Connor Holohan

Cs300

4/15/25

PROJECT 1

**VECTOR SORT UPDATED**

// Function to read from file

FUNCTION readCoursesFromFile(fileName: STRING)

OPEN file fileName FOR READING

IF file is NULL

PRINT ERROR

RETURN

END IF

// Read the file line by line

WHILE NOT EOF(file)

READ a line from the file

SPLIT the line into params

// Check for number and title

IF number is less than 2

PRINT ERROR

CONTINUE to next line

END IF

// Create a new course

CREATE newCourse as Course

SET newCourse.number = param[0]

SET newCourse.title = param[1]

IF the number of param is more than 2

FOR each param after the title

ADD param to newCourse.prereqs

END FOR

END IF

// Add the new course to the list of courses

ADD newCourse to courses

END WHILE

CLOSE file

END FUNCTION

// Function to check if prerequisites are valid (must exist in the list)

FUNCTION validatePrereqs(courses: VECTOR)

// Go through each course and check its prereqs

FOR each course in courses

FOR each prereq in course.prereqs

SET found = FALSE

// Look for the prereq in the list of courses

FOR each otherCourse in courses

IF prereq == otherCourse.number

SET found = TRUE

BREAK

END IF

END FOR

// If the prereq wasn't found, show an error

IF found == FALSE

PRINT ERROR

END IF

END FOR

END FOR

END FUNCTION

// Function to search for a course and print its details

FUNCTION printCourseInfo(courses: VECTOR, courseNumber: STRING)

SET found = FALSE

// Look for the course by course number

FOR each course in courses

IF course.number == courseNumber

PRINT "Course Number: " + course.number

PRINT "Course Title: " + course.title

PRINT "Prerequisites: "

// If no prerequisites, print "None"

IF course.prereqs is EMPTY

PRINT none

ELSE

// Print each prerequisite

FOR each prereq in course.prereqs

PRINT prereq

END FOR

END IF

SET found = TRUE

BREAK

END IF

END FOR

// If the course isn't found, show an error

IF found == FALSE

PRINT ERROR

END IF

END FUNCTION

// Function to sort and print all courses in alphanumeric order

FUNCTION printAllCoursesSorted(courses: VECTOR)

// Sort the courses vector by course number (alphanumerically)

SORT courses BY course.number ascending

// Print each course number and title

FOR each course in courses

PRINT course.number + ": " + course.title

END FOR

END FUNCTION

// Menu function

FUNCTION showMenu()

PRINT "1. Load course data"

PRINT "2. Print course list"

PRINT "3. Print course details"

PRINT "9. Exit"

END FUNCTION

// Main program that runs everything

FUNCTION main()

SET fileName = "courses.txt"

SET courses = empty VECTOR

SET loaded = FALSE

SET choice = 0

WHILE choice != 9

CALL showMenu()

READ user input INTO choice

IF choice == 1

CALL readCoursesFromFile(fileName)

CALL validatePrereqs(courses)

SET loaded = TRUE

ELSE IF choice == 2

IF loaded == FALSE

PRINT "Please load course data first."

ELSE

CALL printAllCoursesSorted(courses)

END IF

ELSE IF choice == 3

IF loaded == FALSE

PRINT "Please load course data first."

ELSE

PRINT "Enter course number:"

READ user input INTO courseNum

CALL printCourseInfo(courses, courseNum)

END IF

ELSE IF choice == 9

PRINT "Goodbye."

ELSE

PRINT "Invalid option."

