# 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**

}

void

//Hash Table - Milestone 2

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

ask user for input;

assign the input to key;

if the key is found;

print course info;

for each prerequisite of the course;

output each prerequisite;

}

//Binary Search Tree – Milestone 3

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

create a left and right node;

left and right node point to null;

create insert method;

if root is null current course is the root;

if courseNumber less than root point left;

else if courseNumber > root root point right;

return root;

Menu:

Create main() method

Set choice equal to 0

While choice is not equal to exit

Prompt user for choice

Output all choices to the user

Create switch case

case 1 loadCourses

break

case 2 printCourse

break

case 3 searchCourse

break

open and read date from a file:

void loadCourses()

use ifstream to open file

if file is not open output error

while not end of file

parse each line

if there are less than two values in a line return error

else read next line

if three values or more

store in vector

else return error

close file

Store course info:

create identifiers (courseID,courseName,preRequisite)

void vector<course> loadCourses() {

for each line in file

parse each line split with “,” and assign to token

course = new course

courseID is in position 0

courseName is in position 1

while not end of line

courseRequisite in position 2

push back coursePrerequisite tokens

print courses alphanumerically;

void printCourses(dataStructure,courseNumber)  
 for each course in dataStructure

sort by courseNumber

for courses in courseNumber

print course.title

print couse.prerequisites

## 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) |

| **HashTable**  **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| ask user for input; | 1 | 1 | 1 |
| assign the input to key; | 1 | n | n |
| if the key is found; | 1 | 1 | 1 |
| print course info; | 1 | 1 | 1 |
| for each prerequisite of the course; | 1 | n | n |
| output each prerequisite; | 1 | n | n |
| **Total Cost** | | | 3n + 3 |
| **Runtime** | | | O(n) |

| **Binary search tree**  **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| create a left and right node; | 1 | n | n |
| left and right node point to null; | 1 | n | n |
| create insert method; | 1 | 1 | 1 |
| if root is null current course is the root; | 1 | n | n |
| if courseNumber less than root point left; | 1 | n | n |
| else if courseNumber > root root point right; | 1 | n | n |
| return root; | 1 | 1 | 1 |
| **Total Cost** | | | 5n + 2 |
| **Runtime** | | | O(n) |

Data structures analysis

The three data structures used for these milestones each have their strengths and weaknesses which make them suitable for different tasks. For example, vectors can provide constant (O)1)) times and have good cache performance but their (O)1)) can become greater when performing insertion and deletion in the middle of the vector. Hash tables has other strengths and weaknesses they provide constant (O(1)) times, are efficient for searching and can handle a large number of elements. However, Hash tables don’t maintain elements ordered, and they can be memory intensive. Then the third option presented is binary search trees (BST) Provide efficient time use, the BST maintains elements sorted and they do not require extra memory for hash functions or resizing. Also, BST has a built-in function called inorder traversal to visit all the nodes and retrieve keys. Even with their advantages BSTs also have their disadvantages, they can become skewed if not self-balancing, more complex to implement compared to the other two options and their time to access data compared to hash tables.

The best approach for this application is to use a vector, since this application has a small data set the data structure can be kept simple and effective. If this was to be used for a whole school where there are hundreds of courses that students can sign up for, the best data structure for that case would be a Hash table. Although hash tables use more memory their speed would make it worth it in a larger data set.