**Project One**

Roger Fisher 6/13/2025

**Milestone One Vector Pseudocode**

**Pseudocode for Reading and Validating the File**

**OPEN** the file “course\_data.txt”

**IF** the file fails to open:

**DISPLAY** an error message: “Unable to open file” (example)

**EXIT** the program

**CREATE** empty list called courseLines

**FOR** each line in the file:

**SPLIT** the line into parts using commas

**IF** the line has fewer than two parts:

**DISPLAY** error: “Wrong format. Not enough data”

**CONTINUE** to the next line

**ADD** tokens to courseLines

**FOR** each line in courseLines:

**FOR** each prerequisite in the line after the course title:

**CHECK** if the prerequisite exists in another line’s course number

**IF NOT** found:

**DISPLAY** error: “Prerequisite not found for course number”

**CLOSE** the file

**Pseudocode for Storing Course Data in a Vector**

**CREATE** an empty vector to courseList

**FOR** each line in the file:

**CREATE** new course object

**SET** course.number to the first token

**SET** course.title to the second token

**SET** course.prerequisites to any remaining tokens

**ADD** course to courseList

**Pseudocode for Searching and Displaying a Course**

**PROMPT** the user for a course number

**SET** found to false

**FOR** each course in the courseList:

**IF** the course number matches the user input:

**DISPLAY** the course.number and course.title

**IF** the course.prerequisites is not empty:

DISPLAY each prerequisite

**ELSE**:

**DISPLAY** “No prerequisites”

**SET** found to true

**EXIT** the loop

**IF** no match was found:

**DISPLAY** “Course not found”

**Milestone two Hash Table Pseudocode**

**Opening the File and Validating the Format Pseudocode:**

**OPEN** the file named “course\_data.txt”

**IF** the file does not open:

**DISPLAY** an error message: “Unable to open file” (example)

**EXIT** the program

**CREATE** an empty list called tempCourses

**FOR** each line in the file:

**SPLIT** the line by commas into a list of tokens

**IF** the line has fewer than 2 items:

**DISPLAY** “Invalid line format”

**CONTINUE** to the next line

**SET** the first token as courseNumber

**SET** the second token as courseTitle

**CREATE** empty list prereqs

**FOR** each remaining token in tokens:

**ADD** token to prereqs list

**ADD (**courseNumber, prereqs) to tempCourses list

**FOR** each course in the tempCousre:

**FOR** each prerequisite in that course:

**IF** prerequisite is not found in any courseNumber in tempCourses:

**DISPLAY** “Missing prerequisite: [name] for course [courseNumber]”

**CLOSE** the file

**Creating and Storing Course Objects Pseudocode:**

DEFINE a course object with:

