CS 201, Spring 2022

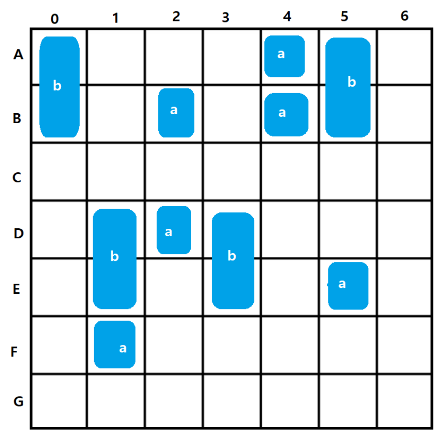
Homework Assignment 1

Due: 23:59, March 22, 2022

NOTE: Before you start the homework, make sure that you study Recitation 1 slides. Remember that your code needs to run smoothly and without any memory leaks on the Dijkstra server before you submit it to Moodle. Otherwise, a considerable grade penalty will be applied.

NOTE: You are only allowed to use dynamic memory allocation in this homework

In this homework, you will design a parking lot management system. The system will allow a user to create parking lots of different sizes and mark individual slots as occupied by certain vehicles. In the world that this parking lot management system is going to be used, there are only two types of cars: light-duty (type a), and heavy-duty (type b). While light-duty cars occupy only a single slot, heavy-duty cars occupy two parking slots in the vertical direction (along columns). For visual purposes, we also assume that the parking lot is a 2D grid with rows marked with the letters of the English alphabet and with columns marked with numbers. A visual illustration of a sample parking lot is displayed in the figure below with light cars marked with 'a' and heavy cars marked with 'b'.



This particular parking lot has 7 rows and 7 columns. But in your program, you will be required to create multiple parking lots with varying sizes. Your program should contain the following functionalities, detailed descriptions of which will be provided in the next section.

- Create a parking lot
- Show the list of parking lots
- Park the car
- Find the car
- Show content of a parking lot

- Remove the car
- Remove a parking lot

Create a parking lot (createLot): The parking management system will allow the user to add a new parking lot by indicating the lot ID (which should be an integer), the number of rows, and the number of columns. Each parking lot needs to have a unique ID number and you need to check the collection of existing parking lots to make sure that the entered ID for the new lot is unique. If the entered ID is not unique, you need to display a warning message and not allow the operation to proceed. You also need to check that the number of rows and columns in the parking lot does not exceed 12. If any of the entered dimensions is larger than 12, you need to display a warning message and not allow the operation to proceed. If the ID, the number of rows, and the number of columns are all valid, you should add the lot and display a message indicating that the operation was a success. At the time of creating the parking lot, you should ONLY allocate a necessary amount of memory for that specific parking lot. For example, DO NOT allocate memory space necessary for a 12x12 parking lot when a 9x9 parking lot is being added to the system. As you might have guessed, memory allocation needs to be dynamic. In addition, make sure that the letters and numbers are in order. That is, only use letters [A, B, C, D, E, F, G, H, I, J, K, L] for rows and numbers [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12] for columns. (Please see the code given below and its output for the full signature of the method and format of the output.)

Show the list of parking lots (listLots): The tracking system should allow the user to see the list of all existing parking lots. For each lot, you should display the ID, dimensions, and the number of empty parking lots in the system. The order in which you display (and store) the list of parking lots is up to you, i.e., you can choose any order you please, as long as you store and display them all. (Please see the code given below and its output for the full signature of the method and format of the output.)

Park the car (parkCar): The system should allow the user to park the car in the parking lot of their choice by specifying the lot ID, location in the lot (e.g. ‘C5’), type of the car (‘a’ for light-duty and ‘b’ for heavy-duty), and a unique car plate number (integer). Note that just like every parking lot has a unique ID, every car has a unique plate number in the system. That is, the plate numbers need to be unique not only for that parking lot but for the entire system. The program should check if the corresponding parking lot ID exists in the system, if the corresponding space in the lot is empty, if that type of car can be parked at that location, and if the car plate number is unique. If any of these checks fail, the system should display a corresponding warning message and not proceed with the operation.

