James Fleming

10/20/2024

CS-300

6-2 Project One

**1.**

**Milestone 1:**

CLASS Course {

INT courseNumber

STRING courseTitle

VECTOR<STRING> prerequisites

}

FUNCTION loadCourses(FILE filePath, VECTOR<Course> courseList) {

OPEN file using fstream

IF file cannot be opened

OUTPUT "Error: file not found"

WHILE not eof {

READ line from file

// Split the line into tokens (course number, course title, prerequisites)

SPLIT line by commas INTO tokens

// Ensure each line has at least two tokens (course number and course title)

IF SIZE of tokens < 2 THEN

OUTPUT "Error: Invalid file format"

// Create a new course object

INITIALIZE newCourse AS Course

newCourse.courseNumber = tokens[0]

newCourse.courseTitle = tokens[1]

// Process prerequisites if any exist

FOR index FROM 2 TO SIZE of tokens {

// Validate if prerequisite exists as a course

IF searchCourse(courseList, tokens[index]) IS NOT NULL THEN

ADD tokens[index] to newCourse.prerequisites

ELSE

OUTPUT "Error: Prerequisite " + tokens[index] + " does not exist as a course"

RETURN

END IF

}

// Add the new course to the courseList

PUSH\_BACK newCourse INTO courseList

}

CLOSE file

}

FUNCTION searchCourse(VECTOR<Course> courses, STRING courseNumber) {

FOR EACH course IN courses {

IF course.courseNumber == courseNumber THEN

RETURN course

END IF

}

RETURN NULL

}

FUNCTION printCourseInfo(VECTOR<Course> courses, STRING courseNumber) {

Course course = searchCourse(courses, courseNumber)

IF course IS NULL THEN

OUTPUT "Course not found"

// Print course information

ELSE

OUTPUT "Course Number: " + course.courseNumber

OUTPUT "Course Title: " + course.courseTitle

// Print prerequisites

IF SIZE of course.prerequisites > 0 THEN

OUTPUT "Prerequisites: "

FOR EACH prerequisite IN course.prerequisites {

OUTPUT prerequisite

}

ELSE

OUTPUT "No prerequisites"

END IF

}

**Milestone 2:**

CREATE structure called Course

Fields: courseNumber, title, prerequisites (if any)

Hash table for storing Course objects

CREATE HashTable to hold Course objects

Function to read course data from a file

FUNCTION readFile(fileName):

OPEN file with name fileName

IF file is not open:

PRINT "Error: File not found"

RETURN

WHILE not at end of file:

READ line from file

SPLIT line into tokens (courseNumber, title, prerequisites)

IF number of tokens is less than 2:

PRINT "Error: Incorrect file format"

CONTINUE

CREATE Course newCourse

SET newCourse.courseNumber = tokens[0]

SET newCourse.title = tokens[1]

IF there are more than 2 tokens:

FOR each token from tokens[2] to end:

IF token does not exist as a course:

PRINT "Error: Prerequisite not found"

ELSE:

ADD token to newCourse.prerequisites

ADD newCourse to HashTable

CLOSE file

Function to print all courses from the hash table

FUNCTION printCourses(HashTable):

FOR each Course in HashTable:

PRINT "Course Number: ", course.courseNumber

PRINT "Title: ", course.title

PRINT "Prerequisites: ", course.prerequisites

Main function to control the program

FUNCTION main():

CALL readFile with file name

CALL printCourses(HashTable)

**Milestone 3:**

FUNCTION loadCourseData(filename, BinarySearchTree \*bst)

OPEN file as inputFile

IF inputFile is NOT open THEN

PRINT "Error: File cannot be opened."

RETURN

SET courseDictionary as an empty dictionary

WHILE NOT end of file (inputFile)

READ line from inputFile

PARSE line into tokens using ',' as delimiter

IF LENGTH(tokens) < 2 THEN

PRINT "Error: Invalid Course"

CONTINUE

SET courseNumber to tokens[0]

SET courseName to tokens[1]

SET prerequisites to empty list

FOR i from 2 to LENGTH(tokens)

SET prerequisite to tokens[i]

IF prerequisite NOT IN courseDictionary THEN

PRINT "Error: Prerequisite not found"

CONTINUE

ADD prerequisite to prerequisites

CREATE Course object using courseNumber, courseName, and prerequisites

ADD course to courseDictionary

INSERT course into Binary search tree using bst->insertCourse(course)

CLOSE inputFile

STRUCT Course

DECLARE courseNumber as STRING

DECLARE courseName as STRING

DECLARE prerequisites as LIST of STRINGS

FUNCTION Course(courseNumber, courseName, prerequisites)

SET this.courseNumber = courseNumber

SET this.courseName = courseName

SET this.prerequisites = prerequisites

CLASS BinarySearchTree

DECLARE root as Node (initially set as NULL)

FUNCTION insertCourse(course)

IF root is NULL THEN

SET root to new Node(course)

ELSE

CALL insertNode(root, course)

FUNCTION insertNode(currentNode, course)

IF course.courseNumber < currentNode.course.courseNumber THEN

IF currentNode.leftChild is NULL THEN

SET currentNode.leftChild to new Node(course)

ELSE

CALL insertNode(currentNode.leftChild, course)

ELSE

IF currentNode.rightChild is NULL THEN

SET currentNode.rightChild to new Node(course)

ELSE

CALL insertNode(currentNode.rightChild, course)

