# 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).

// 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 all courses in courses

print Course object information (courseNumber, title)

for each prerequisite in Course object’s prerequisites

print prerequisite information (courseNumber, title)

}

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**

}

open file

if file is open {

while not end of file

read line from file

if line has at least two parameters

parse line into courseNumber, title, and prerequisites

create new Course object with courseNumber, title, and prerequisites

add Course object to vector data structure

else

print error message

}

// Hashtable pseudocode

int numPrerequisiteCourses(Hashtable<Course> courses) {

}

void printSampleSchedule(Hashtable<Course> courses) {

}

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

}

// Tree pseudocode

int numPrerequisiteCourses(Tree<Course> courses) {

}

void printSampleSchedule(Tree<Course> courses) {

}

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

}

Function readAndValidateFile(filePath): {

coursesHashTable = new Hashtable<Course>()

File file = open(filePath)

For each line in file:

tokens = split line by ","

If length of tokens < 2:

Print "Error: Insufficient parameters on line"

Continue to next line

courseNumber = tokens[0]

title = tokens[1]

prerequisites = []

For i = 2 to length of tokens:

prerequisites.append(tokens[i])

For each prerequisite in prerequisites:

If prerequisite not in coursesHashTable:

Print "Error: Prerequisite not found"

Break from loop

If prerequisites are valid:

course = new Course(courseNumber, title, prerequisites)

coursesHashTable.put(courseNumber, course)

Close file

Return coursesHashTable

}

Function printCourseInformation(coursesHashTable, courseNumber): {

If courseNumber not in coursesHashTable:

Print "Course not found"

Return

course = coursesHashTable.get(courseNumber)

Print "Course Number:", course.courseNumber

Print "Title:", course.title

If course.prerequisites is not empty:

Print "Prerequisites:"

For each prerequisite in course.prerequisites:

Print "- ", prerequisite

}

String courseNumber

String name

List<string> prerequisites

Function readAndValidateFile(filename):

Courses = new Vector<Course>

Try:

Open file with filename for reading

While not end of file:

Read a line from the file

If line is not empty:

Split the line into tokens

If number of tokens < 2:

Print “Error: Line does not have enough parameters”

Continue to next line

courseNumber = first token

name = second token

prerequisites = remaining tokens

create a Course object with courseNumber, name, and prerequisites

add the course object to the courses vector

catch FileNotFound exception:

print “Error: File not found”

finally:

close the file

return courses

Menu:

1. Load Data Structure
2. Print Course List
3. Print Course
4. Exit

Sort the course information by alphanumeric course number from lowest to highest

Print the sorted list to a display

## 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.

| **Code** | **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** | 1 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | 4n + 1 |
| **Runtime** | | | O(n) |

Vector:

1. Opening and reading the file: Reading each line has a cost of O(1) per line, so the total cost is O(n) for every line in the file.
2. Parsing and creating course objects: For each line, splitting it into tokens and creating a course object has a cost of O(1) per line, so the total cost is O(n).

Total: O(n) + O(n) = O(n)

Hashtable:

1. Opening and reading the file: Same as vector.
2. Parsing and creating course objects: Same as vector.

Total: Same as vector.

Tree:

1. Opening and reading the file: Same as vector and hashtable.
2. Parsing and creating course objects: Same as vector and hashtable.

Total: Same as vector and hashtable.

Advantages and Disadvantages:

Vector:

Advantages: Simple, allows random access, maintains insertion order.

Disadvantages: Expensive to insert or remove elements from the middle; resizing can be costly.

Hashtable:

Advantages: Fast lookups, no duplicates, can store key-value pairs.

Disadvantages: Hash collisions can occur, not ordered.

Tree:

Advantages: Ordered, allows for efficient search, insertion, and deletion operations.

Disadvantages: More complex than vectors or hashtables, requires balancing for optimal performance.

Recommendations:

Based on the analysis, I would recommend using a hashtable. It provides fast lookup speeds, which is important for searching for courses by course number. Additionally, it allows storing key-value pairs, which can be useful for mapping course numbers to course objects. The O(n) complexity for reading the file and creating course objects is the same for all three data structures, so the choice can be based on other factors like ease of use and the specific program requirements.