**CS 300 Project One**

**Pseudocode and Runtime Analysis**

**Johanna Rodriguez**

### **1. Vector Data Structure**

#### **File Input and Validation Pseudocode**

FUNCTION LoadCoursesFromFile(filePath)

OPEN file at filePath

IF file cannot be opened

DISPLAY "Error opening file"

RETURN

WHILE not end of file

READ line

SPLIT line by comma into tokens

IF tokens < 2

DISPLAY "Error: Missing course number or title"

CONTINUE

ADD courseNumber to list of known courses

ADD full line to courseLines list

END WHILE

FOR EACH line IN courseLines

SPLIT line into tokens

FOR EACH token from 3rd to end

IF token NOT IN course list

DISPLAY "Invalid prerequisite: " + token

CLOSE file

#### **Course Object and Vector Storage**

DEFINE STRUCTURE Course

STRING courseNumber

STRING title

LIST of STRING prerequisites

INITIALIZE empty vector courseVector

FOR EACH line IN courseLines

SPLIT line into tokens

CREATE Course object

SET fields from tokens

ADD Course to courseVector

#### **Search and Print Pseudocode**

FUNCTION PrintCourse(courseNumber)

FOR EACH course IN courseVector

IF course.courseNumber == courseNumber

DISPLAY course info

IF prerequisites NOT EMPTY

DISPLAY prerequisites

ELSE

DISPLAY "No prerequisites"

RETURN

DISPLAY "Course not found"

#### **Print Sorted Courses**

FUNCTION PrintAllCourses()

SORT courseVector by courseNumber

FOR EACH course IN courseVector

DISPLAY course.courseNumber + ": " + course.title

### **2. Hash Table Data Structure**

#### **Load and Validate File**

* Uses a hash table with computed index
* Checks prerequisites exist in course list

#### **Course Storage**

* Each course is hashed and inserted into the corresponding bucket

#### **Search and Print**

* Course is found by computing its hash index and searching the list

#### **Print All Courses**

FUNCTION PrintAllCourses()

FOR EACH bucket IN hashTable

FOR EACH course IN bucket

DISPLAY course.courseNumber + ": " + course.title

### **3. Binary Search Tree (BST) Data Structure**

#### **Load and Validate File**

* Same logic as above but inserts into BST using courseNumber as key

#### **Course Object and Storage**

* Insert function places Course in left/right based on alphanumeric order

#### **Search and Print**

* Tree traversal used to find course and display info

#### **Print All Courses**

FUNCTION InOrderTraversal(node)

IF node NOT NULL

CALL InOrderTraversal(node.left)

DISPLAY course info

CALL InOrderTraversal(node.right)

### **4. Menu Pseudocode (Shared Across All)**

DISPLAY "Menu"

DISPLAY "1. Load Courses"

DISPLAY "2. Print All Courses"

DISPLAY "3. Print Specific Course"

DISPLAY "9. Exit"

GET user input

IF input == 1 THEN CALL LoadCourses

IF input == 2 THEN CALL PrintAllCourses

IF input == 3 THEN PROMPT for course ID THEN CALL PrintCourse

IF input == 9 THEN EXIT program

### **5. Runtime Analysis Chart**

| **Line** | **Cost** | **Times Executed** | **Total Cost** |
| --- | --- | --- | --- |
| Read line from file | 1 | n | n |
| Split line into tokens | 1 | n | n |
| Create Course object | 1 | n | n |
| Insert into data structure | 1 | n | n |
| Validate prerequisites | n | n | n^2 |
| **Total Runtime** |  |  | **O(n^2)** |

### **6. Evaluation: Pros and Cons**

**Vector:**

* Easy to implement
* Good for small data
* Slow searches (O(n))
* Must re-sort to maintain order

**Hash Table:**

* Fast access and search (O(1))
* Efficient for large data
* Order not preserved
* Requires good hash function

**Binary Search Tree:**

* Keeps data sorted
* Fast search (O(log n)) if balanced
* Slower insertions than hash table
* Complex to implement/delete

### **7. Final Recommendation**

For this project, I recommend using the **Binary Search Tree**. It offers a good balance between fast search and maintaining sorted order, which is ideal for advisors who want to browse and query course information. It also scales well and supports prerequisite validation through traversal.