// initialize variables

Initialize class Course {

string courseID

string courseName

vector<string> coursePrereqs

};

Initialize vector<Course> courses;

Initialize string line variable for file navigation;

// load file and place data into courses vector

Open file into file variable using ifstream;

while going through each line in the file {

vector<string> courseLine

Course newCourse

parse each line seperated by “,” into courseLine vector

if courseLine.size < 2, return error “invalid data file”

for each element in courseLine {

if element index equals 0, add to newCourse.courseID

if element index equals 1, add to newCourse.courseName

if courseLine.size > 2

if element at index courseLine.size -1 does not equal courses.Course.courseID, return “prerequisites not found”

if element index > 2, pushback to newCourse.course

}

}

pushback newCourse to courses

};

Close file;

// search vector for matching course and print

get string searchID from input

bool courseFound equals false

for every course in courses {

if course.courseID equals searchID {

courseFound equals true

then print

“Course ID: “ << course.courseID

“Course Name: “ << course.courseName

“Prerequisites: “

for every prereq in course.coursePrereqs, print prereq

}

}

if courseFound equals false, print “Course not found”

Hash Table Pseudocode

1. **Design pseudocode to define how the program opens the file, reads the data from the file, parses each line, and checks for file format errors.**

// initialize variables

\Initialize class Course {

Int courseNumber

string courseID

string courseName

vector string coursePrereqs

};

Read Date file function()

{

Initialize string for fData variable for navigation of file

///Load the file and place the data

Open file using fstream

WHILE file is open

Read file data

Parse the lines

}

i = 0, j = 0

Flag = TRUE

WHILE (NOT END OF FILE f)

courseInfo[] = SPLIT (READLINE(f, line), DELIMETER = , )

APPEND line TO lines

IF (LENGTH of courseInfo < 2)

Flag = FALSE

BREAK

END IF

courseNumbers[i] = courseInfo[0]

courseTitles[i] = courseInfo[1]

INCREMENT i

IF (LENGTH of courseInfo > 2)

FOR k = 2 to LENGTH of courseInfo

prerequisites[j] = courseInfo[k]

INCREMENT j

END FOR

END IF

END WHILE

IF Flag == TRUE

FOR each P in prerequisites

IF P NOT IN courseNumbers

Flag = FALSE

BREAK

END IF

END FOR

END IF

RETURN Flag

END FUNCTION

1. **Design pseudocode to show how to create course objects and store them in the appropriate data structure**.

CLASS Course

Number: String

Title: String

Prerequisites[]: String[]

CONSTRUCTOR Course(line)

Number = SPLIT(line, DELIMETER = ,)[0]

Title = SPLIT(line, DELIMETER = ,)[1]

IF LENGTH of SPLIT(line, DELIMETER = ,) > 2

Prerequisites = SPLIT(line)[ 2 to LENGTH of SPLIT (line, DELIMETER = ,)]

END IF

END CONSTRUCTOR

END CLASS

FUNCTION createObject(Courses <Course>, File f)

Lines[] = " "

IF readFile(f, Lines) == TRUE

FOR each Line in Lines

APPEND NEW Course(Line) TO Courses

END FOR

END IF

ELSE PRINT("File cannot be read")

END ELSE

END FUNCTION

1. **Design pseudocode that will print out course information and prerequisites**.

FUNCTION MAIN()

Filename = INPUT()

File F = NEW File(Filename)

Courses <Course> : vector

CALL : createObject(Courses, F)

CourseNumber = INPUT()

IF Courses is EMPTY

PRINT ("No objects read from the file")

END IF

ELSE

printCourseInformation (Courses, CourseNumber)

END ELSE

END FUNCTION

Tree Pseudocode

1. **Design pseudocode to define how the program opens the file, reads the data from the file, parses each line, and checks for file format errors.**

// initialize variables

Initialize class Course {

Int courseNumber;

string coursed;

string courseName;

vector <string> coursePrereqs;

struct\_tree node\*left;

struct\_tree nide\*right;

tree root = null

}

ReadDataFile Function()

{

Initialize string for fData variable for navigation of file

///Load the file and place the data

Open file using fstream

WHILE file is open

Read file data

Parse the lines

Verify the course name and course number

If root!=null

Verify if it’s a CoursePrereqs

CoursePrereqs go to right node

If course < 2

Add to left node

Else

Display message error

Else

Add CourseName, CourseNumber, and CoursePrereqs to right node

Print Results

File closed

End

1. **Design pseudocode to show how to create course objects and store them in the appropriate data structure**.

