Project One

CS300

Jake Whaley

8/10/2025

—----------------------—----------------------

**Shared Pseudocode**

—----------------------—----------------------

—----------------------—----------------------

**Open the file**

—----------------------—----------------------

Invoke Ifstream to read input files

Void loadCourseInfo(String csvPath, VectorDataStructure)

If file cannot be opened

Cout << ERROR, CANNOTOPEN FILE

Else

Read File

End If

—----------------------—----------------------

**Read the file**

—----------------------—----------------------

While not at End of file

Read line

If line is not empty

Add line to fileLines

Split line by comma into tokens

If tokens.size <2

Cout << MISSING DATA

Return

End If

Add tokens[0] (course number) to courseNumbers

End If

Close file

—----------------------—----------------------

**Validating prerequisites**

—----------------------—----------------------

For each line in fileLines

Split line by comma into tokens

For index from 2 to tokens.size - 1

Prerequisite = tokens[index]

If prerequisite is not in courseNumbers

Cout << prerequisite not found in file

Return

End If

End For

End For

Call StoreCourses(fileLines)

End Function

—----------------------—----------------------

**Define course objects**

—----------------------—----------------------

Struct Course

String courseNumber

String courseTitle

vector<string> prerequisites

End Struct

—----------------------—----------------------

—----------------------—----------------------

**Vector Specific Pseudocode**

—----------------------—----------------------

—----------------------—----------------------

**Load data from file into vector**

—----------------------—----------------------

void loadCourses(Vector<Course> &courses, string fileName) {

open fileName for reading

if file cannot be opened

print error and return

vector<string> allCourseNumbers

—----------------------—----------------------

**Store all course numbers**

—----------------------—----------------------

while not end of file

read line

if line is empty

continue

split line by "," into tokens

if tokens.size < 2

print error and exit

add tokens[0] to allCourseNumbers

rewind file to beginning

—----------------------—----------------------

**Create Course objects**

—----------------------—----------------------

while not end of file

read line

if line is empty

continue

split line by "," into tokens

Course newCourse

newCourse.courseNumber = tokens[0]

newCourse.courseTitle = tokens[1]

for i from 2 to tokens.size - 1

if tokens[i] not in allCourseNumbers

print error and exit

add tokens[i] to newCourse.prerequisites

add newCourse to courses

close file

}

—----------------------—----------------------

**Search for and print course information**

—----------------------—----------------------

void searchCourse(Vector<Course> courses, string courseNumber) {

for all course in courses

if course.courseNumber == courseNumber

print courseNumber and courseTitle

if course.prerequisites is empty

print "No prerequisites"

else

print "Prerequisites:"

for each prereq in course.prerequisites

print prereq

return

print "Course not found"

}

—----------------------—----------------------

—----------------------—----------------------

**Hash Table Specific Pseudocode**

—----------------------—----------------------

—----------------------—----------------------

**Store course objects in hash table**

—----------------------—----------------------

Function StoreCourses(vector<string> fileLines)

Declare HashTable courseTable

For each line in fileLines

Split line by comma into tokens

Create new Course object

course.courseNumber = tokens[0]

course.courseTitle = tokens[1]

For index from 2 to tokens.size - 1

Add tokens[index] to course.prerequisites

End For

Insert course into courseTable using course.courseNumber as the key

End For

End Function

—----------------------—----------------------

**Search for a course**

—----------------------—----------------------

Function searchCourse(HashTable<Course> Courses, String courseNumber) {

Take User Input

Assign input as key

For all courses

Check to see if key matches hash

If no matching hash is found

cout << ERROR, NO SUCH COURSE

Return

End If

Print course info (title/number)

If course has prereqs

cout << COURSE PREREQS HERE

Else

cout << NO REQUIRED PREREQS

End If

}

End Function

—----------------------—----------------------

**Print all courses**

—----------------------—----------------------

Function PrintCourses(HashTable courseTable)

Declare vector<Course> allCourses

For each bucket in courseTable

For each Course in bucket

Add Course to allCourses

End For

End For

Sort allCourses by courseNumber

For each Course in Courses

Cout << Course.courseNumber

Cout << Course.courseTitle

End For

End Function

—----------------------—----------------------

**Print one course**

—----------------------—----------------------

Function PrintCourseDetails(HashTable courseTable, string courseNumber)

Course = courseTable.Search(courseNumber)

