## Vector Pseudocode:

FUNCTION loadCourses(filePath: STRING) -> Vector<Course>

OPEN the file at filePath FOR reading

IF file cannot be opened

PRINT "Error: Unable to open file."

RETURN an empty vector

INITIALIZE coursesVector as an empty vector of Course

WHILE there are lines in the file

READ the current line as lineData

SPLIT lineData by commas INTO tokens

IF tokens.size < 2

PRINT "Error: Line has less than 2 parameters."

CONTINUE to the next line

INITIALIZE courseNumber as tokens[0]

INITIALIZE courseTitle as tokens[1]

INITIALIZE prerequisites as an empty vector

FOR each token starting from index 2 to tokens.size

ADD token to prerequisites vector

INITIALIZE course as a new Course object

SET course.number = courseNumber

SET course.title = courseTitle

SET course.prerequisites = prerequisites

ADD course to coursesVector

RETURN coursesVector

FUNCTION validateCourses(coursesVector: Vector<Course>) -> BOOLEAN

INITIALIZE courseNumbers as an empty set

FOR each course IN coursesVector

ADD course.number to courseNumbers

FOR each course IN coursesVector

FOR each prerequisite IN course.prerequisites

IF prerequisite is not in courseNumbers

PRINT "Error: Prerequisite " + prerequisite + " does not exist as a course."

RETURN FALSE

RETURN TRUE

FUNCTION searchCourse(coursesVector: Vector<Course>, courseNumber: STRING)

FOR each course IN coursesVector

IF course.number == courseNumber

PRINT "Course Number: " + course.number

PRINT "Course Title: " + course.title

PRINT "Prerequisites: "

IF course.prerequisites is empty

PRINT "None"

ELSE

FOR each prerequisite IN course.prerequisites

PRINT prerequisite

RETURN

PRINT "Error: Course " + courseNumber + " not found."

FUNCTION main()

INITIALIZE filePath as "path\_to\_course\_file.txt"

INITIALIZE coursesVector as loadCourses(filePath)

IF coursesVector is empty

PRINT "Error: No courses loaded."

RETURN

IF NOT validateCourses(coursesVector)

PRINT "Error: Course data validation failed."

RETURN

PRINT "Course data loaded and validated successfully."

INITIALIZE userChoice as 0

WHILE userChoice != 9

PRINT "Menu:"

PRINT "1. Search for a Course"

PRINT "9. Exit"

PRINT "Enter your choice: "

READ userChoice

SWITCH userChoice

CASE 1:

PRINT "Enter course number to search: "

READ courseNumber

CALL searchCourse(coursesVector, courseNumber)

CASE 9:

PRINT "Goodbye!"

DEFAULT:

PRINT "Invalid choice. Please try again."

## Hash Table Pseudocode:

FUNCTION loadCoursesToHashTable(filePath: STRING) -> HashTable<String, Course>

OPEN the file at filePath FOR reading

IF file cannot be opened

PRINT "Error: Unable to open file."

RETURN an empty hash table

INITIALIZE coursesHashTable as an empty hash table with String keys and Course objects as values

WHILE there are lines in the file

READ the current line as lineData

SPLIT lineData by commas INTO tokens

IF tokens.size < 2

PRINT "Error: Line has less than 2 parameters."

CONTINUE to the next line

INITIALIZE courseNumber as tokens[0]

INITIALIZE courseTitle as tokens[1]

INITIALIZE prerequisites as an empty vector

FOR each token starting from index 2 to tokens.size

ADD token to prerequisites vector

INITIALIZE course as a new Course object

SET course.number = courseNumber

SET course.title = courseTitle

SET course.prerequisites = prerequisites

INSERT (courseNumber, course) INTO coursesHashTable

RETURN coursesHashTable

FUNCTION validateHashTable(coursesHashTable: HashTable<String, Course>) -> BOOLEAN

INITIALIZE courseNumbers as a set containing all keys in coursesHashTable

FOR each course IN coursesHashTable.values

FOR each prerequisite IN course.prerequisites

IF prerequisite is not in courseNumbers

PRINT "Error: Prerequisite " + prerequisite + " does not exist as a course."

