**Pseudocode Milestone 1**

START

// Initializes the vector data structure to store all course objects

SET courseList as empty VECTOR

// Opens the course data input file

OPEN "course\_data.txt" AS inputFile

// Checks to see if the file opens successfully

IF inputFile CANNOT be opened THEN

PRINT "Error: Cannot open file."

EXIT

// Reads and processes each line from the file

FOR EACH line IN inputFile

SPLIT line BY comma INTO tokens

// Validates that lines have at least a course number and name

IF LENGTH(tokens) < 2 THEN

PRINT "Error: Invalid line format. Must contain at least course number and course name."

CONTINUE TO NEXT line

END IF

SET courseNumber = tokens[0]

SET courseName = tokens[1]

// Validates that each prerequisite exists in courseList

FOR index FROM 2 TO LENGTH(tokens) - 1

SET prereq = tokens[index]

SET prereqFound = FALSE

FOR EACH existingCourse IN courseList

IF existingCourse.courseNumber == prereq THEN

SET prereqFound = TRUE

BREAK

END IF

END FOR

IF prereqFound == FALSE THEN

PRINT "Error: Prerequisite " + prereq + " not defined in any previous course line."

CONTINUE TO NEXT line

END IF

END FOR

// Creates course object and stores it

CREATE courseObject AS STRUCT

SET courseObject.courseNumber = courseNumber

SET courseObject.courseName = courseName

SET courseObject.prerequisites = empty VECTOR

FOR index FROM 2 TO LENGTH(tokens) - 1

ADD tokens[index] TO courseObject.prerequisites

END FOR

END STRUCT

ADD courseObject TO courseList

END FOR

CLOSE inputFile

// Allows user to search for a course

PRINT "Enter course number to search:"

READ userInput

SET found = FALSE

FOR EACH course IN courseList

IF course.courseNumber == userInput 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

SET found = TRUE

BREAK

END IF

END FOR

IF found == FALSE THEN

PRINT "Course " + userInput + " not found."

END IF

END

**Pseudocode Milestone 2**

START

DEFINE STRUCT Course:

courseNumber

courseTitle

list of prerequisites

DEFINE STRUCT Node:

Course course

Node\* next

DECLARE hashTable as array of linked list buckets (fixed size)

FUNCTION hash(courseNumber):

EXTRACT numeric portion of courseNumber (e.g., "CSCI300" → 300)

CONVERT to integer

RETURN integer MODULO size of hashTable

FUNCTION readCourseFile(fileName):

OPEN the file using fileName

IF file fails to open:

PRINT "Error: Cannot open file"

RETURN

FOR each line in the file:

SPLIT the line into tokens using commas

IF number of tokens < 2:

PRINT "Format error: Missing course number or title"

CONTINUE to next line

SET courseNumber = tokens[0]

SET courseTitle = tokens[1]

CREATE empty list prereqList

FOR each token starting from index 2:

ADD token to prereqList

CALL validatePrerequisites(prereqList)

CALL insertCourse(courseNumber, courseTitle, prereqList)

CLOSE the file

FUNCTION validatePrerequisites(prereqList):

FOR each prereq in prereqList:

IF prereq is not found in hash table:

PRINT "Warning: Prerequisite " + prereq + " not found"

FUNCTION insertCourse(courseNumber, courseTitle, prereqList):

CREATE new Course object

SET course.courseNumber = courseNumber

SET course.courseTitle = courseTitle

SET course.prerequisites = prereqList

COMPUTE index = hash(courseNumber)

IF hashTable[index] is empty:

INSERT new Node with course into hashTable[index]

ELSE:

SET current = head of hashTable[index]

WHILE current.next is not null:

MOVE to next node

APPEND new Node with course at the end

FUNCTION printAllCourses():

FOR each bucket in hashTable:

SET current = head of bucket

WHILE current is not null:

PRINT current.course.courseNumber + ": " + current.course.courseTitle

IF current.course.prerequisites is not empty:

PRINT "Prerequisites:"

FOR each prereq in current.course.prerequisites:

PRINT " - " + prereq

ELSE:

PRINT "No prerequisites"

MOVE to next node

MAIN PROGRAM:

INITIALIZE hashTable with empty buckets

SET fileName = "courses.txt"

CALL readCourseFile(fileName)

CALL printAllCourses()

END

**Pseudocode Milestone 3**

START

STRUCT Course:

STRING courseNumber

STRING courseName

LIST of STRING prerequisites

STRUCT Node:

Course course

Node\* left

Node\* right

CONSTRUCTOR Node(Course aCourse):

course = aCourse

left = nullptr

right = nullptr

CLASS BinarySearchTree:

PRIVATE:

Node\* root

PUBLIC:

CONSTRUCTOR BinarySearchTree():

SET root = nullptr

FUNCTION Insert(Course course):

IF root is nullptr:

SET root = new Node(course)

ELSE:

CALL addNode(root, course)

FUNCTION addNode(Node\* node, Course course):

IF course.courseNumber < node.course.courseNumber:

IF node.left is nullptr:

node.left = new Node(course)

