Pseudocode Document

Fred Wahab

CS-300

//initialize global objects and variables

struct Course {

string courseNumber;

string courseName;

vector<string> coursePrereqs;

};

vector<Course> course;

**VECTOR**

**//LOAD VECTOR**

void LoadVector() {

//local variables

string fileName;

//get filename

output: “Please enter name of file: “;

input: fileName;

//open file

ifstream infile;

infile.open(fileName);

//insert data into courses

if infile is open {

string line;

//read file line by line

while there is a line infile {

vector <string> tokens;

string token;

stringstream ss(line);

//tokenize every string at the comma

while reading the string, seperate at ‘,’ {

push each token to tokens

}

//ensure there are at least two parameters on each line

if size of tokens is less than 2 {

break;

}

//create a new course and add data from tokens

Course newCourse equals Course();

newCourse.courseNumber equals tokens[0];

newCourse.courseName equals tokens[1];

//if prereqs exist in tokens, add to course

if size of tokens is greater than 2 {

for every token[i] after the first 2 {

//ensure any prerequisite that is provided on a line exists

//as a course in the file

for every course[j] in courses {

if courses[j].courseNumber equals token[i] {

push token to newCourse.coursePrereqs

}

}

}

}

push newCourse to courses

}

output: “Data Structure successfully loaded!”;

}

//if the fileName does not exist

else {

output: “Error: Filename not found.”;

}

infile.close();

}

**//PRINT COURSE LIST**

void PrintVector() {

//validate Load Data Structure was successful.

if size of courses is greater than 0 {

//sort courses alphanumerically

for every course[i] in courses {

for every course[j] before course[i] {

if courses[j] is greater than courses[i] {

swap course indexes

}

}

}

//loop courses to print courseNumber and courseName

for every course[i] in courses {

output: courses[i].courseNumber “,”;

output: courses[i].courseName;

}

}

//if data structure hasn’t been loaded

else {

output: “Error: Load Data Structure first.”;

}

}

**//PRINT COURSE**

void PrintVectorNode() {

//validate Load Data Structure was successful.

if courses size is greater than 0 {

string courseInput;

int courseFound equals 0;

output: “What course do you want to know about? “;

input: courseInput;

//loop through courses to find courseInput match

for every course[i] in courses {

if courseNumber[i] equals courseInput {

output: courses[i].courseNumber “, ”;

output: courses[i].courseName;

output: “Prerequisites: “;

if courses[i].coursePrereqs(except last) has data {

for every course[j] in courses[i].coursePrereqs {

output: courses[i].coursePrereqs[j] “,”;

}

//eliminate comma from last entry

output: courses[i].coursePrereqs[size of coursePrereqs - 1];

//flag for course was found

courseFound equals 1;

}

//print none if no prereqs

else {

output: “None”;

}

}

}

//course input validation

if courseFound equals 0 {

output: “Error: Invalid Course”;

}

}

//if data structure hasn’t been loaded

else {

output: “Error: Load Data Structure first.”;

}

}

**HASHTABLE**

const int DEF\_SIZE equals 10;

class HashTable {

public:

//initialize class objects and variables

struct Node {

Course course;

int key;

Node\* next;

}

int tableSize equals DEF\_SIZE;

int hash(int key);

vector<Node> nodes;

//initialize constructor

Node() {

key equals UINT\_MAX;

next equals nullptr;

}

//initialize Node with a Course

Node(Course newCourse) : Node() {

course equals newCourse;

}

//initialize Node with a Course and a Key

Node(Course newCourse, int newKey) : Node(newCourse) {

key equals newKey;

}

};

**//LOAD HASH TABLE**

void LoadHashTable() {

//local variables

string fileName;

//get filename

output: “Please enter name of file: “;

input: fileName;

//open file

ifstream infile;

infile.open(fileName);

//insert data into hash table

if infile is open {

string line;

//calculate table size

while there is a line infile {

incriment tableSize by 1;

}

resize hashTable to updated table size

//read file line by line

while there is a line infile {

vector <string> tokens;

string token;

stringstream ss(line);

//tokenize every string at the comma

while reading the string, seperate at ‘,’ {

push each token to tokens

}

//ensure there are at least two parameters on each line

if size of tokens is less than 2 {

break;

}

//create a newCourse and add data from tokens

course equals Course.newCourse;

HashTable:key equals tokens[0];

newCourse.courseName equals tokens[1];

//if prereqs exist in tokens, add to course

if size of tokens is greater than 2 {

for every token[i] after the first 2 {

//ensure any prerequisite that is provided on a line exists

//as a course in the file

for every course[j] in courses {

if HashTable.key equals token[i] {

add token to newCourse.coursePrereqs

}

}

}

}

add newCourse to HashTable

}

}

//if the fileName does not exist

else {

output: “Error: Filename not found.” << endl;

}

infile.close();

