# CS 300 Pseudocode Document Jones

## Example Function Signatures

Below is an example of a function signature that you can use as a guide to help address the program requirements using each data structure for the milestones. The pseudocode for finding and printing course information is also given below and depicted in bold to help you get started. The provided pseudocode is for a vector data structure, so you may use this pseudocode in your first milestone as is. The hash table and tree structures are also shown below. But these structures are left for you to do in future milestones.

//Vector - Milestone 1

void searchCourse(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**

}

//Hash Table - Milestone 2

void searchCourse(HashTable<Course> courses, String courseNumber) {

}

//Binary Search Tree – Milestone 3

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

## Example Runtime Analysis

When you are ready to analyze the runtime for the Project One data structures for which you created the pseudocode, use the example chart below to support your work. This particular 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. The example only covers the search function for the vector structure. You do not have to complete your runtime analysis until Project One. However, working on your analysis now may help you understand the changes as you complete the milestones. Don’t forget to include your charts in Project One. You will submit Project One in Module Six.

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 1 | n | n |
| **if the course is the same as courseNumber** | 1 | n | n |
| **for each prerequisite of the course** | 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) |

Brandon Jones 9/21/2024  
  
// Struct to represent a Course

// A Course has a course number, title, and a list of prerequisites

STRUCT Course

String courseNumber // The course's identification (e.g., "CS101")

String title // The course's name (e.g., "Intro to Computer Science")

Vector<String> prerequisites // A list of course numbers that are prerequisites

// Function to load courses from a file

// Signature: Function loadCourses(String fileName) -> Vector<Course>

Function loadCourses(String fileName) -> Vector<Course>

Vector<Course> courses // A list to store all the courses

File file = open(fileName) // Open the file with the given name

IF file is not open // Check if the file is successfully opened

PRINT "Error: Cannot open the file." // Display an error message

RETURN empty courses vector // Return an empty list if the file can't be opened

// Read through each line in the file

FOR each line in the file

Split the line by commas or another delimiter // Break the line into pieces (course number, title, prerequisites)

IF there are fewer than 2 pieces after the split

PRINT "Error: The line format is incorrect." // Print an error message if the line is not properly formatted

CONTINUE to the next line // Skip this line and go to the next one

String courseNumber = the first part // Get the course number (the first part of the line)

String title = the second part // Get the course title (the second part of the line)

Vector<String> prerequisites // Create a list to hold the prerequisites

// Check the remaining parts of the line for prerequisites

FOR each part after the title

IF this part is a valid course number that exists in the courses list

ADD this part to the prerequisites list // Add the prerequisite to the list

ELSE

PRINT "Error: Prerequisite " + this part + " does not exist in the course list." // Print an error for invalid prerequisites

CONTINUE with the next line // Skip to the next line if an invalid prerequisite is found

// Create a new course and fill it with data

Course newCourse

newCourse.courseNumber = courseNumber

newCourse.title = title

newCourse.prerequisites = prerequisites // Add the prerequisites

ADD the newCourse to the courses list // Add the course to the list of courses

CLOSE the file // Close the file after reading all lines

RETURN the courses list // Return the list of courses after processing the file

// Function to search for a course and display its information

// Signature: Function searchCourse(Vector<Course> courses, String courseNumber) -> void

Function searchCourse(Vector<Course> courses, String courseNumber) -> void

// Go through each course in the courses list

FOR each course in the list

IF the course number matches courseNumber

PRINT "Course Number: " + course.courseNumber // Print the course number

PRINT "Course Title: " + course.title // Print the course title

IF the course has no prerequisites

PRINT "This course has no prerequisites." // Print a message if there are no prerequisites

ELSE

PRINT "Prerequisites:" // Print a list of prerequisites

FOR each prerequisite in the course's prerequisites

PRINT prerequisite // Print each prerequisite

RETURN // Stop the function once the course is found and printed

PRINT "Error: Course not found." // If the course is not found, print an error message

// Main function to run the program

// Signature: Function main() -> void

Function main() -> void

String fileName = "courses.txt" // Specify the name of the file to read from

Vector<Course> courses = loadCourses(fileName) // Load all courses from the file

IF the courses list is empty // Check if no courses were loaded

PRINT "Error: No courses were loaded from the file." // Print an error message

RETURN // Stop the program if no courses were loaded

String courseToSearch = INPUT "Enter the course number to search: " // Get the course number to search from the user

searchCourse(courses, courseToSearch) // Search for the course and display its information  
  
HASHTABLE Pseudocode:  
  
Function loadCoursesFromFile(String fileName):

Open file with name fileName

Initialize empty HashTable<Course> courses

For each line in the file:

Split line into tokens (courseNumber, courseTitle, prerequisites)

# Validate the line has at least two tokens (course number and title)

If number of tokens < 2:

Print "Error: Invalid line format. Each course must have at least a course number and title."

