Edwin Jones

CS300 3-3: Project One Milestone Three

//define structures

struct CourseInfo {

string code

string name

vector<string> prerequisites

}

struct Node {

CourseInfo course

Node \*left

Node \*right

Node(){

left = nullptr

right = nullptr

}

Node(CourseInfo aCourse) inherit Node() {

course = aCourse

}

}

// define class with methods to create and add nodes to binary search tree

class BST {

**public:**

BST() { // default constructor

root = nullptr

}

virtual ~BST() { //destructor

while root != nullptr

create pointer current type Node = root

while not a leaf node

if current->left != nullptr

set current = current-> left

continue

if current ->right != nullptr

set current = current->right

delete current

}

void ValidateCourses() {

pass root through validateCourses()

}

void InOrder() {

pass root through inOrder

}

void Insert(CourseInfo course) {

if root = nullptr

set root = pass course through a new Node

else

pass root and course through addNode()

}

CourseInfo Search(string code) {

create pointer current type Node = root

while current != nullptr

if current code = code

return current code

else if current code > code

set current = current->left

else

current = current->right

create empty course

return course

}

**private:**

Node\* root;

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

if node != nullptr

pass node->left through validateCourses()

for each prereq in course.prerequisites pointed to by node

if Search(prereq).code is empty

throw an error

pass node->right through validateData()

}

//addNode compares string values (e.g. A < B, B < C, "1" (49 in ascii) < "2" (50 in ascii)).

//assumes all letters in each course code is either uppercase or lowercase

void addNode(Node\* node, CourseInfo course) {

create pointer current type Node = node

while current != nullptr

if current code > course's code

if current->left = nullptr

set current->left = course passed through a new Node()

return

else

current = current->left

else

if current->right = nullptr

set current->right = course passed through a new Node()

return

else

set current = current->right

}

void inOrder(Node \*node) {

if node != nullptr

pass node->left through inOrder()

print node's course information

pass node->right through inOrder()

}

void loadCSVToBST(string csvFile, BST \*bst) { //format CSV file for BST

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 bst

close file

}

int main {

//how to load csv file into binary search tree

set csvFile to the CSV file

create pointer bst type BST = new BST object

create empty course with type CourseInfo

set int userChoice = 0

while userChoice != 9

print menu

switch(userChoice)

case 1:

call the loadCSVToBST(csvFile, bst) method

case 2:

call the ValidateDate() method pointed to by bst

case 3:

call the InOrder() method pointed to by bst

case 4:

set searchCode = the user input

call the Search(searchCode) method pointed to by bst

set course = the info returned by Search()

if results found

print the course's info using course

else

print message that nothing was found

return 0

}