class Course {

String couseNumber

String courseName

List<String> prerequisites

// Constructor

Course(String courseNumber, String courseName) {

this.courseNumber = courseNumber

this.courseName = courseName

this.prerequisites = new List<String>()

}

// Add prerequisites

void addPrerequisites(String prerequisites) {

prerequisites.add(prerequisite)

}}

**// Binary Search Tree**

class Node {

Course course

Node left

Node right

Node(Course course) {

this.course = course

this.left = null

this.right = null

}}

class BinarySearchTree {

Node root

BinarySearchTree() {

this.root = null

}

// Insert method for adding a new course to the BST

void Insert(Course course) {

// if the tree is empty, set the root to the new course

if (root == null) {

root = new Node(course)

} else {

addNode(root, course)

}}

void addNode(Node\* node, Bid bid) {

if (course.courseNumber < currentNode.course.courseNumber) {

if (currentNode.left == null) {

currentNode.left = new Node(course)

} else {

addNode(currentNode.left, course)

}

} else {

// Insert into the right subtree

if (currentNode.right == null) {

currentNode.right = new Node(course)

} else {

addNode(currentNode.right, course)

}}}}

**// Design pseudocode to define how the program opens the file, reads the data from the file, parses each line, and checks for file format errors & Design pseudocode to show how to create course objects and store them in the appropriate data structure.**

void loadCourseData(String csvPath, BinarySearchTree\* bst) {

// Print CSV path

print(“Loading CSV file: “ + csvPath)

// Initialize the CSV parser using the given path

csvParser file = csvParser(csvPath)

**// Reads the data from the file, parses each line.**

for (int i = 0; I < file.rowCount(); i++) {

String courseNumber = file[i][0]

String courseName = file[i][1]

Course course = new Course(courseNumber, courseName)

// **Check for file errors)**

if (file[i].size() < 2) {

print(“Error: line cannot have more than 2 prerequisites”)

continue

}

**// check if any prerequisite that is provided on a line exists as a course in the file. In other words, any prerequisite at the end of a line must have another line in the file that starts with that course number.**

for (int j = 2; j < file[i].size(0; j++) {

String prerequisite = file[i][j]

if (!isCourseInFile(prerequisite)) {

print(“Error: Prerequisite “ + prerequisite + " does not exist in the file.")

} else {

course.addPrerequisite(prerequisite)

}}

// Insert into the binary search tree

bst->Insert(course)

}

}}

**// Design pseudocode that will print out course information and prerequisites**

void printAllCourses(Node currentNode) {

if (currentNode != null) {

// Traverse the left subtree

printAllCourses(currentNode.left)

// Print the course information

print(“CourseNumber: “ + currentNode.course.courseNumber)

print(“Course Name: “ + currentNode.course>courseName)

if (currentNode.course.prerequisites.size() > 0) {

print(“Prerequisites: “)

for each prerequisite in currentNode.course.prerequisites “

print(“ – “ + prerequisite)

}

} else {

print(“No prerequisites”)’

}

// traverse right

printAllCourses(currentNode.right)

}}

// Pseudo code

void displayMenu() {

println(“Menu: “)

println(“Option 1: Load the file data into the data structure. “)

println(“Option 2: Print an alphanumerically ordered list. “)

println(“Option 3: Print the course title and the prerequisites for any individual course.”)

printlln(“Option 9: exit Menu. “)

void main() {

BinarySearchTree\* bst = newBinarySearchTree()

int choice = 0

while choice !=9 {

displayMenu()

println(“Switch case choice: “)

choice = get input

switch choice {

case 1:

loadCoursedata(filePath, bst)

break

case 2:

if bst->root == null {

print(“No course data found.”)

} else {

printAllCoursesAlphanumeric( (bst->root)

}

break

case 3:

if bst->root == null {

println(“No course data available.”)

} else {

println(“Enter course number: “)

String courseNumber = get user input

printCourseInfo(bst->root, courseNumber)

}

break

case 9:

print(“Exiting program.”)

break

default:

print(“Invalid option.”)

} } }

void printAllCoursesAlphanumeric(Node\* currentNode) {

if (currentNode != null) {

// traverse the left subtree

printAllCourses(currentNode->left)

// print the course information

print(“Course Number: “ + currentNode->course.courseNumber)

print(“Course Title: “ + currentNode->course.courseName)

// Traverse the right subtree

printAllCourses(currentNode->right)

} }

void printCourseInfo(Node\* currentNode, String courseNumber) {

if (currentNode == null) {

println(“Course not found.”)

return

}

if (courseNumber == currentNode->course.courseNumber) {

// print course information

print(“Course Number: “ + currentNode->course.courseNumber)

print(“Course title: “ + currentNode->course.courseName)

// Print prerequisites if any

if (currentNode->course.prerequisites.size() > 0) {

print(“Prerequisites: “)

for each prerequisite in currentNode->course.prerequisites {

print(“ – “ + prerequisite)

}

} else {

print(“No prerequisites”)

}

} else if (courseNumber < currentNode->course.courseNumber) {

// Search the left subtree

printCourseInfo(currentNode->left, courseNumber)

} else {

// Search the right subtree

printCourseInfo(currentNode->right, courseNumber)

} }