# CS 300 Project One - Final Pseudocode and Runtime Analysis

## Vector Data Structure Pseudocode

{Define Course Structure}  
Struct Course  
 courseNumber: string  
 title: string  
 prerequisites: list of strings  
End Struct  
  
{Open, Read, and Validate File}  
Function LoadCourses\_Vector(filename)  
 Open file with filename  
 If file cannot be opened  
 Print "File not found"  
 Exit program  
  
 Initialize courseVector as empty vector  
 Initialize prerequisiteSet as empty set  
  
 For each line in file  
 Split line by comma into tokens  
 If number of tokens < 2  
 Print "Invalid line format"  
 Continue to next line  
  
 courseNumber = tokens[0]  
 courseTitle = tokens[1]  
 prerequisites = tokens[2 to end] (if any)  
  
 Create new Course object:  
 course.courseNumber = courseNumber  
 course.title = courseTitle  
 course.prerequisites = prerequisites  
  
 Add course to courseVector  
  
 For each prerequisite in prerequisites  
 Add prerequisite to prerequisiteSet  
 End For  
  
 For each prereq in prerequisiteSet  
 If prereq not in any course in courseVector  
 Print "Missing prerequisite definition: ", prereq  
 End For  
  
 Close file  
End Function  
  
{Option 2: Print Sorted Course List}  
Function PrintCourseList\_Vector(courseVector)  
 Sort courseVector by courseNumber in ascending alphanumeric order  
 For each course in courseVector  
 Print course.courseNumber + ": " + course.title  
End Function  
  
{Option 3: Print Course Info}  
Function PrintCourseDetails\_Vector(courseVector, targetCourseNumber)  
 For each course in courseVector  
 If course.courseNumber == targetCourseNumber  
 Print "Course Number: " + course.courseNumber  
 Print "Title: " + course.title  
 If course.prerequisites is not empty  
 Print "Prerequisites:"  
 For each prereq in course.prerequisites  
 Print " - " + prereq  
 Else  
 Print "No prerequisites"  
 Return  
 Print "Course not found"  
End Function

## Hash Table Data Structure Pseudocode

Struct Course  
 courseNumber : string  
 title : string  
 prerequisites : list of strings  
End Struct  
  
Function LoadCourses\_Hash(filename)  
 courseTable = empty hash table  
 lineNumber = 0  
  
 For each line in file  
 lineNumber = lineNumber + 1  
 tokens = split line by comma  
 If length of tokens < 2  
 Print "Error on line " + lineNumber + ": Invalid format."  
 Continue  
  
 courseNumber = tokens[0]  
 title = tokens[1]  
 prerequisites = empty list  
  
 If length of tokens > 2  
 For i from 2 to length of tokens - 1  
 prerequisites.append(tokens[i])  
  
 course = new Course  
 course.courseNumber = courseNumber  
 course.title = title  
 course.prerequisites = prerequisites  
  
 courseTable[courseNumber] = course  
 Return courseTable  
End Function  
  
Function PrintCourseList\_Hash(courseTable)  
 keys = all keys from courseTable  
 Sort keys in ascending alphanumeric order  
 For each key in keys  
 course = courseTable[key]  
 Print course.courseNumber + ": " + course.title  
End Function  
  
Function PrintCourse\_Hash(courseNumber, courseTable)  
 If courseNumber not in courseTable  
 Print "Course not found."  
 Return  
 course = courseTable[courseNumber]  
 Print "Course Number: " + course.courseNumber  
 Print "Title: " + course.title  
 If prerequisites empty  
 Print "Prerequisites: None"  
 Else  
 Print "Prerequisites:"  
 For each prereq in course.prerequisites  
 Print "- " + prereq  
End Function

## Binary Search Tree Pseudocode

Function LoadCourses\_BST(filename)  
 Open file with filename  
 If file cannot be opened  
 Print "Error: File not found"  
 Exit program  
  
