# 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:**

**Reading pseudocode:**

//Opening a file to read contents. Using ifstream as we only need to read the course information from

the file.

Ifstream myfile

myFile.open(“CourseInformation.csv”)

IF file doesn’t doesn’t open/exist

Print error

ELSE //when file is found.

WHILE not the end of the file

IF the line in the csv file has less than 2 OR greater than 3 length then error out and print error

**Create Course Objects pseudocode:**

SET a vector for Courses.

Set the variables such as courseName, courseNumber, preReqCourse

Course \*courseName = new Course

FOREach line in the file

IF the line size is == 2 ;

Course newCourse ;

Course courseName = to line[0];

Course courseNumber = line[1];

SET course prereqs to an empty vector;

Return newCourse ;

ELSE

IF the line size is 3;

Course newCourse ;

Course courseName = to line[0];

Course courseNumber = line[1];

Course coursePrereqs =line[3];

SET course prereqs to an empty vector;

Return newCourse ;

**Search and print pseudocode:**

Request an input from the user for what course they will look up

Make a for loop that will loop through all the courses

FOREach courseName in courses

IF courseName == user input then

FOREach prereq

Print course name, number, and prereqs (if found)

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **ifstream** | 1 | 1 | 1 |
| **Open the file** | 1 | 1 | 1 |
| If the file fails | 1 | 1 | 1 |
| **While not the end of file** | 1 | n | n |
| **Set vector** | 1 | 1 | 1 |
| **Set variables** | 1 | 1 | 1 |
| **Set course node** | 1 | 1 | 1 |
| **ForEach function** | 15 | n | N |
| **Request user input** | 1 | 1 | 1 |
| **Loop through all couses** | 5 | n | n |
|  |  |  |  |
|  |  |  |  |
| **Total Cost** | | | 3n + 7 |
| **Runtime** | | | O(n) |

**Hash**

**Reading pseudocode:**

Using fstream, open the file to extract the contents

If the file returns a -1 then print out an error and break

ELSE

WHILE not the end of the file

IF the line in the csv file has less than 2 OR greater than 3 values then error out and print error

Close the file

**Create Course Objects pseudocode:**

DEFING: Create a structure with the name Course, this will hold the variables below:

STRING courseName;

STRING courseNumber;

//use a vector as not all classes have prerequisites.

VECTOR<STRING> prerequisites;

Construct an unsigned integer named DEFAULT\_SIZE equaling 8

CLASS: create a class with the name HashTable, inside this class have a structure with the node variables:

STRUCT Node

Course\* course;

Node\* next;

Unsigned integer key;

// add a default node with the node containing the course node and key

Node()

Node(Course course, unsigned int key)

//assign the default size to the unsigned size variable then store a node vector

Unsigned int size = DEFAULT\_SIZE;

Vector<Node> nodes;

HashTable();

Insert(Course course);

Int prerequisiteCourses(HashTable<Course> courses string preReqs){

Produce a key to hash the prereq then pull a node and set the new key to the new node with a pointer.

WHILE the node doesn’t equal the null pointer

IF the node for the course number is the same (preReqs == 0) then

The totalPrereqs will equal the node for the amount of prereqs

For each of these prereqs you will need to add the prereq to the total list of all prereqs

Then print all prereqs

Else the node will equal the next node pointer

}

**Search and print pseudocode:**

Void printCourses(HashTable<Course> course, Str courseNumber){

Produce a key to hash the course number then pull a node and set the new key to the new node with a pointer.

WHILE the node is not equal to the null pointer

IF the node is current course number is equal to the courseNumber

Print out the courses

For each prereq of the course print it ELSE node will equal the next node pointer

}

Int main() {

HasTable\* table = new HashTable();

Vector<string>temp;

String line ;

Ifstream infile(“class information file”);

WHILE(getline(infile, line));

Stringsteam ss(line);

//while the streamstream is good continue

WHILE(ss good());

String subst

Getline(ss, subst)

Temp.push\_back(substr)

Table.insert(parseLine(temp));

Temp clear();

}

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
|  |  |  |  |
| **Fstream to open file** | 1 | 1 | 1 |
| **If the file doesn’t open** | 1 | 1 | 1 |
| **Else while not the end** | 4 | n | N |
| **Define struct and variables for course** | 5 | 1 | n |
| **Class for hashTable** | 1 | 1 | 1 |
| **Struct for nodes** | 11 | 1 | 1 |
| **Prereq course** | 8 | n | N |
| **Search and print** | 7 | n | N |
| **Main** | 7 | n | n |
| **While getline** | 6 | n | N |
| **Total Cost** | | | 6n + 4 |
| **Runtime** | | | O(n) |

