Madison Marbach

SNHU CS 300

Project One

**Reading file:**

INITATE FSTREAM

Create method void loadCourses(string csvPath, dataStructre)

Make call to open file, if the return value is “-1”, file is not found

Else file is found

While it is not EOF

Read each line

IF There are less than two values in a line, return ERROR

ELSE read parameters

IF there is a third or more parameter

IF third or more parameter is in first parameter elsewhere,

continue

ELSE return Error

Close file

**Hold course info:**

Create struct Course{}

Create Identifiers: Course num, Course Name, Prerequisite

**//Vector**

vector<Course> loadCourses(string csvPath)

for (int i = 0; i < file.rowCount(); i++) {

Create a data structure and add to the collection of courses

Course course;

course.coursenum = file[i][1];

course.name = file[i][0];

while not end of line

course.prereq. = file[i][8];

courses.push\_back(course);

void printCourseInformation(Vector<Course> courses, String coursenum) {

for all courses

if the course is the same as coursenum

print out the course information

for each prerequisite of the course

print the prerequisite course information

**//HashTable**

Create Hashtable

Create Node struct

Course course

Unsigned int key

Vector<Node> nodes

Define tableSize

Unsigned int has(int key)

Create Insert method to insert items to HashTable

create the key for the given course, search for node with the key value

if no entry found for the key

assign this node to the key position

else if node is used

assign old node key to UINT\_MAX, set to key, set old node to course and old

node next to

null pointer

else find the next open node

add new newNode to end

void loadCourses(string csvPath, HashTable\* hashTable)

loop to read rows of a CSV file

for (unsigned int i = 0; i < file.rowCount(); i++) {

Create a data structure and add to the collection of courses

Course course;

course.coursenum = file[i][1];

course.name = file[i][0];

while not end of line

course.prereq. = file[i][8];

hashTable->Insert(course);

Create method void printCourseInformation(Hashtable<Course> courses, String coursenum)

Get input for coursenum

Assign key = coursenum

Assign node to the node.at(key)

if current node matches key

Return course, displayCourse(nodes[key].course)

If node points to null, return null

Else while the node is not Null, check against the key

If the key matches the couseId, Return course, displayCourse(nodes[key].course)

Point to next node

**//Tree**

Define a binary search tree to hold all courses

BinarySearchTree\* bst;

bst = new BinarySearchTree();

Course course;

Create add node method void BinarySearchTree::addNode(Node\* node, Course course)

If root is null, add root

if node is less than root then add to left

if no left node

this node becomes left

if node is greater than root add right

if no right node

this node becomes right

void loadCourses(string csvPath, BinarySearchTree\* bst)

loop to read rows of a CSV file

for (unsigned int i = 0; i < file.rowCount(); i++) {

Create a data structure and add to the collection of courses

Course course;

course.coursenum = file[i][1];

course.name = file[i][0];

while not end of line

course.prereq. = file[i][8];

bst->Insert(course);

Create method void printCourseInformation(Tree<Course> courses, String coursenum)

Get input for coursenum

Assign current node to root

While current is not NULL

If course.coursenum matches current

Return current, output course.coursenum << output course.name

while (prereq = true)

out put course.prereq

If coursenum is less than root

Set current to left

Else set current to right

**2. Pseudocode for menu:**

Menu:

print "Menu:"

print "1. Load Data Structure"

print "2. Print Course List"

print "3. Print Course"

print "4. Exit"

choice = input("Enter your choice:")

if choice == 1:

loadCoursesFromFile(filename)

else if choice == 2:

printCourseList()

else if choice == 3:

courseNumber = input("Enter the course number:")

printCourseInformation(courseNumber)

else if choice == 4:

exit()

**3. Pseudocode to print in alphanumeric order**

Start at head node

While current is not null

If node greater than current, move left

If node less than current, move right

Continue until no more move right

Print list

**4. ADVANTAGES AND DISADVANTAGES:**

VECTOR

Advantages: Simple, keeps order of insertion, fast access

Disadvantages: slow searching

HASHTABLE:

Advantages: fast instert and retrieve, good for large datasets

Disadvantages: No order, memory

TREE:

Advantages: Sorted order, more efficient search and retrieval

Disadvantages: Complex implementation, slower insertion and deletion

Recommendation:

Although the vector data structure preserves the order and the tree data structure provides a sorted order, I would recommend the hashtable data structure. Because of the hashtable data structure having fast insertion and retrieval I believe it would be the most suitable. The fast insertion and retrieval will be efficient for printing course information and prerequisites. The hashtable allows direct access information based on course numbers which allows better efficiency when searching for individual course lookups.