**ABCU Computer Science Course Data Pseudocode and Runtime Analysis**

**Course Object Definition**

**Vector Implementation**

LoadCourses(fileName)

1. Open file
2. If file cannot be opened, display error and exit
3. Create empty set courseIDs and empty vector courseList
4. For each line in file:
    - Split line by commas
    - If tokens < 2, display format error and exit
    - Store first token as courseID
    - Store second token as courseTitle
    - Store remaining tokens as prerequisites
    - Add courseID to courseIDs
    - Create Course object and append to courseList
5. For each course in courseList:
    - For each prerequisite:
        - If prerequisite not in courseIDs, display error and exit
6. Close file

FindCourse(courseID)

- Loop through courseList
- If courseID matches, return course
- Return null if not found

PrintCourse(courseID)

- Call FindCourse
- If null, display message
- Otherwise, print courseNumber, courseTitle, prerequisites

PrintAllCourses()

- Sort courseList by courseNumber
- Print each courseNumber and courseTitle

Menu Options

1. Load file
2. Print all courses (sorted)
3. Print single course information
4. Exit

**Hash Table Implementation**

LoadCourses(fileName, courseTable)

1. Open file
2. If file cannot be opened, display error and exit
3. Create empty list allCourseNumbers
4. First pass: store each courseNumber in allCourseNumbers
5. Reset file pointer, second pass:
    - Split line into tokens
    - If < 2 tokens, display format error and continue
    - Create Course object
    - For each remaining token:
        - If token exists in allCourseNumbers, add to prerequisites
        - Else, display error
    - Insert course object into courseTable with key = courseNumber
6. Close file

PrintAllCourses(courseTable)

- For each course in courseTable, print courseNumber and courseTitle

PrintCourse(courseNumber, courseTable)

- If courseNumber not in table, display error
- Otherwise, print courseNumber, courseTitle, prerequisites

Menu Options

1. Load file
2. Print all courses
3. Print single course information
4. Exit

**Binary Search Tree Implementation**

InsertCourse(tree, course)

- If tree.root null, assign new node
- Else, recursively insert based on courseNumber

LoadCoursesFromFile(fileName, tree)

1. Open file
2. If file cannot be opened, display error and exit
3. For each line:
    - Split line into tokens
    - Validate format
    - Create Course object
    - Add courseNumber to courseNumbers list
    - Insert course into tree
4. Close file
5. Validate prerequisites against courseNumbers list

PrintCourseList(tree)

- Perform in-order traversal to print courses in sorted order

PrintCourseInformation(tree, courseNumber)

- Search for course in tree
- Print courseNumber, courseTitle, prerequisites

Menu Options

1. Load file
2. Print all courses (sorted)
3. Print single course information
4. Exit

**Runtime Analysis**

| **Data Structure** | **Load Data** | **Search Single Course** | **Print All Courses (Sorted)** | **Memory Usage** |
|---|---|---|---|---|
| Vector | O(n * m) | O(n) | O(n log n) | O(n) |
| Hash Table | O(n * m) | O(1) average | O(n) | O(n) |
| BST | O(n log n) avg | O(log n) avg | O(n) | O(n) |

*Notes:*

n = number of courses, m = average number of prerequisites

**Advantages and Disadvantages**

**Vector**

- Advantages: Simple to implement; easy iteration
- Disadvantages: Slow search for single courses (O(n)); sorting needed for alphanumeric list

**Hash Table**

- Advantages: Very fast course lookup (O(1) average); simple insertion
- Disadvantages: Not inherently sorted; printing in order requires extra sorting

**Binary Search Tree**

- Advantages: Automatically maintains sorted order; efficient search in average case (O(log n))
- Disadvantages: Can become unbalanced, leading to O(n) worst-case search; slightly more complex to implement

**Recommendation**

The advising program at ABCU requires the hash table because it enables advisors to retrieve course information at the fastest speed. The binary search tree becomes essential when the need arises to maintain an alphanumeric order to track data but users must perform extra work to keep the tree structure in balance. The vector format delivers basic functionality yet demonstrates poor performance when handling extensive data and conducting multiple searches.