Augustus Mendy

CS-300-11373-M01

Southern New Hampshire University

June 2, 2024

**5-3 Project One Milestone Three Tree Data Structure Pseudocode**

“START

//reading files

LOAD files

INITIALIZE call to open file

If (return value is -1, the file is not found)

ELSE file is found

READ each line of file

IF (less than two values in a line)

RETURN error;

ELSE IF

Parameter has a third or more values continue first line with courseNumber

(prerequisite)

ELSE

Continue to read file

CLOSE file

PRINT all courses including prerequisites courseNumber at end of line

// Tree data structure

CREATE struct that holds course information

CREATE courseId variable

CREATE courseTitle variable

CREATE prerequisite1 variable

CREATE prerequisite2 variable

CREATE prerequisiteCount variable

INITIALIZE prerequisiteCount equal to 0

DEFINE internal structure for tree node

CREATE course variable

CREATE Node \*left variable

CREATE Node \*right variable

INITIALIZE a Node

DECLARE method Node()

INITIALIZE left = nullptr

INITIALIZE right = nullptr

INITIALIZE method Node() to take a course as a parameter

Node (course aCourse): Node()

INITIALIZE course = aCourse

DEFINE class for Binary Search Tree

PRIVATE

CREATE Node\* root

CALL void addnode (Node\* node, Course course)

CALL void inOrder (Node\* node)

CALL Node\* removeNode (Node\* node, string courseId)

PUBLIC

CALL method BinarySearchTree()

CALL virtual ~ BinarySearchTree()

CALL void InOrder()

CALL void Insert(Course course)

CALL void Remove(courseId)

CALL search method(courseId)

DEFINE BinarySearchTree() default constructor

INITIALIZE housekeeping variables

SET root = nullptr

DEFINE BinarySearchTree() default destructor

CALL recurse from root and delete every node

DEFINE InOrder()

CALL inOrder function and pass root

DEFINE Insert (Course course)

IF root equals nullptr

SET root equal to new Node (course)

ELSE

this-> addNode (root, course)

DEFINE Remove(courseId)

CALL removeNode(root, courseId)

DEFINE Search() method

SET current node equal to root

//keep looping downward until bottom is reached or match is found

WHILE current node does not equal nullptr

IF current courseId and courseId is equal to 0

RETURN current courseId

IF courseId is smaller than current node traverse left

SET current equal to current -> left

ELSE courseId is larger traverse right

SET current = current->right

RETURN course

DEFINE addNode (Node\* node, Course course)

IF node courseId is greater than 0

IF node left is equal to nullptr

SET node-> left = new Node (course)

ELSE

this -> addNode (node -> left, course)

DEFINE inOrder (Node\* node)

IF node is not equal to nullptr

CALL inOrder (node -> left)

PRINT courseId, courseTitle, prerequisite1, prerequisite2

CALL inOrder (node -> right)

//Print course information and prerequisites

DEFINE displayCourse(Course course)

PRINT courseId, courseTitle, prerequisite1, prerequisite 2

RETURN”