CS 300: Project One

Kenya Craw

Instructor Elia Shahbazi

December 7, 2025

Milestone One – Vector Data Structure

STRUCT Course

courseNumber : STRING
title : STRING
prerequisites : VECTOR of STRING

Purpose: Each Course object holds the course number, title, and zero or more prerequisites.

FUNCTION loadCoursesVector(fileName: STRING) RETURNS VECTOR of Course

```
OPEN file fileName FOR READING
DECLARE courses AS VECTOR of Course
DECLARE allCourseNumbers AS SET of STRING

// First pass: load courses
FOR each line IN file
   SPLIT line BY comma INTO tokens
   IF LENGTH(tokens) < 2 THEN
      PRINT "Error: Invalid line format"
      CONTINUE
   ENDIF

   CREATE newCourse AS Course
   newCourse.courseNumber = tokens[0]
   newCourse.title = tokens[1]

   IF LENGTH(tokens) > 2 THEN
      FOR i = 2 TO LENGTH(tokens) - 1
         ADD tokens[i] TO newCourse.prerequisites
      END FOR
   ENDIF

   ADD newCourse.courseNumber TO allCourseNumbers
   APPEND newCourse TO courses
END FOR
```

```
// Second pass: validate prerequisites
FOR EACH course IN courses
   FOR EACH prereq IN course.prerequisites
      IF prereq NOT IN allCourseNumbers THEN
         PRINT "Error: Prerequisite " + prereq + " does not exist for course " +
course.courseNumber
      ENDIF
   END FOR
END FOR

CLOSE file
RETURN courses
```

FUNCTION searchCourseVector(courses: VECTOR of Course, courseNumber: STRING)

```
FOR EACH course IN courses
   IF course.courseNumber == courseNumber THEN
      PRINT "Course Number: " + course.courseNumber
      PRINT "Title: " + course.title
      IF LENGTH(course.prerequisites) > 0 THEN
         PRINT "Prerequisites:"
         FOR EACH prereq IN course.prerequisites
            FOR EACH c IN courses
               IF c.courseNumber == prereq THEN
                  PRINT "  " + c.courseNumber + ": " + c.title
               ENDIF
            END FOR
         END FOR
      ELSE
         PRINT "Prerequisites: None"
      ENDIF
      RETURN
   ENDIF
END FOR
PRINT "Course " + courseNumber + " not found."
```

Milestone Two – Hash Table Data Structure

STRUCT Course

courseNumber: STRING
title: STRING
prerequisites: LIST of STRING

CLASS HashNode

course: Course
next: HashNode pointer

CLASS HashTable

buckets: ARRAY of HashNode pointers
tableSize: INTEGER

FUNCTION hash(key: STRING) RETURNS INTEGER

Compute numeric hash from key
RETURN hashValue MOD tableSize

FUNCTION insertCourse(hashTable: HashTable, course: Course)

key = hash(course.courseNumber)
node = hashTable.buckets[key]

IF node IS NULL THEN
  CREATE new HashNode(course)
  hashTable.buckets[key] = newNode
ELSE
  CREATE newNode(course)
  newNode.next = node.next
  node.next = newNode

```
ENDIF
```

FUNCTION loadCoursesHash(fileName: STRING, hashTable: HashTable)

```
// First pass: collect all course numbers
DECLARE allCourseNumbers AS SET of STRING
OPEN file FOR READING
FOR each line IN file
   tokens = SPLIT line BY comma
   IF LENGTH(tokens) >= 2 THEN
      ADD tokens[0] TO allCourseNumbers
   ENDIF
END FOR
CLOSE file

// Second pass: insert courses into hash table with validation
OPEN file FOR READING
FOR each line IN file
   tokens = SPLIT line BY comma
   IF LENGTH(tokens) < 2 THEN
      PRINT "Invalid course entry"
      CONTINUE
   ENDIF

   courseNumber = tokens[0]
   courseTitle = tokens[1]
   prerequisites = tokens[2 to end]

   // Validate prerequisites exist
   FOR EACH prereq IN prerequisites
      IF prereq NOT IN allCourseNumbers THEN
         PRINT "Error: Prerequisite " + prereq + " not found"
      ENDIF
   END FOR

   course = new Course(courseNumber, courseTitle, prerequisites)
   insertCourse(hashTable, course)
END FOR
```

