Dan Story

Project 1  
13APR2025

Pseudocode

Load libraries and headers

**Vector**

Make a struct to hold relevant course data

courseId

courseName

reqNum

reqList

constructor Course()CourseId= ; courseName = “”; reqNum = 0; reqList = “”

Make a new list, courseList, of structure

Get CSV file  
 If no file provided, use default address

call parseText, using CSV file path

call validateList

Get value to search for and store it as courseSearch

call printCourse with courseSearch

end

parseText

Create local tempList

open file in String using parser libraries  
 Loop each row until eof (end of file)

if first and second string are present (seperated by ,)

add first String to struct at courseId

add second String to Struct at courseName

loop until there’s no value in column

if there’s any more strings after courseName

increment reqNum in Struct

concatenate into reqList

return tempList

end

courseSearch

create tempCourse of Course

Loop through list for each course

if String is same as courseId

set tempCourse to Course

return tempCourse

end

printCourse

create tempCourse of Course

set tempCourse equal to courseSearch

output courseId

output courseName

loop using reqNum

output values in reqList

end

validateList

create tempCourse of course

create boolean and set to true

for each course

if boolean is false, break

loop 0 to reqNum

set tempCourse equal to courseSearch

if tempCourse courseID is empty, set boolean to False

return valid

end

Partition()

Get lowest and highest index  
 determine midpoint  
 set pivot equal to midpoint  
 While Lowest index is less than Highest index  
 Loop from lowest until a vector larger than pivot is found  
 overwrite lowest index with that elements position

loop from lowest until an element smaller than pivot is found  
 overwrite highest index with that elements position

swap the elements at highest and lowest index  
 overwrite the lowest index by incrementing it one  
 overwrite the highest by lowering it by one  
 return highest index  
   
   
  
sortList()  
 get vector to sort, lowest and highest indexes  
 if lowest index is greater than or equal to highest

return nothing

call Partition()  
 Recursively call quicksort passing vector, lowest index, and value passed from Partition  
 recursively call quicksort, passing vector, value passed from Partition+1, and highest index  
  
printList()  
 call sortList  
 loop through vector

if courseId [0] = c and courseName [1] = s  
 output courseId

output courseName

loop using reqNum

output values in reqList

**Hash Table**

Make a struct to hold relevant course data

courseId

courseName

reqNum

reqList

constructor Course()CourseId= ; courseName = “”; reqNum = 0; reqList = “”

Make a class HashTable  
 struct bucket

Course

key

next pointer

Define a hash function  
 make a printall function  
 Make a list hashTable

Make a new list named courseList of the struct type CourseMap

Get CSV file  
 If no file provided, use default address

call parseText, using CSV file path

call validateList

Get value to search for and store it as courseSearch

call printCourse with courseSearch

end

parseText

Create local tempList

open file in String using parser libraries  
 Loop each row until eof (end of file)

if first and second string are present (seperated by ,)

add first String to struct at courseId

add second String to Struct at courseName

loop until there’s no value in column

if there’s any more strings after courseName

increment reqNum in Struct

concatenate into reqList

return tempList

end

courseSearch

create tempCourse of type bucket

set tempCourse to bucket at the hash location   
 loop through list  
 if string = courseId

set tempCourse to course

return temp course

printCourse

create tempCourse of type bucket

set tempCourse to bucket at the hash location

loop through chains to check for string

output courseId

output courseName

check and if not 0, loop using reqNum to zero

output values in reqList

end

validateList

create tempCourse of type bucket

create boolean and set to true

for each course

if boolean is false, break

while tempCourse != null

loop 0 to reqNum

set tempCourse equal to courseSearch

if tempCourse courseID is empty, set boolean to False

return valid

end

printList()

make a new node pointer  
 set address of nodes at the beginning  
 loop through list, starting at beginning  
 if courseId [0] = c and courseName [1] = s  
 output courseId

output courseName

loop using reqNum

output values in reqList

int Hash(key)

implement logic to has string courseId, easiest is to courseId modulo tableSize

taking asci values of 4 letters would allow the course types to be hashed together in their own chain.

Return hash

**Binary Search Tree**

Make a struct to hold relevant course data

courseId

courseName

reqNum

reqList

constructor Course()CourseId= ; courseName = “”; reqNum = 0; reqList = “”

-struct Node

Course

Right

Left

Make a class BinarySearchTree

-root = nullptr

+printCourse  
 +BinarySearchTree

Make a new tree, courseTree of Course  
get CSV file

if no file provided, use default address

call parseText, using CSV file path

call validateList

Get value to search for and store it as courseSearch

call printCourse with courseSearch

end

parseText

create Local tempList

open file

loop each row until end of file

if first and second string are present

add first string to struct at courseId

add second string struct at courseName

loop until there is no value in the column

increment precount for each pre-req found in reqnum

concetenate a local string with in reqList

return tempList

end

courseSearch

Create tempCourse Node

start at root

if tempCourse,courseId !=nullptr

if tempCourse.courseId == courseId

return tempCourse

if tempCourse.courseId < search term

point left  
 return searchList

else

point right

return searchList

printCourse

create tempCourse  
 call searchList

if tempCourse == nullptr  
 return course not found

else  
 output courseId

output courseName

check if reqNum is zero, loop to zero

ourput values in reqList  
end

validateList

create tempCourse node  
 create boolean and set to true

for each course

if bool is false break

while tempCourse!= nullptr

loop 0 to reqNum

set tempCourse equal to courseSearch

if tempCourse courseID is empty, set bool to false

return valid

printList

create new Node pointer for root  
 set root to null  
 check if node is null  
 recursively call node’s left pointer to find left most node  
 if courseId [0] = c and courseName [1] = s  
 output courseId

output courseName

loop using reqNum

output values in reqList

recursively call nodes right pointer to find right most node

**Main()**

int choice = 0

loop while choice isn’t 9

output menu options  
 “1)Load file data  
 2)Print an alphanumerically ordered list of all the courses in the Computer Science department.   
 3)Print the course title and the prerequisites for any individual course.   
 9) exit  
 if choice is not 1-3 or 9

output error message

if choice = 1

check for file path  
 if no file path found

use default file path provided

call parseText  
 if choice = 2   
 call validateList

call printList

if choice = 3

call validateList  
 Call courseSearch  
 call printCourse

if choice = 9

output goodbye  
 exit program