RETURN FALSE

RETURN TRUE

FUNCTION searchCourseInHashTable(coursesHashTable: HashTable<String, Course>, courseNumber: STRING)

IF coursesHashTable.CONTAINS(courseNumber)

INITIALIZE course as coursesHashTable[courseNumber]

PRINT "Course Number: " + course.number

PRINT "Course Title: " + course.title

PRINT "Prerequisites: "

IF course.prerequisites is empty

PRINT "None"

ELSE

FOR each prerequisite IN course.prerequisites

PRINT prerequisite

ELSE

PRINT "Error: Course " + courseNumber + " not found."

FUNCTION main()

INITIALIZE filePath as "path\_to\_course\_file.txt"

INITIALIZE coursesHashTable as loadCoursesToHashTable(filePath)

IF coursesHashTable is empty

PRINT "Error: No courses loaded."

RETURN

IF NOT validateHashTable(coursesHashTable)

PRINT "Error: Course data validation failed."

RETURN

PRINT "Course data loaded and validated successfully."

INITIALIZE userChoice as 0

WHILE userChoice != 9

PRINT "Menu:"

PRINT "1. Search for a Course"

PRINT "9. Exit"

PRINT "Enter your choice: "

READ userChoice

SWITCH userChoice

CASE 1:

PRINT "Enter course number to search: "

READ courseNumber

CALL searchCourseInHashTable(coursesHashTable, courseNumber)

CASE 9:

PRINT "Goodbye!"

DEFAULT:

PRINT "Invalid choice. Please try again."

## Binary Search Tree Pseudocode:

CLASS TreeNode

ATTRIBUTE course: Course

ATTRIBUTE left: TreeNode

ATTRIBUTE right: TreeNode

CLASS BinarySearchTree

ATTRIBUTE root: TreeNode

FUNCTION insert(course: Course)

IF root is NULL

SET root to a new TreeNode with course

ELSE

CALL insertRecursive(root, course)

FUNCTION insertRecursive(node: TreeNode, course: Course)

IF course.number < node.course.number

IF node.left is NULL

SET node.left to a new TreeNode with course

ELSE

CALL insertRecursive(node.left, course)

ELSE

IF node.right is NULL

SET node.right to a new TreeNode with course

ELSE

CALL insertRecursive(node.right, course)

FUNCTION search(courseNumber: STRING) -> Course

RETURN searchRecursive(root, courseNumber)

FUNCTION searchRecursive(node: TreeNode, courseNumber: STRING) -> Course

IF node is NULL

RETURN NULL

IF courseNumber == node.course.number

RETURN node.course

ELSE IF courseNumber < node.course.number

RETURN searchRecursive(node.left, courseNumber)

ELSE

RETURN searchRecursive(node.right, courseNumber)

FUNCTION inorderTraversal(node: TreeNode)

IF node is not NULL

CALL inorderTraversal(node.left)

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

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

PRINT "Prerequisites: "

IF node.course.prerequisites is empty

PRINT "None"

ELSE

FOR each prerequisite IN node.course.prerequisites

PRINT prerequisite

CALL inorderTraversal(node.right)

FUNCTION loadCoursesToTree(filePath: STRING) -> BinarySearchTree

OPEN the file at filePath FOR reading

IF file cannot be opened

PRINT "Error: Unable to open file."

RETURN an empty BinarySearchTree

INITIALIZE coursesTree as a new BinarySearchTree

WHILE there are lines in the file

READ the current line as lineData

SPLIT lineData by commas INTO tokens

IF tokens.size < 2

PRINT "Error: Line has less than 2 parameters."