END IF

END WHILE

END FUNCTION

**HASH UPDATE**

class Item:

id: STRING

name: STRING

requirements: list<STRING>

FUNCTION readDataFromFile(fileName: STRING)

OPEN file fileName FOR READING

IF file is NULL

PRINT "ERROR: Can't open file"

RETURN

END IF

WHILE NOT EOF(file)

READ a line from the file

SPLIT the line into parts

IF length of parts < 2

PRINT "ERROR: Missing ID or Name"

CONTINUE to next line

END IF

CREATE newItem as Item

SET newItem.id = parts[0]

SET newItem.name = parts[1]

IF length of parts > 2

FOR each part from 2 to length of parts - 1

ADD part to newItem.requirements

END FOR

END IF

ADD newItem to dataList

END WHILE

CLOSE file

END FUNCTION

FUNCTION checkRequirements(dataList: VECTOR)

FOR each item in dataList

FOR each req in item.requirements

SET found = FALSE

FOR each otherItem in dataList

IF req == otherItem.id

SET found = TRUE

BREAK

END IF

END FOR

IF found == FALSE

PRINT "ERROR”

END IF

END FOR

END FOR

END FUNCTION

FUNCTION displayItemInfo(dataList: VECTOR, itemId: STRING)

SET found = FALSE

FOR each item in dataList

IF item.id == itemId

PRINT "ID: " + item.id

PRINT "Name: " + item.name

PRINT "Requirements: "

IF item.requirements is EMPTY

PRINT "None"

ELSE

FOR each req in item.requirements

PRINT req

END FOR

END IF

SET found = TRUE

BREAK

END IF

END FOR

IF found == FALSE

PRINT "ERROR: Item " + itemId + " not found"

END IF

END FUNCTION

FUNCTION displayAllItemsSorted(dataList: VECTOR)

SORT dataList BY item.id ascending

FOR each item in dataList

PRINT item.id + ": " + item.name

END FOR

END FUNCTION

FUNCTION showMenu()

PRINT "====== Menu ======"

PRINT "1. Load data from file"

PRINT "2. Display all items (sorted)"

PRINT "3. Display item info"

PRINT "9. Exit"

END FUNCTION

FUNCTION main()

SET fileName = "data.txt"

SET dataLoaded = FALSE

SET dataList = empty VECTOR

SET choice = 0

WHILE choice != 9

CALL showMenu()

PRINT "Enter your choice:"

READ user input INTO choice

IF choice == 1

CALL readDataFromFile(fileName)

CALL checkRequirements(dataList)

SET dataLoaded = TRUE

PRINT "Data loaded successfully"

ELSE IF choice == 2

IF dataLoaded == FALSE

PRINT "ERROR: Load data first using option 1"

ELSE

CALL displayAllItemsSorted(dataList)

END IF

ELSE IF choice == 3

IF dataLoaded == FALSE

PRINT "ERROR: Load data first using option 1"

ELSE

PRINT "Enter item ID:"

READ user input INTO itemId

CALL displayItemInfo(dataList, itemId)

END IF

ELSE IF choice == 9

PRINT "Exiting program..."

ELSE

PRINT "Invalid choice. Please select again."

END IF

END WHILE

END FUNCTION

**TREEDATA UPDATE**

CLASS Course:

num: STRING

title: STRING

prereqs: list<STRING>

CLASS Node:

course: Course

left: Node

right: Node

CLASS Tree:

root: Node

FUNCTION insert(c: Course)

IF root == NULL

root = NEW Node

root.course = c

RETURN

END IF

curr = root

WHILE TRUE

IF c.num < curr.course.num

IF left == NULL

left = NEW Node

left.course = c

RETURN

ELSE

curr =.left

END IF

ELSE

IF right == NULL

right = NEW Node

right.course = c

RETURN

ELSE

curr =right

END IF

END IF

END WHILE

END FUNCTION

FUNCTION print(Node)

IF node == NULL RETURN

CALL print(left)

PRINT node.course.num+ node.course.title

IF node.course.prereqs IS EMPTY

PRINT "Prereqs None"

ELSE

PRINT PREREQ

END IF

CALL print(right)

END FUNCTION

END CLASS

FUNCTION loadCourses(fileName, Tree, Hashtable)

OPEN fileName FOR READING AS file

IF file == NULL

PRINT "Can't open”

