# 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).

**// Menu**

Void Menu{

Int userChoice and set to -1

Bool programDone and set to false

Print “Welcome to the CS course search and print system.” And newline

Print “1. Load courses” and newline

Print “2. Print Course List” and newline

Print “3. Print Course” and newline

Print “4. Exit Program” and newline

Prompt user with “Please make a selections: “

Set userChoice to user input

While (!programDone){

Switch (userChoice) {

Case 1:

If (checkFile(filename) != -1){

Run loadCourses(filename)

}

Else{

Set programDone to true

}

Break

Case 2:

Call printCourseList()

Break

Case 3:

String whichCourse

Prompt user with “Which course do you want to print: “

Set whichCourse to user input

Call printCourseInformation(whichCourse)

Break

Case 4:

Print “Goodbye”

Set programDone to true

Break

Default:

Print “Please make a valid choice between 1 and 4” and newline

Break

}

}

}

void PrintCourseList{

call quickSort(courses)

for x in courses

call printCourseInformation(x)

}

**// Course Class**

Class Course{

String courseNum

String courseName

Vector<string> prereqs

Void printCourse{

Print “Course#: “ and courseNum and newline

Print “Course Name: “ and courseName and newline

If prereqs is not empty

Print “prerequisites: “

For x in prereqs

Print x and “ “

Print a newline

}

**// Universal checkFile code**

int checkFile(string filename){

vector<string> courseNumbers

vector<string> prereqs

if file is not empty

while not end of file

parse string up to first ‘,’ or newline

add string to courseNumbers

parse string up to next ‘,’ or newline

if next parse is new line

print error “File formatted incorrectly: no course name found”

return -1

parse string up to next ',’ or newline

while string not equal to newline

search prereqs for string

if string is not in prereqs

add string to prereqs

parse string up to next ',’ or newline

else

print error “File is empty”

return -1

for string in prereqs

search courseNumbers

if string is not in courseNumbers

print error “File issue: prerequisite course not in file”

return -1

return 1

}

**// Vector pseudocode**

void PrintCourseList{

call quickSort(courses)

for x in courses

call printCourseInformation(x)

}

Int partition(Vector<course> courses, int start, int end){

Create int pivotIndex and set it to start + (end-start)/2

Create int pivotValue and set it to courses.at(pivotIndex) courseNum

Create int temp

While(start <= end) {

While(courses.at(start) courseNum < pivotValue){

Start++

}

While(courses.at(end) courseNum > pivotValue) {

End++

}

If(start <= end){

Temp = courses.at(start)

Courses.at(start) = courses.at(end)

Courses.at(end) = temp

Start++

End++

}

}

Return pivotIndex

}

Void quicksort(Vector<course> courses, int start, int end){

If(start < end){

Create int pivotIndex and set it to partition(course, start, end)

Quicksort(courses, start, pivotIndex – 1)

Quicksort(courses, pivotIndex, end)

}

}

void loadCourses(Vector<course> courses, string filename){

open filename

if checkFile(filename) is not equal to 1

close filename

exit program

else

close filename

open filename

while not end of file

Course tempCourse

parse string up to first ‘,’

set tempCourse.courseNum to string

parse string up to next ‘,’ or newline

set tempCourse.courseName to string

parse next string up to ‘,’ or newline

if string is not a newline

pushback string to prereq vector of tempCourse

pushback tempCourse to courses

close filename

}

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 printCourseInformation(Vector<Course> courses, String courseNumber) {

for x in courses

if courseNum of x is equal to courseNumber

call print function for x

for y in prereqs of x

call printCourseInformation(courses, y)

else

print “This course is not on the list!”

