Foster Hare

CS 300

Professor Griffith

12/5/2024

**CS 300 Project One Pseudocode**

Structure Course {

courseId (String)

courseName (String)

preCount (Integer)

preList (List of Strings)

Constructor Course(courseId = "", courseName = "", preCount = 0, preList = [])

}

Main {

Create new list named courseList of structure type Course

Get CSV file path from user input

If no input, use default location ("default.csv")

Call txtParser(filePath) and store the result in courseList

Call validateList(courseList)

If validation fails, output "Invalid course list" and terminate

Get user input for courseId to search, store in userSearch

Call printCourse(userSearch, courseList)

}

txtParser(filePath)

create local list tempList

Open file at filePath

If file cannot be opened

Output "Error: Unable to open file" and terminate

Loop through rows in file until end-of-file

If the row contains at least two values

Create a new Course instance

Set courseId to the first value

Set courseName to the second value

Initialize preList as an empty list

Loop through remaining values in the row

Increment preCount

Add each value to preList

Add the populated Course instance to tempList

Close the file

Return tempList

End

searchList(courseList, courseId)

For each course in courseList

If courseId matches course.courseId

Return course

Return empty Course instance

End

printCourse(courseId, courseList, visitedCourses = Set)

If courseId in visitedCourses

Output "Circular dependency detected for course: " + courseId

Return

Add courseId to visitedCourses

Create tempCourse

Set tempCourse = searchList(courseList, courseId)

If tempCourse.courseId is empty

Output "Course not found: " + courseId

Return

Output "Course ID: " + tempCourse.courseId

Output "Course Name: " + tempCourse.courseName

If tempCourse.preCount > 0

Output "Prerequisites:"

For each prereq in tempCourse.preList

Call printCourse(prereq, courseList, visitedCourses)

Else

Output "No prerequisites"

End

validateList(courseList)

For each course in courseList

For each prereq in course.preList

Create tempCourse

Set tempCourse = searchList(courseList, prereq)

If tempCourse.courseId is empty

Function LoadData(filename):

Try to Open the file (filename) in read mode

If file cannot be opened:

Print "Error: Unable to open file."

Exit the function

Initialize an empty list called `lines`

For each line in the file:

Trim any leading/trailing whitespace

Add the line to `lines`

Close the file

Return `lines`

Function ParseData(lines):

Initialize an empty dictionary called `courseMap`

For each line in `lines`:

Split the line into tokens by commas

If the number of tokens is less than 2:

Print "Error: Line does not contain sufficient parameters."

Skip to the next line

Assign the first token to `courseNumber`

Assign the second token to `courseTitle`

Assign the remaining tokens (if any) to `prerequisites` as a list

For each `prerequisite` in `prerequisites`:

If `prerequisite` is not a key in `courseMap`:

Print "Error: Prerequisite [prerequisite] does not exist."

Skip the current line

Create a Course object with `courseNumber`, `courseTitle`, and `prerequisites`

Add the Course object to `courseMap` using `courseNumber` as the key

Return `courseMap`

Function InsertToTree(courseMap):

Initialize an empty Binary Search Tree (BST) called `courseTree`

For each key in `courseMap`:

Retrieve the Course object from `courseMap`

Insert the Course object into `courseTree`

Return `courseTree`

Function PrintCourses(tree):

If the tree is empty:

Print "No courses available."

Exit the function

Perform an In-Order Traversal of the tree:

For each Course object:

Print "Course Number: [courseNumber]"

Print "Course Title: [courseTitle]"

If `prerequisites` is not empty:

Print "Prerequisites: [comma-separated list of prerequisites]"

Else:

Print "Prerequisites: None"

Define Class Course:

Attributes:

- courseNumber: String

- courseTitle: String

- prerequisites: List of Strings

Constructor:

- Initialize courseNumber, courseTitle, and prerequisites

Procedure LoadCoursesIntoHashTable(fileName):

Create an empty hash table courseTable

Try to open file with name fileName

If file cannot be opened:

Print "Error: Unable to open file."

