# Project One:

# Matthew A Keaton

Southern New Hampshire University

CS300: DSA: Analysis and Design

Professor Jack Lusby

February 16, 2025

# CS 300 Pseudocode Document

//Vector - Milestone 1

//Pseudocode of open, read, and parse file && creating course objects

**START**

DEFINE STRUCT Course {

STRING courseNumber

STRING courseName

STRING coursePreReqs vector

END STRUCT

}

Vector<Course> loadCourses(STRING txtFilePath) {

DISPLAY “Loading text file” txtFilePath

vector<Course> courses

TRY

OPEN file(txtFilePath)

IF (file != open) THEN

THROW exception

ENDIF

WHILE file has lines

READ each line of file

INITIALIZE object to parse current line

STRING courseNumber, courseName

STRING coursePreReqs vector

READ line for courseNumber STOP at “,”

IF not courseNumber THEN

THROW exception

ENDIF

READ line for courseName STOP at “,”

IF not courseName THEN

THROW exception

ENDIF

WHILE READ line for any coursePreReqs

ADD to coursePreReqs vector

END WHILE

INITIALIZE course object with courseNumber, courseName, coursePreReqs

ADD course to courses vector

END WHILE

CATCH error& e

DISPLAY e.what

CLOSE file

RETURN vector courses

}

//Pseudocode of search && printing course information

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

FOR all (Course, course, courses)

IF (course.courseNumber is the same as courseNumber) THEN

PRINT out “Course Number: “ && course.courseNumber

PRINT out the course.courseName

IF (course.coursePreReqs is empty) THEN

DISPLAY “there are no prereqs”

ELSE

DISPLAY “Prerequisites: “

ENDIF

FOR STRING coursePreReqs in course

PRINT the prerequisite course information

ENDFOR

RETURN

ENDIF

DISPLAY message for course not found

ENDFOR

**END**

}

//Hash Table - Milestone 2

**START**

//Create hash table class with chaining

CREATE CLASS HashTable {

PRIVATE:

STRUCT Node {

Bid bid

Hash key

Node pointer

CONSTRUCT Node()

INIT int Key

SET next to nullptr

CALL CONSTRUCT Node()

INIT Bid

CONSTRUCT Node(Bid & key)

SET key to value

}

Vector<Node>nodes

DEFINE HashTable size

DECLARE Hash function with int key

PUBLIC:

INIT HashTable()

HashTable(int size)

void PrintAll()

void searchCourse(STRING courses, INT courseNumber)

}

FUNCTION HashTable() {

INIT Node STRUCT

}

FUNCTION HashTable(int size) {

SET size using pointer

RESIZE nodes(size)

}

FUNCTION HashTable(int key) {

RETURN key MOD size of table

}

//load, read, parse txt file

Vector<Course> loadCourses(STRING txtFilePath) {

DISPLAY “Loading text file” txtFilePath

vector<Course> courses

TRY

OPEN file(txtFilePath)

IF (file != open) THEN

THROW exception

ENDIF

WHILE file has lines

READ each line of file

INITIALIZE object to parse current line

STRING courseNumber, courseName

STRING coursePreReqs vector

READ line for courseNumber STOP at “,”

IF not courseNumber THEN

THROW exception

ENDIF

READ line for courseName STOP at “,”

IF not courseName THEN

THROW exception

ENDIF

WHILE READ line for any coursePreReqs

ADD to coursePreReqs vector

END WHILE

INITIALIZE course object with courseNumber, courseName, coursePreReqs

ADD course to courses vector

END WHILE

CATCH error& e

DISPLAY e.what

CLOSE file

RETURN vector courses

}

FUNCTION PrintALL(){

FOR (INIT ITERATE begin();CONDITION ITERATE to end();INCREMENT ITERATE){

IF (iter.key != UINT\_MAX) THEN

DISPLAY courseNumber, courseName, and preReqs

SET node to next.iterate

WHILE (Node != nullptr)

DISPLAY courseNumber, courseName, and preReqs

SET node to node.next

ENDWHILE

ENDIF

ENDFOR

}

//Search for courses

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

SET key and COMPUTE hash

IF (node != nullptr && node.key != UINT\_MAX && COMPARE == 0) THEN

RETURN node

ENDIF

IF (node == nullptr || node.key == UINT\_MAX) THEN

RETURN object

ENDIF

WHILE (node != nullptr)

IF (node.key != UINT\_MAX && COMPARE == 0) THEN

RETURN node

ENDIF

SET node to node.next

ENDWHILE

}

**END**

//Binary Search Tree – Milestone 3

**START**

STRUCT Course

STRING courseNumber

STRING courseName

STRING coursePreReqs Vector

STRUCT Tree

Course object

Node Left pointer

Node Right pointer

//load, read, parse txt file

Vector<Course> loadCourses(STRING txtFilePath) {

DISPLAY “Loading text file” txtFilePath

vector<Course> courses

TRY

OPEN file(txtFilePath)

IF (file != open) THEN

THROW exception

ENDIF

WHILE file has lines

READ each line of file

INITIALIZE object to parse current line

STRING courseNumber, courseName

STRING coursePreReqs vector

READ line for courseNumber STOP at “,”