}

**// Hashtable pseudocode**

Class HashTable{

Private:

Vector<Node> nodes

int tableSize

int hash(int key)

Public:

HashTable()

HashTable(int size)

Void Insert(Course course)

}

Int hash(int key){

Return key % tableSize

}

Void Insert(Course course){

Int key = hash(courseNum)

If(nodes.at(key) is empty){

Set nodes.at(key) course to equal course

Set nodes.at(key) key to equal key

Set nodes.at(key) next to null

}

Else{

Set tempPtr to nodes.at(key)

While(tempPtr next != null){

Set tempPtr to tempPtr next

}

Set tempPtr next to point to new node

Set new node course to course

Set new node key to key

Set new node next to null

}

}

Void printCourseList(HashTable hash){

For(i = 0; i < tableSize; i++){

If nodes.at(i) is not empty

Call printCourseInformation(nodes.at(i) course)

Create tempPtr and point it to nodes.at(i)

While tempPtr’s next is not null

Set tempPtr to tempPtr’s next

Call printCourseInformation(tempPtr course)

}

}

void loadCourses(Hashtable hash, string filename){

open filename

if checkFile(filename) is not equal to 1

close filename

exit program

else

close filename

open filename

while not end of file

Course tempCourse

parse string up to first ‘,’

set tempCourse.courseNum to string

parse string up to next ‘,’ or newline

set tempCourse.courseName to string

parse next string up to ‘,’ or newline

if string is not a newline

pushback string to prereq vector of tempCourse

call hash->Insert(tempCourse)

close filename

}

**// Tree pseudocode**

Void inOrder(Node node){

If(node not equal to null){

inOrder(node left child)

printCourseInformation(node courseNum)

inOrder(node right child)

}

}

Void printCourseList(){

inOrder(root)

}

void loadCourses(BinarySearchTree bst, string filename){

open filename

if checkFile(filename) is not equal to 1

close filename

exit program

else

close filename

open filename

while not end of file

Course tempCourse

parse string up to first ‘,’

set tempCourse.courseNum to string

parse string up to next ‘,’ or newline

set tempCourse.courseName to string

parse next string up to ‘,’ or newline

if string is not a newline

pushback string to prereq vector of tempCourse

call bst->Insert(tempCourse)

close filename

}

Class BinarySearchTree{

Private:

Node root

Void addNode(Node node, Course course)

public:

BinarySearchTree()

~BinarySearchTree()

Void PrintCourseInformation(string courseNum)

Void Insert(Course course)

Void Remove(string courseNum)

Course Search(string courseNum)

}

Void Insert(Course course){

If BST root is empty

Point root to a new Node

Add course data to node

Else

Call addNode(root node, course)

}

Void addNode(Node node, Course course){

If(node courseNum is greater than course courseNum)

If node’s left child is empty

Point node’s left child to new node

Add course to the new node

Else

Call addNode(node’s left child, course)

Else

If node’s right child is empty

Point node’s right child to new node

Add course to the new node

Else

Call addNode(node’s right child, course)

}

Void Remove(string courseNum){

Create node parent and point to null

Create node current and point it to BST root

While the current node is not empty

If current courseNum matches removal courseNum

If current has no children

If parent pointer is empty

Point BST root to null

Else if current is parent’s left child

Point parent’s left child to null

Else

Point parent’s right child to null

Else if current has no right child

If parent pointer is empty

Point BST root to null

Else if current is parent’s left child

Point parent’s left child to current left child

Else

Point parent’s right child to current right child

Else if current has no left child

If parent pointer is empty

Point BST root to null

Else if current is parents left child

Point parent’s left child to current right child

Else

Point parent’s right child to current right child

Else

Create node suc and point it to current right child

While suc has a left child

Point suc to its left child

Create tempCourse to hold suc’s course

Call Remove(suc’s courseNum)

Copy suc’s course data to current node

Exit function

Else if current courseNum is less than removal courseNum

Point parent to current

Point current to its right child

Else

Point parent to current

Point current to its left child

Exit function

}

Course Search(string courseNum){

Create node current and point it to BST root

While current is not empty

If current courseNum matches searched courseNum

Return current course

Else if current courseNum is less than searched courseNum

Point current to its left child

Else

Point current to its right child

If course is not found create and empty course and return it

}

void printCourseInformation(string courseNum){

Course toPrint = Search(courseNum)

If toPrint is empty

Print “Course could not be found”

Else

Print toPrint course info

for x in prereqs of toPrint

call printCourseInformation(x courseNum)

}

## Example Runtime Analysis

When you are ready to begin analyzing the runtime for the data structures that you have created pseudocode for, use the chart below to support your work. This 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.