If Course is not found

Cout << COURSE NOT FOUND

End If

Cout << “Course number: ” + courseNumber

Cout << “Title: “ + courseTitle

If Course.prerequisites is empty

Cout << NO PREREQS

Else

Cout << “Prerequisites: “+ join Course.prerequisites with “,”

End If

End Function

—----------------------—----------------------

**Binary Search Tree Specific Pseudocode**

—----------------------—----------------------

—----------------------—----------------------

**Tree node Structure**

—----------------------—----------------------

Struct TreeNode

Course courseData

TreeNode left

TreeNode right

End Struct

—----------------------—----------------------

**Define Binary search tree class**

—----------------------—----------------------

Class BinarySearchTree

TreeNode root

—----------------------—----------------------

**Insert Node Function**

—----------------------—----------------------

Function insertNode(node, course):

If course.courseNumber < node.courseData.courseNumber:

If node.left is Null:

node.left-> NEW TreeNode(course)

Else:

Call insertNode(node.left, course)

Else If course.courseNumber > node.courseData.courseNumber:

If node.right is Null:

node.right -> NEW TreeNode(course)

Else:

Call insertNode(node.right, course)

End If

—----------------------—----------------------

**Insert course Function**

—----------------------—----------------------

Function insert(course):

If root is Null:

root-> NEW TreeNode(course)

Else:

Call insertNode(root, course)

—----------------------—----------------------

**Search node Function**

—----------------------—----------------------

Function searchNode(node, courseNumber):

If node is Null:

Return Null

Else If courseNumber == node.courseData.courseNumber:

Return node.courseData

Else If courseNumber < node.courseData.courseNumber:

Return searchNode(node.left, courseNumber)

Else:

Return searchNode(node.right, courseNumber)

—----------------------—----------------------

**Search course Function**

—----------------------—----------------------

void searchCourse(Tree<Course> courses, String courseNumber) {

Return searchNode(root, courseNumber)

}

—----------------------—----------------------

**Printing a course Function**

—----------------------—----------------------

Function inOrderPrint(node):

If node is not Null:

Call inOrderPrint(node.left)

Cout <<”Course number:” + courseNumber

Cout << “Title” + courseTitle

Call inOrderPrint(node.right)

—----------------------—----------------------

**Print courses in order**

—----------------------—----------------------

print "All Courses:"

Call courseTree.inOrderPrint(courseTree.root)

Function printCourse(courseTree, courseNumber):

foundCourse -> courseTree.search(courseNumber)

If course not found:

Cout << COURSE NOT FOUND

Else:

Cout <<”Course number:” + courseNumber

If Length(foundCourse.prerequisites) > 0:

print "Prerequisites:", + join Course.prerequisites with “,”

Else:

Cout << NO PREREQS

—----------------------—----------------------

**Project One Specific Pseudocode**

—----------------------—----------------------

—----------------------—----------------------

**Print courses in alphanumeric order: Vector**

—----------------------—----------------------

Void printCourseList(Vector<course> courses) {

If courses is empty

Cout << “Error: You must load data before attempting to print the courses.”

Return

Sort courses by courseNumber(ascending)

For each course in courses

Cout << course.courseNumber + “,” + course.courseTitle

}

—----------------------—----------------------

**Print courses in alphanumeric order: Binary Search Tree**

—----------------------—----------------------

Void printCourseListBinarySearchTree(Node\* root) {

If root is null

Cout << "Error: You must load data before attempting to print the courses.”

Return

inOrderPrint(root)

}

Void inOrderPrint(Node\* node){

If node is not Null:

Call inOrderPrint(node.left)

Cout <<”Course number: ” + courseNumber << endl;

Cout << “Title: ” + courseTitle << endl;

Call inOrderPrint(node.right)

}

—----------------------—----------------------

**Display Menu**

—----------------------—----------------------

Void displayMenu() {

Cout << “1. Load Data Structure” << endl;

Cout << “2. Print Course List” << endl;

Cout << “3. Print Course” << endl;

Cout << ”9. Exit” << endl;

}

—----------------------—----------------------

**Main method**

—----------------------—----------------------

Vector<Course> courses

String fileName = “courses.txt”

Int choice = 0

While choice != 9

displayMenu()

Take user input = choice

If choice = 1

loadCourses(courses, fileName)

Cout << “Courses loaded”

Else if choice = 2

If courses is empty

Cout << “Error, must load data before printing course list”

Else

Sort courses by courseNumber (ascending)

For each course in courses

Cout << course.coursenumber + “,” + course.courseTitle

Else if choice = 3

If courses is empty

Cout << “Error, must load data before printing a course”

Else

Cout << “Enter course number”

Input = findNum

searchCourse(courses, findNum)

Else if choice = 9

Cout << “Goodbye”

Else

Cout << “You entered: “ + choice + “ which is invalid. Please enter a valid input.”

