***Vector sorting pseudocode***

class Course {

int courseNumber

string courseName

vector<string>PreRequisite

}

Course Search (vector <Course> courses, string courseNum) {

create empty course

for (each Course in courses)

if(current courseNumber ==courseNum)

return course

return empty

}

//Open file. Store info in vector  
Vector <string> OpenReadFile (string filename) {

initialize vector<string>

initialize string line to hold single line

initialize ifStream instream to capture contents of file

open file with instream with file name

if unable to open file

output “Unable to open file”

pull line from instream until all info received

push line to back

close file

return info

}

//Store info and create a course for each line and store in vector  
vector <Course\*> CreateSchedule (vector<string> contents) {

initialize vector<Course>

initialize stringstream lineStream

initialize string token to store words from line

initialize int count to track tokens per line

for (every new course)

set count to 0

creater new Course course

fill lineStream with line contents

pull token from LineStream until end of line

if(count = 0)

course courseNumber = token

++count

else if(count == 1)

courseName = token

++count

else

if(token = existing courseNumber)

push token to back of course PreRequisite

else

output “PreRequisite must be previously taken.”

++count

If (count < 2)

Output “File format error. Check course number and name”

empty lineStream

push course to end of courses

return courses

}

Void Print(vector <course> courses, string courseNum){

Create course object

If(course returned is empty)

Output “Course not in schedule.”

Return

Output course courseNumber and courseName

For (PreRequisite in prerequisite)

Output prerequisite

}

***Hash table pseudocode***

Program start

Open file

Read data

Parse lines

Check course title

Check course number

If prerequisite found

Add to array

If course parameters are less than two

Skip course

Error message displayed: “File not formatted correctly.”

End

Else

Add course name, number, prerequisite to hash table

If prerequisite found

Check if prerequisite prior to course

Add to hash table

If prerequisite not found

Skip course

Display error “Prerequisite course not found.”

Create constructor and parameters

Call constructor GenerateCourseObj

Initialize variables for courses and read file

Open the file to reread

While file is open

Store the course object in hash table

Create constructor and parameters

Change constructor to LocateSpecificCourse

Initialize variables to open file

Open file

While file open

Print course info

Store collected data in hash table

***Tree table pseudocode***

FUNCTION readFile(File A, lines[])

courseTitles[],courseNumbers[],prerequisites[], line

C = 0, B = 0

Code = TRUE

WHILE

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

APPEND line TO lines

IF (LENGTH of courseInfo < 2)

code = FALSE

BREAK

END IF

courseNumbers[C] = courseInfo[0]

courseTitles[B] = courseInfo[1]

INCREMENT A

IF (LENGTH of courseInfo > 2)

FOR k = 2 to LENGTH of courseInfo

prerequisites[B] = courseInfo[k]

INCREMENT B

END FOR

END IF

END WHILE

IF Flag == TRUE

FOR each K in prerequisites

IF K NOT IN courseNumbers

Flag = FALSE

BREAK

END IF

END FOR

END IF

RETURN Code

END FUNCTION

}

}

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 A)

Lines[] = " "

IF readFile(f, Lines) == TRUE

FOR each Line in Lines

APPEND NEW Course(Line) TO Courses

END FOR

END IF

ELSE PRINT("File unable to be read")

END ELSE

END FUNCTION

FUNCTION MAIN()

Filename = INPUT()

File A = NEW File(Filename)

Courses <Course> : vector

CALL : createObject(Courses, A)

CourseNumber = INPUT()

IF Courses is Null

PRINT ("No.")

END IF

ELSE

printCourseInformation (CourseNumber, Courses)

END ELSE

END FUNCTION

}

***Menu pseudocode***

Create integer for switch statement named MenuInput, set to 0

While choice does not equal 9

Print “1. Load Data”

Print “2. Course List”

Print “3. Course”

Print "9. Exit”

Print "Please make selection”

Switch (MenuInput)

Case 1:

Loads course data

Case 2:

Print course list

Case 3:

Print Course

Case 9:

Print “Goodbye”

***Arrange courses in alphanumeric order pseudocode***

Create string used for sorting, string s

Create char to set length +1

Create string to array

Sort array

Create integers alphabet and numbers

While alphabet < 97

Set alphabet +1

If I < 97, set number +1

Else

Set alphabet +1

Return

Create string Classes

Print classes in alphanumeric order

***Vector runtime analysis***

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **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 info | 2 | n | n |
|  |  | **Total** **Cost**: | 6n +1 |
|  |  | **Runtime**: | 1(n) |

***Hash table runtime analysis***

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **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 | 1 | 1 | 1 |
| For each prerequisite of the course | 2 | n | n |
| Print the prerequisite course info | 4 | n | n |
|  |  | **Total** **Cost**: | 9n +1 |
|  |  | **Runtime**: | 0(n) |

***Binary Tree runtime analysis***

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **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 info | 4 | n | n |
|  |  | **Total** **Cost**: | 8n +1 |
|  |  | **Runtime**: | 0(n) |

***Vector***

* Pro: one-dimensional which is advantageous since only dealing with courses and reading files
* Con: unable to delete elements, unable to handle multiple media types

***Hash table***

* Pro: allows organization and storage of info, ability to create, delete, call, create unique elements, and synchronize
* Con: can be slow thanks to synchronization

***Binary tree***

* Pro: better organization, can be expanded, ability to run searches
* Con: can be slow to modify

Although using vector data structures can be less costly, it has limitations. Therefore, I would utilize hash tables. Yes, they can be a bit slower but their ability to modify and organize as the client wishes give it an advantage. Hash tables gives the client to the ability to add to and modify the table as the project expands in functionality.