Aero Berry

10/12/25

CS – 300

Project One

# Milestone One:

STRUCT Course

STRING courseNumber

STRING courseName

LIST<STRING> prerequisites

END STRUCT

CLASS CourseManager\_Vector

LIST<Course> courseList

METHOD LoadCourses(filename)

OPEN file "filename" FOR reading

IF file cannot be opened THEN

PRINT "Error: Unable to open file."

RETURN

tempCourses ← []

courseNumbers ← []

WHILE NOT end of file DO

line ← READLINE(file)

IF line IS EMPTY THEN CONTINUE

tokens ← SPLIT(line, ",")

IF LENGTH(tokens) < 2 THEN

PRINT "Error: Invalid line format."

CONTINUE

END IF

courseNumber ← TRIM(tokens[0])

courseName ← TRIM(tokens[1])

prereqs ← tokens[2..end]

newCourse ← NEW Course

newCourse.courseNumber ← courseNumber

newCourse.courseName ← courseName

newCourse.prerequisites ← prereqs

APPEND newCourse TO tempCourses

APPEND courseNumber TO courseNumbers

END WHILE

CLOSE file

// Validate prerequisites AFTER full load

FOR EACH c IN tempCourses DO

FOR EACH prereq IN c.prerequisites DO

IF prereq NOT IN courseNumbers THEN

PRINT "Error: prerequisite " + prereq + " not found for " + c.courseNumber

RETURN

END IF

END FOR

END FOR

courseList ← tempCourses

PRINT "Courses successfully loaded."

END METHOD

METHOD PrintAllCourses()

SORT courseList BY courseNumber

FOR EACH c IN courseList DO

PRINT c.courseNumber + ": " + c.courseName

END FOR

END METHOD

METHOD PrintCourseInfo(courseNumber)

FOR EACH c IN courseList DO

IF c.courseNumber = courseNumber THEN

PRINT "Course Number: " + c.courseNumber

PRINT "Course Name: " + c.courseName

IF c.prerequisites IS EMPTY THEN

PRINT "Prerequisites: None"

ELSE

PRINT "Prerequisites:"

FOR EACH p IN c.prerequisites DO

PRINT " - " + p

END FOR

END IF

RETURN

END IF

END FOR

PRINT "Course not found."

END METHOD

END CLASS

MAIN

manager ← NEW CourseManager\_Vector

choice ← 0

WHILE choice ≠ 9 DO

PRINT "1. Load file data"

PRINT "2. Print all courses (sorted)"

PRINT "3. Print specific course information"

PRINT "9. Exit"

choice ← READ INPUT

IF choice = 1 THEN

PRINT "Enter filename:"

filename ← READ INPUT

manager.LoadCourses(filename)

ELSE IF choice = 2 THEN

manager.PrintAllCourses()

ELSE IF choice = 3 THEN

PRINT "Enter course number:"

num ← READ INPUT

manager.PrintCourseInfo(num)

ELSE IF choice = 9 THEN

PRINT "Goodbye!"

ELSE

PRINT "Invalid selection."

END IF

END WHILE

END MAIN

# Milestone Two:

STRUCT Course

STRING courseNumber

STRING courseName

LIST<STRING> prerequisites

END STRUCT

CLASS CourseManager\_Hash

HashTable<STRING, Course> courseTable

METHOD LoadCourses(filename)

OPEN file "filename" FOR reading

IF file cannot be opened THEN

PRINT "Error: Unable to open file."

RETURN

tempCourses ← []

courseNumbers ← []

WHILE NOT end of file DO

line ← READLINE(file)

IF line IS EMPTY THEN CONTINUE

tokens ← SPLIT(line, ",")

IF LENGTH(tokens) < 2 THEN

PRINT "Error: Invalid line format."

CONTINUE

END IF

courseNumber ← TRIM(tokens[0])

courseName ← TRIM(tokens[1])

prereqs ← tokens[2..end]

newCourse ← NEW Course

newCourse.courseNumber ← courseNumber

newCourse.courseName ← courseName

newCourse.prerequisites ← prereqs

APPEND newCourse TO tempCourses

APPEND courseNumber TO courseNumbers

END WHILE

CLOSE file

// Validate prerequisites AFTER full load

FOR EACH c IN tempCourses DO

FOR EACH prereq IN c.prerequisites DO

IF prereq NOT IN courseNumbers THEN

PRINT "Error: prerequisite " + prereq + " not found for " + c.courseNumber

RETURN

END IF

END FOR

END FOR

// Insert into hash table

courseTable ← NEW HashTable<STRING, Course>

FOR EACH c IN tempCourses DO

courseTable.insert(c.courseNumber, c)

END FOR

PRINT "Courses successfully loaded."

