Struct Course

courseNumber String

courseTitle String

prereq List of Strings

Fn loadCourses

Create empty list courses

Create empty set courseNumbers

Open the file

If the file can't open

Print "Could not open file"

Return empty list

For each line in the file

Split line by commas into tokens

If there are fewer than 2 tokens

Print "Invalid input on line: " + line

Continue

Create new Course

course.courseNumber = tokens[0]

course.courseTitle = tokens[1]

For i = 2 to end of tokens

Add tokens[i] to course.prereq

Add course.courseNumber to courseNumbers

Add course to courses

For each course in courses

For each prereq in course.prereq

If prereq not in courseNumbers

Print "Invalid prerequisite: " + prereq

Return empty list

Close the file

Return courses

Fn searchCourse courses courseNumber

For each course in courses

If course.courseNumber == courseNumber

Print course.courseNumber + ": " + course.courseTitle

Print "Prerequisites:"

If course.prereq is empty

Print "None"

Else

For each prereq in course.prereq

Print prereq

Return

Print "Course not found"

Fn printAllCourses courses

Sort courses by courseNumber

For each course in courses

Print course.courseNumber + ": " + course.courseTitle

Fn showMenu

courses = empty

dataLoaded = false

While true

Print "1. Load course data"

Print "2. Print all courses"

Print "3. Print course info"

Print "9. Exit"

Read userInput

If userInput is 1

Print "Enter file name:"

Read fileName

courses = loadCourses

If courses is not empty

dataLoaded = true

Print "Course data loaded"

Else

Print "Failed to load data"

If userInput is 2

If dataLoaded is false

Print "Please load course data first"

Else

Call printAllCourses courses

If userInput is 3

If dataLoaded is false

Print "Please load course data first"

Else

While true

Print "Enter course number or 'back' to return:"

Read courseNumber

If courseNumber == "back"

Break

Call searchCourse courses courseNumber

If userInput is 9

Print "Goodbye"

The overall time complexity for loading courses from a file into a vector is O(n + p), where n is the number of courses and p is the total number of prerequisites across all courses. Reading and parsing the file takes O(n) time since each line represents one course. Adding each course to the vector takes constant time per course, so storing all courses also takes O(n) time. While loading, course numbers are saved in a set to help quickly check if each prerequisite is valid. Looking up each prerequisite in the set takes O(1) on average, so checking all prerequisites takes O(p) time.

Searching for a specific course in the vector requires checking each course one by one, which takes O(n) time in the worst case. Once the course is found, printing its title is quick, but printing its prerequisites takes time based on how many there are. If a course has, for example, three prerequisites, it takes time to print all three, making the time O(number of prerequisites).

In summary, vectors are simple and fast for loading and storing course data, but they are slower for searching because they do not support fast lookups. This makes them less ideal when the program needs to find specific courses quickly, especially as the number of courses grows.