**CS 300 Project One: Pseudocode and Runtime Analysis**

**Student: Jay Morocho**

### **Vector Data Structure Pseudocode**

**1. Load File and Parse Data**

Begin

* Open the file named "course\_data.txt" for reading
* Create an empty list called "courseList" to store all course objects
* While not at the end of the file:
  + Read a line from the file
  + Split the line into tokens using a comma as the delimiter
  + If the number of tokens is less than 2:
    - Display an error message and exit
  + Set courseNumber to the first token
  + Set courseTitle to the second token
  + If more tokens exist:
    - Set prerequisites to the remaining tokens
  + Else:
    - Set prerequisites to an empty list
  + Create a new Course object with courseNumber, courseTitle, and prerequisites
  + Add Course to courseList
* End While
* Close the file

**2. Print Course Information**

Function PrintCourse(courseNumber):

* For each course in courseList:
  + If course.courseNumber equals courseNumber:
    - Display courseNumber, courseTitle, and prerequisites
    - Return
* Display "Course not found."

**3. Menu**

Function DisplayMenu():

* Display:
  + 1. Load Data
    2. Print Course List
    3. Print Course Info
    4. Exit
* Get user input
* If input is 1:
  + Call LoadFile()
* Else if input is 2:
  + Call PrintSortedCourses()
* Else if input is 3:
  + Get courseNumber from user
  + Call PrintCourse(courseNumber)
* Else if input is 9:
  + Exit program

**4. Print Sorted Course List**

Function PrintSortedCourses():

* Sort courseList by course.courseNumber
* For each course in the sorted courseList:
  + Display courseNumber and courseTitle

### **Hash Table Data Structure Pseudocode**

**1. Load File and Parse Data**

Function LoadFile():

* Open the file for reading
* Create an empty hash table called coursesTable
* While not at the end of the file:
  + Read line and split into tokens
  + If tokens are less than 2:
    - Display error and skip line
  + Set courseNumber to tokens[0]
  + Set courseTitle to tokens[1]
  + Set prerequisites to tokens[2:]
  + Create Course object
  + Validate each prerequisite exists in the table
  + Add Course to hash table using courseNumber as key
* Close the file

**2. Print Course Information**

Function PrintCourse(courseNumber):

* If courseNumber exists in coursesTable:
  + Retrieve course
  + Display courseNumber, title, and prerequisites
* Else:
  + Display "Course not found."

**3. Menu**

Function DisplayMenu():

* Display options 1, 2, 3, 9
* Get user input
* If input is 1:
  + Call LoadFile()
* Else if input is 2:
  + Call PrintSortedCourses()
* Else if input is 3:
  + Get courseNumber
  + Call PrintCourse(courseNumber)
* Else if input is 9:
  + Exit

**4. Print Sorted Course List**

Function PrintSortedCourses():

* Get all keys from coursesTable
* Sort keys alphanumerically
* For each key in the sorted list:
  + Display courseNumber and course title

### **Binary Search Tree (BST) Pseudocode**

**1. Load File and Parse Data**

Function LoadFile():

* Open file for reading
* Create empty BST called courseTree
* While not at the end of file:
  + Read line and split into tokens
  + If tokens are less than 2:
    - Display error and skip line
  + Set courseNumber to tokens[0]
  + Set courseTitle to tokens[1]
  + Set prerequisites to tokens[2:]
  + Create Course object
  + Insert into BST using courseNumber as key
* Close the file

**2. Print Course Information**

Function PrintCourse(courseNumber):

* Search BST for courseNumber
* If found:
  + Display courseNumber, title, and prerequisites
* Else:
  + Display "Course not found."

**3. Menu**

Function DisplayMenu():

* Display options
* If input is 1:
  + Call LoadFile()
* Else if input is 2:
  + Call PrintSortedCourses()
* Else if input is 3:
  + Get courseNumber and call PrintCourse(courseNumber)
* Else if input is 9:
  + Exit

**4. Print Sorted Course List**

Function PrintSortedCourses():

* Perform in-order traversal on BST
* For each visited node:
  + Display courseNumber and courseTitle

### **Runtime Analysis Chart**

| **Operation** | **Cost per Line** | **Times Executed** | **Big O Total** |
| --- | --- | --- | --- |
| File reading and parsing | 1 | n | O(n) |
| Creating Course object | 1 | n | O(n) |
| Inserting into vector/hash/tree | 1 | n | O(n) |
| Searching in vector (linear) | 1 | n | O(n) |
| Searching in hash table | 1 | 1 | O(1) |
| Searching in BST (balanced) | 1 | log n | O(log n) |

### **Advantages and Disadvantages**

Vector:

* Easy to implement and understand.
* Not ideal for large datasets due to linear search.
* Requires manual sorting to maintain order.

Hash Table:

* Fast lookup time for most operations.
* Does not maintain order naturally.
* Requires extra logic for sorting the course list.

Binary Search Tree:

* Maintains order automatically using in-order traversal.
* Offers efficient search if balanced.
* More complex to implement compared to vector or hash tables.

### **Recommendation**

Based on the requirements and the evaluation of each data structure, I recommend using the Binary Search Tree (BST) for the final implementation. The BST is ideal because it allows courses to be displayed in alphanumeric order through in-order traversal and provides efficient search capabilities. While hash tables offer faster lookup, they do not maintain order. Vectors, on the other hand, require sorting and are inefficient for large searches. Therefore, the BST is the most appropriate structure for this application.