# CS 300 Pseudocode Document

**Project 1**

Annaliese Pintar

Struct Course

Declare String courseNumber

Declare String courseName

Declare String Vector coursePreReqs

Course ( ){

courseName = “”

coursePreReqs = {null}

}

End Struct Course

**Vector**

**Read From File**

Course Vector Function vectorReadFile

Declare String filename

Declare File courseInfoFile

Declare String info

Declare Course Vector courses

fileName = name of file containing course info

Open courseInfoFile named fileName

If courseInfoFile does not open properly

Display error message “File (filename) was not found.”

Else

While the end of the file has not been reached

Push parse(info) to the back of courses;

End While

End if

Close courseInfoFile

Return courses

End Function vectorReadFile

**Parse Information**

Course Function parse (Vector rawInfo)

Declare Course course

Declare Integer elementNum

Declare Integer start

Declare Integer position

Declare String info

Declare String Vector courseNums

elementNum = 0

start = 0

For the size of vector rawInfo starting at elementNum, increment elementNu

position = Find first occurrence of “,” from rawInfo at elementNum

position = position - 1

info = rawInfo at elementNum from start to position

push info to the back of courseNums

End For

Declare String number

Declare String name

Declare String Vector preReq

elementNum = 0

position = 0

For the size of vector rawInfo starting at elementNum, increment elementNum

number = “”

name = “”

position = Find first occurrence of “,” from rawInfo at elementNum

position = position – 1

number = rawInfo at elementNum from start to position

start = position + 1

position = Find occurrence of “,” after start from rawInfo at elementNum

position = position – 1

name = rawInfo at elementNum from start to position

Boolean commaFound

commaFound = true

While commaFound is true

Boolean match

match = false

Declare Integer count

count = 0

start = position + 1

position = Find occurrence of “,” after start from rawInfo at elementNum

If “,” is found or newline is found

position = position – 1

info = rawInfo at elementNum from start to position

While match is false or count does not equals size of courseNums - 1

If courseNums at count matches info

match = true

Push course info to the back of preReq

End If

count = count + 1

End While

Else

commaFound = false

End Else

End While

course = vectorCreate(number, name, preReq)

End For

Return course

End parse function

**Create Object**

Course Function vectorCreate(number, name, preReq)

Declare Course course

course courseNumber = number

course courseName = name

course courePreReqs = preReq

Return course

End Function vectorCreate

**Sort**

Integer Function partition(Course Vector courses, integer begin, integer end)

Declare integer low

Declare integer high

Declare integer midPoint

Declare integer pivot

Declare Boolean sortDone

low = begin

high = end

midPoint = low + (high – low) / 2

pivot = midpoint

sortDone = false

While sortDone is false

While title at low index is less than pivot

Increment low

End While

While title at pivot is less than high

Decrement high

End While

If low index is greater than high

sortDone = true

Else

Swap low and high

Increment low

Decrement high

End If

End While

Return high

End function partition

Void Function quicksort(Course Vector courses, integer begin, integer end)

Declare integer mid

mid = 0

If begin is greater than or equal to end

Return

End If

mid = partition(courses, begin, end)

quickSort(courses, begin, mid)

quickSort(courses, mid + 1, end)

End quicksort

**Print**

Void Function vectorPrint(Course vector courses)

For the size of courses

Display courseNumber “, ”

Display courseName “, “

For the size of courses coursePreReqs

Display coursePreReqs at position

If position is not equal to size of cousePreReqs – 1

Display “, “

End

End For

Newline

End For

End Function vectorPrint

**Hash Table**

**Read File**

Vector Course Function hashTableReadFile

Declare String filename

Declare File courseInfoFile

Declare String info

Declare Course Vector courses

fileName = name of file containing course info

Open courseInfoFile named fileName

If courseInfoFile does not open properly

Display error message “File (filename) was not found.”