The general rules for parking the car are as follows: When parking a light-duty car, which only occupies a single space, the user simply provides necessary information and location where the car needs to be parked. For heavy-duty cars, which occupy 2 parking slots, along with the necessary information, the user must provide the location of the lower slot that the car will occupy. For example, in the grid below, if the user wants to park a heavy-duty car at locations B6 and C6, they need to enter C6 as the location where they want to park the car. Naturally for heavy-duty cars, the algorithm needs to check both locations to ensure that the car can indeed be parked there.

```
  1 2 3 4 5 6 7
A a + b + b a +
B b + b + b + +
C b a + + + + +
D + + b + b a +
E a a b + b + +
```

In your method always use capitalized letter-first format to indicate location e.g C4, E6, do not use formats like c4, e6, 4c, 6E, 6e, etc. (Please see the code given below and its output for the full signature of the method and format of the output.)

Find the car (findCar): The system should allow the user to find the location of the car using the car's plate number. The program should display the parking lot ID and location within that parking lot if the car is found, and if it does not exist in the system, it should display a warning message. (Please see the code given below and its output for the full signature of the method and format of the output.)

Show the contents of a parking lot (lotContents): The tracking system should allow the user to see the contents and detailed information about a specific parking lot by specifying the ID of that lot. If a lot with that ID does not exist, you should display a warning message. Otherwise, you should display the ID, dimensions, number of empty slots, list of cars, and a printout that is similar to this:

```
  1 2 3 4 5 6 7
A c + c + r r +
B r + r + r + c
C + r r + + + r
D + + r + c r +
E c r c + r r c
```

(Please see the code given below and its output for the full signature of the method and format of the output.)

Remove the car (removeCar): The system should allow the user to remove a certain car using its plate number and display a warning message if that plate number does not exist in the system. (Please see the code given below and its output for the full signature of the method and format of the output.)

Remove the parking lot (removeLot): The parking system should allow the user to remove a certain lot from the collection using the lot ID. If the lot with the specified ID does not exist, you should display a warning message and do not proceed with the removal. If it exists, you first remove all the cars from that parking lot (while displaying a message that that car has been removed) and then remove the lot itself while displaying a suitable message. (Please see the code given below and its output for the full signature of the method and format of the output.)

Below is the header file for the ParkingSystem class (ParkingSystem.h)

```
class ParkingSystem{
public:
        ParkingSystem();
        ~ParkingSystem();

        void createLot(int id, int rows, int columns);
        void removeLot(int id);
        void listLots();
        void lotContents(int id);
        void parkCar(int lotId, string location, string carType, int plateNumber);
        void findCar(int plateNumber);
        void removeCar(int plateNumber);
        // ...
        //you may define additional member functions and data members, if necessary
}
```

