Kyle Raja

CS 300

08/03/2022

6-2 Submit Project One

// Class Course

class Course

declare courseNumber;

declare course Name;

declare the vector preReqs;

// declare and define the Search function of type Course

for all courses

if the course is the same as courseNumber

returns the course

else

return

**// Deals with how the file is opened and what the vector will contain**

Declare and define vector FileOpen that contains the name of the file as the parameter

Vector that stores file info is initialized

Declare and define variable of type string for line

Declare the instream variable

Declare the ifstream variable

// if statement that will try to open the file

If can not open file

Throw error message

Put the lines that are stored in the instream variable

Close file

Finally returns the info in the file

// Declare the function that will create the schedule of the course and places them in vector

Declare and Define the CreateSchedule vector and store the vector as the parameter

Declare pointer vector of class Course

Declare and initialize the stringstream variable

Declare and initialize token to have line contents

Declare and initialize the count for token

Use for loop

Make course object of class Course

lineStream stores the info

Take the lineStream token

If statement to check if the count is null

Make the courseNum assigned to the token

Increase count by 1

else if statement to see if the count is at one

assign the courseName to token

Increase count by 1

else if statement to see if the courseNumber is token and is already there

place in the prerequisites to have the token at the end

Increase count by 1

If the count is less than 2

Display error message

Remove contents of the lineStream

Place the course at the end

Ends by returning the courses

**Defining HashTable class**

Defining the Struct Node making if private

Defining the default object Node()

Define the integer key

Define the Node object with two parameters key and the course

Defining Course\* course

Defining the Node\* next

Defining the vector named nodes of type Node

Defining the HashTable() object

Defining the insert object with the course object as the parameter

Declaring and Defining the function prereqNumCourses with two parameters Hashtable with Course Type named courses and string parameter named courseNum

Define the key that hashes with the courseNum

Node set to new node

Use a while loop that loops as long as the node is not null

Start if statement checking if the node pointer courseNum is equal to courseNum

The totalPrerequistes = the size of the node prerequisites

Start For each loop that loops every prerequisite in the totalPrerequisites

Display the prerequisite and the number of totalPrerequisite.

Else statement

The node = next // next is the pointer of node.

Declare and define the printInfoCourse function with two parameters(Hashtable course with

Course as the type, and String courseNum

Define the key that hashes with the courseNum

Node set to new node

Use a while loop that loops as long as the node is not null

Start if statement checking if the node pointer courseNum is equal to courseNum

The totalPrerequistes = the size of the node prerequisites

Start For each loop that loops every prerequisite in the totalPrerequisites

Display the prerequisite and the number of totalPrerequisite.

Else statement

The node = next // next is the pointer of node.

Declare the integer hashNum with one parameter that is key of type integer

Return key

Declare and define the insert function with one parameter course of type Course

Set key integer to the hasNum with coursed

Start if statement see if node = ‘ ‘

Nested If statement to see if node = ‘ ‘

Then node includes the current course

Nested if statement to see if node = ‘ ‘

Start while loop that loops till node = ‘ ‘

Then the course is included into the node

Declare and define the parseStatement vector that references one parameter line. Vector is type string

Start if statement that checks if the sizeOfLine = 2

Declare new object secondCourse

courseName = paramLine[0]

courseNum = paramLine[1]

make the coursePrereq vector empty

finally we return secondCourse

Else statement

Declare and define vector called cachePrereq

Start for loop

take cacheprerq and add paramLine[i] to end of it.

Declare another secondCourse object

courseName = paramLine[0]

courseNum = paramLine[1]

course prereq = cacheprerq

finally we return secondCourse

Main function of type integer

Set the pointer HashTable\* table = a new HashTable

Declare and define vector called tempVector

Declare string variable fileLine

Use Ifstream infile with one parameter that is the name of the file

Use While loop that will keep getting the information in the file until done Use Stringstream

Use nested while loop that checks Stringstream object

Declare substring variable

Get the line of Stringstream using substring variable

Use tempVector and place substring variable at the end.

Use table and the insert function to insert the parsed line and tempVector

Empty the tempVector

**//Binary Tree code**

Starting by defining the class which will be called BinaryFindTree

Declaring the private members of the class

Object root which is of type Node

addANode using the node and course objects of type void

printOrder using the node object of type void

Declaring the public members of the class

The default constructor which will be called BinaryFindTree

Declaring the destructor which is also called BinaryFindTree

Declaring in the printOrder method of type void

Declaring the method called InsertValue that uses the course object of type void

Declaring the method called SearchValue that uses the courseNum of type int

Defining the BinaryFindTree

Defining the method called OrderNode

Defining the InsertValue method

Starting if statement to check if the node = null

Assign the root = the new Node

Assign the leftside & the rightside\* = null

Else statement

Use the addANode method which uses the node and course as arguments

Defining the BinaryFindTree with the addANode method that uses the node and course as arguments

Declaring the nowCourseNum variable to the node course id

Declaring the addCourseNumNode variable to the course id

Starting if statement to see if the addCourseNumNode < nowCourseNum

Next if statement to check if the leftSide\* = null

Assign leftSide\* = the new node

Assign leftSide\* and rightSide\* = null

Else

Invoke the addANode method that passes the leftSide\* and the course

Else Starting if statement to check if the rightSide\* = null

Assign the rightSide\* = node that is passed in course

Assign the rightSide\* and leftSide\* = null

Else

Invoke the addANode that will pass the rightSide\* node and course

Defining the SearchValue method of type BinaryFindTree that takes the courseNum as a string variable

