**Introduction to Computer Science – 150005**

**Homework Assignment #8**

**Pointers**

**Comments:**

1. Use meaningful variable names
2. Comment each program (including a comment before the main program explaining its purpose and how it works.
3. Be careful on code readability and appearance (indentation)
4. Make sure to compute exactly what is requested in each question.
5. You should check correctness of input (in this exercise, the following inputs are incorrect: negative number of lines, stations, and length of travel times, and selection choice not between 1 and 7).
6. Submit the solution according to the directions in moodle.

The Bus-Lev bus company wants to keep computer records of its bus lines. The company has a number of different lines, and for each line, the company needs to record the line number, the number of stations in its route, and the estimated travel time from its starting point to its destination.

The company is interested in getting the following information based on what is stored in the system:

* Given a line number, does it exist in the system.
* What is the average travel time of the company’s buses.
* What is the average number of stops for its lines.
* Which line has the shortest route.
* Print all the data in the system.

In addition, the company wants to be able to update the system. Therefore the program needs to handle the following:

* Add a new bus line.
* Delete a bus line.

You will write a program that contains the following data and actions:

The data will be kept in a two-dimensional array called *buses***.** The maximum number of bus lines that can be recorded in the system is 10. The amount of data for each bus line is exactly 3, where the first column records its bus line number, the second its number of stops, and the third, its travel time. The matrix will be kept sorted according to the bus line numbers.

The program should implement the following functions:

1. addLine – adds a new bus line to the system. The function receives the matrix buses and the current number of lines in the system, as well as information for the new line: line number, number of stations, and duration. The function adds the new line and its information to the system. If the bus line already exists in the system, the function prints an error message and does not modify the system’s database. (Note, after inserting the new bus line, the system is still sorted by bus line numbers.)
2. deleteLine – Removes an existing line from the system. The function receives the matrix buses and the current number of lines in the system and the line number to be removed. The function deletes all the information regarding the given bus line from the system. The system must remain without holes in the matrix. This means that all bus lines that appear after the bus line to delete must be shifted up to replace the one that has been deleted.  
   Note, in the event that the given bus line to be deleted does not exist in the system, the function makes no changes to the system with no error message and no request to input a correct line number.
3. print – Prints all information stored in the system. The function receives the matrix buses and the current number of lines in the system, and prints for each bus line its line number, number of stops, and duration.
4. search – Searches for a given line number in the system. The function receives the matrix buses and the current number of lines in the system and a line number and returns the row number where the line is stored in the system. If the line number does not exist in the system, the function returns -1.
5. averageTravel – Calculates the average duration of the bus lines that exist in the system. The function receives the matrix buses and the current number of lines in the system and prints the average duration for all the lines.
6. averageStops – Calculates the average number of stops of the bus lines that exist in the system. The function receives the matrix buses and the current number of lines in the system and prints the average number of stops for all the lines rounded to its lower integer value.
7. shortest – Find the line with the shortest travel time. The function receives the matrix buses and the current number of lines in the system and returns the bus line with the shortest duration. In the event of an empty system, the function returns -1.

The main program should declare the system matrix (see code below). It then continuously asks the user to input a desired action (a number from 0-7) and performs the following:

* On input 0, the program terminates.
* On input 1, the main program reads in the details for a new bus line and calls the function addLine and then the function print.
* On input 2, the main program reads in the bus line number to delete and calls the function deleteLine and then the function print.
* On input 3, the main program calls the function print.
* On input 4 the main program reads in the bus line number to search and calls the function search and prints the value returned.
* On input 5, the main program calls the function averageTravel and prints the value returned.
* On input 6, the main program calls the function averageStops and prints the value returned.
* On input 7, the main program calls the function shortest and prints the value returned.

1. For action 1 (add bus line), the program prints the following, after each line printed it reads in a value from the user (total of 3 values corresponding to the line number, number of stops and, and duration), and prints the updated system information.  
   enter the line to add

enter the number of stops

enter the duration of the ride

1. For action 2 (delete bus line), the program prints the following, reads in the line to delete and prints the updated system information.  
   enter the line to delete
2. For action 3 (print system), the program prints the system information as follows:
   1. All information about a bus line (number, number of stops, and duration) separated by a space on the same line. All bus lines on a separate line.
   2. For example, for 2 lines 2 and 5 with line #2 having 10 stops and taking 55 minutes and line #5 12 stops and 70 minutes, the output is:  
      2 10 55  
      5 12 70
3. For action 4 (search bus line), the program prints following, reads the line number, and prints the row number for the line if it is found and -1 otherwise. (Note, the main program needs to print the value returned from the function called).  
   enter the line to search for
4. For action 5 (calculate average travel time), the program prints the value returned from the function averageTravel.
5. For action 6 (calculate average number of stops), the program prints the value returned from the function averageStops.
6. For action 7 (shortest route), the program prints the value returned from the function shortest.
7. For action 0 (exit), the program exits without further output.
8. For any other number, the program prints:  
   ERROR  
   enter 0-7 .