IF not courseNumber THEN

THROW exception

ENDIF

READ line for courseName STOP at “,”

IF not courseName THEN

THROW exception

ENDIF

WHILE READ line for any coursePreReqs

ADD to coursePreReqs vector

END WHILE

INITIALIZE course object with courseNumber, courseName, coursePreReqs

ADD course to courses vector

END WHILE

CATCH error& e

DISPLAY e.what

CLOSE file

RETURN vector courses

}

FUNCTION CONSTRUCT BinarySearchTree()

root = nullptr

FUNCTION DESTRUCT BinarySearchTree()

WHILE (root != nullptr)

REMOVE (root->node)

ENDWHILE

//Traverse Tree

FUNCTION InOrder()

inOrder(root)

FUNCTION PostOrder()

postOrder(root)

FUNCTION PreOrder()

preorder(root)

FUNCTION BinarySearchTree Insert(Course courses)

IF (root == nullptr)THEN

SET root = new Node

ELSE

ADD node

ENDIF

//Search for course

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

SET current node = root

WHILE (current != nullptr)

IF (current->Course.courses == courses) THEN

RETURN current->Course

ENDIF

IF (courses < current->Course.courses) THEN

SET current = current->left

ELSE

SET current = current->right

ENDIF

ENDWHILE

RETURN Course()

//Print courses with prerequisites using InOrder

FUNCTION PrintInOrder(Node node)

IF (node == nullptr) THEN

DISPLAY “no courses”

RETURN

ENDIF

PrintInOrder(node->left)

DISPLAY node

//IF preReqs stored in vector

IF (node.coursePreReqs.size() > 0) THEN

FOR i = 0 TO node.coursePreReqs.size() - 1

DISPLAY node.coursePreReqs[i]

IF (i < node.coursePreReqs.size() - 1) THEN

DISPLAY ","

ENDIF

ENDFOR

ENDIF

PrintInOrder(node->right)

**END**

//Project One PrintMenu

**START**

WHILE (choice != 9)

DISPLAY Menu

DISPLAY 1: Load File

DISPLAY 2: Print All Courses In Order

DISPLAY 3: Print Single Course

DISPLAY 9: EXIT

DISPLAY Enter choice:

GET cin choice

switch(choice)

case 1:

String filename

DISPLAY Enter file path

GET line

loadFile(filePath)

break

case 2:

call method to display all courses

break

case 3:

call method to search for course

break

DISPLAY Goodbye.

**END**

## Example Runtime Analysis

| **Vector** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 1 | n | n |
| **if the course is the same as courseNumber** | 1 | n | n |
| **If 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) |

| **Hash Table** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create class HashTable** | 1 | 1 | 1 |
| **SET size using pointer** | 1 | 1 | 1 |
| **RESIZE nodes(size)** | 1 | 1 | 1 |
| **SET key and compute hash** | 1 | 1 | 1 |
| **IF (node != nullptr && node.key != UINT\_MAX && COMPARE == 0)** | 1 | 1 | 1 |
| **RETURN node** | 1 | 1 | 1 |
| **IF node == nullptr** | 1 | 1 | 1 |
| **RETURN object** | 1 | 1 | 1 |
| **WHILE node != nullptr** | 1 | n | n |
| **IF (node.key != UINT\_MAX && COMPARE == 0)** | 1 | n | n |
| **SET node to node.next** | 1 | n | n |
| **Total Cost** | | | 11n + 1 |
| **Runtime** | | | O(n) |

| **BST** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **SET current node = root** | 1 | 1 | 1 |
| **WHILE current != nullptr** | 1 | n | n |
| **If current->Course.courses == courses** | 1 | n | n |
| **If current < current->Course.courses** | 1 | n | n |
| **SET current = current->left ELSE current->right** | 1 | n | n |
| **RETURN current->course** | 1 | 1 | 1 |
| **RETURN course()** | 1 | 1 | 1 |
| **Total Cost** | | | 4n + 1 |
| **Runtime** | | | O(n) |

**Evaluate each structure**

The vector allows for dynamic resizing and has fast access to elements by indexing. However, the array must be sorted and appending elements to the front or middle would require shifting of other elements resulting in O(n) complexity time. A hash table has a fast runtime of O(1) for inserting, deleting, and searching for elements using a unique key value to store elements. However, collisions may occur where using chaining causes a runtime of O(n) to search through a linked list of chained elements. A binary search tree (BST) keeps elements sorted in nodes allowing for in-order traversal and has a fast runtime of O(log n) for inserting, deleting, and searching. However, creating a BST is harder than a vector or hash table and traversal must be correct to receive the proper output.

After analyzing all three data structures of vector, hash table, and binary search tree (BST) I will use the BST data structure. A BST maintains sorted order during insertion and allows me to use in-order traversal to print all the courses in alphanumeric order. A balanced BST has a search runtime that is O(log n) which is considered fast. A BST allows efficient memory storage by using nodes that are proportional to the number of elements in the tree. Using a BST is the best option for this program because it maintains a sorted order, has fast searching runtime, and uses memory efficiently and dynamically.