# CS 300 Pseudocode Document

## Function Signatures

displayMenu()

print "Menu:"

print "1. Load Data Structure"

print "2. Print Course List"

print "3. Print Course"

print "4. Exit"

getChoice()

choice = -1

while choice < 1 or choice > 4 do

print "Enter choice (1-4): "

choice = readInteger()

end while

return choice

menuLoop()

dataLoaded = false

while true do

choice = getChoice()

if choice == 1 then

loadDataStructure()

dataLoaded = true

print "Data loaded successfully."

else if choice == 2 then

if not dataLoaded then

print "Data structure not loaded. Please load data structure first."

else

printCourseList()

end if

else if choice == 3 then

if not dataLoaded then

print "Data structure not loaded. Please load data structure first."

else

printCourse()

end if

else if choice == 4 then

print "Exiting program."

exit()

end if

end while

loadDataStructure()

// Load data from file into data structure

// Open file for reading

file = open("data.txt", "r")

// Loop through file line by line

while not end of file do

line = readLine(file)

course = parseLine(line) // Parse line to extract course information

addCourseToDataStructure(course) // Add course to data structure

end while

// Close file

close(file)

printCourseList()

// Print alphanumerically ordered list of courses in the Computer Science department

courses = getCoursesInDepartment("Computer Science") // Retrieve courses in Computer Science department

sortedCourses = sortCourses(courses) // Sort courses alphabetically

// Print sorted list of courses

print "Course List:"

for each course in sortedCourses do

print course.name

end for

// 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) {

HashTable<Course> prerequistes = new Hashtable<Course>();

Stack<Course> stack = new Stack<Course>();

For each course in courses:

Stack.push(course);

While(!stack.isEmpty()){

Course course = stack.pop();

For each prerequistes p in current.getPrerequistiests() {

If(!prerequisites.contains(p)){

Prerequisites.put(p, p);

Stack.push();

}

}

}

Return preqreuistes.size();

}

void printSampleSchedule(Hashtable<Course> courses) {

for each course in courses:

print(course.courseNumber)

print(course.courseName)

print(course.instructor)

print(course.schedule)

print(course.location)

print(“\n”)

}

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

course = courses.get(courseNumber)

if course is not null:

print(course.courseNumber)

print(course.courseName)

print(course.instructor)

print(course.schedule)

print(course.location)

else:

print(“Course not found.”)

}

// Tree pseudocode

int numPrerequisiteCourses(Tree<Course> courses) {

total prerequisites = set of prerequisites of all courses in the tree

stack<Node<Course>> stack

stack.push(courses.root)

while stack is not empty

current = stack.pop()

if current.value.prerequisites is not empty and current.value.courseNumber not in totalPrerequisites

add current.value.prerequisites to totalPrerequisites

if current has left child

stack.push(left child of current)

if current has right child

stack.push(right child of current)

print number of totalPrerequisites

}

void printSampleSchedule(Tree<Course> courses) {

stack<Node<Course>> stack

stack.push(courses.root)

while stack is not empty

current = stack.pop()

print current.value

if current has left child

stack.push(left child of current)

if current has right child

stack.push(right child of current)

}

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

stack<Node<Course>> stack

stack.push(courses.root)

while stack is not empty

current = stack.pop()

if current.value.courseNumber == courseNumber

print current.value

return

if current has left child

stack.push(left child of current)

if current has right child

stack.push(right child of current)

print "Course not found"

}

## Example Runtime Analysis

| **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) |

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Stack<Node<Course>> stack** | 1 | 1 | 1 |
| **Stack.push(courses.root)** | 1 | 1 | 1 |
| **While stack is not empty** | 1 | n | 1 |
| **Current = stack.pop()** | 1 | n | n |
| **If current.value.courseNumber == courseNumber** | 1 | 1 | 1 |
| **Print current.value** | 1 | 1 | 1 |
| **return** | 1 | 1 | 1 |
| **If current has left child** | 1 | n-1 | n-1 |
| **Stack.push(left child of current)** | 1 | n-1 | n-1 |
| **If current has right child** | 1 | n-1 | 1 |
| **Stack.push(right child of current)** | 1 | n-1 | n-1 |
| **Print “course not found”** | 1 | 1 | 1 |
| **Total Cost** | | | 4n + 5 |
| **Runtime** | | | O(n) |

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Courses = course.get(courseNumber)** | 1 | 1 | 1 |
| **if course is not null:** | 1 | 1 | 1 |
| **Print(course.courseNumber)** | 1 | 1 | 1 |
| **Print(course.courseName)** | 1 | 1 | 1 |
| **Print(course.Instructor)** | 1 | 1 | 1 |
| **Print(course.schedule)** | 1 | 1 | 1 |
| **Print(course.location)** | 1 | 1 | 1 |
| **Else:** | 1 | 1 | 1 |
| **Print(“course not found.”)** | 1 | 1 | 1 |
| **Total Cost** | | | 7 |
| **Runtime** | | | O(1) |

Assuming I did the runtime analysis correctly then hash table would be the best-case scenario since it is not using a loop in the printcourseinformation function. However, other than that I believe vector data structure is the best one to use as it simple to use and provides the best functionality when working with something as simple as the courses. As the problem gets more complex it may be better to use a different data structure. The tree data structure is poor when it comes to runtime, and this may be because basic tree structure implementation are very poor use and needs optimizing to make it a better data structure for efficient usage. For this project I am going to use the vector data structure as there are only 8 courses however, if this project was for a large university, then hash table or tree would be the recommended data structure as they would be better at handling the bigger data set.