**Project One Pseudocode and Runtime Analysis**

## 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).

1. // Open file and read file pseudocode

// create an object and fstream with filename

vector<Course> LoadDataStructure()

{

Ifstream(“filename.txt”, ios::in);

Vector<course>courses;

While get each line in the file

If end of the line is reached

Then break the loop

Courses;

// get information separated by commas;

Course.coursenumber = info[0];

Course.coursename = info[1]; // store course structure information

For each course check if there are prerequisites

Then give them a new course number

Append the course prerequisites into list of courses

File.close()

Return courses;

}

// Open file and read file pseudocode end

1. // print courses into correct format

Void Printcourse( course course) {

// get the variable name

String courseNumber = course.courseNumber;

String courseName = course.courseName;

Vector <string> prerequisites = course.prerequisites;

Print course Number;

Print course name;

For each prerequisites list size

Print the prerequisite course;

// end;

// 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) {

}

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) {

// make a key by course number

Get the node by using the key and set it to the new node variable

While node != null // loop until course is not found in the hash table

{

If node-> course.courseNumber is equal to courseNumber

Then totalPrerequisites = node.Prerequisites.size();

For each prerequisites p in totalPrerequisites

Then add prerequisites of p to totalPrerequisites

Display number of total prerequisites;

Break;

Else

Set node to the next node

}

}

void printSampleSchedule(Hashtable<Course> courses) {

CONSTRUCTOR Course(line)

Number = SPLIT(line, DELIMETER = ,)[0]

Title = SPLIT(line, DELIMETER = ,)[1]

IF LENGTH of SPLIT(line, DELIMETER = ,) > 2

Prerequisites = SPLIT(line)[ 2 to LENGTH of SPLIT (line, DELIMETER = ,)]

END IF

END CONSTRUCTOR

END CLASS

FUNCTION createObject(Courses <Course>, File f)

Lines[] = " "

IF readFile(f, Lines) == TRUE

FOR each Line in Lines

APPEND NEW Course(Line) TO Courses

END FOR

END IF

ELSE PRINT("File cannot be read")

END ELSE

END FUNCTION

}

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

// make a key for courseNumber

Get the node by using key and set to a new node variable

// set the while loop to interate until node is not null

While node != null{

If node ->course.courseNumber is not equal to courseNumber

Then display the course info

For each prerequisite of the course

Print the prerequisite course info

Break;

Else

Set the node to the next

Node = node->next

}

}

// Tree pseudocode

int numPrerequisiteCourses(Tree<Course> courses) {

Course course = courses.search(courseNum)

While(course->Prerequisites != 0){

For each prerequisites in course->Prerequisites{

Courses.search(course->prerequisites->courseNum)}

++ totalPrequisites

}

}

void printSampleSchedule(Tree<Course> courses) {

Get input for courseId Assign current node to root

While current != NULL{

If course.courseId == current

then current,

print course.courseId

print course.name

while (prereq = true)

print course.prereq

If courseIid < root

current = left

Else

current = right

}

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

Get input for courseId Assign current node to root

While current != NULL{

If course.courseId == current

then current,

print course.courseId

print course.name

while (prereq = true)

print course.prereq

If courseIid < root

current = left

Else

current = right

}

// Menu Pseudocode

Choice == 0

While choice != 0{

// print menu choices

Print Load course file

Print course

Print individual course

Print exit

}

Switch (choice):

Case 1: Loadcourses(courseFile, datastructure

Case2: print sorted call

Case3: print curse info(course id)

## 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** |
| --- | --- | --- | --- |
| **Create vector** | 1 | 1 | 1 |
| **For each line in file** | 1 | n | n |
| **Create vector course item** | 1 | n | n |
| **While prep exist** | 1 | n | n |
| **Append prereq** | 1 | n | n |
| **Total Cost** | | | 5n + 1 |
| **Runtime** | | | O(n) |
|  | | |  |

| **Hash Table** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Create hash table | **1** | **1** | **1** |
| Insert method | 0 | 0 | 0 |
| Create key for course | 1 | N | N |
| If no entry found for key | 1 | N | N |
| Assign node to key | 1 | N | N |
| Else | 1 | N | N |
| Assign old node key to uni\_max, set to key,set old node and next to null pointer | 4 | N | 4n |
| Else | 1 | N | N |
| Find the next open node and new NewNode to end | 1 | N | n |
|  |  |  |  |
| For each line in file | 1 | n | n |
| Create vector course item | 1 | n | n |
| While prep exist | 1 | n | n |
| Append prereq | 1 | n | n |
| **Total Cost** | | | 16n + 1 |
| **Runtime** | | | O(n) |
|  | | |  |

| **Tree** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| If node is greater than the root add right | **1** | **N** | **N** |
| If no right node | 1 | N | N |
| This node becomes right | 1 | N | N |
| For each line in file | 1 | n | n |
| Create vector course item | 1 | n | n |
| While prep exist | 1 | n | n |
| Append prereq | 1 | n | n |
| **Total Cost** | | | 11n + 1 |
| **Runtime** | | | O(n) |
|  | | |  |

The vector method is one of the fastest methods to read file and add course object. It takes less memory and easy to read. However, the disadvantage is the search for a course. The program has to go through the whole list just to pick one. Hash table are a bit hard to perform sorting. Binary tree is even faster than the vector.