**Binary Tree pseudocode:**

Create a struct with the name Course, this will hold the variables below:

Struct Course

String courseName

String courseNumber

Vector<String>preRequisutes

Couse()

Create a struct with the name node with the variables

Struct Node

Course course

Node\* right

Node\* left

Create a class for BinarySearchTree

Class courseBinarySearchTree

Void printCourseInformation(Tree<Course> course, str courseNum){

Course course search for courseNum

Print the course number

WHILE the prereqs doesn’t equal 0

FOREach course search for the prereqs associated to the couse and print them

}

Void BinarySearchTree::addNode(Node\* node, Course course){

IF the node course courseNumber is greater then 0

IF the left node equals null then have it set a new node and move left

Else have the left node add a node and move left

ELSE

IF the tight node is null then make a new node and move right

ELSE have a node made and move the node right

}

Void BinarySearchTree::PrintCourseInformation(Node\* current, string courseNumber)

While the current node is not equal to null

IF current node is the same as courseNumber

Print courseNumber, courseName

Unsigned int size equals preReqs

Print prereqs

Unsigned int I = 0

FOREach course print available course prereqs

ELIF courseNumber is less than 0 then current = current -> left

ELSE current = current -> right

Itn main ()

Tree node equals new tree

Vector<string>temp

String line

Fstream infile(“CourseInformationFile”)

While(getline(infile, line))

Stringsteam ss(line)

While(ss.good())

Getling(ss, substr,’.’)

Temp.push\_back(substr)

Tree.insert(parseLine(temp))

Temp.clear

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Struct for course | 6 | 1 | 1 |
| Struct for node | 3 | 1 | 1 |
| Class for bst | 2 | 1 | 1 |
| Print course information | 5 | n | n |
| Add node | 7 | n | n |
| printCourseinformation | 10 | n | n |
| Main | 13 | n | n |
|  |  |  |  |
| **Total Cost** | | | 4n + 3 |
| **Runtime** | | | O(n) |

**Menu**

While user input != 9

Print:

1. Load Data Structure
2. Print course list
3. Print course

9. Exit

Please select an option

Cin >> option

IF option != 1, 2, 3, 9

Print please enter a valid option

ELSE

Case 1:

Load course data from csv

Break;

Case 2:

Print courseName and number

Break;

Case 3:

Cin >> request user to input course to search

Print courseNumber, Name, and preReqs

Break;

Case 4:

Print exiting, goodbye

Exit, leave the program

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| While loop | 2 | n | N |
| **Print options** | 7 | n | n |
| **If check** | 2 | n | n |
| **Else through cases** | 14 | n | n |
|  |  |  |  |
| **Total Cost** | | | 4n |
| **Runtime** | | | O(n) |

**Print in alphabetical order**

**//this can be done through a 2d array for simplicity sake**

Set an empty list []

FOREach courseNumber in the length of course

Loop through the empty list,

IF the currentCourseNumber is > the index of the list increase the index

Else

If the currentCouseNumber is < the index then add the couse number to that

Index, replacing it and moving the list to the right.

Add to the index of couseNumber

Return the list and print it

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Set and empty list | 1 | 1 | 1 |
| For each loop | 2 | n | N |
| Loop through list | 6 | n | n |
|  |  |  |  |
|  |  |  |  |
| **Total Cost** | | | 3n |
| **Runtime** | | | O(n) |
|  | | |  |

Vectors:

One of the main advantages of vectors is that its fast and that reading files and creating course object to the system. It can be slow while searching and looping through large files and will cause issues if theres a lot of information to parse through.

Hash tables:

Hash tables are insanely fast when information is set and called on, the main disadvantage would be that the tables aren’t sorted traditionally and will take a long time to go though to find the correct information when the set is not called on.

Binary Tree:

One big advantage to binary tress is that they can sort information quickly and that allows us to search and call information faster than hash tables, one problem is that setting this information would be slower than hash tables.

I would recommend Hash tables, they are incredibly fast when storing information and calling it. One this for hashtables is that any sorting of the information should be done before hand to store this to be called at a latter date when needed as it will take far more time to sort after the fact.