This document contains pseudocode and runtime analysis for implementing ABC University's advising system using three data structures: vector, hash table, and binary search tree. Each implementation loads course data from a file, validates formatting, stores course objects, and supports searching and printing course information.

STRUCT Course

STRING courseNumber

STRING courseName

VECTOR<STRING> prerequisites

END STRUCT

FUNCTION LoadCourses(filePath)

DECLARE VECTOR<Course> courseList

DECLARE SET<STRING> validCourseNumbers

OPEN file at filePath FOR reading

IF file cannot be opened THEN

PRINT "Error: Cannot open file."

RETURN empty courseList

END IF

WHILE NOT end of file

READ line

SPLIT line by comma INTO tokens

IF LENGTH(tokens) < 2 THEN

PRINT "Error: Line missing course name or number."

CONTINUE

END IF

DECLARE Course newCourse

SET newCourse.courseNumber = tokens[0]

SET newCourse.courseName = tokens[1]

INITIALIZE newCourse.prerequisites as empty vector

FOR i FROM 2 TO LENGTH(tokens) - 1

ADD tokens[i] TO newCourse.prerequisites

END FOR

ADD newCourse TO courseList

ADD newCourse.courseNumber TO validCourseNumbers

END WHILE

CLOSE file

CALL ValidatePrerequisites(courseList, validCourseNumbers)

RETURN courseList

END FUNCTION

FUNCTION ValidatePrerequisites(courseList, validCourseNumbers)

FOR EACH course IN courseList

FOR EACH prereq IN course.prerequisites

IF prereq NOT IN validCourseNumbers THEN

PRINT "Warning: Prerequisite " + prereq + " for " + course.courseNumber + " not found."

END IF

END FOR

END FOR

END FUNCTION

FUNCTION searchCourse(courseList, targetCourseNumber)

FOR EACH course IN courseList

IF course.courseNumber == targetCourseNumber THEN

PRINT "Course Number: " + course.courseNumber

PRINT "Course Name: " + course.courseName

IF LENGTH(course.prerequisites) == 0 THEN

PRINT "Prerequisites: None"

ELSE

PRINT "Prerequisites:"

FOR EACH prereq IN course.prerequisites

PRINT " - " + prereq

END FOR

END IF

RETURN

END IF

END FOR

PRINT "Course " + targetCourseNumber + " not found."

END FUNCTION

FUNCTION PrintAllCourses(courseList)

SORT courseList BY course.courseNumber

FOR EACH course IN courseList

PRINT course.courseNumber + ": " + course.courseName

END FUNCTION

FUNCTION LoadCourses(filename)

OPEN file with name 'filename'

IF file cannot be opened:

Display error and exit

Initialize empty list courseLines

FOR each line in file:

Trim whitespace

IF line is not empty:

Add line to courseLines

Close file

FUNCTION ValidateCourseLines(courseLines)

Initialize empty set courseNumbers

FOR each line in courseLines:

Split line by comma into tokens

IF number of tokens < 2:

Display format error and exit

Add first token (course number) to courseNumbers

FOR each line in courseLines:

Split line by comma into tokens

FOR each token after the second:

IF token not in courseNumbers:

Display missing prerequisite error and exit

FUNCTION StoreCourses(courseLines)

Initialize empty hash table courseTable

FOR each line in courseLines:

Split line by comma into tokens

Create new Course object

Set Course.courseNumber ← tokens[0]

Set Course.title ← tokens[1]

FOR each token after the second:

Add token to Course.prerequisites

Insert Course into courseTable using courseNumber as key

FUNCTION PrintCourse(courseNumber)

Search courseTable for courseNumber

IF course not found:

Display "Course not found"

Exit

Display courseNumber and title

IF prerequisites list is not empty:

Display "Prerequisites:"

FOR each prerequisite in course.prerequisites:

Display prerequisite

ELSE:

Display "No prerequisites"

FUNCTION PrintAllCourses()

Extract all keys from courseTable

Sort keys alphanumerically

FOR each key:

Display courseTable[key].courseNumber and title

FUNCTION loadCourses(filePath)

Open file at filePath

IF file cannot be opened:

Print "Error: Cannot open file"

Exit function

Initialize empty list courseLines

FOR each line in file:

IF line is not empty:

Append line to courseLines

Close file

FUNCTION validateCourseLines(courseLines)

Initialize empty set validCourseNumbers

FOR each line in courseLines:

Split line by comma into tokens

IF number of tokens < 2:

Print "Error: Line missing course number or title"

Exit function

Add tokens[0] to validCourseNumbers

FOR each line in courseLines:

Split line by comma into tokens

FOR each token after the second:

IF token not in validCourseNumbers:

Print "Error: Prerequisite " + token + " not found"

Exit function

FUNCTION buildCourseTree(courseLines)

Initialize empty BinarySearchTree courseTree

FOR each line in courseLines:

Split line by comma into tokens

Create new Course object

Set Course.courseNumber = tokens[0]

Set Course.title = tokens[1]

FOR each token after the second:

Append token to Course.prerequisites

Insert Course into courseTree using courseNumber as key

Return courseTree

FUNCTION searchCourse(courseTree, courseNumber)

Search courseTree for courseNumber

IF course not found:

Print "Course not found"

Exit function

Print course.courseNumber + ": " + course.title

IF course.prerequisites is empty:

Print "No prerequisites"

ELSE:

FOR each prereq in course.prerequisites:

Search courseTree for prereq

IF found:

Print prereq.courseNumber + ": " + prereq.title

ELSE:

Print "Prerequisite " + prereq + " not found"

FUNCTION PrintAllCourses(courseTree)

Perform in-order traversal of courseTree

FOR each node visited:

Print course.courseNumber + ": " + course.title

FUNCTION DisplayMenu()

LOOP until user selects Exit

PRINT options:

1. Load Data

2. Print Course List

3. Print Course Details

9. Exit

GET user input

IF input == 1 THEN

SET courses = LoadCourses("courses.txt")

ELSE IF input == 2 THEN

CALL PrintAllCourses(courses)

ELSE IF input == 3 THEN

PROMPT for course number

CALL searchCourse(courses, courseNumber)

ELSE IF input == 9 THEN

EXIT program

ELSE

PRINT "Invalid option"

Runtime Analysis Chart

| Operation | Vector | Hash Table | Binary Search Tree |

|------------------------|--------------|------------------|------------------------|

| Load & Parse File | O(n) | O(n) | O(n log n) |

| Insert Course | O(1) | O(1) avg | O(log n) avg |

| Search Course | O(n) | O(1) avg | O(log n) avg |

| Print Sorted List | O(n log n) | O(n log n) | O(n) |

| Memory Usage | Moderate | High | Moderate |

Evaluation and Recommendation

Vector: Easy to implement, but linear search and sorting are inefficient for large datasets.

Hash Table: Fast lookup but requires extra sorting logic.

Binary Search Tree: Efficient search and naturally sorted output.

Recommended Structure: Binary Search Tree balances performance and simplicity while meeting all the advisor’s goals.