Else

While the end of the file has not been reached

Push parse(info) to the back of courses

End While

End if

Return courses

End Function hashTableReadFile

**Parse Information**

Course Function parse (Vector rawInfo)

Declare Course course

Declare Integer elementNum

Declare Integer start

Declare Integer position

Declare String info

Declare String Vector courseNums

elementNum = 0

start = 0

For the size of vector rawInfo starting at elementNum, increment elementNu

position = Find first occurrence of “,” from rawInfo at elementNum

position = position - 1

info = rawInfo at elementNum from start to position

push info to the back of courseNums

End For

Declare String number

Declare String name

Declare String Vector preReq

elementNum = 0

position = 0

For the size of vector rawInfo starting at elementNum, increment elementNum

number = “”

name = “”

position = Find first occurrence of “,” from rawInfo at elementNum

position = position – 1

number = rawInfo at elementNum from start to position

start = position + 1

position = Find occurrence of “,” after start from rawInfo at elementNum

position = position – 1

name = rawInfo at elementNum from start to position

Boolean commaFound

commaFound = true

While commaFound is true

Boolean match

match = false

Declare Integer count

count = 0

start = position + 1

position = Find occurrence of “,” after start from rawInfo at elementNum

If “,” is found or newline is found

position = position – 1

info = rawInfo at elementNum from start to position

While match is false or count does not equals size of courseNums - 1

If courseNums at count matches info

match = true

Push course info to the back of preReq

End If

count = count + 1

End While

Else

commaFound = false

End Else

End While

course = hashTableCreate(number, name, preReq)

End For

Return course

End parse function

**Create Hashtable**

Void Function hashTableCreate(Course course, HashTable\* hashTable)

Call insert function to insert course into hashtable

End Function hashTableCreate

Void Function Insert(Course course)

Create key for course

Get node with key

If entry for key is not found

Assign node to key position

Else an entry is found for key

If key is still at default value

Node’s key is now key

Node’s bid is now course

Node’s next is now null pointer

Else if key has a key value

While next of Node is not null ptr

Node is Node’s next

End While

End If

End If

End Function Insert

**Print**

Void Function hashTablePrint

For all nodes begin to end

Key is key of Node iteration

If key is not default value

Display information of current node iteration

Set Node to next iteration

While Node is not empty

Display Node’s information

Node is next Node

End While

End If

End For

End Function hashTablePrint

**Tree**

**Read File**

Course Vector Function treeReadFile

Declare String filename

Declare File courseInfoFile

Declare String info

Declare Vector Course courses

fileName = name of file containing course info

Open courseInfoFile named fileName

If courseInfoFile does not open properly

Display error message “File (filename) was not found.”

Else

While the end of the file has not been reached

Push parse(info) to the back of courses

End While

End if

Return courses

End Function treeReadFile

**Parse Information**

Course Function parse (Vector rawInfo)

Declare Course course

Declare Integer elementNum

Declare Integer start

Declare Integer position

Declare String info

Declare String Vector courseNums

elementNum = 0

start = 0

For the size of vector rawInfo starting at elementNum, increment elementNu

position = Find first occurrence of “,” from rawInfo at elementNum

position = position - 1

info = rawInfo at elementNum from start to position

push info to the back of courseNums

End For

Declare String number

Declare String name

Declare String Vector preReq

elementNum = 0

position = 0

For the size of vector rawInfo starting at elementNum, increment elementNum

number = “”

name = “”

position = Find first occurrence of “,” from rawInfo at elementNum

position = position – 1

number = rawInfo at elementNum from start to position

start = position + 1

position = Find occurrence of “,” after start from rawInfo at elementNum

position = position – 1

name = rawInfo at elementNum from start to position

Boolean commaFound

commaFound = true

While commaFound is true

Boolean match

match = false

Declare Integer count

count = 0

start = position + 1

position = Find occurrence of “,” after start from rawInfo at elementNum

If “,” is found or newline is found

position = position – 1

info = rawInfo at elementNum from start to position

While match is false or count does not equals size of courseNums - 1

If courseNums at count matches info

match = true

Push course info to the back of preReq

End If

count = count + 1

End While

Else

commaFound = false

End If

End While

course = treeCreate(number, name, preReq)

End For

Return course

End parse function

**Create Tree**

Function treeCreate(BinarySearchTree bst)

Declare Course course

course = hashTableReadFile

Call insert function to insert course into bst

End Function treeCreate

Function insert(Course course)

If root is null

root = new node course

Else

Add Node root and course

End insert function

Void Function addNode(Node nose, Course theCourse)

