Lars Dela Cruz

CS-300: Analysis and Design

Southern New Hampshire University

***Project One Pseudo:***

**Hash Table Pseudocode**

The purpose of the following pseudocode is to implement a hash table with logic that allows the searching, modification and printing of the instances in the hash table.

-

Use fstream to open file.

Call to open file

IF return value = -1 then file is not found

ELSE file is found

While file is not end of file

Read lines

IF there are less than 2 values in line ERROR

ELSE reads parameters

IF 3 or more parameters

IF 3rd parameter or more is on 1st parameter resume

ELSE return ERROR

END

**Create/Hold Course Objects**

INITIALIZE vector nodes

Create variables for course objects

Create root and set to null

Create left and right node variables

OPEN file

WHILE open

Read file

Parse

IF root == null

Search for course

IF course found

Create course object

Place on right node

IF course not found

PRINT error

END

**Print**

Create root variable and set to null

Create left and right node variables

Open File

WHILE open

Read file

Parse lines

IF root == null

Check course info

IF root == null

Check for prerequisites

Place prereq to right node

IF parameters are less than 2

Place course to left node

Print Error msg

ELSE

Add name, # and prereq info to right node

PRINT Validation

END

**Tree Data Pseudocode**

**Read file:**

Call to open file with fstream

READ each line

IF less than 2 values in line return an error

ELSE read

IF 3rd or more parameters is in the 1st

ELSE return an error

END

**Add nodes and tree:**

DEFINE Binary tree class

CREATE root that points to nul

Create insert method

IF root null, current course is root

ELSE course number less than root, place left

IF left null add course number

ELSE

IF course number is greater than, place right

IF course number is less than, place left

Else course number is greater than root, place right

IF right null add course number

ELSE

IF course number is greater than, place right

IF course number is less than, place left

**Prereq Pseudocode**

Create function with 2 parameters

Open file

Read data

Parse each line

Check course title

Check course number

IF file runs into no errors

Check if prereq exists in course file

CreateCourseObj parameters

Initialize variables for courses, and read file

Open the file to read

WHILE file is open

Storecourse obj in vector data

Open file

WHILE open

PRINT course info

Store data

**Pseudocode Menu**

Create an integer for switch statement and set to 0

Create WHILE loop

WHILE choice is NOT equal to 4

Print choices (Load, print courses or exit menu)

PRINT 1 loadCourses

PRINT 2 PrintSorted. Call function to print sorted list

PRINT 3 Print course id/info

PRINT 4 END program

**Alphanumeric order Pseudo**

CREATE sorting string

CREATE character that sets length +1

CREATE a string to character array

CREATE 2 integers for alphabet and #’s

CREATE WHILE loop for if the alphabet integer is less than 97

set alphabet int value to +1

CREATE for loop

IF ‘i’ is less than 97 then set the number int to +1

ELSE set the alpha int to +1

and return sorting string

CREATE string for the classes

PRINT courses in order.

Considering we are only dealing with courses, and reading files the use of our vector would prove to be efficient enough. There's also less memory that is being used when this data structure is utilized. The main downside is that elements cannot be removed and its inability to use different data types.

The use of the hash table allows information to not only be stored but organized as well. A key is used in our project for creating, removing and calling within our data, making it synchronized but also slowing down the said synchronization. A little give and take but it does not seem like a huge problem for our project.

A tree, in this case, can store our data in nodes (either left or right) which gives the most organization out of the 3. Very similar to a hash table with the ability to expand data but usually slows down modifications.

That being said, I believe Hash tables is the appropriate pick since we are able to keep the organization that we need for this project and have our program run slower instead of a tree where the speed of modifying data will be affected instead. I also think that this choice will help in the future if/when any more functionality is requested by the client. Even then, the hit on the program's speed should not be that obvious.