 Initialize empty list validCourseNumbers  
 Initialize empty list rawCourseLines  
  
 For each line in file:  
 Split line by comma into tokens  
 If number of tokens < 2:  
 Print "Error: Line has insufficient data"  
 Continue  
 courseNumber = tokens[0]  
 Add courseNumber to validCourseNumbers  
 Add line to rawCourseLines  
  
 Initialize empty BinarySearchTree courseTree  
  
 For each line in rawCourseLines:  
 Split line by comma into tokens  
 courseNumber = tokens[0]  
 courseName = tokens[1]  
 prerequisites = tokens[2 to end]  
  
 For each prereq in prerequisites:  
 If prereq not in validCourseNumbers:  
 Print "Error: Prerequisite", prereq, "not found"  
 Continue to next line  
  
 Create new Course object  
 course.courseNumber = courseNumber  
 course.name = courseName  
 course.prerequisites = prerequisites  
  
 Insert course into courseTree using courseNumber as key  
End Function  
  
Function PrintCourseList\_BST(courseTree)  
 Perform in-order traversal of courseTree  
 For each node visited:  
 Print node.course.courseNumber + ": " + node.course.name  
End Function  
  
Function SearchCourse\_BST(courseTree, courseNumber)  
 course = courseTree.search(courseNumber)  
 If course is null  
 Print "Course not found"  
 Return  
 Print course.courseNumber + ": " + course.name  
 If prerequisites not empty  
 Print "Prerequisites:"  
 For each prereqNumber in course.prerequisites  
 prereqCourse = courseTree.search(prereqNumber)  
 If prereqCourse not null  
 Print prereqCourse.courseNumber + ": " + prereqCourse.name  
 Else  
 Print "Warning: prerequisite", prereqNumber, "not found"  
End Function

## Menu Pseudocode (Common to All)

Function Menu()  
 While True  
 Print "1. Load file data"  
 Print "2. Print alphanumeric list of courses"  
 Print "3. Print course details"  
 Print "9. Exit"  
 choice = get user input  
 If choice == 1  
 filename = prompt "Enter filename:"  
 Call LoadCourses\_<Structure>(filename)  
 Else If choice == 2  
 Call PrintCourseList\_<Structure>()  
 Else If choice == 3  
 courseNumber = prompt "Enter course number:"  
 Call PrintCourse\_<Structure>(courseNumber)  
 Else If choice == 9  
 Print "Exiting program"  
 Break  
 Else  
 Print "Invalid option"  
End Function

## Runtime Analysis - File Read & Course Creation

|  |  |  |  |
| --- | --- | --- | --- |
| Data Structure | Read/Parse/Create | Insert Cost per Course | Total Runtime |
| Vector | O(n) | O(1) amortized | O(n) |
| Hash Table | O(n) | O(1) average | O(n) average |
| BST Balanced | O(n) | O(log n) | O(n log n) |
| BST Unbalanced | O(n) | O(n) | O(n^2) worst |

## Advantages and Disadvantages

Vector:  
+ Simple to implement, low memory overhead  
+ Good cache performance  
- Slow lookups if unsorted (O(n))  
- Maintaining sort on insert is costly (O(n))  
  
Hash Table:  
+ O(1) average-case lookups and inserts  
+ Ideal for frequent search  
- Unordered; requires sorting for ordered output (O(n log n))  
- Higher memory usage  
  
Binary Search Tree (Balanced):  
+ Maintains sorted order for free  
+ O(log n) lookups and inserts  
- Slightly higher memory and complexity  
- Slower lookups than hash table in average case

## Recommendation

A balanced binary search tree is recommended for this application.  
It naturally maintains the courses in sorted order for Option 2 in O(n) time  
and supports lookups in O(log n) for Option 3. While hash tables are faster  
on average for single lookups, they require extra sorting work for ordered output,  
making them less efficient when both sorted listing and searching are frequent tasks.