**// VECTOR DATA STRUCTURE**

Begin Struct

Declare Struct Courses

int courseNumber

string courseName

vector<string> coursePrerequisites

End Struct

// Declare vector object

vector<Courses> courses

// Declare struct object

Struct Courses structObject

Open file in READ Mode by calling the open() method

     BEGIN IF

if file doesn't exist:  
          DISPLAY "Unable to open the file"  
          return;

     ELSE:  
               BEGIN WHILE

while file is not an empty

                        Read a record as a string from the file and split it  
                        Check for At least two parameters courseNumber and CourseName

BEGIN IF

if number of parameters < 2

                                   DISPLAY ERROR Message  
                                   RETURN

                        ELSE

                                    Store the record into the structObject object  
                                    courses.push\_back(structObject)

                        END IF

                END OF WHILE

BEGIN LOOP

for each course

BEGIN LOOP

for each prerequisite

DISPLAY ERROR MESSAGE "Prerequisite course does not exist"

RETURN

END LOOP

END LOOP

END IF

Close file

End Struct

request user to input course number

loop through vector for all courses

IF the input is a valid position in the courseNumber vector.

print course information and prerequisites at the position in the vector specified by the user

ELSE

return error message stating invalid input or vector position choice

END IF

**// HASH TABLE DATA STRUCTURE**

Struct Courses structObject

Open file in READ Mode by calling the open() method

     IF file doesn't exist:  
          DISPLAY "Unable to open the file"  
          return;

     ELSE:  
               WHILE file is not an empty

                        Read a record as a string from the file and split it  
                        Check for At least two parameters courseNumber and CourseName

**IF Number of parameters < 2**

                                   DISPLAY ERROR Message  
                                   RETURN

                        ELSE

                                    Store the record into the structObject object  
                                    hashTable.add(structObject)

                        END IF

                END WHILE

BEGIN LOOP

for each course

BEGIN LOOP

for each prerequisite

BEGIN IF

if prerequisite does not exist

DISPLAY ERROR MESSAGE "Prerequisite course does not exist"

RETURN

END IF

END LOOP

END LOOP

END IF

Close file

End Struct

request user to input course number

find course in hashmap

BEGIN IF the input is a valid position in the courseNumber hash table

print course information and prerequisites at the position in the hash table specified by the user

ELSE

return error message stating invalid input or hash table position choice

END IF

**// TREE DATA STRUCTURE**

//Reading File:

Open file

Read data

IF the value is -1, then the file is not able to be found.

ELSE

WHILE file is not an empty

Read and parse each line

IF line has < two parameters

Return ERROR

ELSE

Read Each parameter

END IF

IF line has >= two parameters and they exist as course in file

continue

ELSE

Return Error

END IF

END IF

Close file

// Create Course Objects and store

Create a course object by asking user to input at least two parameters

IF user enters less than two parameters, return error

INT Course Vector

LOOP through the file

For every line in file

if size < two

return error

if size > two

pushback to add value( like course name, id, etc)

// search vector and print course info

Request user input

Loop through the vector for all course

If the input is the same as course number.

print course information

for each prerequisite of the course

print the prerequisite course information.

**// MENU**

int choice = 0

BEGIN WHILE choice != 9

PRINT “Menu”

PRINT “1. Load Courses”

PRINT “2. Display All Courses”

PRINT “3. Display a Particular Course”

PRINT “9. Exit”

PRINT “Enter Choice: ”

getline(choice)

END WHILE

BEGIN SWITCH (choice)

CASE 1

courses = loadCourses(csvPath)

PRINT “Courses loaded”

CASE 2

selectionSort(courses)

BEGIN LOOP

if i in courses list

PRINT courses[i]

END LOOP

CASE 3

int user input = 0

PRINT “Enter Course Identification Number”

user input = getNextLine()

BEGIN LOOP

for i in courses list

BEGIN IF

if user input = = courses[i]

PRINT courses[i]

break

END IF

BEGIN IF

if courses[i] is the last object in the courses list

PRINT ”Course Not Found”

END IF

END LOOP

END SWITCH

**// PRINT ALPHANUMERICALLY**

**// VECTOR**

vector<courses> courses

int minIndex

BEGIN LOOP

for i in courses

minIndex = i

BEGIN LOOP

for j = i + 1 in courses

BEGIN IF

if courses[j].title < courses[minIndex].title

minIndex = j

END IF

END LOOP

swap(courses[i], courses[minIndex])

END LOOP

BEGIN LOOP

for i in courses

PRINT courses[i]

END LOOP

**// HASH TABLE**

Add every entry from the hashmap to a vector called vector<courses> courses

int minIndex

BEGIN LOOP

for i in courses

minIndex = i

BEGIN LOOP

for j = i + 1 in courses

BEGIN IF

if courses[j].title < courses[minIndex].title

minIndex = j

END IF

END LOOP

swap(courses[i], courses[minIndex])

END LOOP

BEGIN LOOP

for i in courses

PRINT courses[i]

END LOOP

**// TREE**

BEGIN STRUCT

struct Course

string courseId

string title

string prerequisites

END STRUCT

BEGIN STRUCT

struct Node

Course course

Node \*left

Node \*right

BEGIN CONSTRUCTOR

Node()

left = nullptr

right = nullptr

END CONSTRUCTOR

Node (course aCourse) :

Node()

Course = aCourse

END STRUCT

BEGIN CLASS

class BinarySearchTree

private:

Node\* root

void printSortedList(Node\* node)

public:

BinarySearchTree()

virtual ~BinarySearchTree()

void printSortedList

BEGIN FUNCTION

BinarySearchTree::BinarySearchTree()

Root = nullptr

END FUNCTION

BEGIN FUNCTION

void BinarySearchTree::printSortedList()

printSortedList(root)

END FUNCTION

BEGIN FUNCTION

void printSortedList(Node\* node)

BEGIN IF

if node != nullptr

printSortedList(node -> left)

PRINT node -> courses.courseId “:” node -> courses.title “|” courses.prerequisites

printSortedList(node -> right)

END IF

END FUNCTION

END CLASS

BEGIN FUNCTION

Main()

BinarySearchTree\* tree

tree = new BinarySearchTree()

tree -> printSortedList

END FUNCTION

**// BIG O ANALYSIS**

**Vector:**

* Reading the file: O(n)
* Creating course objects: O(n)

**Hash Table:**

* Reading the file: O(n)
* Creating course objects: O(n)

**Tree:**

* Reading the file: O(n)
* Creating course objects: O(n log n)

**// Evaluation**

**Vectors:**

Advantages:

* Has a large memory capacity.
* Performs better when it comes to sequentially accessing data objects.

Disadvantages:

* It is expensive to insert and delete objects in the middle of the vector.
* Searching a vector is slower than trees or hash tables.

**Hash Tables:**

Advantages:

* Very fast at finding and returning data objects.

Disadvantages:

* Requires a vast amount of memory space.

**Trees:**

Advantages:

* Very efficient at sorting data objects.
* Very efficient at adding and removing data objects.

Disadvantages:

* Requires more memory space as opposed to a vector.

**// Evaluation**

While all three of the data structures can certainly get the job done, and fairly efficiently at that, I would recommend using the hash table data structure. The reason for this is mainly because hash tables are much faster at looking up, finding, and returning the course objects as opposed to trees and vectors. And ultimately, our main goal should be to provide the end user with the greatest experience possible. One way we can do this is to provide them with the fastest lookup times possible for course information, and hash tables, I believe, are the best choice for this.