Return an empty hash table

For each line in the file:

If line is empty:

Skip to the next line

Split the line into tokens using a delimiter (e.g., space or comma)

If the number of tokens is less than 2:

Print "Error: Line is improperly formatted."

Continue to the next line

Set courseNumber to the first token

Set courseTitle to the second token

Set prerequisites to any remaining tokens

For each prerequisite in prerequisites:

If prerequisite is not a key in courseTable:

Print "Error: Prerequisite course not found - " + prerequisite

Return an empty hash table

Create a new Course object using courseNumber, courseTitle, and prerequisites

Add courseNumber as the key and the Course object as the value to courseTable

Close the file

Return courseTable

Procedure PrintAllCourses(courseTable):

Get all keys from courseTable as a list courseKeys

Sort courseKeys alphabetically (optional, for readability)

For each key in courseKeys:

Retrieve the Course object from courseTable using the key

Print "Course Number: " + courseObject.courseNumber

Print "Course Title: " + courseObject.courseTitle

If courseObject.prerequisites is not empty:

Print "Prerequisites: " + Join(courseObject.prerequisites, ", ")

Else:

Print "Prerequisites: None"

Print a blank line for spacing

Procedure Main:

Set fileName to "courses.txt"

Call LoadCoursesIntoHashTable(fileName) and store the result in courseTable

If courseTable is not empty:

Call PrintAllCourses(courseTable)

FUNCTION CreateCourseObjects(inputFile):

courseList ← []

OPEN inputFile FOR reading

WHILE NOT EndOfFile(inputFile):

line ← ReadLine(inputFile)

components ← Split(line, ',')

course ← {

"courseID": components[0],

"courseName": components[1],

"prerequisites": components[2:] // Remaining components as prerequisites

}

Append(courseList, course)

CLOSE inputFile

RETURN courseList

FUNCTION FindAndPrintCourse(courseList, targetCourseID):

FOR EACH course IN courseList:

IF course["courseID"] = targetCourseID THEN:

PRINT "Course ID: " + course["courseID"]

PRINT "Course Name: " + course["courseName"]

PRINT "Prerequisites: " + Join(course["prerequisites"], ", ")

RETURN

PRINT "Course not found: " + targetCourseID

**Vector (Dynamic Array)**

1. **Advantages**:
   * Easy to implement and straightforward to iterate.
   * Suitable for preserving the order of input data.
2. **Disadvantages**:
   * Searching for a specific course has O(n)O(n)O(n) complexity in the worst case.
   * Not efficient for large datasets when frequent searches are required.

**Hash Table**

1. **Advantages**:
   * Efficient for lookups, with an average-case complexity of *O*(1) for insertions and searches.
   * Can store key-value pairs (e.g., courseID as key and Course object as value).
2. **Disadvantages**:
   * Hash collisions may degrade performance to O(n) in the worst case.
   * No inherent order of data; additional steps are needed if order is important.

**Binary Search Tree (BST)**

1. **Advantages**:
   * Data is stored in a sorted manner, which makes range queries efficient.
   * Lookups, insertions, and deletions are *O*(log *n*) in a balanced tree.
2. **Disadvantages**:
   * Requires balancing to maintain *O*(log *n*) complexity; otherwise, performance may degrade to *O*(*n*).
   * More complex to implement compared to a vector or hash table.

A hash table is the best choice for this project because it is very efficient for searching courses by courseID, with an average time complexity of *O*(1). It also handles creating and inserting course objects quickly. A vector is not ideal because searching in a vector takes *O*(*n*), which can be slow for large datasets. While a Binary Search Tree (BST) allows for sorted data and has *O*(log *n*) complexity for searches and insertions, it requires extra work to keep it balanced, which adds complexity. Overall, a hash table is simple to use and provides the best performance for the project’s needs.

|  |  |
| --- | --- |
| **Operation** | **Total Cost** |
| Opening, Reading, Parsing, and Checking Format | *O(n\*m)* |
| Creating Course Objects | *O(n)* |
| Finding and Printing Course Information | O(n) |