CS 300

MODULE SIX

Project one.

1. **Resubmit pseudocode from previous pseudocode assignments and update as necessary**.

Vector Pseudocode:

// Function to count the total number of prerequisites for a course

int numPrerequisiteCourses(Vector<Course> courses, Course c) {

int count = 0

for each prerequisite p in c.prerequisites  
count += numPrerequisiteCourses(courses, courses[getIndex(courses, p)])  
return count + c.prerequisites.length

}

// Function to print a sample schedule of all courses  
void printSampleSchedule(Vector<Course> courses) {  
for each course c in courses

print(c.courseNumber + " - " + c.name)  
}

// Function to print course information and prerequisites  
void printCourseInformation(Vector<Course> courses, String courseNumber) {

for each course c in courses  
if c.courseNumber == courseNumber  
print(c.courseNumber + " - " + c.name)

if c.prerequisites.length > 0  
print("Prerequisites:")  
for each prerequisite p in c.prerequisites  
print(p + " - " + courses[getIndex(courses, p)].name)

}

Hashtable Pseudocode:

// Function to count the total number of prerequisites for a course  
int numPrerequisiteCourses(Hashtable<Course> courses, Course c) {

int count = 0  
for each prerequisite p in c.prerequisites  
count += numPrerequisiteCourses(courses, courses.get(p))  
return count + c.prerequisites.length

}

// Function to print a sample schedule of all courses  
void printSampleSchedule(Hashtable<Course> courses) {  
for each course c in courses

}

print(c.courseNumber + " - " + c.name)  
}

// Function to print course information and prerequisites  
void printCourseInformation(Hashtable<Course> courses, String courseNumber) {  
Course c = courses.get(courseNumber)  
if c != null  
print(c.courseNumber + " - " + c.name)  
if c.prerequisites.length > 0  
print("Prerequisites:")  
for each prerequisite p in c.prerequisites  
print(p + " - " + courses.get(p).name)  
}

Tree Pseudocode:

// Function to count the total number of prerequisites for a course  
int numPrerequisiteCourses(Tree<Course> courses, Course c) {  
int count = 0  
for each prerequisite p in c.prerequisites  
count += numPrerequisiteCourses(courses, courses.get(p))  
return count + c.prerequisites.length  
}

// Function to print a sample schedule of all courses  
void printSampleSchedule(Tree<Course> courses) {  
for each course c in courses.inorderTraversal()  
print(c.courseNumber + " - " + c.name)  
}

// Function to print course information and prerequisites  
void printCourseInformation(Tree<Course> courses, String courseNumber) {  
Course c = courses.get(courseNumber)  
if c != null  
print(c.courseNumber + " - " + c.name)  
if c.prerequisites.length > 0  
print("Prerequisites:")  
for each prerequisite p in c.prerequisites  
print(p + " - " + courses.get(p).name)  
}

2. **Create pseudocode for a menu**.

DECLARE file\_data\_loaded AS BOOLEAN = FALSE

DECLARE course\_bst AS BinarySearchTree

FUNCTION display\_menu()

PRINT "MENU:"

PRINT "1. Load file data into data structure"

PRINT "2. Print alphanumerically ordered list of Computer Science courses"

PRINT "3. Print course title and prerequisites"

PRINT "9. Exit"

END FUNCTION

FUNCTION load\_file\_data(file\_path)

DECLARE bst AS BinarySearchTree = process\_file(file\_path)

IF bst IS NOT NULL

SET course\_bst = bst

SET file\_data\_loaded = TRUE

PRINT "File data loaded successfully."

ELSE

PRINT "Failed to load file data."

END IF

END FUNCTION

FUNCTION print\_computer\_science\_courses()

IF NOT file\_data\_loaded

PRINT "Please load file data first."

ELSE

DECLARE computer\_science\_courses AS LIST OF Course

// Collect Computer Science courses into a list

FOR EACH course IN course\_bst.GET\_SORTED\_COURSES()

IF course.course\_number STARTS WITH "CS"

ADD computer\_science\_courses, course

END IF

END FOR

// Print Computer Science courses

PRINT "Alphanumerically ordered list of Computer Science courses:"

FOR EACH course IN computer\_science\_courses

CALL course\_bst.print\_course(course)

END FOR

END IF

END FUNCTION

FUNCTION print\_course\_info(course\_number)

IF NOT file\_data\_loaded

PRINT "Please load file data first."

ELSE

DECLARE course\_found AS Course = course\_bst.SEARCH(course\_number)

IF course\_found IS NULL

PRINT "Course not found."

ELSE

PRINT "Course Information:"

CALL course\_bst.print\_course(course\_found)

END IF

END IF

END FUNCTION

// Main program

DECLARE choice AS INTEGER

LOOP

CALL display\_menu()

INPUT "Enter your choice: " INTO choice

SWITCH choice

CASE 1:

DECLARE file\_path AS STRING

INPUT "Enter file path: " INTO file\_path

CALL load\_file\_data(file\_path)

BREAK

CASE 2:

CALL print\_computer\_science\_courses()

BREAK

CASE 3:

DECLARE course\_number AS STRING

INPUT "Enter course number: " INTO course\_number

CALL print\_course\_info(course\_number)

BREAK

CASE 9:

PRINT "Exiting program."