Declaring the currentNode of type Node

Starting while loop that loops to check if the currentNode is not null

Declaring the addCourseNumNode using the course id

Declaring the nowCourseNum using the course id

Starting if statement that checks if addCourseNumNode = nowCourseNum

Returns the now course num

Starting if else statement to check is addCoursNumNode < nowCourseNum

Assigns the nowCourseNum = leftSide\*

Starting else statement

Assigns the nowCourseNum = rightSide\*

Declaring the course object and then returning

Defining the printOrder method

Starting if statement to check if the node != null

Pass leftSide\*

Display node information

Invoke the printOrder function using the rightSide\*

// Displaying the vector information

void printCourseInformation(Vector<Course> courses, String courseNumber)

{

for all courses

if the course is the same as courseNumber

print out the course information

for each prerequisite of the course

print the prerequisite course information

}

//

void printSampleSchedule(Hashtable<Course> courses)

{

Use for each loop for key that matches the course

Display the key and course

Use if statement to see if that key has prereqs

Use for each loop for all prerequisite

Display the prerequisite

}

void printCourseInformation(Tree<Course> courses, String courseNumber)

{

Use the for loop to go through nodes

Use if statement to see if the course matches #

Display the information of the node

Use if statement to see if is left

Display the node (using the class info)

Use if statement to see if is right

Display the node (using the class info)

Break;

Use else statement

Use if statement to see if the node is the left

Left node

Use if statement to see if node is the right

Right node

}

**// Menu code to show list of choices**

Start by assigning the variable userChoice to 0

The while loop starts, stays in loop as long as userChoice does not choose to end program

Show the menu choices from 1-4

Start the switch statement using userChoice as the parameter

The first case shows “loadCourses”

The second case shows “printSorted”

The third case shows “printCourseInformation”

The fourth case shows “Close the program”

The loop ends if user choses fourth cases

**// The Print Sorted List code using vector**

First creating the function of printSorted with the parameter of courses

Creating the divideCourses function of type int with the vector, begin, and end parameters

Assign the first and last elements to variables

Assign the middle element

Assign the pivot to the middle element

Starting while loop that checks if the top index, sees if the pivot < top index

Implement the swap method to deal with the low and top index using a temp variable.

Next creating a quicksort function that uses the course, begin, and end parameters

Assign the low, top, and middle indexes

Starting the if statement to make sure the low is not greater than or equal to the top.

Assignin the low index to the divideCourses function

Using recursion that goes through the method again.

After that we create the display method to show the courses

Starting for loop that displays each course

Finally we create the inOrder function

Starting the if statement to check the node is not null

Calls the inOrder function using node and point to left

Display course info

Calls inOrder function is node right then points to right

Display course info

**Runtime analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| The Vector Section | Line Cost | Times Executes | Total Cost |
| Start by making the vector | 1 | 1 | 1 |
| for loop to manage the file | 1 | n | n |
| Another creation of a vector | 1 | n | n |
| for loop to manage the prerequisites | 1 | n | n |
| placing the prerequisities where they need to go in the vector | 4 | n | n |
| Also managing other course info such as name and # | 2 | n | n |
|  |  | Total | 6n+1 |
|  |  | R | O(n) |
|  |  |  |  |
| The HashTable section | Line Cost | Times Executes | Total Cost |
| Basically Declaring and Defining the hashtable | 1 | 1 | 1 |
| Then dealing with the insert function | 1 | n | n |
| Next we add the course variable of a key inside that method | 1 | n | n |
| dealing with nodes and making sure we are locating the keys | 1 | n | n |
| looping through nodes and assigning | 3 | n | n |
| using while loop to go through prerequisites | 3 | n | n |
| adding the item of the course to the vector | 1 | n | n |
|  |  | Total | 12n+1 |
|  |  | R | O(n) |
|  |  |  |  |
| The Tree Section | Line Cost | Times Executes | Total Cost |
| Starting by using the if statement to compare the nodes | 1 | 1 | 1 |
| Checking if the node is right or left | 1 | n | n |
| assigning the node | 4 | n | n |
| Using a search method | 2 | n | n |
| checking the nodes of courses | 1 | n | n |
| adding the nodes | 1 | n | n |
|  |  | Total | 10n+1 |
|  |  | R | O(n) |

**Summary of the pros and cons**

From this project, I saw that there were many pros and cons from the various ways to organize and manipulate the data. I have always preferred using vectors over arrays because I find them to be better structured, less room for error. Using vectors also help improve performance time for example, reading files and applying the data to the objects that we have for our courses. I found that manipulating data in vectors to be particularly excellent. Adding and removing elements throughout the vector. Looking for elements in a vector is also quite quick. I noticed that the runtime between the vector, hashtable, and tree, vector had the fastest runtime.

Then we move on to Hash tables, have a challenging topic to understand but have started to make more sense. Hash tables deal with mainly keys that are assigned to each element in the table. These keys are very important in order to find what value within the table one is trying to work with. In the table, I mainly dealt with the course and was able to locate it via the key. We can find, add, remove courses. The problems with hash tables I found was that I was not able to easily organize the table without a fair amount of effort on my part.

The binary trees I particularly liked. It was quite effective to work with nodes and courses.

I found the performance to not be as fast as the vector but it was effective to search courses and also work with nodes, assigning left and right. I found it naturally was easier to understand than hash tables. After working with all three of these, I have to say I still value the vector the most because of its performance and simplicity. You do not need to know too much more than that. However, I think that in professional programming, hash tables and binary trees must be all over the place so its good for one to familiarize it.