Return

# Create course object

course = Course()

course.number = courseNumber

course.title = courseTitle

# Validate and store prerequisites

If prerequisites exist:

For each prerequisite in prerequisites:

If prerequisite does not exist in courses:

Print "Error: Prerequisite course not found in file."

Return

Add prerequisite to course.prerequisites

# Store course in the HashTable

Insert course into courses with courseNumber as the key

Close file

Return courses  
  
Struct Course:

String number

String title

List<String> prerequisites  
  
Function searchCourse(HashTable<Course> courses, String courseNumber):

If courseNumber exists in courses:

course = courses[courseNumber]

Print "Course Number: " + course.number

Print "Course Title: " + course.title

If course.prerequisites is empty:

Print "No prerequisites"

Else:

Print "Prerequisites: "

For each prerequisite in course.prerequisites:

Print prerequisite

Else:

Print "Course not found."  
  
### Struct to Define a Course Object ###

struct Course:

courseNumber (String)

courseTitle (String)

prerequisites (List of Strings)

constructor(courseNumber, courseTitle):

this.courseNumber = courseNumber

this.courseTitle = courseTitle

this.prerequisites = empty list

### Function to Open File, Parse Data, and Store in Binary Search Tree ###

function readCourseFile(fileName):

open file with name fileName in read mode

create an empty binary search tree called courseTree

while not end of file:

read current line from file

if line is empty or contains only whitespace:

continue to next line

split the line by commas into courseTokens

if the number of tokens is less than 2:

print "Error: Line does not have at least a course number and title"

continue with the next line

# Create a new Course object

courseNumber = courseTokens[0]

courseTitle = courseTokens[1]

course = new Course(courseNumber, courseTitle)

# Handle Prerequisites

if the number of tokens > 2:

for each token from courseTokens[2] to end:

prerequisite = token

if checkCourseExists(courseTree, prerequisite):

add prerequisite to course.prerequisites

else:

print "Error: Prerequisite " + prerequisite + " does not exist as a course"

# Insert the Course object into the courseTree

insertCourse(courseTree, course)

close the file

return courseTree

### Function to Check if a Course Exists in the Tree ###

function checkCourseExists(courseTree, courseNumber):

search courseTree for a node with courseNumber

if found:

return true

else:

return false

### Function to Insert a Course into the Binary Search Tree ###

function insertCourse(courseTree, course):

if courseTree is empty:

set course as root node of courseTree

else:

recursively insert course into the tree based on courseNumber

if course.courseNumber < current node's courseNumber:

insert course in left subtree

else:

insert course in right subtree

### Function to Print Course Information ###

function printCourseInfo(courseTree, courseNumber):

course = searchCourse(courseTree, courseNumber)

if course is found:

print "Course Number: " + course.courseNumber

print "Course Title: " + course.courseTitle

if course.prerequisites is not empty:

print "Prerequisites: "

for each prerequisite in course.prerequisites:

print prerequisite

else:

print "No prerequisites."

else:

print "Course not found."

### Helper Function to Search for a Course in Binary Search Tree ###

function searchCourse(courseTree, courseNumber):

if courseTree is empty:

return null

if courseNumber == courseTree.root.courseNumber:

return courseTree.root

else if courseNumber < courseTree.root.courseNumber:

return searchCourse(courseTree.leftSubtree, courseNumber)

else:

return searchCourse(courseTree.rightSubtree, courseNumber)  
  
Final implementation of pseudocode:  
  
Struct Definition for Course  
  
TRUCT Course

String courseNumber // Example: "CS300"

String title // Example: "DSA: Analysis and Design"

List<String> prerequisites // List of prerequisite course numbers

Constructor(courseNumber, title)

this.courseNumber = courseNumber

this.title = title

this.prerequisites = empty list  
  
File Loading Function for All Data Structures  
  
Function loadCourses(String fileName) -> Vector<Course>

Vector<Course> courses

Open file with fileName

IF file is not open

PRINT "Error: Cannot open the file."

RETURN empty courses

FOR each line in the file

Split line by delimiter (comma) -> tokens

IF tokens < 2

PRINT "Error: Invalid line format."

CONTINUE

courseNumber = tokens[0]

title = tokens[1]

Vector<String> prerequisites

FOR each token after the title

IF course with token exists in courses

ADD token to prerequisites

ELSE

PRINT "Error: Invalid prerequisite."

