**Design pseudocode to define how the program opens the file, reads the data from the file, parses each line, and checks for file format errors**

Function validateFileFormat(filename):

try:

Open file with name filename

for each line in file:

Split line by comma to get parameters

if length of parameters < 2:

print "Error: Insufficient parameters in line"

return false

for each parameter in parameters:

if parameter is not a valid courseNumber:

print "Error: Prerequisite", parameter, "not found"

return false

Close file

return true

catch FileNotFound:

print "Error: File not found"

return false

catch IOError:

print "Error: Input/Output error"

return false

**Design pseudocode to show how to create course objects and store them in the appropriate data structure.**

Function readCourseData(filename):

courses = empty Vector of Course objects

try:

Open file with name filename

for each line in file:

Split line by comma to get parameters

courseNumber = parameters[0]

name = parameters[1]

prerequisites = parameters[2:] (if any)

course = new Course(courseNumber, name, prerequisites)

Add course to courses Vector

Close file

return courses

catch FileNotFound:

print "Error: File not found"

return empty Vector

catch IOError:

print "Error: Input/Output error"

return empty Vector

**Design pseudocode that will print out course information and prerequisites.**

Function printCourseInformation(Tree<Course> courses, String courseNumber):

course = courses.search(courseNumber)

if course is not null:

print "Course:", course.courseNumber, "-", course.name

if course has prerequisites:

for each prerequisite in course.prerequisites:

printCourseInformation(courses, prerequisite)

else:

print "Error: Course", courseNumber, "not found"

**Pseudocode for a menu**

Menu Pseudocode:

Function displayMenu():

Display "Menu:"

Display "1. Load Data Structure"

Display "2. Print Course List"

Display "3. Print Course"

Display "4. Exit"

Display "Enter your choice:"

Function loadDataSet(dataStructure, filename):

if dataStructure is Vector:

courses = readCourseData(filename) // Use the function from previous pseudocode

else if dataStructure is Hashtable:

// Implement loading data into Hashtable (pseudocode not provided)

else if dataStructure is Tree:

// Implement loading data into Tree (pseudocode not provided)

return courses

Function printAlphanumericCourseList(dataStructure):

if dataStructure is Vector:

sortedCourses = sortCoursesAlphanumerically(dataStructure) // Use the function from previous pseudocode

for each course in sortedCourses:

Print course information

else if dataStructure is Hashtable:

// Implement printing course list from Hashtable (pseudocode not provided)

else if dataStructure is Tree:

// Implement printing course list from Tree (pseudocode not provided)

Function printCourseInformation(dataStructure, courseNumber):

if dataStructure is Vector:

printCourseInformation(courses, courseNumber) // Use the function from previous pseudocode

else if dataStructure is Hashtable:

// Implement printing course information from Hashtable (pseudocode not provided)

else if dataStructure is Tree:

// Implement printing course information from Tree (pseudocode not provided)

Function main():

coursesDataStructure = null

choice = 0

while choice != 4:

displayMenu()

choice = input() // Assume input from user

switch choice:

case 1:

if coursesDataStructure is null:

filename = input("Enter filename: ") // Assume input from user

coursesDataStructure = chooseDataStructure() // Assume input from user to choose data structure

loadDataSet(coursesDataStructure, filename)

else:

print "Data structure is already loaded."

case 2:

if coursesDataStructure is not null:

printAlphanumericCourseList(coursesDataStructure)

else:

print "Data structure is not loaded yet."

case 3:

if coursesDataStructure is not null:

courseNumber = input("Enter course number: ") // Assume input from user

printCourseInformation(coursesDataStructure, courseNumber)

else:

print "Data structure is not loaded yet."

case 4:

print "Exiting program."

default:

print "Invalid choice."

Function chooseDataStructure():

Display "Choose data structure:"

Display "1. Vector"

Display "2. Hashtable"

Display "3. Tree"

Display "Enter your choice:"

dataStructureChoice = input() // Assume input from user

switch dataStructureChoice:

case 1:

return Vector

case 2:

return Hashtable

case 3:

return Tree

**Pseudocode that will print out the list of the courses in the Computer Science program in alphanumeric order**

Sort Courses Alphanumerically Pseudocode:

Function sortCoursesAlphanumerically(courses):

sortedCourses = empty Vector

for each course in courses:

sortedCourses.add(course)

sortedCourses.sort() // Sort the courses alphanumerically by course number

return sortedCourses

|  | **Vector** | **Hashtable** | **Tree** |
| --- | --- | --- | --- |
| **Advantages** | * Simple implementation * Allows random access to elements | * Fast lookups (average-case O(1)) * Efficient insertion and deletion (average-case O(1)) | * Efficient for searching, insertion, and deletion (average-case O(log n)) * Maintains order of elements |
| **Disadvantages** | * Insertions and deletions can be expensive if not at the end of the vector * Resizing the vector can be costly * Searching can be slow for large vectors | * Memory usage might be inefficient, especially if the load factor is high * Not ordered, so retrieving items in a specific order requires additional work | * Memory overhead due to pointers * Slower than hashtables for lookups in large datasets |

Hashtable offers a balanced combination of efficiency, simplicity, flexibility, and suitability for the task of storing and retrieving course data, making it the recommended choice for this scenario.