Daniel Loranger

CS-300

Project 1.2

*In this milestone, you will continue working on a task for ABC University (ABCU). ABCU is looking for software to help its computer science advisors access course information for students. To do this, you will use what you have learned about data structures. In this milestone, you will create pseudocode for the Computer Science department at ABCU. This code will demonstrate your ability to import data from a file and store it in the tree data structure.*

*For this milestone, you will create pseudocode to load data into the* ***tree data structure****. There is no programming work in this milestone. You will develop pseudocode to help you carry your design in Project One.*

1. **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.** The Course Information document linked in the Supporting Materials section contains all the information about how the courses required in the Computer Science curriculum for ABCU will be stored in the data file. Each line will consist of information about a single course including the course number, title, and prerequisites. The Course Information document includes the course data and a diagram of how the program will execute.

Your pseudocode will need to validate the sample file to make certain it is formatted correctly and check for the following items:

* There are at least two parameters on each line. Some courses may not have any prerequisites.
* 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.

1. **Design pseudocode to show how to create course objects and store them in the appropriate data structure.** Your pseudocode should show how to create course objects so that one course object holds data from a single line from the input file. Knowing the file format will help you parse and store each token of data in the appropriate course object instance variable. You should store each course object in the binary search tree data structure. Once the entire file has been processed, the binary search tree data structure will have multiple course objects, one per line in the file. Hint: You will need a loop to process all lines from the file and a struct to hold all the data from a single course.
2. **Design pseudocode that will print out course information and prerequisites.** Pseudocode for printing course information using a vector data structure is provided as an example in the Pseudocode Document linked in the Supporting Materials section. Develop the pseudocode for printing course information for the tree data structures using the base code that has been provided.



* PseudoCode

//#includes, etc to setup the environment

// create a structure that contains each course instance details

Struct Course{

courseID: string

courseName: string

numPrereq: int

preReqList: list<string>

//create a constructor to initialize to empty fields.

Course(): courseID=””;courseName=””;numPrereq =0; preReqList =””

}

Class BinaryTree{}

- struct binaryTree

course

right

left

- parseCSVData()

- testCSVPath()

- printCourse() &Course

+ userCourseSearch(): &Course

+ printAllCourses()

// main behavioral loop

Main() {

create a named binary tree of type courses (struct type)

//parse the input data file

BinaryTree::parseCSVData();

//request the user search criteria and execute the search

BinaryTree::userCourseSearch()

} // end main

// function that is designed to specifically accept the input of a CSV file

// which holds a unique format as specified by the customer, need to parse

// the data and also perform validations on each entry when its read.

BinaryTree::parseCSVData()

//read in the CSV file to memory

BinaryTree::testCSVPath()

//open the csv file read only

//For each line

// validate the contents for validity (what constitutes a valid entry?)

test each preReq by searching the entire file for a courseID that matches the prereq value

//fail if match doesn’t exist.

// if valid

//parse the data into a new course(struct) object (how to handle duplicates?)

// first field goes to course.courseID field

// second field goes to course.courseName field

// remaining data is the prerequisite list in CSV format

// count the number of commas and set class.numPrereq

//compute the hash key of the new course based on courseID

//Look up the key in the hash table

//append new course to the hash table list at the given key

}

// helper function that handles OS specific file path handling, etc. to ensure the provided

//CSV path is valid for use.

BinaryTree::testCSVPath(){

//determine path, either user input or predefined default path

//test the path for validity

}

// dedicated function that is responsible for accepting as input a course ID

// then outputting the data in the format as specified by the customer.

BinaryTree::printCourse(course){

//pretty print the course details based on requirements

// function that accepts a user input course ID

BinaryTree::userCourcsSearch(&course)

{

//Request the user to enter course ID if not provided (null) to search for

Get the users input

//Validate the user input

//Start at the current node and compare the courseID fields (ASCII compare)

//If course ID matches the root ID

PrintCourse()

return

// else if courseID <= the current node

If left != Null

//Travel the left path

//Recursively check the path

BinaryTree:: userCourcsSearch (left pointer){

Else

Return

Else

If right != Null

//Travel the right path

//Recursively check the path

BinaryTree:: userCourcsSearch (right pointer){

Else

Return

}