Project One

CS300

Jeremy Reaban

#2315571

April 9th, 2023

Create Course Object Class

Consists of:

String for Course ID,

String for Course Name,

Vector of Strings for Course Prerequisites

Input Course Data File Function

Open File

If Doesn’t Exist, Return Error

Read Line (While It Exists)

Parse Line Into a Number of Fields (values separated by commas)

Check to See it Has At least Two Fields

If Not Return Error

Create Course Object

Check If First Field is Proper Format for Course ID

If Not, Return Error

If Yes, Set Course ID of Course Object to First Field

Check If Second Field is Proper Format for Course Name

If Not, Return Error

If Yes, Set Course Name of Course Object to Second Field

Check If Third Field is Proper Format for Course Prerequisite

If Not, Return Error

If Yes, Set Course Name of Course Object to Second Field

Repeat Above For Remaining Fields

Insert Course Object into chosen Data Structure

Go Back to Top of Loop

Print Sorted Course List Function

Sort Course List by Course ID

Quick Sort Course Data Structure

Set Pivot

Partition Data Structure Into Low and High

Sort Low Partition

Sort High Partition

Iterate Through Course Data Structure

Print Course Information Of Each Course Object

Display Menu Function

Get Vector Of Menu Items to Display from Function Call

Print Menu Header

Loop Through Vector of Menu Items

Print Menu Item

Print Menu Footer

Display Menu

Define Menu Items As Vector of Strings

Call Display Menu Function with Vector of String

Input from Keyboard

Check for Valid Input

If Not Valid, Print Error Message and Display Menu Again

If Input == a, Call Input Course Data Function

If Input == b, Call Sorted Print Course List Function

If Input == c, Input Course ID

Check for Valid Input

If Not Valid, Print Error Message and Ask Again

Call Search Course Function With Course ID

If Input == d

Exit Program

Evaluation / Analysis

Vectors are the easiest of the data structures to implement, but offer the worst performance in terms of searching, and while insertion is quick, it’s only quick if it’s at the end of the vector, although in this case order shouldn’t matter. It also has the disadvantage of require more space depending on how the data is stored, there could be large numbers of empty space.

Hash Tables offer quick search, insertion, and deletion, but requires more complex code and may require re-sizing and re-hashing as more and more items are added to it, potentially slowing it down in a worst case scenario. It can also feature some wasted space.

Binary Search Trees are somewhat difficult to implement but offer good performance in terms of searching, insertion, and deleting. It’s also quicker to iterate over a sorted binary search tree, since the nature of the tree means it’s already somewhat sorted.

For this project, with a fairly limited number of courses, a vector should be sufficient.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Structure | Search Time | Insert Time | Delete Time |
| Vector | O(n) | O(1) | O(1) |
| Hash Table | O(1) | O(1) | O(1) |
| Binary Search Tree | O(log n) | O(log n) | O(log n) |