# CS 300 Pseudocode Document

## Function Signatures

Below are the function signatures that you can fill in to address each of the three program requirements using each of the data structures. The pseudocode for printing course information, if a vector is the data structure, is also given to you below (depicted in bold).

READ FILE

vector<string> readFile(string CourseList.txt) {

initialize vector<string> lines

initialize string line

initialize ifstream fileStream

open fileStream with CourseList.txt

if unable to open fileStream:

output "Cannot open file."

return empty vector

while pull line from fileStream until end of file:

push line to back of the lines

close fileStream

return lines

}

HOLD COURSE INFORMATION

Create struct Course{}

Create Identifiers: Course ID, 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.courseId = file[i][1];

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

while not end of line

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

courses.push\_back(course);}

// Vector pseudocode

int numPrerequisiteCourses(Vector<Course> courses, Course c) {

totalPrerequisites = prerequisites of course c

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

void printSampleSchedule(Vector<Course> courses) {

for each course c in courses

print(c.courseNumber + " - " + c.name)

}

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**

}

// Hashtable pseudocode

int numPrerequisiteCourses(Hashtable<Course> courses) {

int count = 0

for each prerequisite p in c.prerequisites

count += numPrerequisiteCourses(courses, courses.get(p))

return count + c.prerequisites.length

}

void printSampleSchedule(Hashtable<Course> courses) {

for each course c in courses

print(c.courseNumber + " - " + c.name)

}

void printCourseInformation(Hashtable<Course> courses, String courseNumber) {

Course c = courses.get(courseNumber)

if c != null

print(c.courseNumber + " - " + c.name)

if c.prerequisites.length > 0

print("Prerequisites:")

for each prerequisite p in c.prerequisites

print(p + " - " + courses.get(p).name)

}

// Tree pseudocode

int numPrerequisiteCourses(Tree<Course> courses) {

int numPrerequisiteCourses(Tree<Course> courses, Course c) {

int count = 0

for each prerequisite p in c.prerequisites

count += numPrerequisiteCourses(courses, courses.get(p))

return count + c.prerequisites.length

}

void printSampleSchedule(Tree<Course> courses) {

for each course c in courses.inorderTraversal()

print(c.courseNumber + " - " + c.name)

}

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

Course c = courses.get(courseNumber)

if c != null

print(c.courseNumber + " - " + c.name)

if c.prerequisites.length > 0

print("Prerequisites:")

for each prerequisite p in c.prerequisites

print(p + " - " + courses.get(p).name)

}

2. CREATE PSEUDOCODE FOR A MENU

Set user choice to a 0

Create a while loop for menu

While user choice is not 4

Output menu choices 1. Load Course File, 2. Print Course List, 3. Print Individual Course, and 4. Exit

Selection 1: load courses of the data structure chosen by user

Selection 2: print the sorted class list

Selection 3: print the course information

Selection 4: End the program

3. DESIGN PSEUDOCODE THAT WILL PRINT OUT THE LIST OF THE COURSES IN THE COMPUTER SCIENCE PROGRAM IN ALPHANUMERIC ORDER

//VECTOR

sortCourses():

sort the courses in an alphanumeric order based on the courseNum

printCourseList():

sortCourses()

For each course that is in the vector

Print that course information

//HASHTABLE

printCourseList():

For each of the keys located in the hashtable, sorted in alphanumeric order

Retrieve course from the hashtable with the key

Print that course information

//TREE

printCourseList():

inOrderTraversal(root)

inOrderTraversal(node):

if not does not equal to null:

inOrderTraversal(node.left)

Print node.course information

inOrderTraversal(node.right)

## Example Runtime Analysis

When you are ready to begin analyzing the runtime for the data structures that you have created pseudocode for, use the chart below to support your work. This example is for printing course information when using the vector data structure. As a reminder, this is the same pairing that was bolded in the pseudocode from the first part of this document.

| **Vector** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 1 | n | n |
| **if the course is the same as courseNumber** | 1 | n | n |
| **print out the course information** | 2 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 2 | n | n |
| **Total Cost** | | | 6n + 1 |
| **Runtime** | | | O(n) |

| **HashTable** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 2 | n | n |
| **if the course is the same as courseNumber** | 1 | n | n |
| **print out the course information** | 1 | 1 | 1 |
| **for each prerequisite of the course** | 2 | n | n |
| **print the prerequisite course information** | 4 | n | n |
| **Total Cost** | | | 9n + 1 |
| **Runtime** | | | O(n) |

| **Tree** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 1 | n | n |
| **if the course is the same as courseNumber** | 1 | n | n |
| **print out the course information** | 2 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 4 | n | n |
| **Total Cost** | | | 8n + 1 |
| **Runtime** | | | O(n) |

All three data structures that were evaluated have their own advantages and disadvantages for the project. The vector data structure has the advantage of being a faster method for adding the course objects and reading the file. However, a disadvantage of the vector data structure is that it has to search the entire list for the specific course, until the match is found the program will have to check each item. This method is also the most straightforward method.

Hash tables are able to search a list very fast. It can create keys so that the locations can be easily printed and searched. However, this is a slower implementation and has to create the list and place each course in a certain spot in memory. We are unable to sort hash tables by the table itself. To be able to print the list in an alphanumeric order the values would have to be extracted and then sorted and then printed.

Binary trees are able to sort the list quickly, a little faster than the vector data structure. However it takes a bit longer to be able to implement and insert and delete items into the structure versus the vector and hash table structures.

For this project, I would recommend the vector sorting and data structure. This is because being able to sort the list and print it quickly is more beneficial. Even with the time loss during searching the list. The vector sort also has a lower total cost than the rest of the data structures.