Course newCourse(courseNumber, title, prerequisites)

ADD newCourse to courses

CLOSE file

RETURN courses  
  
Hash Table Implementation  
  
Function loadCoursesFromFile(String fileName) -> HashTable<Course>

HashTable<Course> courses

Open file with fileName

IF file is not open

PRINT "Error: Cannot open the file."

RETURN empty hash table

FOR each line in the file

Split line into tokens

IF tokens < 2

PRINT "Error: Invalid line format."

CONTINUE

courseNumber = tokens[0]

title = tokens[1]

List<String> prerequisites

FOR each token after title

IF token exists in courses

ADD token to prerequisites

ELSE

PRINT "Error: Invalid prerequisite."

Course newCourse(courseNumber, title, prerequisites)

INSERT courseNumber -> newCourse in courses

CLOSE file

RETURN courses  
  
Binary Search Tree (BST) Implementation  
  
Function readCourseFile(String fileName) -> BinarySearchTree<Course>

BinarySearchTree<Course> courseTree

Open file with fileName

IF file is not open

PRINT "Error: Cannot open the file."

RETURN empty tree

FOR each line in the file

Split line into tokens

IF tokens < 2

PRINT "Error: Invalid line format."

CONTINUE

courseNumber = tokens[0]

title = tokens[1]

Course newCourse(courseNumber, title)

FOR each token after title

IF checkCourseExists(courseTree, token)

ADD token to newCourse.prerequisites

ELSE

PRINT "Error: Invalid prerequisite."

INSERT newCourse into courseTree

CLOSE file

RETURN courseTree  
  
Search Functions for All Data Structures  
  
Function searchCourse(Vector<Course> courses, String courseNumber)

FOR each course in courses

IF course.courseNumber == courseNumber

PRINT course info (number, title, prerequisites)

RETURN

PRINT "Error: Course not found."  
  
Hash Table Search:  
  
Function searchCourse(HashTable<Course> courses, String courseNumber)

IF courseNumber exists in courses

PRINT course info (number, title, prerequisites)

ELSE

PRINT "Error: Course not found."  
  
Binary Search Tree Search:  
  
Function searchCourse(BinarySearchTree<Course> courseTree, String courseNumber) -> Course

IF courseTree is empty

RETURN null

IF courseNumber == courseTree.root.courseNumber

RETURN courseTree.root

ELSE IF courseNumber < courseTree.root.courseNumber

RETURN searchCourse(courseTree.left, courseNumber)

ELSE

RETURN searchCourse(courseTree.right, courseNumber)  
  
Print All Courses (Alphanumeric Order)  
  
Function printCourses(Vector<Course> courses)

SORT courses by courseNumber

FOR each course in courses

PRINT courseNumber + ": " + title  
  
Menu System  
  
Function mainMenu()

Vector<Course> courses

HashTable<Course> courseHashTable

BinarySearchTree<Course> courseTree

WHILE true

PRINT "1. Load Courses"

PRINT "2. Print All Courses (Alphanumeric Order)"

PRINT "3. Search Course"

PRINT "9. Exit"

choice = INPUT "Enter choice: "

IF choice == 1

courses = loadCourses("courses.txt")

courseHashTable = loadCoursesFromFile("courses.txt")

courseTree = readCourseFile("courses.txt")

IF courses is empty

PRINT "No courses loaded."

ELSE IF choice == 2

printCourses(courses)

ELSE IF choice == 3

courseNumber = INPUT "Enter course number: "

PRINT "Searching in Vector:"

searchCourse(courses, courseNumber)

PRINT "Searching in Hash Table:"

searchCourse(courseHashTable, courseNumber)

PRINT "Searching in Binary Search Tree:"

searchCourse(courseTree, courseNumber)

ELSE IF choice == 9

PRINT "Exiting..."

BREAK

ELSE

PRINT "Invalid choice."  
  
Data Structure Analysis: Big O Performance Evaluation  
  
Vector:

Load Time: O(n)

Search Time: O(n)

Pros: Simple and easy to implement.

Cons: Inefficient search for large datasets.

Hash Table:

Load Time: O(n)

Search Time: O(1) average, O(n) worst-case.

Pros: Fast search and insertion.

Cons: Requires more memory, performance affected by collisions.

Binary Search Tree:

Load Time: O(n log n)

Search Time: O(log n) (if balanced).

Pros: Balanced search times.

Cons: May become unbalanced, increasing search time.  
  
Conclusion:  
  
Based on the performance evaluation, I would recommend Hash Table for this project:

* It provides the fastest search time on average (O(1)).
* Quick insertion is essential given the frequency of course lookups.
* Although it uses more memory, the performance trade-off is worthwhile.