**Pseudocode for the menu:**

FUNCTION DisplayMenu():

PRINT “Select an option:”

PRINT “1: Load the file data into the data structure”

PRINT “2: Print an alphanumerically ordered list of all the courses”

PRINT “3: Print the course title and the prerequisites for any individual course”

PRINT “9: Exit the program”

PRINT “Enter your choice:”

END FUNCTION

FUNCTION Menu():

DECLARE choice AS INTEGER

DECLARE isDataLoaded AS BOOLEAN = FALSE

WHILE TRUE

CALL DisplayMenu()

INPUT choice

SWITCH choice

CASE 1:

PRINT “Loading file data into the data structure…”

CALL LoadFileData()

isDataLoaded = TRUE

PRINT “File data loaded successfully”

BREAK

CASE 2:

IF isDataLoaded == FALSE THEN

PRINT “Error: Data must be loaded first. Select Option 1.”

ELSE

PRINT “Printing alphanumerically ordered list courses…”

CALL PrintSortedCourses()

END IF

BREAK

CASE 3:

IF isDataLoaded == FALSE THEN

PRINT “Error: Data must be loaded first. Select Option 1”

ELSE

PRINT “Enter the course number:”

INPUT courseNumber

PRINT “Searching for course information…”

CALL PrintCourseInfo(courseNumber)

END IF

BREAK

CASE 9:

PRINT “Exiting program. Goodbye!”

EXIT

DEFAULT:

PRINT “Invalid option. Please enter a valid choice (1,2,3 or 9):”

BREAK

END SWITCH

END WHILE

END FUNCTION

FUNCTION LoadFileData():

// Open the file

// Parse each line and check for errors

// Create course objects and store them in the data structure

END FUNCTION

FUNCTION PrintSortedCourses(binarySearchTree):

PRINT “Courses in alphanumeric order:”

CALL InOrderTraversal(binarySearchTree.root)

END FUNCTION

FUNCTION InOrderTraversal(node):

// If node is NULL, return

IF node IS NULL:

RETURN

END IF

// Recursively traverse the left subtree

CALL InOrderTraversal (node.left)

PRINT “Course Number:”, node.course.courseNumber,

“Course Name:”, node.course.courseName

// Recursively traverse the right subtree

CALL InOrderTraversal (node.right)

END FUNCTION

FUNCTION PrintCourseInfo(courseNumber):

// Search the data structure for the course

// If found, print course number, title and prerequisites

// If not found, print “Course not found”

END FUNCTION

**Evaluate the run time and memory of data structures that could be used to address the requirements:**

|  |  |  |
| --- | --- | --- |
| Data Structure | Reading File | Creating Course objects |
| Vector | O(n) | O(1) |
| Hash Table | O(n) | O(1) |
| Tree | O(n) | O(n) |

**Advantages and Disadvantages of each structure in my evaluation:**

* Vector:
  + Advantages: Simple to implement, easy to use for sequential data
  + Disadvantages: Slower insertion and deletion time
* Hash Table:
  + Advantages: Fast search and insertion time, flexible data types using key-value pairs
  + Disadvantages: Collisions, not in order
* Tree:
  + Advantages: Sorted data, fast searching and sorting, hierarchical relationships
  + Disadvantages: Tree balancing can add complexity, requires additional pointers for nodes which increases memory usage

**Make a recommendation for which data structure you plan to use in your code:**

I would recommend to use the Tree data structure because this structure aligns well with the program’s requirements. It will naturally maintain data in sorted order. It will print alphanumerically sorted courses efficiently and also quickly search for course details. The hierarchical relationships are also beneficial for prerequisites handling. It will use slightly more memory because of the node pointers but the benefits will outweigh the memory usage.