Some advice: Write the program using top-down design. Start with the main program and then write the individual functions. You should use the skeleton code provided below. Handle one action/function at a time and make sure that it works before continuing to the next (you may want to starts with exit first and print).

In addition to the main program, the functions addLine and print have also been provided.

Comments

* Before you begin writing the program, it is worthwhile to go over the provided main program to understand what the program is expected to do.
* It is also worthwhile to go over the functions addLine and print to understand how to use the program.
* It is recommended to submit a partial program in the event you were not able to get a complete working program. A partial program will get an appropriate grade on the manual grading.

#define MAX 10

#include <iostream>

using namespace std;

enum choices {EXIT, ADD, DELETE, PRINT, SEARCH, AVG\_TRAVEL, AVG\_STOPS, SHORTEST\_TRAVEL};

// adds a new bus line to the array buses

// parameters buses - the 2D array that holds the database of buses

// numOfBuses - the total number of buses stored in the array buses

// line - the line being added

// stops - the number of stops in the line to be added

// duration - the duration of the line to be added

// function updates the array buses to contain the new line being added

// and updates numOfBuses to reflect the number of buses stored in the array

// buses after the new bus line was added

void addLine(int buses[][3], int & numOfBuses, int line, int stops, int duration){

// checking for errors

if (numOfBuses == MAX){

cout << "Error! Cannot add any more buses\n";

return;

}

if (search(buses, numOfBuses, line) != -1){

cout << "Error! Bus number " << line << " already exists\n";

return;

}

// shifting right all bus lines that are larger than the bus line to be

// inserted. allows for placing new bus line in its correct position

int i = numOfBuses;

while ((i > 0) && buses[i-1][0] > line)){

for(int j=0; j<3; j++)

buses[i][j] = buses[i-1][j];

// \*((\*(buses+i))+j) = \*((\*(buses+i-1))+j);

i--;

}

// correct index to place new bus is found

// inserting new bus line in correct position (index)

buses[i][0] = line;

buses[i][1] = stops;

buses[i][2] = duration;

numOfBuses++; // updates numOfBuses since one more bus was added to array

}

// prints out the buses that are saved in the database

void print(int buses[][3], int numBuses){

for(int i=0;i<numBuses;i++){ // for each bus that is present in the database

for(int j=0;j<3;j++){ // prints the 3 pieces of data of the bus

cout << buses[i][j] << ‘ ‘;

}

cout << endl;

}

}

int main(){

int buses[MAX][3]; // 2D array of buses -

// 10 rows to hold 10 different bus lines

// each row holds bus number, number of stops,

//and duration of ride

int numBuses = 0; // number of buses that is currently stored in the array bus

int line;

int stops;

int durationOfRide;

int choice;

do {

cout << "enter 0-7" << endl;

cin >> choice;

switch(choice){

case ADD : // add a line to the array of buses

cout << "enter the line to add" << endl;

cin >> line;

cout << "enter the number of stops" << endl;

cin >> stops;

cout << "enter the duration of the ride" << endl;

cin >> durationOfRide;

addLine(buses, numBuses, line, stops, durationOfRide);

print(buses,numBuses);

break;

case DELETE : // delete a line from the array of buses

cout << "enter the line to delete" << endl;

cin >> line;

deleteLine(buses, numBuses, line);

print(buses,numBuses);

break;

case PRINT: // print all lines

print(buses,numBuses);

break;

case SEARCH : // search for a particular line

cout << "enter the line to search for" << endl;

cin >> line;

cout << search(buses, numBuses, line)<<endl;

break;

case AVG\_TRAVEL : // calculate average travel time of all buses

cout << averageTravel(buses, numBuses)<<endl;

break;

case AVG\_STOPS: // calcuate average stops of all buses

cout << averageStops(buses, numBuses)<<endl;

break;

case SHORTEST\_TRAVEL: // calculate the bus with the shortest travel time

cout << shortest(buses,numBuses)<<endl;

break;

case EXIT: break; // exit the program

default : cout << "ERROR"<<endl;

}// switch

} while (choice != 0);

return 0;

}