Project One Milestone One: Guidelines and Rubric

Web Page

**1 rst pseudocode…**.

// Define a structure for Course

struct Course {

String courseNumber

String name

Vector<String> prerequisites

}

// Main procedure to manage course data, will be the actual Main

Procedure Main()

// Load courses from a file into a vector

Vector<Course> courses = LoadCoursesFromFile("courses.txt")

// Prompt the user to enter a course number

String courseNumber = GetInput("Enter course number to search: ")

// Search for the course in the vector

Course foundCourse = searchCourse(courses, courseNumber)

// Check if the course was found and print details

if foundCourse is not null then

PrintCourseInfo(foundCourse)

else

Print("Course not found.")

// Function to load courses from a file, ensuring no duplicate course numbers

// vector should work fine, there should not be a lot of insertions or deletions so vectors should be more //than efficient

Function Vector<Course> LoadCoursesFromFile(String filePath)

Open file with path filePath

Vector<Course> courses

Set<String> courseNumbers // To track course numbers and ensure no duplicates

while not end of file

String line = Read next line from file

if IsValidLine(line)

Course course = ParseLineToCourse(line, courses)

if course is not null

if courseNumbers.contains(course.courseNumber)

Print("Error: Duplicate course number found -> " + course.courseNumber)

else

courses.add(course)

courseNumbers.add(course.courseNumber)

//error check

else

Print("Error: Line format is incorrect -> " + line)

Close file

return courses

// Function to validate the line format

Function Boolean IsValidLine(String line)

// Split line into parts and check for at least two elements (course number and name)

String[] parts = line.split(",")

return (parts.length >= 2)

// Function to parse a line to a Course object, checking prerequisites, making sure the courses were //taken, possible add to this in the future to incorporate and order the courses may have to be taken in?

Function Course ParseLineToCourse(String line, Vector<Course> existingCourses)

String[] parts = line.split(",")

Course course = new Course()

course.courseNumber = parts[0]

course.name = parts[1]

// Add prerequisites, checking each one to ensure it exists in the list of courses

for i from 2 to parts.length - 1

if existingCourses.contains(parts[i])

course.prerequisites.add(parts[i])

//error check below

else

Print("Error: Prerequisite course not found -> " + parts[i])

return course

// Function to search for a course by course number

Function Course searchCourse(Vector<Course> courses, String courseNumber)

// Iterate through courses to find a match by course number

for course in courses

if course.courseNumber == courseNumber

return course

return null

// Function to print course information and prerequisites

Procedure PrintCourseInfo(Course course)

// Display the basic course info and list all prerequisites

Print("Course Number: " + course.courseNumber)

Print("Name: " + course.name)

if course.prerequisites.size() > 0

Print("Prerequisites: ")

for prerequisite in course.prerequisites

Print(prerequisite)