FUNCTION inOrderTraversal(node)

IF node is NOT NULL THEN

CALL inOrderTraversal(node.leftChild)

PRINT "Course ID: " + node.course.courseNumber

PRINT "Course Name: " + node.course.courseName

PRINT "Prerequisites: " + JOIN(node.course.prerequisites, ", ")

CALL inOrderTraversal(node.rightChild)

FUNCTION printAllCourses(BinarySearchTree bst)

CALL inOrderTraversal(bst.root)

**2.**

FUNCTION Main ()

WHILE userChoice NOT 9

PRINT "Menu Options:"

PRINT "1. Load course data from file"

PRINT "2. Print all courses in alphanumeric order"

PRINT "3. Print course details"

PRINT "9. Exit"

INPUT userChoice

SWITCH userChoice

CASE 1:

CALL loadCoursesFromFile()

CASE 2:

CALL printAllCourses()

CASE 3:

INPUT courseNumber

CALL printCourseDetails(courseNumber)

CASE 9:

PRINT "Exiting the program"

BREAK

DEFAULT:

PRINT "Invalid option. Please try again."

**3.**

**Vector Example**

FUNCTION printAllCourses(Vector<Course> courseList)

CALL sortCourses(courseList)

FOR EACH course IN courseList

PRINT "Course Number: " + course.courseNumber

PRINT "Course Title: " + course.courseTitle

FUNCTION sortCourses(Vector<Course> courseList)

FOR i FROM 0 TO SIZE(courseList) - 1

FOR j FROM 0 TO SIZE(courseList) - i - 1

IF courseList[j].courseNumber > courseList[j+1].courseNumber THEN

SWAP courseList[j] WITH courseList[j+1]

**Hash Table Example:**

FUNCTION printAllCourses(HashTable<Course> courseTable)

INITIALIZE courseList AS Vector<Course>

FOR EACH course IN courseTable

PUSH\_BACK course INTO courseList

CALL sortCourses(courseList)

FOR EACH course IN courseList

PRINT "Course Number: " + course.courseNumber

PRINT "Course Title: " + course.courseTitle

FUNCTION sortCourses(Vector<Course> courseList) //same sort as vector example

FOR i FROM 0 TO SIZE(courseList) - 1

FOR j FROM 0 TO SIZE(courseList) - i - 1

IF courseList[j].courseNumber > courseList[j+1].courseNumber THEN

SWAP courseList[j] WITH courseList[j+1]

**BST Example:**

FUNCTION printAllCourses(BinarySearchTree bst)

CALL inOrderTraversal(bst.root)

FUNCTION inOrderTraversal(Node node)

IF node IS NOT NULL THEN

CALL inOrderTraversal(node.leftChild)

PRINT "Course Number: " + node.course.courseNumber

PRINT "Course Title: " + node.course.courseTitle

CALL inOrderTraversal(node.rightChild)

**4 and 5.**

| **Code**  **(vector)** | **Line Cost** | **# Times Executes** | | **Total Cost** |
| --- | --- | --- | --- | --- |
| **Open file** | 1 | 1 | | 1 |
| **Read each line** | 1 | n | | n |
| **Split line into tokens** | 1 | n | | n |
| **Create course object** | 1 | n | | n |
| **Insert course** | 1 | n | | n |
| **Total Cost** | | | | 4n + 1 |
| **Runtime** | | | | O(n) |
| **Advantages:** simple to implement and add items to  **Disadvantages:** cumbersome if frequent resizing is necessary. Can cost time and memory because vector must be scanned in order | | | |  |
| **Code**  **(hash table)** | **Line Cost** | **# Times Executes** | | **Total Cost** |
| **Open file** | 1 | 1 | | 1 |
| **Read each line** | 1 | n | | n |
| **Split line into tokens** | 1 | n | | n |
| **Create course object** | 1 | n | | n |
| **Insert course** | 1 | n | | n |
| **Total Cost** | | | | 4n + 1 |
| **Runtime** | | | | O(n) |
| **Advantages:** very fast at finding specific course  **Disadvantages:** high collision rate in the worst case could severely degrade speed | | | |  |
| **Code**  **(binary search tree)** | **Line Cost** | **# Times Executes** | **Total Cost** | |
| **Open file** | 1 | 1 | 1 | |
| **Read each line** | 1 | n | n | |
| **Split line into tokens** | 1 | n | n | |
| **Create course object** | 1 | n | n | |
| **Insert course** | log(n) | n | N\*log(n) | |
| **Total Cost** | | | N log(n) + 3n + 1 | |
| **Runtime** | | | O(n log(n)) | |
| **Advantages:** inherently keeps courses in sorted order allowing for in-order traversal to print. searching and inserting are O(n) for balanced trees.  **Disadvantages**: in the worst case insertion and search can be slow. BST is more complex to implement | | |  | |

**6.**

Based on my research this week I think the best data structure to utilize for this application is a hash table. Hash tables offer O(1) lookups which is faster than both the vector and the binary search tree which is important because searching for an individual class is a frequent need. Even though storing the data as a binary search tree would also maintain it's alphanumeric order it is unlikely a common request to print the entire catalog in alphanumeric order. The additional complexity of setting up a binary search tree and ensuring that it remains balanced after insertions and deletions is an unneeded level of complication for this purpose. The hash table is also faster for insertions in comparison to the binary search tree as there's no sorting logic involved