If node’s courseNumber is larger than theCourse courseNumber

If left of node equals null

Left is new node with node course being theCourse

Else

Recurse down the left node

End If

Else

If right of node is null

Right is new node with node course being theCourse

Else

Recurse down left node

End If

End If

End Function addNode

**Print in Order**

Void Function treePrint(Node node)

If node is not null pointer

Call treePrint(for left of current node)

Display courseNumber “, “

Display courseName “, “

For size of coursePreReqs

Display coursePreReqs at position

If position is equal to coursePreReqs size – 1

Display “, “

End If

End for

Call treePrint(for right of current node)

End If

End Function treePrint

**Menu**

Void Function menu

Display “Menu”

Switch case

If 1 is selected

Load data

Break

If 2 is selected

Call print course list function

Break

If 3 is selected

Call print Course function

Break

If 4 is selected

Exit

Break

End Switch

End menu function

Traversing Runtime Analysis

Vector

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 1 | n | n |
| **if courseNumber of courses in i is same as courseNumber** | 1 | n | n |
| **Return course in i** | 1 | 1 | 1 |
| **Dispay Message “course not found”** | 1 | 1 | 1 |
| **Return empty bid** | 1 | 1 | 1 |
| **Total Cost** | | | 2n + 3 |
| **Runtime** | | | O(n) |

Hash Table

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **If entry exists for key** | 1 | 1 | 1 |
| **Return findNode’s courseNumber** | 1 | 1 | 1 |
| **If entry does not exist for key** | 1 | 1 | 1 |
| **Return course** | 1 | 1 | 1 |
| **While there is an entry in findNode** | 1 | n | n |
| **If correct node is found** | 1 | n | n |
| **Return course** | 1 | 1 | 1 |
| **findNode is findNode’s next** | 1 | n | n |
| **Return bid** | 1 | 1 | 1 |
| **Total Cost** | | | 3n + 6 |
| **Runtime** | | | O(n) |

Tree

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **While thisNode does not equal null** | 1 | logn | logn |
| **If thisNode’s courseNumber matches courseNumber** | 1 | n | n |
| **Return thisNode’s course info** | 1 | 1 | 1 |
| **If courseNumber is less than thisNode’s courseNumber** | 1 | n | n |
| **thisNode is now thisNode’s left node** | 1 | n | n |
| **Else** | 1 | n | n |
| **thisNode is node thisNode’s right node** | 1 | n | n |
| **Declare Bid bid** | 1 | 1 | 1 |
| **Return bid** | 1 | 1 | 1 |
| **Total Cost** | | | 5(logn) + 3 |
| **Runtime** | | | O(logn) |

The main advantage of vectors is that they are easy to work with and often quicker to write. However, they are less efficient for traversing through the data then binary search trees and about the same as hash tables, as shown above. When it comes to searching, vectors are the least efficient out of the three because in the worst case, you would have to compare the value you are searching for with each element.

For Hash Tables, if you need to be able to search data for a specific element, Hash Tables will be the most effective of the three. Worst case, it has a constant complexity for searching while vectors have a linear complexity and Binary Search Trees have a logarithmic complexity. However, if you wish to transverse through each element, Hash Tables would be less efficient than Binary Search Trees (worst case logarithmic complexity) and about the same as Vectors (both have worst case linear complexities). Hash Tables are also not ideal for data that needs to be in a specific order. The location of an element in a Hash Table is dependent on the value of the element modulo the number of elements in the hash table, so the elements will not be any specific order. You would have to transverse through the table and compare the elements to put the elements in a specific order.

Binary Search Trees are great for data that needs to be in a specific order. As shown above, Binary Search Trees worst case for traversing is the best of the three. For searching for a specific element, it is better than vectors, but not as good as Hash Table. Also, the code for Binary Search Trees can be a bit heftier.

**Recommendation**

Of the three data structures, the one I would recommend for this specific project would be Binary Search Trees. The application needs to display an ordered list of the courses. This project does not call for searching for a specific course, so the main advantage of Hash Tables will be irrelevant. Vectors fall in the middle or back of the pack in all categories, so it is not the most ideal. Data structures a great for creating ordered lists. This can be done as you are inserting new nodes, unlike with vectors. For vectors, it is not easy to insert an element in the middle of the vector so creating an order list can be troublesome.