CS 300 – Project One Milestone 2: Hash Table Pseudocode

Overview  
This document provides pseudocode for reading ABCU course data from a file, validating the data, creating Course objects, storing them in a hash table, and printing course information with prerequisites. The file format is CSV with the schema: courseNumber, name, prerequisite1, prerequisite2, ...  
  
Assumptions  
• Some courses have zero prerequisites.  
• Prerequisites are listed by courseNumber only.  
• Hash table uses separate chaining (buckets of lists) and a simple string hash.

## Data Structures

STRUCT Course  
 STRING number // e.g., "CSCI200"  
 STRING title // e.g., "Data Structures"  
 LIST<STRING> prereqs // e.g., ["CSCI101"]  
END STRUCT  
  
STRUCT HashNode  
 STRING key // courseNumber  
 Course value  
 HashNode next  
END STRUCT  
  
STRUCT HashTable  
 LIST<HashNode> buckets[CAPACITY] // array of bucket heads  
END STRUCT  
  
FUNCTION hash(key: STRING) -> INTEGER  
 sum <- 0  
 FOR each character c in key  
 sum <- sum + ASCII(c)  
 END FOR  
 RETURN sum MOD CAPACITY  
END FUNCTION  
  
FUNCTION ht\_insert(table: HashTable, key: STRING, value: Course)  
 idx <- hash(key)  
 node <- table.buckets[idx]  
 WHILE node != NULL  
 IF node.key == key THEN  
 node.value <- value // replace on duplicate key  
 RETURN  
 END IF  
 node <- node.next  
 END WHILE  
 newNode <- NEW HashNode(key, value, table.buckets[idx])  
 table.buckets[idx] <- newNode  
END FUNCTION  
  
FUNCTION ht\_find(table: HashTable, key: STRING) -> Course or NULL  
 idx <- hash(key)  
 node <- table.buckets[idx]  
 WHILE node != NULL  
 IF node.key == key THEN  
 RETURN node.value  
 END IF  
 node <- node.next  
 END WHILE  
 RETURN NULL  
END FUNCTION  
  
FUNCTION ht\_keys(table: HashTable) -> LIST<STRING>  
 keys <- EMPTY LIST  
 FOR i <- 0 TO CAPACITY - 1  
 node <- table.buckets[i]  
 WHILE node != NULL  
 APPEND node.key TO keys  
 node <- node.next  
 END WHILE  
 END FOR  
 RETURN keys  
END FUNCTION

## File Loading and Validation

// Returns (HashTable courses, LIST<STRING> errors)  
FUNCTION loadCourses(filePath: STRING) -> (HashTable, LIST<STRING>)  
 errors <- EMPTY LIST  
 courses <- NEW HashTable with CAPACITY = reasonable prime (e.g., 101)  
  
 FILE f <- OPEN filePath FOR READING  
 IF f == NULL THEN  
 APPEND "ERROR: Unable to open file." TO errors  
 RETURN (courses, errors)  
 END IF  
  
 lineNumber <- 0  
 WHILE NOT EOF(f)  
 raw <- READLINE(f)  
 lineNumber <- lineNumber + 1  
  
 // Trim whitespace and skip empty/comment lines  
 line <- TRIM(raw)  
 IF line == "" OR STARTS\_WITH(line, "#") THEN  
 CONTINUE  
 END IF  
  
 // Split by comma  
 tokens <- SPLIT(line, ",")  
  
 // Validate: at least courseNumber and title  
 IF LENGTH(tokens) < 2 THEN  
 APPEND "ERROR (line " + TO\_STRING(lineNumber) + "): fewer than 2 fields." TO errors  
 CONTINUE  
 END IF  
  
 courseNum <- TRIM(tokens[0])  
 courseTitle <- TRIM(tokens[1])  
  
 // Validate basic non-empty fields  
 IF courseNum == "" THEN  
 APPEND "ERROR (line " + TO\_STRING(lineNumber) + "): missing course number." TO errors  
 CONTINUE  
 END IF  
 IF courseTitle == "" THEN  
 APPEND "ERROR (line " + TO\_STRING(lineNumber) + "): missing course title." TO errors  
 CONTINUE  
 END IF  
  
 // Parse prerequisites (if any)  
 prereqList <- EMPTY LIST  
 FOR i <- 2 TO LENGTH(tokens) - 1  
 prereq <- TRIM(tokens[i])  
 IF prereq != "" THEN  
 APPEND prereq TO prereqList  
 END IF  
 END FOR  
  
 // Create Course object  
 c <- NEW Course  
 c.number <- courseNum  
 c.title <- courseTitle  
 c.prereqs <- prereqList  
  
 // Insert course (replace duplicates)  
 ht\_insert(courses, c.number, c)  
 END WHILE  
 CLOSE(f)  
  
 // PASS 2: prerequisite existence validation  
 keys <- ht\_keys(courses)  
 keySet <- SET(keys) // for O(1) contains checks  
  
 FOR each k IN keys  
 c <- ht\_find(courses, k)  
 FOR each p IN c.prereqs  
 IF NOT CONTAINS(keySet, p) THEN  
 APPEND "ERROR: prerequisite '" + p + "' for course '" + c.number + "' not found in file." TO errors  
 END IF  
 END FOR  
 END FOR  
  
 RETURN (courses, errors)  
END FUNCTION

## Printing Course Information (Hash Table)

// Prints a single course and its prerequisites (numbers and titles when available)  
FUNCTION printCourseInfo(courses: HashTable, courseNumber: STRING)  
 course <- ht\_find(courses, courseNumber)  
 IF course == NULL THEN  
 PRINT "Course " + courseNumber + " not found."  
 RETURN  
 END IF  
  
 PRINT course.number + ", " + course.title  
  
 IF LENGTH(course.prereqs) == 0 THEN  
 PRINT "Prerequisites: None"  
 ELSE  
 PRINT "Prerequisites:"  
 FOR each p IN course.prereqs  
 pc <- ht\_find(courses, p)  
 IF pc != NULL THEN  
 PRINT " - " + pc.number + ": " + pc.title  
 ELSE  
 // Should not happen if validation passed, but handle gracefully  
 PRINT " - " + p + " (missing details)"  
 END IF  
 END FOR  
 END IF  
END FUNCTION

## Search Function Signature (Hash Table)

FUNCTION searchCourse(courses: HashTable, courseNumber: STRING)  
 printCourseInfo(courses, courseNumber)  
END FUNCTION

## Main Program Flow

FUNCTION main()  
 (courses, errors) <- loadCourses("courses.csv")  
  
 IF LENGTH(errors) > 0 THEN  
 PRINT "File validation failed with the following issues:"  
 FOR each e IN errors  
 PRINT " - " + e  
 END FOR  
 EXIT PROGRAM WITH ERROR CODE  
 END IF  
  
 // Example interactive loop (optional)  
 WHILE TRUE  
 PRINT "Enter a course number to view (or 'q' to quit): "  
 input <- READ\_INPUT()  
 input <- TRIM(input)  
 IF input == "q" OR input == "Q" THEN  
 BREAK  
 END IF  
 searchCourse(courses, input)  
 END WHILE  
END FUNCTION

## Notes on Complexity (High Level)

• ht\_insert and ht\_find are expected O(1) average-case with a good hash and load factor; worst-case O(n) with heavy collisions.  
• loadCourses runs in O(n + m) where n is number of courses and m total prerequisites, assuming O(1) average hash operations. The second pass validates all prerequisites in O(n + m).  
• printCourseInfo executes in O(1 + d) where d is the number of prerequisites for the course.