RETURN

END IF

WHILE NOT EOF(file)

line = READ line

parts = SPLIT line BY space

IF LENGTH(parts) < 2

PRINT "Invalid

CONTINUE

END IF

c = NEW Course

c.num = parts[0]

c.title = parts[1]

c.prereqs = parts[2 TO END]

map[c.num] = c

CALL tree.insert(c)

END WHILE

CLOSE file

END FUNCTION

FUNCTION checkPrereqs(Hashtable)

FOR EACH course IN map

FOR EACH p IN course

IF p NOT IN map

PRINT "Missing prereq: " + p

END IF

END FOR

END FOR

END FUNCTION

FUNCTION printCourseInfo(map, courseNum)

course = map[courseNum]

PRINT "Course Number: " + course.num

PRINT "Course Title: " + course.title

IF course.prereqs IS EMPTY

PRINT "Prereqs: None"

ELSE

PRINT "Prereqs: "

FOR EACH p IN course.prereqs

PRINT p

END FOR

END IF

END FUNCTION

FUNCTION showMenu()

PRINT "1. Load course data"

PRINT "2. Print all courses (sorted)"

PRINT "3. Print course info"

PRINT "9. Exit"

END FUNCTION

FUNCTION main()

tree = NEW Tree

courseMap = NEW Hashtable

choice = 0

WHILE choice != 9

CALL showMenu()

READ user input INTO choice

IF choice == 1

CALL loadCourses("courses.txt", tree, courseMap)

CALL checkPrereqs(courseMap)

ELSE IF choice == 2

CALL tree.print(tree.root)

ELSE IF choice == 3

PRINT "Enter course number:"

READ user input INTO courseNum

CALL printCourseInfo(courseMap, courseNum)

ELSE IF choice == 9

PRINT "Exiting program."

ELSE

PRINT "Invalid choice."

END IF

END WHILE

END FUNCTION

**RUN TIME**

### VECTOR SORT

readFromFile: O(n)

processCourseLine: O(1)

insertCourse: O(1)

validatePrereqs: O(n²)

printCourseInfo: O(n)

printAllCoursesSorted: O(n log n)

Total costt O(n²)

### HASH

* readDataFromFile: O(n)
* processLine: O(1)
* insertItem: O(1)
* checkRequirements: O(n²)
* displayItemInfo: O(n)
* displayAllItemsSorted: O(n log n)

Total cost O(n²)

### TREEDATA

* loadCourses: O(n log n)
* processCourseLine: O(1)
* insertCourse: O(log n)
* insertRecursive: O(log n)
* checkPrereqs: O(n²)
* printCourseInfo: O(1)
* printCourseInformation: O(n)

Total cost O(n²)

### **Vector**

Using a vector is straightforward and easy to work with, especially for basic tasks like loading, displaying, and sorting courses. It works well when the dataset is small because everything is done linearly(adding items, searching, and printing). However, when you start validating prerequisites or looking up specific courses, it becomes inefficient since you have to loop through the entire list each time. Vectors will struggle with scaling.

### **Hash Table**

Hash tables are efficient for fast lookups and quick validation. Instead of looping through a list, they access a course using its ID, which saves time. This makes it ideal when there are many courses or when validations happen frequently. The only downside is that the data isn’t stored in order, so if you want everything sorted, you have to add an extra step to sort it. It is a good choice for performance and accuracy balance.

### **Tree Summary**

Trees offer a nice balance between keeping data organized and allowing faster searches than vectors. They keep items sorted automatically and make insertions and lookups efficient. However, trees are a bit more complicated to code, especially with recursive logic and the risk of unbalanced structures. While they’re great for sorted output, they’re not always the most practical unless sorted data is a priority throughout the program.

**Recommendation**

I would choose a hashtable. The capability to utilize direct searching via ID is a strong point, and is offered with a fast lookup and insertion. An additional step for sorting can be considered a downside, but the hashtable will be efficient and scalable for its primary use. With the scale of the overall project any of these methods would be acceptable, but with scaling being a primary consideration I would choose a hashtable.