courseNumber as string

courseTitle as string

prerequisites as list of strings

**CREATE** empty hash table called courseTable

**FOR** each entry in tempCourses:

**SET** courseNumber as entry.courseNumber

**SET** courseTitle as entry.courseTitle

**SET** prereqs as entry.prereqs

**CREATE** a new course object

**SET** the course.number to courseNumber

**SET** the course.title to courseTitle

**SET** course.prerequisites to prereqs

**CALCULATE** hash key using course.number

**INSERT** course into the courseTable at hash key

**Printing Course Information and Prerequisite Pseudocode:**

**PROMPT** the user to enter a course number

**READ** user input into searchCourse

**CALCULATE** course from courseTable using the hash key

**IF** course is found:

**DISPLAY** course.number and course.title

**FOR** each prereq in course.prerequisites:

**DISPLAY** prereq

**ELSE**:

**DISPLAY** “No prerequisites”

**ELSE**:

**DISPLAY** “No course found”

**Milestone Three Tree Data Structure Pseudocode**

**Pseudocode for Opening, Reading and Validating the File:**

**OPEN** the file named “course\_data.txt”

**IF** the file doesn’t open:

**DISPLAY** error message “Unable to open file”

**EXIT** the program

**CREATE** empty list called tempCourses

**FOR** each line in the file:

**SPLIT** the line into tokens using commas

**IF** the line has less than 2 parts:

**DISPLAY** error about missing course number or title

**CONTINUE** to the next line

**SET** courseNumber to first token

**SET** courseTitle to second token

**CREATE** empty list prereqs

**FOR** each token after that:

**FOR** each prerequisite in that course:

**IF** prerequisite is not found in any courseNumber or tempCourses:

**DISPLAY** “Missing prerequisites for course”

**CLOSE** the file

**Pseudocode for Creating and Storing Course Objects:**

**DEFINE** a course object with:

courseNumber as string

courseTitle as string

prerequisites as list of strings

**CREATE** an empty binary search tree called courseTree

**FOR** each entry in tempCourses:

**CREATE** new course object

**SET** course.number to entry.courseNumber

**SET** course.title to entry.courseTitle

**SET** course.prerequisite to entry.prereqs

**INSERT** course into courseTree using cousre.number as the key

**Pseudocode for Printing Course Information and Prerequisites:**

**FUNCTION** PrintAllCourses:

**IF** node is not null:

**RETURN**

**CALL** PrintAllCourses(node’s left child)

**DISPLAY** node.course.number and node.course.title

**IF** node.course.prerequisites is not empty:

**DISPLAY** “Prerequisites:”

**FOR** each prereq in node.course.prerequisites:

**DISPLAY** prereq

**ELSE**:

**DISPLAY** “No prerequisites”

**CALL** PrintAllCourses(node’s right child)

**Pseudocode for the Main Menu**

**DISPLAY** menu options:

1. Load data into the selected data structure

2. Print course list in alphanumeric order

3. Print course title and prerequisite

9. Exit the program

**REPEAT** until user selects 9 to exit

**PROMPT** user to enter an option number

**READ** user input into menuOption

**IF** menuOption is 1:

**CALL** function to load data into data structure

**DISPLAY** “Courses loaded successfully” message

**ELSE IF** menuOption is 2:

**CALL** function to print all courses in alphanumeric order

**ELSE IF** menuOption is 3:

**PROMPT** user to enter a course number

**CALL** function to search and display that course and its prerequisites

**ELSE IF** menuOption is 9:

**DISPLAY** “Exiting program”

**EXIT** the loop

**ELSE**:

**DISPLAY** “Invalid option. Please choose a correct number”

**Pseudocode for Printing Courses in Alphanumeric Order**

**FUNCTION** printCoursesAlphanumerically:

**IF** course list is empty:

**DISPLAY** “No courses loaded”

**RETURN**

**CREATE** a temporary list called cortedCourses

**FOR** each course in the course list:

**ADD** course to sortedCourses

**SORT** sortedCourses by course.number in alphanumeric order

**FOR** each course in sortedCourses:

**DISPLAY** course.number and course.title

**Vector Runtime Analysis Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **Number of times Executed** | **Total Cost** |
| Create Empty vector courseList | 1 | 1 | 1 |
| For each line in the file | 1 | n | n |
| Split line by commas | 1 | n | n |
| Create course object | 1 | n | n |
| Set course number and title | 1 | n | n |
| For each remaining token (prerequisites) | 1 | n | n |
| Append each prerequisite to course object | 1 | n | n |
| Add course object to courseList | 1 | n | n |
| Total cost |  |  | 6n +1 |
| Runtime |  |  | O(n) |

**Hash Table Runtime Analysis Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **Number of times Executed** | **Total Cost** |
| Create empty hash table | 1 | 1 | 1 |
| For each entry in tempCourses | 1 | n | n |
| Set courseNumber, courseTitle, prereqs from entry | 1 | n | n |
| Create new course object | 1 | n | n |
| Set course.number, course.title and prerequisites | 1 | n | n |
| Calculate hash key using course number | 1 | n | n |
| Insert course into courseTable at hash key | 1 | n | n |
| Total Cost |  |  | 6n+1 |
| Runtime |  |  | O(n)\* |

\*Although inserting and lookup operations in a hash table take time in the average O(1), it can deteriorate to O(n) in the worst case because of hash collisions. This is because when many items map to one single bucket, it can lead to the chaining of operations in entries.

**Binary Search Tree Runtime Analysis Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **Number of Times Executed** | **Total Cost** |
| For each line in the file | 1 | n | n |
| Split line into tokens | 1 | n | n |
| Create new course object | 1 | n | n |
| Set course.number and course.title | 1 | n | n |
| For each prerequisite in the line | 1 | n | n |
| Add prerequisite to course.prerequisites | 1 | n | n |
| Insert course into binary search tree | Log n | n | n log n |
| Total cost |  |  | 6n + n log n |
| Runtime |  |  | O(n log n) |

**Advantages and Disadvantages**

Both vector, hash table, and binary search tree data structures discussed in this project have certain advantages and disadvantages concerning the performance, memory consumption, and applicability in the management of course-related data. The simplest and most intuitive structure is one of the vectors. It is amenable and works fine when used with small volumes of data. Vectors also offer predictable effects in terms of index because vectors preserve the insertion order, and this enables constant time access. Nevertheless, any search within a vector is linear, and its complexity is O(n), which cannot be efficient when it comes to large data or frequent searches. Also, vectors do not require any uniqueness in them; duplicates should be handled on their own. The hash table is fast, with both insertions and lookups having an average-case complexity of O (1). This is quite useful, especially when one wants to find a particular course number. Hash tables have the unique keys requirement, and this is ideal for course data. Hash tables run as fast as O(1) in the best case, but O(n) in the worst case. They also fail to adhere to any order, which implies that some additional work is required in case one needs to be sorted out. The use of a hash table is more complicated than with vectors as well. Binary search tree (BST) perfectly maintains sorted data, and this is highly convenient when an alphanumerical list is needed. Its operations are O(log n) on average and thus it becomes efficient to search, insert, and delete data. In more extreme cases (e.g. putting sorted data in unbalanced), it may perform as poorly as O(n). Also, BSTs are harder to implement and need precise logic to keep them in a balanced situation to avoid performance degrading. In short, each data structure has its application; vectors are good when simple and small data sets are involved as well as when frequent sorted output is required, hash table is best applied when the quick look up is needed whereas a binary search tree is best when ordered output is the demanded case.

**Recommendation**

According to the analysis, a vector is what I would recommend using in this project. It provides the easiest and quickest deployment of reading, storing and displaying of course data. Although it does not have the search capabilities of a hash table, its stable, easy-to-understand structure, low implementation overhead, and simple sorting lends itself perfectly to this advising utility, as the course list is not very large, and is mostly a display list.