};

**//PRINT Hash Table**

void PrintHashTable() {

//validate Load Data Structure was successful.

if size of HashTable is greater than 0 {

//sort nodes alphanumerically

for every key[i] in HashTable {

for every key[j] before course[i] {

if key[j] is greater than key[i] {

swap nodes

}

}

}

//loop hash table to print courseNumber and courseName

for node in HashTable {

output: node.key “,”;

output: node.value[0];

}

}

//if data structure hasn’t been loaded

else {

output: “Error: Load Hash Table first.”;

}

}

**//PRINT Hash Table Node**

void PrintHTNode() {

//validate Load Data Structure was successful.

if courses size is greater than 0 {

string courseInput;

int courseFound equals 0;

output: “What course do you want to know about? “;

output: courseInput;

output: endl;

//loop through hash table to find courseInput match

for every node[i] in HashTable {

if node[i]->key equals courseInput {

output: node[i]->key “, ”;

output: node[i]->courseName;

output: “Prerequisites: “;

if node[i]->coursePrereqs(except last) has data {

for every node[j] in HashTable[i]->coursePrereqs {

output:node[i]->coursePrereqs[j] “,”;

}

//eliminate comma from last entry

output: last element of node;

//flag for course was found

courseFound equals 1;

}

//print none if no prereqs

else {

output: “None”;

}

}

}

//course input validation

if courseFound equals 0 {

output: “Error: Invalid Course”;

}

}

//if hash table hasn’t been loaded

else {

output: “Error: Load Hash Table first.”;

}

}

**TREE**

//initialize objects and variables

struct TreeNode {

Course course;

TreeNode\* left;

TreeNode\* right;

int data;

TreeNode(int data) {

this->data equals data;

left equals right equals nullptr;

}

//create key for course

int key;

TreeNode\* node;

//if key entry found

if node has data and key does not equal UINT\_MAX {

return node->course

//if key entry is not found

if node has no data or key equals UINT\_MAX {

return course

}

//loop for matching node

while node has data {

if node->key does not equal UINT\_MAX and key equals 0 {

return node->course;

}

node equals node->next;

}

return course;

}:

//preorder traversal

void preorderTrav(struct TreeNode\* node) {

if node is empty {

return;

}

preorderTrav(node->left);

preorderTrav(nod->right

}

//postorder traversal

void postorderTrav(struct TreeNode\* node) {

if node is empty {

return;

}

postorderTrav(node->left);

postorderTrav(node\_>right);

}

//inorder traversal

void inorderTrav(struct TreeNode\* node) {

if node is empty {

return;

}

inorderTrav(node->right);

**//LOAD Tree**

void LoadTree() {

//local variables

string fileName;

//get filename

output: “Please enter name of file: “;

input: fileName;

//csv Parse file

csv::Parser infile equals csv::Parser(fileName);

//create tree and add to collection

try {

for every row infile {

Course course;

course.courseNumber equals infile[i][0];

course.courseName equals infile[i][1];

if size of infile is greater than 2 {

for every index[j] infile after 1 {

course.courePrereqs equals infile[i]index[j];

}

}

} catch(csv::Error& e) {

e.what();

}

}

}

**//PRINT Tree**

void PrintTree() {

struct TreeNode\* root equals new Node();

root->left equals new TreeNode(node);

root->right equals new TreeNode(node);

root->left->left equals new TreeNode(node);

root->left->right equals new TreeNode(node);

//validate Load Data Structure was successful.

if size of Tree is greater than 0 {

//sort nodes alphanumerically

for every key[i] in Tree {

for every key[j] before course[i] {

if key[j] is greater than key[i] {

swap nodes

}

}

}

//loop tree to print courseNumber and courseName

for node in Tree {

output: node.key “,”;

output: node.value[0];

}

}

//if data structure hasn’t been loaded

else {

output: “Error: Load Tree first.”;

}

}

**//PRINT Tree Node**

void PrintTreeNode() {

//validate Load Data Structure was successful.

if courses size is greater than 0 {

string courseInput;

int courseFound equals 0;

output: “What course do you want to know about? “;

output: courseInput;

output: endl;

//loop through tree to find courseInput match

for every node[i] in Tree {

if node[i]->key equals courseInput {

output: node[i]->key “, ”;

output: node[i]->courseName;

output: “Prerequisites: “;

if node[i]->coursePrereqs(except last) has data {

for every node[j] in HashTable[i]->coursePrereqs {

output:node[i]->coursePrereqs[j] “,”;

}

//eliminate comma from last entry

output: last element of node;

//flag for course was found

courseFound equals 1;

}

//print none if no prereqs

else {

output: “None”;

}

}

}

//course input validation

if courseFound equals 0 {

output: “Error: Invalid Course”;

}

}

//if tree hasn’t been loaded

else {

output: “Error: Load Tree first.”;

}

}