CLOSE file

FUNCTION searchCourseHash(hashTable: HashTable, courseNumber: STRING)

```
key = hash(courseNumber)
node = hashTable.buckets[key]

WHILE node != NULL
   IF node.course.courseNumber == courseNumber THEN
      PRINT course info
      FOR EACH prereq IN node.course.prerequisites
         PRINT prereq info
      END FOR
      RETURN
   ENDIF
   node = node.next
END WHILE
PRINT "Course not found"
```

Milestone Three – Binary Search Tree

STRUCT Bid

bidId: STRING
title: STRING
fund: STRING
amount: DOUBLE

STRUCT BSTNodeBid

bid: Bid
left: BSTNodeBid pointer
right: BSTNodeBid pointer

CLASS BinarySearchTreeBids

root: BSTNodeBid pointer

FUNCTION InsertBid(bst, bid)

```
IF bst.root IS NULL THEN
   bst.root = new BSTNodeBid(bid)
ELSE
   addBidNode(bst.root, bid)
ENDIF
```

FUNCTION addBidNode(node, bid)

```
IF bid.bidId < node.bid.bidId THEN
   IF node.left IS NULL THEN
      node.left = new BSTNodeBid(bid)
   ELSE
      addBidNode(node.left, bid)
   ENDIF
ELSE
   IF node.right IS NULL THEN
      node.right = new BSTNodeBid(bid)
   ELSE
      addBidNode(node.right, bid)
   ENDIF
ENDIF
```

STRUCT CourseBSTNode

course: Course
left: CourseBSTNode pointer
right: CourseBSTNode pointer

CLASS BinarySearchTreeCourses

root: CourseBSTNode pointer

FUNCTION InsertCourseBST(node, course)

```
IF node IS NULL THEN
   RETURN new CourseBSTNode(course)
ENDIF

IF course.courseNumber < node.course.courseNumber THEN
   node.left = InsertCourseBST(node.left, course)
ELSE
   node.right = InsertCourseBST(node.right, course)
ENDIF

RETURN node
```

FUNCTION searchCourseBST(node, courseNumber)

```
IF node IS NULL THEN
   RETURN "Course not found"
ENDIF
IF node.course.courseNumber == courseNumber THEN
   PRINT course info
   FOR EACH prereq IN node.course.prerequisites
      searchCourseBST(root, prereq)
   END FOR
ELSE IF courseNumber < node.course.courseNumber THEN
   searchCourseBST(node.left, courseNumber)
ELSE
   searchCourseBST(node.right, courseNumber)
ENDIF
```

FUNCTION printAllCoursesBST(node)

```
IF node != NULL THEN
   printAllCoursesBST(node.left)
   PRINT node.course.courseNumber, node.course.title
   IF LENGTH(node.course.prerequisites) > 0 THEN
```

```
        PRINT "Prerequisites: " + join(node.course.prerequisites, ", ")
    ELSE
        PRINT "Prerequisites: None"
    ENDIF
    printAllCoursesBST(node.right)
ENDIF
```

Main Program Flow

```
START

// Vector demonstration
coursesVector = loadCoursesVector("CourseInformation.txt")
PRINT "Vector courses loaded"
INPUT searchVectorCourse
CALL searchCourseVector(coursesVector, searchVectorCourse)

// Hash table demonstration
hashTable = new HashTable(size=10)
loadCoursesHash("CourseInformation.txt", hashTable)
PRINT "Hash table courses loaded"
INPUT searchHashCourse
CALL searchCourseHash(hashTable, searchHashCourse)

// BST demonstration
courseTree = new BinarySearchTreeCourses()
FOR EACH course IN coursesVector
    courseTree.root = InsertCourseBST(courseTree.root, course)
END FOR
PRINT "BST courses loaded"
INPUT searchBSTCourse
CALL searchCourseBST(courseTree.root, searchBSTCourse)
CALL printAllCoursesBST(courseTree.root)

END
```