EXIT

DEFAULT:

PRINT "Invalid choice. Please enter a valid option."

END SWITCH

END LOOP

3. **Design pseudocode that will print out the list of the courses in the Computer Science program in alphanumeric order.**

DEFINE STRUCT Course

STRING course\_number

STRING course\_title

LIST OF STRING prerequisites

END STRUCT

DEFINE CLASS TreeNode

Course course

TreeNode left

TreeNode right

FUNCTION CONSTRUCTOR(course)

this.course = course

this.left = NULL

this.right = NULL

END FUNCTION

END CLASS

DEFINE CLASS BinarySearch

Tree TreeNode root

FUNCTION INSERT(course)

IF root IS NULL

root = NEW TreeNode(course)

ELSE

CALL insert\_recursive(root, course)

END IF

END FUNCTION

FUNCTION insert\_recursive(node, course)

IF course.course\_number < node.course.course\_number

IF node.left IS NULL

node.left = NEW TreeNode(course)

ELSE

CALL insert\_recursive(node.left, course)

END IF

ELSE

IF node.right IS NULL

node.right = NEW TreeNode(course)

ELSE

CALL insert\_recursive(node.right, course)

END IF

END IF

END FUNCTION

FUNCTION SEARCH(course\_number)

RETURN search\_recursive(root, course\_number)

END FUNCTION

FUNCTION search\_recursive(node, course\_number)

IF node IS NULL

RETURN NULL

ELSE IF node.course.course\_number == course\_number

RETURN node.course

ELSE IF course\_number < node.course.course\_number

RETURN search\_recursive(node.left, course\_number)

ELSE

RETURN search\_recursive(node.right, course\_number)

END IF

END FUNCTION

FUNCTION PRINT\_COURSES()

CALL print\_inorder(root)

END FUNCTION

FUNCTION print\_inorder(node)

IF node IS NOT NULL

CALL print\_inorder(node.left)

CALL print\_course(node.course)

CALL print\_inorder(node.right)

END IF

END FUNCTION

FUNCTION print\_course(course)

PRINT "Course Number: " + course.course\_number

PRINT "Course Title: " + course.course\_title

IF LENGTH(course.prerequisites) > 0

PRINT "Prerequisites: " + JOIN(course.prerequisites, ", ")

ELSE

PRINT "Prerequisites: None"

END IF

PRINT "-----------------------"

END FUNCTION

FUNCTION GET\_SORTED\_COURSES()

DECLARE sorted\_courses AS LIST OF Course

CALL collect\_inorder(root, sorted\_courses)

RETURN sorted\_courses

END FUNCTION

FUNCTION collect\_inorder(node, sorted\_list)

IF node IS NOT NULL

CALL collect\_inorder(node.left, sorted\_list)

ADD sorted\_list, node.course

CALL collect\_inorder(node.right, sorted\_list)

END IF

END FUNCTION

END CLASS

DEFINE FUNCTION process\_file(file\_path)

DECLARE bst AS BinarySearchTree

DECLARE line\_number AS INTEGER = 0

TRY

OPEN file at file\_path FOR READING AS file

FOR EACH line IN file

INCREMENT line\_number

SPLIT line BY comma INTO parts

IF LENGTH(parts) < 2

PRINT "Error on line " + line\_number + ": Less than two parameters."

CONTINUE

END IF

DECLARE course\_number AS STRING = TRIM(parts[0])

DECLARE course\_title AS STRING = TRIM(parts[1])

DECLARE course\_prerequisites AS LIST OF STRING

IF LENGTH(parts) > 2

course\_prerequisites = TRIM(parts[2:])

END IF

DECLARE new\_course AS Course

new\_course.course\_number = course\_number

new\_course.course\_title = course\_title

new\_course.prerequisites = course\_prerequisites

bst.INSERT(new\_course)

PRINT "All courses have been processed and inserted into the BST."

RETURN bst

CATCH Exception AS e

PRINT "An error occurred while reading the file: " + e.message

RETURN NULL

FINALLY

CLOSE file

END FUNCTION

// Main execution

DECLARE file\_path AS STRING = "path/to/course\_data.txt"

DECLARE course\_bst AS BinarySearchTree = process\_file(file\_path)

IF course\_bst IS NOT NULL

PRINT "Binary search tree has been created successfully."

PRINT "Course Information:"

// Print courses in sorted order

DECLARE sorted\_courses AS LIST OF Course = course\_bst.GET\_SORTED\_COURSES()

FOR EACH course IN sorted\_courses

CALL course\_bst.print\_course(course)

END FOR

ELSE

PRINT "Failed to create the binary search tree."

END