# Jacob Hasbrook CS-300 Project One

## Introduction

This project contains the final pseudocode and runtime analysis for the advising program required by ABCU's Computer Science department. It integrates the pseudocode for vector, hash table, and binary search tree data structures, incorporating updates for Milestones 1, 2, and 3.

## Vector Implementation

### Course Structure

DEFINE STRUCT Course  
 String courseID  
 String courseName  
 Integer prerequisiteCount  
 List<String> prerequisiteList  
  
 FUNCTION Course() // Constructor  
 courseID = ""  
 courseName = ""  
 prerequisiteCount = 0  
 prerequisiteList = []  
 END FUNCTION  
END STRUCT

### LoadCourses Function

FUNCTION LoadCourses(String filePath)  
 CREATE List<Course> courseList  
  
 OPEN file at filePath FOR reading  
 IF file cannot be opened THEN  
 PRINT "Error: Unable to open file at " + filePath  
 RETURN empty list  
 END IF  
  
 WHILE NOT end of file  
 READ line FROM file  
 IF line IS blank THEN  
 CONTINUE  
 END IF  
  
 SPLIT line INTO tokens USING comma as delimiter  
 IF number of tokens < 2 THEN  
 PRINT "Warning: Skipping malformed line: " + line  
 CONTINUE  
 END IF  
  
 CREATE Course tempCourse  
 SET tempCourse.courseID = tokens[0]  
 SET tempCourse.courseName = tokens[1]  
  
 IF tempCourse.courseID IS empty OR tempCourse.courseName IS empty THEN  
 PRINT "Error: Invalid course data: " + line  
 CONTINUE  
 END IF  
  
 FOR i = 2 TO length(tokens) - 1  
 ADD tokens[i] TO tempCourse.prerequisiteList  
 END FOR  
 SET tempCourse.prerequisiteCount = length(tempCourse.prerequisiteList)  
  
 ADD tempCourse TO courseList  
 END WHILE  
  
 CLOSE file  
 RETURN courseList  
END FUNCTION

### PrintCourse Function

FUNCTION PrintCourse(List<Course> courseList, String targetID)  
 SET course = SearchCourse(courseList, targetID)  
 IF course IS null THEN  
 PRINT "Error: Course with ID " + targetID + " not found."  
 RETURN  
 END IF  
  
 PRINT "Course ID: " + course.courseID  
 PRINT "Course Name: " + course.courseName  
 IF course.prerequisiteCount == 0 THEN  
 PRINT "Prerequisites: None"  
 ELSE  
 PRINT "Prerequisites: " + JOIN(course.prerequisiteList, ", ")  
 END IF  
END FUNCTION

## Hash Table Implementation

Pseudocode from Milestone 2 remains unchanged except updated handling for duplicate course IDs.

### Insertion Function

FUNCTION Insert(Course course)  
 CALCULATE key = Hash(course.courseID)  
 FOR EACH existingCourse IN table[key]  
 IF existingCourse.courseID == course.courseID THEN  
 PRINT "Error: Duplicate course ID detected: " + course.courseID  
 RETURN  
 END IF  
 END FOR  
 ADD course TO table[key]  
END FUNCTION

## Binary Search Tree Implementation

Pseudocode from Milestone 3 remains unchanged except updated handling for duplicate course IDs and validation.

### Insertion Function

FUNCTION InsertNode(Node currentNode, Course course)  
 IF course.courseID == currentNode.courseData.courseID THEN  
 PRINT "Error: Duplicate course ID detected: " + course.courseID  
 RETURN  
 END IF  
  
 IF course.courseID < currentNode.courseData.courseID THEN  
 IF currentNode.left == NULL THEN  
 CREATE Node  
 currentNode.left.courseData = course  
 ELSE  
 CALL InsertNode(currentNode.left, course)  
 END IF  
 ELSE  
 IF currentNode.right == NULL THEN  
 CREATE Node  
 currentNode.right.courseData = course  
 ELSE  
 CALL InsertNode(currentNode.right, course)  
 END IF  
 END IF  
END FUNCTION

## Runtime Analysis

The table below summarizes the Big O notations for file loading, searching, and printing sorted operations for each data structure: Vector, Hash Table, and Binary Search Tree. Both average and worst case scenarios are included where applicable.

| Operation | Vector | Hash Table | Binary Search Tree |
| --- | --- | --- | --- |
| File Loading | O(n) | O(n) avg O(n²) worst | O(n log n) avg O(n²) worst |
| Searching | O(n) | O(1) avg O(n) worst | O(log n) avg O(n) worst |
| Printing Sorted | O(n log n) | O(n log n) | O(n) |

## Recommendation

Based on the analysis, the Binary Search Tree is recommended for this application. It provides a good balance between efficient searching and sorting. Although it requires proper balancing for optimal performance, it ensures an efficient way to generate sorted outputs via in order traversal.