Program start

// initialize variables

Initialize class Course {

Int courseNumber;

string coursed;

string courseName;

vector <string> coursePrereqs;

struct\_tree node\*left;

struct\_tree nide\*right;

tree root = null

}

Initialize string for fData variable for navigation of file

///Load the file and place the data

Open file using fstream

WHILE file is open

Read file data

Parse the lines

Verify the course name and course number

IF root == null

Search for COURSE

IF COURSE found

Create object for COURSE;

COURSE is added to right node

IF COURSE not found

Print error

File close

Print Objects

End

1. **Design pseudocode that will print out course information and prerequisites**.

// initialize variables

Initialize class Course {

Int courseNumber;

string coursed;

string courseName;

vector <string> coursePrereqs;

struct\_tree node\*left;

struct\_tree nide\*right;

tree root = null

}

Initialize string for fData variable for navigation of file

///Load the file and place the data

Open file using fstream

WHILE file is open

Read file data

Parse the lines

Verify the course name and course number

IF root == null

Verify courseName and CourseNumber

IF root == null

Look for a CoursePrereqs

CoursePrereqs go to right node

If Course < 2

Add course to left node

Else

Add courseName, CourseNumber, and CoursePrereqs to right node

Print Results

File Close

End

**Pseudocode for Menu**

User Prompt Menu

WHILE user input != 9;

{

Print 1. “Load the data”

Print 2. “Course List”

Print 3. “Course”

Print 9. “Exit Program”

}

User prompt “Menu Selection”

User Selection 1

Program loads course data

Break

User prompt “Menu Selection”

User Selection 2

Program prints out course list

Break

User prompt “Menu Selection”

User Selection 3

Program prints out course info

Break

User Prompt “menu Selection”

User selction 9

Print “Thank You. Goodbye.

**c**

1. **Runtime Analysis**

| **Code(vector)** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 1 | n | n |
| **if the course is the same as courseNumber** | 1 | n | n |
| **print out the course information** | 2 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 2 | n | n |
| **Total Cost** | | | 6n + 1 |
| **Runtime** | | | O(n) |
|  | | |  |
|  | | |  |

| **Code(Hash Table)** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 2 | n | n |
| **if the course is the same as courseNumber** | 1 | n | n |
| **print out the course information** | 2 | 1 | 1 |
| **for each prerequisite of the course** | 2 | n | n |
| **print the prerequisite course information** | 2 | n | n |
| **Total Cost** | | | 8n + 1 |
| **Runtime** | | | O(n) |
|  | | |  |
|  | | |  |

| **Code(Tree)** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 2 | n | n |
| **if the course is the same as courseNumber** | 1 | n | n |
| **print out the course information** | 2 | 1 | 1 |
| **for each prerequisite of the course** | 2 | n | n |
| **print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | 7n + 1 |
| **Runtime** | | | O(n) |
|  | | |  |

1. Evaluations

Vectors

Vectors are unidimensional, which is an advantage since we are creating a vector for reading files and creating a table for courses. Vectors also use less memory when created, but the disadvantage comes when deleting elements, which cannot be deleted, and not able to change it into different data types.

Hash Table

Hash tables are created with a unique key that helps with organizing and storing the information, this is an advantage as you can search for a specific item by that unique key. The only disadvantage of the hash table is the speed of the run time.

Tree

The advantage of a tree is that the organization is a lot better than a hash table and a vector, but the disadvantage of the hash table is that the run time is quite long and makes for a longer time to modify it.

1. Considering all 3 of the data structures, hash table would be the best option as it offers organization without sacrificing the runtime as the tree structure, but also gives a better organization than a vector structure. The table will offer better sorting for the courses and allow us to modify it easily without causing any problems.