END METHOD

METHOD PrintAllCourses()

keys ← courseTable.keys()

SORT keys

FOR EACH k IN keys DO

c ← courseTable.get(k)

PRINT c.courseNumber + ": " + c.courseName

END FOR

END METHOD

METHOD PrintCourseInfo(courseNumber)

c ← courseTable.get(courseNumber)

IF c IS NULL THEN

PRINT "Course not found."

RETURN

END IF

PRINT "Course Number: " + c.courseNumber

PRINT "Course Name: " + c.courseName

IF c.prerequisites IS EMPTY THEN

PRINT "Prerequisites: None"

ELSE

PRINT "Prerequisites:"

FOR EACH p IN c.prerequisites DO

PRINT " - " + p

END FOR

END IF

END METHOD

END CLASS

MAIN

manager ← NEW CourseManager\_Hash

choice ← 0

WHILE choice ≠ 9 DO

PRINT "1. Load file data"

PRINT "2. Print all courses (sorted)"

PRINT "3. Print specific course information"

PRINT "9. Exit"

choice ← READ INPUT

IF choice = 1 THEN

PRINT "Enter filename:"

filename ← READ INPUT

manager.LoadCourses(filename)

ELSE IF choice = 2 THEN

manager.PrintAllCourses()

ELSE IF choice = 3 THEN

PRINT "Enter course number:"

num ← READ INPUT

manager.PrintCourseInfo(num)

ELSE IF choice = 9 THEN

PRINT "Goodbye!"

ELSE

PRINT "Invalid selection."

END IF

END WHILE

END MAIN

# Milestone Three:

STRUCT Course

STRING courseNumber

STRING courseName

LIST<STRING> prerequisites

END STRUCT

STRUCT Node

Course data

Node\* left

Node\* right

END STRUCT

CLASS CourseManager\_BST

Node\* root

METHOD insertCourse(root, newCourse)

IF root IS NULL THEN

node ← NEW Node

node.data ← newCourse

RETURN node

END IF

IF newCourse.courseNumber < root.data.courseNumber THEN

root.left ← insertCourse(root.left, newCourse)

ELSE

root.right ← insertCourse(root.right, newCourse)

END IF

RETURN root

END METHOD

METHOD LoadCourses(filename)

OPEN file "filename" FOR reading

IF file cannot be opened THEN

PRINT "Error: Unable to open file."

RETURN

tempCourses ← []

courseNumbers ← []

WHILE NOT end of file DO

line ← READLINE(file)

IF line IS EMPTY THEN CONTINUE

tokens ← SPLIT(line, ",")

IF LENGTH(tokens) < 2 THEN

PRINT "Error: Invalid line format."

CONTINUE

END IF

courseNumber ← TRIM(tokens[0])

courseName ← TRIM(tokens[1])

prereqs ← tokens[2..end]

newCourse ← NEW Course

newCourse.courseNumber ← courseNumber

newCourse.courseName ← courseName

newCourse.prerequisites ← prereqs

APPEND newCourse TO tempCourses

APPEND courseNumber TO courseNumbers

END WHILE

CLOSE file

// Validate prerequisites AFTER full load

FOR EACH c IN tempCourses DO

FOR EACH prereq IN c.prerequisites DO

IF prereq NOT IN courseNumbers THEN

PRINT "Error: prerequisite " + prereq + " not found for " + c.courseNumber

RETURN

END IF

END FOR

END FOR

// Build BST

root ← NULL

FOR EACH c IN tempCourses DO

root ← insertCourse(root, c)

END FOR

PRINT "Courses successfully loaded."

END METHOD

METHOD searchCourse(root, courseNumber)

IF root IS NULL THEN RETURN NULL

IF courseNumber = root.data.courseNumber THEN RETURN root.data

IF courseNumber < root.data.courseNumber THEN

RETURN searchCourse(root.left, courseNumber)

ELSE

RETURN searchCourse(root.right, courseNumber)

END IF

END METHOD

METHOD inOrderTraversal(root)

IF root IS NULL THEN RETURN

inOrderTraversal(root.left)

PRINT root.data.courseNumber + ": " + root.data.courseName

inOrderTraversal(root.right)

END METHOD

METHOD PrintAllCourses()

inOrderTraversal(root)

END METHOD

METHOD PrintCourseInfo(courseNumber)

c ← searchCourse(root, courseNumber)

IF c IS NULL THEN

PRINT "Course not found."

