**Vector Data Pseudocode**

Function CreateCourseObjectsFromFile(filename):

Initialize an empty list called courses

Try to open file with courses

Display error message if unable to open file

While there are more than 2 pieces in a line

Read a line from courses file

Split line and use separator

If there are less than 2 pieces in a line

Skip task

Create a new course object

Set first piece to courseNumber

Set second piece to courseName

For remaining pieces, set as preReqs

Add preReqs to course’s list of preReqs

Add course to course’s list

Return

Function SearchAndPrintCourseInfo(courses, courseNumber):

For each course in courses

If course.courseNumber == courseNumber

Print course.courseNumber

Print course.courseName

If course has preReqs

Print preReqs from course’s preReqs

Main

Declare ‘courseData’ list

Declare ‘courseToSearch’ string

courseData = to contain file directory of course information

courseToSearch = user input for navigating through courses

SearchAndPrintCourseInfo(courseData, courseToSearch)

**Hash Table Pseudocode**

Define hash table structure for storing course information

Define function for loading data from file to hash table

Open file for reading

For each line in file

Split line into courseNumber, title, and preReqs

If length of preReqs is less than 2

Print error message

Continue

For each preReq in preReqs

If preReq is not in courseHashTable

Print error message

Continue

Create new course object named NewCourse

Add new course to the hash table

Close file

Define function for printing course info and preReqs

If courseHashTable contains courseNumber

Print course number, title, and preReqs

For each preReq

Print variable + preReq to display multiple preReqs

Else

Print error message if course is not found in courseHashTable

Define Main()

Initialize courseHashTable as new hash table

Load data from file into hash table

Print course information and preReqs

Receive input from user for courseNumber

Print courseInfo using input from courseNumber

**Tree Data Pseudocode**

Function openAndReadFile(filename)

Try to open filename

IF file cannot be opened

Print error message “file could not be opened”

Exit the program

Initialize an empty tree data structure

courseTree = InitializeTree()

For each line in filename

IF line has less than two parameters

PRINT Error message for invalid format

Exit the program

ELSE IF line has more than two parameters

Split line into courseNumber, courseTitle, and prerequisites

For each prerequisite in prerequisites

IF prerequisite is not in courseTree

PRINT error message, “Prerequisite is not found for course”

Exit the program

Create a new course Object with courseNumber, courseTitle, and prerequisites

newCourse = CreateCourse(courseNumber, courseTitle, prerequisites)

Insert course object into tree

InsertIntoTree(courseTree, newCourse)

Close filename

Function numPrerequisiteCourses(courseTree)

Initialize totalPrerequisites to 0

Traverse search tree to count prerequisites for courses

In each course, count prerequisites and add to totalPrerequisites

Return totalPrerequisites

Function createCourse(courseNumber, courseTitle, prerequisites)

Create new course object

newCourse = Course(courseNumber, courseTitle, prerequisites)

Return newCourse

Function printCourseInformation(courseTree, courseNumber)

Find course with specific course number in tree

foundCourse = findCourse(courseTree.root, courseNumber)

IF foundCourse != null

Print Course Information

Print prerequisite course information

ELSE

Print Course was not found in tree

Function printSampleSchedule(courseTree)

Initialize an empty schedule

Schedule = []

Traverse tree to obtain course information for schedule

Append course information to the schedule

PRINT collected schedule

**Menu Pseudocode**

Menu():

PRINT “Menu: “

PRINT “1. Load Data Structure”

PRINT “2. Show Course List”

PRINT "3. Show Course”

PRINT "4. Exit”

OBTAIN input from user

IF input == 1:

loadCoursesFromFile(filename)

ELSE IF input == 2:

printCourseList()

ELSE IF input == 3:

courseNumber = input(“Enter course number: “)

ELSE IF input == 4:

Exit menu

**Courses Pseudocode**

Function PrintSourtedCourseList(dataStructure):

IF dataStructure is “Vector”:

Sort vector data structure based on courseNumber alphanumerically

For each course in dataStructure:

PRINT course information

ELSE IF data Structure is “Hash Table”:

Create sorted list from hashtable data structure alphanumerically

For each key in Hash Table

OBTAIN course using key

PRINT course information

ELSE IF data Structure is “Tree”:

Call inOrderTraversal(dataStructure.root)

Function inOrderTraversal(node):

IF node != null:

inOrderTraversal(node.left)

PRINT course information

inOrderTraversal(node.right)

**Create Course Pseudocode**

Function CreateCourseObjectsFromFile(filename):

INITALIZE empty list called courses

Open filename with courses

DISPLAY error if file is unable to open

WHILE there are more than 2 pieces in line

Read line from course filename

Split line using separator

IF there are less than 2 pieces in line

Skip reading line and splitting line

CREATE new course object

First piece is set to courseNumber

Second piece is set to courseName

IF there are more pieces

Set pieces to preReqs

Add preReqs to course’s list of preReqs

Add course to the course’s list

Return

The total cost would be 6n+1. This is found by ‘n’ being used to represent the number of courses, the number of times the lines will execute, while excluding the lines that are used to call a function.

**Evaluation of Each Structure**

For a vector data structure, it is easy for implementation and is also efficient for accessing information by index. Vector data structures also tend to use less memory compared to hash table and tree. The cost of inserting and deleting can be costly due to the nature of the number of courses being added and dropped. Vectors are normally in a fixed size, which would make it very hard to make the courses list malleable to change.

For a hash table structure, they provide fast access to data stored and can be much better for data that is changing and resizing. In this case where we are using courseNumber to identify a course, a hash table is able to use courseNumber as a key to find the information regarding that searched course. Due to hash tables being more of a search structure by using keys, order of data seems to be a lesser feature if we wanted to sort the courses alphabetically. Since hash tables are much more susceptible to changes adding and dropping courses, they will use more memory compared to a vector structure.

For a tree structure, this is most suitable for a list that would want to be in some sort of alphabetical order. Tree structures tend to be the most complex out of the three structures due to the efficient insertions and deletions and they use extra node structures as well, which means more lines to process. A tree data structure is most suitable for data that has to be maintained ordered and be able to be changed for the future.

Each one of the data structures provides their own pros and cons when it comes to developing code. There will always be a faster alternative, but there will always be a more user-friendly alternative. It is important to note that including several of these structures may be a better method in the implementation of these data structures compared to only using one for your entire program. I would definitely recommend a combination of both tree structure and hash table structure for my code. The Hash Table will still maintain the accessible speed of retrieving information. The tree structure will allow the sorting of courses to be in alphabetical order which is important to have since we have a function to print all courses in alphanumeric order. Both structures also allow for insertions and deletions of courses and their prerequisites, which is an important feature to maintain in this project.