—----------------------—----------------------

**Worst-Case Runtime Analysis**

—----------------------—----------------------

—----------------------—----------------------

**Data Structure: Vector**

—----------------------—----------------------

| Step | Cost per Execution | Executions | Total Cost | Big O |
| --- | --- | --- | --- | --- |
| Open the file | 1 | 1 | 1 | O(1) |
| Read line from file | 1 | n | n | O(n) |
| Parse course number | 1 | n | n | O(n) |
| Parse course title | 1 | n | n | O(n) |
| Parse prerequisites | P (number of prerequisites) | n | n\*p | O(n) |
| Create course object | 1 | n | n | O(n) |
| Add to vector | 1 | n | n | O(n) |
| Total |  |  | 3\*n + n\*p + 1 | O(n) |

—----------------------—----------------------

**Data Structure: Hash Table**

—----------------------—----------------------

| Step | Cost per Execution | Executions | Total Cost | Big O |
| --- | --- | --- | --- | --- |
| Open the file | 1 | 1 | 1 | O(1) |
| Read line from file | 1 | n | n | O(n) |
| Parse course number | 1 | n | n | O(n) |
| Parse course title | 1 | n | n | O(n) |
| Parse prerequisites | P (number of prerequisites) | n | n\*p | O(n) |
| Create course object | 1 | n | n | O(n) |
| Add to hash table | O(1) | n | n | O(n) |
| Total |  |  | 4\*n + n\*p | O(n) |

—----------------------—----------------------

**Data Structure: Binary Search Tree**

—----------------------—----------------------

| Step | Cost per Execution | Executions | Total Cost | Big O |
| --- | --- | --- | --- | --- |
| Open the file | 1 | 1 | 1 | O(1) |
| Read line from file | 1 | n | n | O(n) |
| Parse course number | 1 | n | n | O(n) |
| Parse course title | 1 | n | n | O(n) |
| Parse prerequisites | P (number of prerequisites) | n | n\*p | O(n) |
| Create course object | 1 | n | n | O(n) |
| Add to Binary Search Tree | O(log c)  (c courses as of yet) | n | n\*log(n) | O(n log(n)) |
| Total |  |  | n(log (n)) + 3\*n + n\*p | O(n log(n)) |

—----------------------—----------------------

**Analysis**

—----------------------—----------------------

There are advantages and disadvantages that accompany the use of any one of the three data structures that have been analyzed. Assuming that the number of courses is not incredibly large, advantages of vector data structures include their simplicity and efficiency for storing courses. These advantages diminish as the size of the data set increases, as the quick sorting algorithms that can be implemented with vector data structures are not incredibly efficient when used to sort large data sets. This is not a large issue if the courses do not need to be sorted frequently, but the vector will need to be sorted again every time that a new course is added or a course ID is changed.

Hash tables are incredibly efficient in terms of inserting new courses into the data set and looking up individual courses. If the administrators did not require the courses to be sorted alphanumerically, this would be my recommendation. However, hash tables do not inherently store the data that they contain in a sorted manner, meaning that the data would need to be exported to a different data structure to be sorted. This means that hash tables do not satisfy the administrator’s requirements, and would not be suitable for their desired system.

A binary search tree would inherently keep the courses sorted alphanumerically, which would enable the administrators to efficiently add, remove, and sort through the courses. However, this is only true if the tree is well balanced, meaning that in order for it to perform optimally there would need to be additional overhead implemented. Due to the nature of the client’s business, I do not believe that it would be worthwhile to implement the features required to dynamically balance the tree. This is because while the content of the courses may change, the course numbers, IDs, and prerequisites would be the same for the length of the university’s term, meaning that insertions, deletions, and sorting would not need to be done very frequently.

Overall, I would recommend using a vector data structure for use in my code. I make this recommendation based on the simplicity and practicality that comes with implementing and using vector data structures. While the Big O analysis for a vector is not ideal when compared to a balanced tree, the administrator’s use case does not seem to require sorting the data very frequently. A hash table would be better for quickly looking up a course, but it does not support maintaining an alphanumerically sorted list in the same way that a vector does