RETURN

END IF

PRINT "Course Number: " + c.courseNumber

PRINT "Course Name: " + c.courseName

IF c.prerequisites IS EMPTY THEN

PRINT "Prerequisites: None"

ELSE

PRINT "Prerequisites:"

FOR EACH p IN c.prerequisites DO

PRINT " - " + p

END FOR

END IF

END METHOD

END CLASS

MAIN

manager ← NEW CourseManager\_BST

choice ← 0

WHILE choice ≠ 9 DO

PRINT "1. Load file data"

PRINT "2. Print all courses (sorted)"

PRINT "3. Print specific course information"

PRINT "9. Exit"

choice ← READ INPUT

IF choice = 1 THEN

PRINT "Enter filename:"

filename ← READ INPUT

manager.LoadCourses(filename)

ELSE IF choice = 2 THEN

manager.PrintAllCourses()

ELSE IF choice = 3 THEN

PRINT "Enter course number:"

num ← READ INPUT

manager.PrintCourseInfo(num)

ELSE IF choice = 9 THEN

PRINT "Goodbye!"

ELSE

PRINT "Invalid selection."

END IF

END WHILE

END MAIN

# Menu Pseudocode Breakdown:

FUNCTION Menu()

WHILE TRUE DO

PRINT "1. Load file data into data structure"

PRINT "2. Print alphanumeric list of courses"

PRINT "3. Print course info and prerequisites"

PRINT "9. Exit"

choice ← READINT()

IF choice = 1 THEN

PRINT "Enter filename:"

filename ← READLINE()

// Call appropriate LoadCourses\_\* depending on current data structure selection.

// Example for Hash:

result ← LoadCourses\_Hash(filename)

IF result = success THEN PRINT "Load successful."

ELSE IF choice = 2 THEN

// Call PrintAllCourses\_\* for active DS

ELSE IF choice = 3 THEN

PRINT "Enter course number to search:"

searchNumber ← READLINE()

// Call PrintCourse\_\* for active DS

ELSE IF choice = 9 THEN

PRINT "Exiting."

BREAK

ELSE

PRINT "Invalid option"

END IF

END WHILE

END FUNCTION

# Runtime Analysis:A screenshot of a computer AI-generated content may be incorrect.

**Assumptions:** The input has n courses. Cost of parsing one line = O(1) per line. Where appropriate we call out average vs worst-case.

* All structures store n Course objects → O(n) storage for actual objects.
* Vector: O(n) + small resizing overhead.
* Hash table: O(n) plus extra bucket overhead, so slightly larger constant factor.
* BST: O(n) nodes + pointers, slightly larger per-node overhead than vector.

# Advantages / Disadvantages:

**Vector**

Advantages:

* Simple to implement.
* Fast to append.
* Compact memory layout for objects.

Disadvantages:

* Searching for a specific course like Option 3 is O(n) linear scan.
* Must sort to produce alphanumeric list like Option 2 → O(n log n) each time unless you maintain sorted order at insert cost.
* Not optimal for frequent lookups.

**Hash Table**

Advantages:

* Very fast average-time lookup for Option 3: O(1) average.
* Insert is average O(1), building the table is O(n) expected.
* Simple to get a course by number.

Disadvantages:

* Hash tables do not preserve order, so to print sorted list like Option 2 you have to extract keys and sort: O(n log n).
* Worst-case insertion/lookup can degrade if hash poorly chosen but should be rare with good hash/resizing.
* Uses extra memory for buckets and overhead.

**Binary Search Tree**

Advantages:

* In-order traversal prints courses in sorted alphanumeric order directly.
* Searching for a course is O(log n) average.
* Keeps guaranteed O(log n) operations, and build is O(n log n) worst-case, in-order traversal O(n).
* Good choice when frequent ordered operations are required.

Disadvantages:

* If supplied keys are in sorted order, a naive BST becomes degenerate→ worst-case O(n) per operation.
* Insert/search worst-case O(n) → building can be O(n²) worst-case unless tree is balanced.
* Slightly more pointer overhead per node.
* More complex to implement than simple BST or hash table.

# Recommendation:

I would recommend a Hash Table with a supplemental sorted-key list when needed:

1. Frequent advisor queries will likely be lookups for course info like Option 3. Hash Table provides O(1) average-time lookup, which gives the snappiest experience for advisors.
2. Loading time is O(n) expected, and Memory cost is modestly higher than vector but I feel acceptable.
3. To satisfy Option 2, simply extract the keys from the hash table and sort them once on request, I think this is acceptable because printing the entire course list is likely less frequent than single course lookups. Alternatively, if printing sorted lists is frequent, you could maintain a separate sorted array of keys that you update when loading. This keeps Option 2 at O(n) retrieval of already-sorted list.

If there is expected to be many sorted operations, then of course the use of more advanced structures, like a self-balancing BST is a good secondary choice and it guarantees O(log n) operations and O(n) in-order listing, but the implementation is more complex.