Edwin Jones

CS300 3-3: Project One Milestone Two

constant unsigned int DEFAULT\_SIZE = desired table size

// Defaults to public members

struct CourseInfo {

string code

string name

vector<string> prerequisites

}

//Structure of each course in courses given respective parameters

class HashTable {

private

struct Node {

CourseInfo course

Unsigned int key

Node type point next

Node() {

key = max possible value

next = nullptr

}

Node(CourseInfo course) : Node() {

this course = course

}

Node(CourseInfo course, unsigned int key) : Node(course) {

this key = key

}

}

vector<Node> courses

unsigned int tableSize = DEFAULT\_SIZE

unsigned int hash(int key)

public

HashTable() {

resize courses to tableSize

}

HashTable(unsigned int size) {

set tableSize = size

resize courses to tableSize

}

~HashTable() { //delete everything in the table when class is destroyed

traverse courses and at each course at index i

create Node pointer current of type Node = the next node

while current != nullptr

create a new pointer toDelete = current

set current = next course after itself

delete toDelete

}

hash(int key) { //use input parameter as the hash key

return the resulting hash value

}

Insert(CourseInfo course) { // Assign course to a bucket and integrate into linkedlist

set unsigned int currentKey = hash(the course info to hash)

if the course at currentKey has key = max possible value

set course at currentKey = Node(course, currentKey)

else

create Node pointer nextCourse = content of course at currentKey

while the course after nextCourse != nullptr

set nextNode = the course after itself

set the course after nextNode = a new pointer for Node(course, currentKey)

}

void validateCourses(){ // Verify that each unique course prerequisite is listed as a course

vector<string> tempPrereqs

vector<string> tempCourses

//The courses to compare prequisites against

for each Node course in courses

add course.code to the end of the tempCourses vector

//check each course for their prerequisites

for each Node course in courses

//check if the prerequisite course is listed as course

for each string prereq in course.prerequisites

if prereq is in tempCourses

continue //check next course

else

throw an error //course not found, invalid

// delete the temp vectors from memory since no errors

vector<string>().swap(tempPrereqs)

vector<string>().swap(tempCourses)

}

void PrintSortedData(HashTable \*hashTable) { //Iterate through all the courses and print the information

set int key = 0

vector<CourseInfo> courseInfo

while key < tabelSize

create point currentNode type Node=address pointed to by hashTable of nodes at key

while currentNode != nullptr

if key pointed at by currentNode = max possible value

break

add course pointed to by currentNode to end of courseInfo

set currentNode = the next node

key++

use built in sort function to sort peopleInfo

for each course type CourseInfo in peopleInfo

print course info

}

void searchCourse(string courseCode) { // iterate through matching bucket for matching code

create an empty CourseInfo course object

set unsigned int currentKey = hash(the course info to hash)

if the code of course at currentKey = courseCode

return the course's info

else

create Node pointer nextCourse = next course after course at currentKey

create Node point prevCourse = course at currentKey

while next != nullptr

if the course's code at nextCourse = courseCode

return nextCourse bid object

set prevCourse = nextCourse

set nextCourse = course after nextCourse

return course //empty values

}

}

void loadCSVToHashTable(string csvFile, HashTable \*hashTable) { //Read the CSV file and format content for the hash table

ifstream file(csvFile)

if file is not open

throw an error

string line

while getline(file, line)

if the line is empty

continue

if the line doesn't end with a comma

add a comma to the end

vector<string> tokens

stringstream ss(line)

string token

while getline(ss, token, ',')

add token to end of tokens vector

if the tokens vector size is < 2

throw an error

create CourseInfo course instance

set course.code = token at index 0 of tokens vector

set course.name = token at index 1 of tokens vector

for i=2, i++ while i is < tokens vector size

if tokens vector at i is not empty

add token at index i to end of course.prerequisites

call the Insert(course) method pointed to by hashTable

close file

}

main { //Follow lines achieve basic functionality

//Process, store, then validate CSV file content

assign variable with CSV file

create pointer hashTable with type HashTable

create an empty CourseInfo course object

set the hash table = a new HashTable() instance

set int userChoice = 0

while userChoice != 9

print menu

set userChoice = to user input

switch(userChoice)

case 1:

call the loadCSVToHashTable(csvFile, hashTable) method

case 2:

call the validateCourses(hashTable) method pointed to by hashTable

case 3:

call the PrintSortedData(hashTable) method pointed to by hashTable

case 4:

set searchCode = the user input

call the searchCourse(searchCode) method pointed to by hashTable

set course = the info returned by searchCourse

if results found

print the course's info using course

else

print message that nothing was found

return 0

}