| **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 loadCourses for vectors** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Open filesName** | **1** | **1** | **1** |
| **If checkFile(filename != 1)** | **1** | **1** | **1** |
| **Close filename** | **1** | **1** | **1** |
| **Exit program** | **1** | **1** | **1** |
| **Else** | **1** | **1** | **1** |
| **Close filename** | **1** | **1** | **1** |
| **Open filename** | **1** | **1** | **1** |
| **While not at end of file** | **1** | **N** | **N** |
| **Course tempCourse** | **1** | **N** | **N** |
| **Parse string up to ‘,’** | 1 | n | n |
| **Set tempCourse courseNum to string** | 1 | n | n |
| **Parse string up to next ‘,’** | 1 | N | N |
| **Set tempcourse courseName** | 1 | n | n |
| **Parse string to next ‘,’ or newline** | 1 | N | N |
| **While string != newline** | 1 | N | N |
| pushback string to prereq vector of tempCourse | 1 | N | N |
| **Parse string to next ‘,’ or newline** | 1 | N | N |
| Pushback tempCourse to courses | 1 | n | n |
| Close fileName | 1 | 1 | 1 |
| **Total Cost** | | | 11n + 8 |
| **Runtime** | | | O(n) |

| **Code loadCourses for hashTable**  **Same as above plus insert function** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Int key = hash(courseNum)** | **1** | **N** | **N** |
| If(nodes.at(key) is empty){ | **1** | **N** | **N** |
| Set nodes.at(key) course to equal course | **1** | **n** | **n** |
| Set nodes.at(key) key to equal key | **1** | **n** | **n** |
| Set nodes.at(key) next to null | **1** | **N** | **N** |
| **Else** | **1** | **N** | **N** |
| Set tempPtr to nodes.at(key) | **1** | **N** | **N** |
| While(tempPtr next != null) | **N** | **N** | **N^2** |
| Set tempPtr to tempPtr next | **N** | **N** | **N^2** |
| Set tempPtr next to point to new node | **1** | **N** | **N** |
| Set new node course to course | **1** | **N** | **N** |
| **Set new node key to key** | **1** | **N** | **N** |
| **Set new node next to null** | **1** | **N** | **N** |
|  |  | **Total Cost** | **11N = 2N^2** |
|  |  | **Runtime** | **O(N^2)** |

| **Code loadCourses for hashTable**  **Same as above plus insert function** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| If BST root is empty | **1** | **n** | **N** |
| Point root to a new Node | **1** | **N** | **N** |
| Add course data to node | **1** | **N** | **N** |
| **Else** | **1** | **N** | **N** |
| **Call addNode(rootNode, course)** | **N^N** | **N** | **N(N^N)** |
|  |  | **Total Cost** | **4N + N(N^N)** |
|  |  | **Runtime** | **O(N^N)** |

**Evaluations**

**Vectors**

Advantages:

* Faster course loading
* Easier to code for
* Least storagespace

Disadvantages:

* Slowest search, sort, and removal algorithms

**Hash Tables**

Advantages:

* Faster course loading than BST
* Can have faster searches than BST dependent on hash algorithm and how collisions are handled
* Faster searches, sorting, and removal as compared to vectors

Disadvantages:

* Slower course loading than vectors
* Possibly hardest to code well
* Posibbly max storagespace requirements

**BST**

Advantages:

* Faster removal than Hash Table
* Easier coding than Hash Table
* Can have faster searching than Hash Table
* Automatically sorts when created
* Faster searches, sorting, and removal than vectors

Disadvantages:

* Slowest course loading
* Larger storage space requirements than vectors
* Removals can become complicated if several children exist for node to be removed

**Final Recommendations**

I would recommend the Vector data structure for this project. While it has the slowest search, sort, and removal speeds, that only matters with large data sets. An average CS course would have well under 100 courses to consider and as such the sort, removal, and search speed differences would be negligible. On the other hand, it is one of the easiest data structures to work with given that most of its functions are built into C++ already and requires minimal overhead for data space. If we were to up the courses considerable as in to the thousands range than I might switch my answer to the a BST as they are sorted during creation and thus are much faster to search later on which would be very important for a system designed to look up courses and print out their information.