CONTINUE to the next line

INITIALIZE courseNumber as tokens[0]

INITIALIZE courseTitle as tokens[1]

INITIALIZE prerequisites as an empty vector

FOR each token starting from index 2 to tokens.size

ADD token to prerequisites vector

INITIALIZE course as a new Course object

SET course.number = courseNumber

SET course.title = courseTitle

SET course.prerequisites = prerequisites

CALL coursesTree.insert(course)

RETURN coursesTree

FUNCTION validateTree(coursesTree: BinarySearchTree) -> BOOLEAN

INITIALIZE courseNumbers as an empty set

FUNCTION populateCourseNumbers(node: TreeNode)

IF node is not NULL

ADD node.course.number to courseNumbers

CALL populateCourseNumbers(node.left)

CALL populateCourseNumbers(node.right)

CALL populateCourseNumbers(coursesTree.root)

FUNCTION validateNode(node: TreeNode) -> BOOLEAN

IF node is not NULL

FOR each prerequisite IN node.course.prerequisites

IF prerequisite is not in courseNumbers

PRINT "Error: Prerequisite " + prerequisite + " does not exist as a course."

RETURN FALSE

RETURN validateNode(node.left) AND validateNode(node.right)

RETURN TRUE

RETURN validateNode(coursesTree.root)

FUNCTION searchCourseInTree(coursesTree: BinarySearchTree, courseNumber: STRING)

INITIALIZE course as coursesTree.search(courseNumber)

IF course is NULL

PRINT "Error: Course " + courseNumber + " not found."

ELSE

PRINT "Course Number: " + course.number

PRINT "Course Title: " + course.title

PRINT "Prerequisites: "

IF course.prerequisites is empty

PRINT "None"

ELSE

FOR each prerequisite IN course.prerequisites

PRINT prerequisite

FUNCTION main()

INITIALIZE filePath as "path\_to\_course\_file.txt"

INITIALIZE coursesTree as loadCoursesToTree(filePath)

IF coursesTree.root is NULL

PRINT "Error: No courses loaded."

RETURN

IF NOT validateTree(coursesTree)

PRINT "Error: Course data validation failed."

RETURN

PRINT "Course data loaded and validated successfully."

INITIALIZE userChoice as 0

WHILE userChoice != 9

PRINT "Menu:"

PRINT "1. Search for a Course"

PRINT "2. Display All Courses"

PRINT "9. Exit"

PRINT "Enter your choice: "

READ userChoice

SWITCH userChoice

CASE 1:

PRINT "Enter course number to search: "

READ courseNumber

CALL searchCourseInTree(coursesTree, courseNumber)

CASE 2:

CALL coursesTree.inorderTraversal(coursesTree.root)

CASE 9:

PRINT "Goodbye!"

DEFAULT:

PRINT "Invalid choice. Please try again."

## Analysis:

A screenshot of a computer

Description automatically generated

## Advantages and Disadvantages

Vector

Advantages:

* Simple and intuitive structure.
* Supports efficient iteration.
* Easy to implement.

Disadvantages:

* Slow search times (O(n)) for finding a course.
* Insertion at arbitrary positions is inefficient (O(n)).

Hash Table

Advantages:

* Constant time complexity for insertions and lookups (O(1)).
* Efficient for large datasets where frequent searches are required.

Disadvantages:

* Higher memory overhead due to hashing and potential collisions.
* Does not maintain any order among elements.

Binary Search Tree (BST)

Advantages:

* Maintains elements in sorted order.
* Efficient search, insertion, and deletion operations (O(log n) for balanced BST).
* Supports in-order traversal to display all courses.

Disadvantages:

* Requires additional complexity to ensure balance (e.g., AVL or Red-Black Trees).
* Worse-case performance degrades to O(n) if the tree becomes unbalanced.

## Recommendation:

The Hash Table is the best fit for this application because it’s super fast for adding and finding courses, thanks to its constant time for those operations. This makes it perfect for the advisor’s need for efficiency. While a BST keeps things nicely sorted and works efficiently most of the time, it can slow down if it’s unbalanced, and it’s a bit more complicated to set up. Vectors, on the other hand, are straightforward but not great for searching, which is a key part of what we need.

In short, the Hash Table is the ideal choice since it combines speed and simplicity.