# CS 300 Pseudocode Document

## 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) {

**OPEN** course document

**IF** the document is not -1

READ each line

PARSE each line

**ELSE**

file is not found

**END IF**

**WHILE** file is not -1

IF the parameters < 2

PRINT "Error"

ELSE IF parameters >= 2

READ next line

**END IF**

**END WHILE**

**CLOSE** file

**INPUT** the variable for the course

CREATE hashTable class

**OPEN** file

READ file

INSERT method to store into class

ITERATE through the method

WHILE file is not -1

USE insert method

ADD to current values

PRINT from hashTable

READ from user input

PRINT parameters

END WHILE

PRINT:

CSCI100,Introduction to Computer Science

CSCI101,Introduction to Programming in C++,CSCI100

CSCI200,Data Structures,CSCI101

MATH201,Discrete Mathematics

CSCI300,Introduction to Algorithms,CSCI200,MATH201

CSCI301,Advanced Programming in C++,CSCI101

CSCI350,Operating Systems,CSCI300

CSCI400,Large Software Development,CSCI301,CSCI350

}

//Binary Search Tree – Milestone 3

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

**OPEN** course document

**IF** the document is not -1

READ each line

PARSE each line

**ELSE**

file is not found

**END IF**

**WHILE** file is not -1

IF the parameters < 2

PRINT "Error"

ELSE IF parameters >= 2

READ next line

**END IF**

**END WHILE**

**CLOSE** file

INPUT the variable for searchCourse

CONSTRUCT searchCourse

DESTRUCT searchCourse

TRAVERSE

INSERT courseNumber

SEARCH courses

RETURN and PRINT courses, courseNumber

//Pseudocode for Menu

OPEN Course Document

WHILE the document is not -1

cout << “Menu: “;

cout << “1. Load Data: “;

cout << “2. Print Course List: “;

cout << “3. Print Course: “;

cout << “4. Exit”

GET userInput

If userInput is C1:

GET FILE

LOAD DATA

CALL FILE

RETURN FILE

END IF

ELSE

cout << “File has already been uploaded.”;

IF userInput is C2:

LOAD DATA

IF course == courseNumber

IF userInput == prerequisiteN

SORT course //to sort courses alphabetically.

DISPLAY (course, presquisiteN)

END IF

ELSE IF userInput:

cout << “Exit”

END IF

END WHILE

CLOSE FILE

// Pseudocode for alphanumerical order

Start

class Course

{

String courseName;

String courseID;

Vector <string Prerequisites>

}

//Declare <vector, courseNumber, courseName, prerequisites>

OPEN text.file

//READ data line by line

Vector <Course> CourseList;

While end of file

Read line

CourseList[i].courseName=string(first comma);

CourseList[i]courseId=string(second comma);

while end of line

CourseList[i].prerequisite=insert string inside before comma

End of while

End while

SORT courseList[i].courseName

Print courseList[i].courseName, courseList[i].courseID

EXIT

Close File

## 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.

|  |  |  |  |
| --- | --- | --- | --- |
| **Vector (Code)** | **Line (Cost)** | # Times Executed | **Total Cost** |
| Reading the file | 1 | O(n) | n |
| Creating course objects | 1 | O(n) | n = 2n |
| **Hashtable** | Line(cost) | # times Executed | Total Cost |
| Reading file | 1 | O(n) |  |
| Course Objects | O(1) | O(n) | = 2n + 1 |
| Trees | Cost O(log n) | # of times executed  O(n) | Total = O(n log n) |

| **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) |

Vectors, Hashtables, and Trees each has some advantages and disadvantages. Vectors are good for memory and storage when it comes to lists. In addition, it is efficient in runtime and can display a dataset with little problems such as collision. The only con is that it could possibly get hectic since there’s no flexibility with adding or deleting elements. However, you can always use the resize() function. In regards to Hashtables, they are great for handling large data sets and are fast for searching, inserting, and deleting an element. However, they have a complex runtime. Regarding trees, this is the best for order of a dataset since many elements are grouped together (root, parent, child order). All in all, I would suggest using a Hashtable.