Psuedocode Development

1. Pseudocode for Prerequeisites

This pseudocode outlines how to read from a file, parse the data, create course objects, and handle formatting errors.

FUNCTION LoadCoursesFromFile(filePath);  
 OPEN file at filePath  
 IF file is not open THEN  
 PRINT “Error: Could not open file.  
 RETURN  
 ENDIF

WHILE NOT end of file:

READ line from file  
 IF line is improperly formatted THEN  
 PRINT “Formatting error in line: “ + line  
 CONTINUE  
 ENDIF  
 courseData = PARSE line into courseNumber, title, prerequisites  
 course = NEW Course(courseNumber, title, prerequisites)  
 ADD course to coursesDataStructure  
 ENDWHILE  
 Close file  
END FUNCTION

1. Pseudocode for Menu  
   This pseudocode presents the menu options for user interaction.

FUNCTION DisplayMenu():

PRINT “1. Load Course Data”  
 PRINT “2. Print Alphanumeric Course List”  
 PRINT “3. Print Course Title and Prerequisites”  
 PRINT “9. Exit:  
 userSelection = GET user input  
 SWITCH userSelection:  
 CASE 1:  
 CALL LoadCoursesFromFile(filePath)  
 CASE 2:  
 CALL PrintCourseList()  
 CASE 3:  
 courseNumber = GET course number input  
 CALL PrintCourseInfo(courseNumber)  
 CASE 9:  
 PRINT “Exiting program.”  
 EXIT  
 DEFAULT:  
 PRINT “Invalid selection. Please try again.”  
 ENDSWITCH  
END FUNCTION

1. Pseudocode for Course list  
   This pseudocode sorts and prints the course list form the data structure.

FUNCTION PrintCourseList():  
 SORT coursesDataStructure by courseNumber in alphanumeric order  
 FOR each course in coursesDataStructure:  
 PRINT course.courseNumber + “: “ + course.title  
 ENDFOR  
END FUNCTION

Runtime Evaluation

|  |  |  |  |
| --- | --- | --- | --- |
| Data Structure | Loading Data (Big O) | Printing Course List (Big O) | Memory Usage |
| Vector | O(n) | O(n log n) | O(n) |
| Hash Table | O(n) | O(n) | O(n) |
| Binary Search Tree | O(n log n) | O(n) | O(n) |

Explanation of Runtime:

* Vector:
  + Loading Data: Each course is appended, resulting in O(n).
  + Printing: Sorting requires O(n log n).
* Hash Table:
  + Loading Data: Each insertion is O(1) on average, leading to O(n).
  + Printing: Direct access to elements allows for O(n).
* Binary Search Tree:
  + Loading Data: Inserting n elements into a BST takes O(n log n) in the worst case (unbalanced).
  + Printing: Traversing the tree takes O(n).

Advantages and Disadvantages

* Vector
  + Advantages:
    - Easy to implement and understand
    - Direct indexing provides O(1) access time for individual elements.
  + Disadvantages:
    - Sorting takes O(n log n), which may be slow for large datasets.
* Hash Table
  + Advantages:
    - Average O(1) access time for retrieval.
    - Fast insertion and deletion operations.
  + Disadvantages:
    - Performance can degrade with many collisions.
    - Requires additional memory for storing pointers.
* Binary Search Tree
  + Advantages:
    - Maintains sorted order, allowing efficient in-order traversal.
    - Dynamic resizing (can grow/shrink)
  + Disadvantages:
    - Requires O(n log n) time for balancing and insertion.
    - Performance can degrade to O(n) if not balanced.

Recommendation

After analyzing the three data structures, **I recommend using a hash table** for this project. The main reason for this choice is its efficient average-case performance for inserting and retrieving course data (O(1) for both operations). Since the program primarily needs to access course titles and prerequisites quickly, the hash table's constant-time complexity for lookups aligns well with the advisor's requirements. While vectors are more straightforward, their slower sorting time could hinder performance with larger datasets. Binary search trees, while suitable for sorted data, may introduce complexity with balancing. Thus, the hash table provides the best balance of efficiency and practicality for the advising program.