BEGIN

// Define structure for storing course information

STRUCT CourseInfo

STRING courseID // Identifier for the course

STRING courseName // Name of the course

LIST<String> prerequisites // List of prerequisite course IDs

END STRUCT

// Define structure for the nodes in the Binary Search Tree

STRUCT CourseNode

CourseInfo course // Data held by this node

CourseNode left // Left child node (for smaller courseID values)

CourseNode right // Right child node (for larger courseID values)

END STRUCT

// Function to create a new CourseInfo object

FUNCTION createCourseInfo(courseID, courseName, prerequisites)

DECLARE CourseInfo newCourse

newCourse.courseID = courseID

newCourse.courseName = courseName

newCourse.prerequisites = prerequisites

RETURN newCourse

END FUNCTION

// Function to insert a course into the Binary Search Tree (BST)

FUNCTION addCourseToBST(BST, CourseInfo newCourse)

IF BST IS NULL THEN

CREATE new CourseNode

new CourseNode.course = newCourse

new CourseNode.left = NULL

new CourseNode.right = NULL

RETURN new CourseNode

END IF

IF newCourse.courseID < BST.course.courseID THEN

BST.left = addCourseToBST(BST.left, newCourse)

ELSE IF newCourse.courseID > BST.course.courseID THEN

BST.right = addCourseToBST(BST.right, newCourse)

END IF

RETURN BST

END FUNCTION

// Function to read course data from a file and populate the BST

FUNCTION loadCourseDataFromFile(FILE\_PATH)

DECLARE CourseNode BST = NULL

OPEN FILE\_PATH FOR READING

IF file cannot be opened THEN

PRINT "Error: Unable to open the file containing course data."

RETURN NULL

END IF

FOR each line in the file DO

SPLIT line into components by commas

// Ensure the line contains at least the course ID and name

IF length of components < 2 THEN

PRINT "Warning: Skipping invalid entry - " + line

CONTINUE

END IF

DECLARE courseID = components[0]

DECLARE courseName = components[1]

DECLARE prerequisites = EMPTY LIST

// If any prerequisites are listed, add them to the list

FOR each component from the third onward DO

APPEND component to prerequisites list

END FOR

DECLARE newCourse = createCourseInfo(courseID, courseName, prerequisites)

// Add the new course to the BST

BST = addCourseToBST(BST, newCourse)

END FOR

CLOSE file

RETURN BST

END FUNCTION

// Function to ensure all course prerequisites are valid

FUNCTION checkCoursePrerequisites(TreeNode BST)

DECLARE SET<String> allCourseIDs

// Retrieve all course IDs stored in the BST

CALL gatherAllCourseIDs(BST, allCourseIDs)

// Check each course’s prerequisites for validity

CALL verifyPrerequisitesExist(BST, allCourseIDs)

END FUNCTION

// Helper function to collect all course IDs in a set for easy lookup

FUNCTION gatherAllCourseIDs(TreeNode BST, SET<String> allCourseIDs)

IF BST IS NOT NULL THEN

ADD BST.course.courseID TO allCourseIDs

CALL gatherAllCourseIDs(BST.left, allCourseIDs)

CALL gatherAllCourseIDs(BST.right, allCourseIDs)

END IF

END FUNCTION

// Helper function to validate if prerequisites exist in the course set

FUNCTION verifyPrerequisitesExist(TreeNode BST, SET<String> allCourseIDs)

IF BST IS NOT NULL THEN

FOR each prereq IN BST.course.prerequisites DO

IF prereq NOT IN allCourseIDs THEN

PRINT "Error: Missing prerequisite " + prereq + " for " + BST.course.courseID

END IF

END FOR

CALL verifyPrerequisitesExist(BST.left, allCourseIDs)

CALL verifyPrerequisitesExist(BST.right, allCourseIDs)

END IF

END FUNCTION

// Function to display the details of a specific course and its prerequisites

FUNCTION showCourseDetails(TreeNode BST, STRING courseID)

IF BST IS NULL THEN

PRINT "Error: Course " + courseID + " not found."

RETURN

END IF

IF courseID < BST.course.courseID THEN

CALL showCourseDetails(BST.left, courseID)

ELSE IF courseID > BST.course.courseID THEN

CALL showCourseDetails(BST.right, courseID)

ELSE

PRINT "Course ID: " + BST.course.courseID

PRINT "Course Name: " + BST.course.courseName

PRINT "Prerequisites:"

IF BST.course.prerequisites IS EMPTY THEN

PRINT "No prerequisites."

ELSE

FOR each prereq IN BST.course.prerequisites DO

PRINT prereq

END FOR

END IF

END IF

END FUNCTION

// Main function to control the program flow

FUNCTION main()

STRING filePath = "courses\_data.txt"

CourseNode BST = loadCourseDataFromFile(filePath)

IF BST IS NULL THEN

RETURN

END IF

CALL checkCoursePrerequisites(BST)

WHILE TRUE DO

PRINT "Enter course ID to search for (or type 'EXIT' to quit): "

STRING userInput = GET USER INPUT

IF userInput == "EXIT" THEN

BREAK

END IF

CALL showCourseDetails(BST, userInput)

END WHILE

END FUNCTION

CALL main()

END

This version of the pseudocode uses a binary search tree (BST) to store course data, allowing for efficient organization and sorted traversal based on course IDs. Reading the file and inserting each course into the BST both take O(n log n) time on average, assuming a balanced tree. Searching for a course also takes O(log n) time on average, which is a clear improvement over a vector’s O(n) linear search. However, in the worst-case scenario, such as inserting already sorted data into an unbalanced BST. The insertion and search time can degrade to O(n). Compared to vectors and hash tables, BSTs offer the benefit of in-order traversal for sorted output, which can be useful if ordering is important. While slightly more complex, a BST is a solid middle ground between the fast lookups of a hash table and the simplicity of a vector, making it a good choice when both search efficiency and ordered data access are required.