CS – 300 Project One

**Vector Data Structure**   
CLASS Course

{

Integer courseID

String courseTitle

Vector<String> prerequisites

}

Vector<Course> courseList

FUNCTION LoadFileData()

{

Declare fData as fstream

Print "Enter the name of the file (include '.txt'): "

Declare fileName as String

Input fileName

IF fileName does NOT contain ".txt"

Print "Invalid file format."

RETURN

ENDIF

fData.open(fileName)

IF fData not found

Print "File not found."

RETURN

ENDIF

String line

WHILE getline(fData, line)

IF line is empty

CONTINUE

ENDIF

Course tempCourse

StringStream ss(line)

Vector<String> tokens

String word

WHILE ss >> word

tokens.push\_back(word)

ENDWHILE

IF tokens.size() >= 2

tempCourse.courseTitle = tokens[0]

tempCourse.courseID = tokens[1]

FOR i = 2 TO tokens.size() - 1

tempCourse.prerequisites.push\_back(tokens[i])

ENDFOR

courseList.push\_back(tempCourse)

ELSE

Print "Invalid line detected, skipping..."

ENDIF

ENDWHILE

fData.close()

}

FUNCTION DisplayAllCourses()

{

FOR each course IN courseList

Print "Course ID: " + course.courseID

Print "Course Title: " + course.courseTitle

Print "Prerequisites: "

FOR each preq IN course.prerequisites

Print " - " + preq

ENDFOR

Print "-----------------------"

ENDFOR

}

FUNCTION SearchCourse()

{

Print "Enter Course ID to search: "

Declare searchID as Integer

Input searchID

Boolean found = false

FOR each course IN courseList

IF course.courseID == searchID

Print "Course Found!"

Print "ID: " + course.courseID

Print "Title: " + course.courseTitle

Print "Prerequisites:"

FOR each preq IN course.prerequisites

Print " - " + preq

ENDFOR

found = true

BREAK

ENDIF

ENDFOR

IF found == false

Print "Course not found."

ENDIF

}  
**Hash Table**  
CLASS Course

{

Integer courseID

String courseTitle

Vector<String> prerequisites

}

HashTable<Integer, Course> courseTable

FUNCTION LoadDataFromFile()

{

Print "Enter file name (.txt): "

Input fileName

IF ".txt" not in fileName

Print "Error: file must end with .txt"

RETURN

ENDIF

fstream file

file.open(fileName)

IF file not found

Print "Unable to open file."

RETURN

ENDIF

String line

WHILE getline(file, line)

IF line empty

CONTINUE

ENDIF

StringStream ss(line)

Vector<String> tokens

String token

WHILE ss >> token

tokens.push\_back(token)

ENDWHILE

IF tokens.size() >= 2

Course courseObj

courseObj.courseTitle = tokens[0]

courseObj.courseID = tokens[1]

FOR i = 2 TO tokens.size() - 1

courseObj.prerequisites.push\_back(tokens[i])

ENDFOR

courseTable.insert(courseObj.courseID, courseObj)

ELSE

Print "Invalid line detected."

ENDIF

ENDWHILE

file.close()

}

FUNCTION FindCourse()

{

Print "Enter Course ID to search: "

Input searchID

IF courseTable.contains(searchID)

Course c = courseTable[searchID]

Print "Course Found!"

Print "ID: " + c.courseID

Print "Title: " + c.courseTitle

Print "Prerequisites:"

FOR each preq IN c.prerequisites

Print " - " + preq

ENDFOR

ELSE

Print "Course not found."

ENDIF

}

Binary Search Tree

CLASS Course

{

Integer courseID

String courseTitle

Vector<String> prerequisites

}

BST<Course> courseTree

FUNCTION ImportCourses()

{

Print "Enter file name (.txt): "

Input fileName

IF ".txt" not in fileName

Print "Error: invalid file type."

RETURN

ENDIF

fstream file

file.open(fileName)

IF file not found

Print "File not found."

RETURN

ENDIF

String line

WHILE getline(file, line)

IF line empty

CONTINUE

ENDIF

StringStream ss(line)

Vector<String> tokens

String token

WHILE ss >> token

tokens.push\_back(token)

ENDWHILE

IF tokens.size() >= 2

Course nodeCourse

nodeCourse.courseTitle = tokens[0]

nodeCourse.courseID = tokens[1]

FOR i = 2 TO tokens.size() - 1

nodeCourse.prerequisites.push\_back(tokens[i])

ENDFOR

courseTree.insert(nodeCourse.courseID, nodeCourse)

ELSE

Print "Skipping invalid record."

ENDIF

ENDWHILE

file.close()

}

FUNCTION SearchCourseTree()

{

Print "Enter Course ID to search: "

Input searchID

Node result = courseTree.find(searchID)

IF result exists

Print "Course Found!"

Print "ID: " + result.courseID

Print "Title: " + result.courseTitle

Print "Prerequisites:"

FOR each preq IN result.prerequisites

Print " - " + preq

ENDFOR

ELSE

Print "Course not found in tree."

ENDIF

}

**Menu System**  
Integer choice = 0

WHILE choice != 9

{

Print "========== COURSE MENU =========="

Print "1. Load Data"

Print "2. Display All Courses"

Print "3. Search Course"

Print "9. Exit"

Print "================================="

Print "Enter your selection: "

Input choice

SWITCH(choice)

{

CASE 1:

LoadFileData()

BREAK

CASE 2:

DisplayAllCourses()

BREAK

CASE 3:

SearchCourse()

BREAK

CASE 9:

Print "Goodbye!"

BREAK

DEFAULT:

Print "Invalid choice, please try again."

BREAK

}

}  
**Vector Structure**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| loop through all courses | 1 | n | n |
| check if oursed equals searchID | 1 | n | n |
| display course information | 2 | 1 | 2 |
| iterate through all prerequisites | 1 | n | n |
| print each prerequisite | 2 | n | 2n |
| **Total Cost** |  |  | **6n + 2** |
| **Runtime** |  |  | **O(n)** |

**Hash Table Structure**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| access courseTable element | 2 | n | 2n |
| check if key (courseID) exists | 1 | n | n |
| display course information | 1 | 1 | 1 |
| loop through prerequisites | 2 | n | 2n |
| print each prerequisite | 3 | n | 3n |
| **Total Cost** |  |  | **8n + 1** |
| **Runtime** |  |  | **O(1)** (average) / **O(n)** (worst) |

**Binary Search Tree Structure**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| traverse nodes in tree | 1 | n | n |
| check if node matches searchID | 1 | n | n |
| output course information | 2 | 1 | 2 |
| loop through prerequisites | 1 | n | n |
| print prerequisite information | 3 | n | 3n |
| **Total Cost** |  |  | **6n + 2** |
| **Runtime** |  |  | **O(log n)** (average) / **O(n)** (worst) |

**Explanation**  
The Vector structure remains the most straightforward and efficient choice for this assignment.  
Its total cost (6n + 2) is slightly lower than the hash and tree structures, meaning fewer operations per run.  
While hash tables and trees can optimize search times under certain conditions, vectors provide simplicity, predictable access patterns, and minimal overhead — making them ideal for small to moderately sized course lists where data lookup is linear and easily maintained.