Start with an empty data structure (e.g., vector, hash table, or binary search tree) for storing course information

Function DisplayMenu():

Print "1: Load course data"

Print "2: Print ordered list of all courses"

Print "3: Print course information and prerequisites"

Print "9: Exit program"

Main():

Start with an empty data structure (choose either vector, hash table, or binary search tree)

While True:

Call DisplayMenu()

userChoice = Get user input

If Choice == 1:

Call LoadCourseData

Else If Choice == 2:

Call PrintOrderedCourses

Else If Choice == 3:

Call PrintCourseInfo

Else If Choice == 9:

Print "Exiting program"

Break

Else:

Print "Invalid option. Please try again."

Call Main()

LoadCourseData:

Open file with name "courses.txt"

For each line in the file:

proces the line to create a course object

Add the course object to the data structure

Close the file

Print "Course data loaded successfully"

PrintOrderedCourses:

If Structure is empty:

Print "No course data available. Please load the data first."

Else If structure is a vector:

Sort the vector by course number in alphanumeric order

For each course in the sorted vector:

Print course number and title

Else If Structure is a hash table:

Create an empty list to store course numbers

For each key in the hash table:

Add the key to the list

Sort the list by course number in alphanumeric order

For each key in the sorted list:

Retrieve the course object from the hash table

Print course number and title

Else If Structure is a binary search tree:

Traverse the tree in-order

For each course in the in-order traversal:

Print course number and title

PrintCourseInfo:

courseNumber = Get user input for the course number

If the course with courseNumber exists in the data structure:

Retrieve the course object

Print course number, title, and prerequisites

Else:

Print "Course not found"

Process Line(line):

Separate the line into parts using commas

If there are less than 2 parts:

Show an error message and stop

courseNumber = first part

courseTitle = second part

prerequisites = remaining parts (if any)

Return a new course object with courseNumber, courseTitle, and prerequisites

AddCourseToDataStructure(course, dataStructure):

If the Structure is a vector:

Add the course object to the vector

Else If Structure is a hash table:

Add the course object to the hash table using the course number as the key

Else If Structure is a binary search tree:

Add the course object to the binary search tree

|  |  |  |  |
| --- | --- | --- | --- |
|  | VECTOR | HASH TABLE | BINARY TREE |
| Cost Per Line | O(1) | O(1) | O(1) |
| # of times line will execute | O(n) | O(n) | O(n log n) |
| Worst Case | O(n) | O(n) | O( n^2) |

Run Time Analysis

Vector  
Advantage:  
-simplicity for the implementation and use

Disadvantage:  
- Insert/Delete slow due to shift element (O(n))

-Not suitable for search/sort  
Binary Search Tree

Advantage:

-Average time for insertion/deletion/search

-Efficient handle on large data set

Disadvantages:

-Worst time complex is O(n) due to collisions  
While all three data structures demonstrate disadvantages/ advantages, the recommendation is using hash table for storing and management of course information/ Running efficiency with hash table, it provides average 0(1) time complex for insert/delete/search operations. Demonstrating simplicity, making it easy to implement and doesn’t really require balancing but this also helps with scalability to handle large numbers of courses efficiently, a good hash function to use to decrease collisions. Binary Tree off ordered storage, vector is simple to use, but hash table gives the best balance for all three recommendations.