TreePrintInOrder(node->left)

Print node

TreePrintInOrder(node->right)

**Evaluation:**

**Run Time Evaluation:**

|  |  |
| --- | --- |
| **Data Structures** | **Results** |
| **Vector** | **Create File:** O(N)  **Storing Object:** O(N)  **Result:** O(N^2) |
| **Hash Table** | **Create File:** O(N)  **Storing Object:** O(N)  **Result:** O(N^2) |
| **Binary Tree** | **Create File:** O(N)  **Storing Object:** O(N \* LogN)  **Result:** O(N^2 \* LogN) |

**Pros and Cons of Each Structure:**

A vector’s main advantage is that it is easy to add and remove elements at the end of a list. So, if you know you are going to only be adding items to a list a vector may be the right choice. Furthermore, any element in a vector can be accessed by its index, which makes finding a particular value very easy. However, removing or adding elements in the middle of a vector list can be very tricky in comparison to the other data structures.

Hash tables are very efficient at storing data. It works by assigning data to a key value and storing it into an array. Storing data this way will sort like values close together in a hash table so that they can be located and retrieved quickly. It’s important to try and avoid collisions when designing your hash tables because they can slow down the process. To avoid collisions, you need to have a big enough hash table beforehand so there is always an available key for data entry.

Binary Trees are good for fast searching and fast inserting and deleting since everything is connected by pointers. Binary trees are relatively simple to set up since you can just add data points as needed and you don’t need to know the size of the data beforehand like a hash table or vector. One big disadvantage of a binary tree is that runtime complexity can increase if the tree is unbalanced. Also, memory usage tends to be higher with giant trees in comparison to other methods.

**Recommendations:**

Since the list of courses is quite small, I think a vector would be the way to go. I personally am much more comfortable coding with vectors in comparison to the other two. Since I am just adding courses to the vector as I go through the file, I don’t need to worry about adding and removing courses from the middle of the list. If I need to add or remove courses, I can just remove the course from the file and read the file again. Hash tables and Binary trees may be better if they wanted me to implement code for adding and removing courses but since that is not needed a vector will work just fine.