else

Print("No prerequisites.")………………………

**Second pseudocode**

**// Function to load course data from a file into a hash table**

**void loadDataFromFile(String filename) {**

**// Open the file**

**file = open(filename)**

**// Initialize an empty hash table to store courses**

**HashTable<Course> courses = new HashTable<Course>()**

**// Read each line from the file**

**while (line = file.readLine()) {**

**// Split the line by commas to extract course information**

**tokens = line.split(',')**

**// Check if the line has at least two parameters (course number and name)**

**if (tokens.length < 2) {**

**print("Error: Invalid line format")**

**continue**

**}**

**// Extract course number and name**

**courseNumber = tokens[0]**

**courseName = tokens[1]**

**// Initialize an empty list for prerequisites**

**List<String> prerequisites = new List<String>()**

**// Add any prerequisites to the list**

**for (i = 2; i < tokens.length; i++) {**

**prerequisites.add(tokens[i])**

**}**

**// Create a new course object**

**Course course = new Course(courseNumber, courseName, prerequisites)**

**// Store the course object in the hash table**

**courses.put(courseNumber, course)**

**}**

**// Close the file**

**file.close()**

**// Validate the prerequisites**

**validatePrerequisites(courses)**

**}**

**// Function to validate prerequisites**

**void validatePrerequisites(HashTable<Course> courses) {**

**// For each course in the hash table**

**for (course in courses.values()) {**

**// For each prerequisite of the course**

**for (prerequisite in course.prerequisites) {**

**// Check if the prerequisite exists in the hash table**

**if (!courses.containsKey(prerequisite)) {**

**print("Error: Prerequisite " + prerequisite + " for course " + course.courseNumber + " does not exist")**

**}**

**}**

**}**

**}**

**// Function to print course information and prerequisites**

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

**// Check if the course exists in the hash table**

**if (!courses.containsKey(courseNumber)) {**

**print("Course " + courseNumber + " not found")**

**return**

**}**

**// Retrieve the course object from the hash table**

**Course course = courses.get(courseNumber)**

**// Print the course information**

**print("Course Number: " + course.courseNumber)**

**print("Course Name: " + course.courseName)**

**// Print the prerequisites**

**if (course.prerequisites.isEmpty()) {**

**print("No prerequisites")**

**} else {**

**print("Prerequisites: " + course.prerequisites.join(", "))**

**}**

**}**

**Third pseudocode**

**Project 1 milestone 3**

**// Pseudocode for loading data into a binary search tree, validating the data, and printing course information**

**// Function to load courses from a file and insert them into a BST**

**function loadCoursesFromFile(filename):**

**open file with name filename**

**for each line in the file:**

**split the line by commas into an array**

**if the array length is less than 2:**

**print "Error: Each line must have at least a course number and a course name"**

**continue to the next line**

**courseNumber = array[0]**

**courseName = array[1]**

**prerequisites = array[2 to end] // This captures all prerequisites**

**create a Course object with courseNumber, courseName, and prerequisites**

**insertCourseIntoBST(root, course) // Insert the Course object into the BST**

**close the file**

**// Function to validate that all prerequisites exist as courses in the file**

**function validatePrerequisites(courses):**

**for each course in courses:**

**for each prerequisite in course.prerequisites:**

**if prerequisite is not in courses:**

**print "Error: Prerequisite " + prerequisite + " for course " + course.courseNumber + " does not exist"**

**// Function to insert a course into the BST**

**function insertCourseIntoBST(root, course):**

**if root is null:**

**root = course**

**else if course.courseNumber < root.courseNumber:**

**if root.left is null:**

**root.left = course**

**else:**

**insertCourseIntoBST(root.left, course)**

**else:**

**if root.right is null:**

**root.right = course**

**else:**

**insertCourseIntoBST(root.right, course)**

**// Function to print course information and prerequisites**

**function printCourseInformation(root, courseNumber):**

**if root is null:**

**print "Course not found"**

**return**

**if courseNumber < root.courseNumber:**

**printCourseInformation(root.left, courseNumber)**

**else if courseNumber > root.courseNumber:**

**printCourseInformation(root.right, courseNumber)**

**else:**

**print "Course Number: " + root.courseNumber**

**print "Course Name: " + root.courseName**

**if root.prerequisites is not empty:**

**print "Prerequisites: "**

**for each prerequisite in root.prerequisites:**

**print " " + prerequisite**

**// Example usage**

**courses = loadCoursesFromFile("courses.txt")**

**validatePrerequisites(courses)**

**printCourseInformation(courses, "CSCI300")**

|  |  |  |  |
| --- | --- | --- | --- |
| MATH201 | Discrete Mathematics |  |  |
| CSCI300 | Introduction to Algorithms | CSCI200 | MATH201 |
| CSCI350 | Operating Systems | CSCI300 |  |
| CSCI101 | Introduction to Programming in C++ | CSCI100 |  |
| CSCI100 | Introduction to Computer Science |  |  |
| CSCI301 | Advanced Programming in C++ | CSCI101 |  |
| CSCI400 | Large Software Development | CSCI301 | CSCI350 |
| CSCI200 | Data Structures | CSCI101 |  |