ELSE:

CALL addNode(node.left, course)

ELSE:

IF node.right is nullptr:

node.right = new Node(course)

ELSE:

CALL addNode(node.right, course)

FUNCTION PrintInOrder():

CALL inOrder(root)

FUNCTION inOrder(Node\* node):

IF node is not nullptr:

CALL inOrder(node.left)

PRINT node.course.courseNumber, node.course.courseName

IF node.course.prerequisites is not empty:

PRINT "Prerequisites: ", each item in prerequisites

CALL inOrder(node.right)

FUNCTION LoadCoursesFromFile(string filePath, BinarySearchTree\* bst):

OPEN file at filePath

IF file cannot be opened:

PRINT "Error: Unable to open file"

RETURN

FOR each line in the file:

SPLIT line by commas into list called tokens

IF size of tokens < 2:

PRINT "Error: Invalid format in line"

CONTINUE

CREATE Course course

SET course.courseNumber = tokens[0]

SET course.courseName = tokens[1]

FOR i FROM 2 to size of tokens - 1:

ADD tokens[i] to course.prerequisites

CALL bst->Insert(course)

CLOSE file

FUNCTION ValidatePrerequisites(BinarySearchTree\* bst, list of Course allCourses):

FOR each course in allCourses:

FOR each prerequisite in course.prerequisites:

IF prerequisite is not found in allCourses:

PRINT "Error: Missing prerequisite ", prerequisite, " for course ", course.courseNumber

MAIN FUNCTION:

SET filePath = "courses.txt"

CREATE BinarySearchTree bst

CREATE empty list allCourses

CALL LoadCoursesFromFile(filePath, &bst)

CALL bst.PrintInOrder()

END

**Pseudocode for Menu**

START

SET userChoice = ""

WHILE userChoice != "3"

PRINT "Welcome to the Course Planner"

PRINT "-----------------------------"

PRINT "1. Load Data Structure"

PRINT "2. Print Course List"

PRINT "3. Exit"

PRINT "Please enter your choice: "

READ userChoice

IF userChoice == "1" THEN

CALL LoadCoursesToBST("course\_data.txt", bst)

PRINT "Data structure loaded successfully."

ELSE IF userChoice == "2" THEN

CALL PrintCourseListInOrder(bst.root)

ELSE IF userChoice == "3" THEN

PRINT "Goodbye."

ELSE

PRINT "Invalid input. Please select 1, 2, or 3."

END WHILE

END

**Pseudocode for Course List**

FUNCTION PrintCourseListInOrder(node)

IF node == NULL THEN

RETURN

END IF

CALL PrintCourseListInOrder(node.left)

PRINT node.course.courseNumber + ", " + node.course.courseName

CALL PrintCourseListInOrder(node.right)

END FUNCTION

**Evaluation**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Open file** | 1 | 1 | 1 |
| **For each line in the file** | 1 | n | n |
| **Split line into tokens** | 1 | n | n |
| **Check format of line** | 1 | n | n |
| **Create course object** | 1 | n | n |
| **Loop through tokens to collect prerequisites** | 1 | p x n | p x n |
| **Insert course into data structure (vector/hash/BST)** | 1 | n | depends |
| **Total Cost (Vector)** | | | 5n + p x n |
| **Total Cost (Hash Table)** | | | 5n + p x n |
| **Total Cost (BST, balanced)** | | | 5n + p x n + log n per insert = n log n |
| **Worst-Case Big O (Vector/Hash)** | | | O(n x p) |
| **Worst-Case Big O (BST)** | | | O(n log n + n x p) |

**Vector**

* **Advantages**:
  + Simple to use and understand.
  + Efficient for appending data at the end.
  + Maintains input order without extra effort.
* **Disadvantages**:
  + Searching or matching prerequisites requires a full linear scan.
  + Poor scalability when data grows large.
  + No automatic sorting unless added manually.

**Hash Table**

* **Advantages**:
  + Constant-time (O(1)) lookups for course and prerequisites.
  + Ideal for validating prerequisite existence quickly.
  + Fast insertions and flexible key-based access.
* **Disadvantages**:
  + Does not maintain any ordering.
  + Requires custom sorting if needed later.
  + Hash collisions or poor distribution can degrade performance.

**Binary Search Tree (BST)**

* **Advantages**:
  + Maintains alphanumeric order inherently.
  + Efficient traversal and printing in sorted order.
  + Balanced trees support O(log n) insert/search.
* **Disadvantages**:
  + Slightly more complex implementation.
  + Worst-case performance degrades to O(n) if tree is unbalanced.
  + Inserting requires maintaining order across left/right branches.

Based on the Big O analysis and structural features, the Binary Search Tree is the recommended data structure. It supports both major requirements: efficient insert and lookup while maintaining alphanumeric ordering. While a hash table offers faster lookups, it sacrifices ordered traversal. The vector structure is too inefficient for large datasets and fails both in lookup time and sorting. The BST balances performance and functionality, particularly with in-order traversal allowing the course list to be printed in order without extra sorting logic. This makes it the most appropriate choice for ABCU’s advising program.