Below is an example of the main.cpp file that we will use to test your program.
Note: It is crucial that you do not change the 'include' statements in the main code when you create your own. We will use a different main.cpp file to test your program, but it will import the same exact libraries like this one. So the code you submit needs to work with that configuration.

```
#include <iostream>
using namespace std;
#include "ParkingSystem.h"


int main(){
        ParkingSystem L;

        L.listLots();

        // Testing add parking lot
        cout << endl;
        cout << "Testing add parking lots" << endl;
        cout << endl;

        L.createLot(980, 7, 7);
        L.createLot(666, 6, 11);
        L.createLot(222, 12, 12);
        cout << endl;

        L.createLot(434, 13, 3);
        L.createLot(222, 5, 4);
        L.createLot(301, 9, 9);
        cout << endl;

        L.createLot(301, 19, 3);
        L.createLot(205, 4, 4);
        cout << endl;
        L.listLots();
        cout << endl;

        // Testing remove parking lot
        cout << endl;
        cout << "Testing remove parking lot" << endl;
        cout << endl;

        L.removeLot(666);
        L.removeLot(101);
        L.removeLot(205);
        cout << endl;
        L.createLot(743, 6, 7);
        cout << endl;

        // Testing park car
        cout << endl;
        cout << "Testing park a car" << endl;
        cout << endl;

        L.parkCar( 980, "C3", "b", 211);
        L.parkCar( 980, "E5", "a", 141);
        L.parkCar( 980, "A3", "b", 199);
        cout << endl;

        L.parkCar( 980, "A3", "a", 199);
        L.parkCar( 980, "G1", "b", 888);
        L.parkCar( 222, "G7", "b", 200);
        L.parkCar( 222, "B2", "a", 156);
        cout << endl;

        L.parkCar( 221, "C3", "a", 177);
        L.parkCar( 222, "C3", "a", 177);
        L.parkCar( 223, "C3", "b", 177);
        cout << endl;

        L.parkCar( 743, "A1", "a", 256);
        L.parkCar( 743, "C4", "b", 159);
        L.parkCar( 743, "B4", "a", 120);
        cout << endl;

        L.listLots();
        cout << endl;

        //  Testing find car
        cout << endl;
        cout << "Testing find a car" << endl;
        cout << endl;


        L.findCar(199);
        L.findCar(156);
        L.findCar(198);
        cout << endl;
        L.findCar(159);
        L.findCar(100);
        L.findCar(177);
        cout << endl;
        // Testing show lot contents
        cout << endl;
        cout << "Testing show lot contents" << endl;
        cout << endl;

        L.lotContents(980);
        cout << endl;
        L.lotContents(222);
        cout << endl;
        L.lotContents(301);
        cout << endl;

        L.lotContents(743);
        cout << endl;

        // Testing remove car
        cout << endl;
        cout << "Testing remove car" << endl;
        cout << endl;

        L.removeCar(156);
        L.removeCar(177);
        L.removeCar(188);
        cout << endl;
        L.removeCar(159);
        L.removeCar(888);
        L.removeCar(127);

        cout << endl;
        cout << "Testing remove car, again, this time with cars in the lot " << endl;
        cout << "Notice how first cars are deleted " << endl;
        cout << endl;

        L.removeLot(980);
        cout << endl;

        return 0;
}
```

Sample output of the program looks like:

```
No lots to list

Testing add parking lots

Parking lot with ID 980 and dimensions (7,7) has been added to the system
Parking lot with ID 666 and dimensions (6,11) has been added to the system
Parking lot with ID 222 and dimensions (12,12) has been added to the system

Cannot create the parking lot 434, dimensions exceed acceptable bounds
Cannot create the parking lot 222, a lot with this ID already exists
Parking lot with ID 301 and dimensions (9,9) has been added to the system
```

```
Cannot create the parking lot 301, a lot with this ID already exists
Parking lot with ID 205 and dimensions (4,4) has been added to the system

List of lots:
ID: 980, Dim: (7,7), number of empty parking spaces: 49
ID: 666, Dim: (6,11), number of empty parking spaces: 66
ID: 222, Dim: (12,12), number of empty parking spaces: 144
ID: 301, Dim: (9,9), number of empty parking spaces: 81
ID: 205, Dim: (4,4), number of empty parking spaces: 16

Testing remove parking lot

Lot 666 has been successfully removed from the system
Lot 101 is not in the system
Lot 205 has been successfully removed from the system

Parking lot with ID 743 and dimensions (6,7) has been added to the system


Testing park a car

Heavy-duty car with number plate 211 has been parked at location C3, lot 980
Light-duty car with number plate 141 has been parked at location E5, lot 980
Cannot park heavy-duty car with number plate 199 at location A3, not enough space

Light-duty car with number plate 199 has been parked at location A3, lot 980
Heavy-duty car with number plate 888 has been parked at location G1, lot 980
Heavy-duty car with number plate 200 has been parked at location G7, lot 222
Light-duty car with number plate 156 has been parked at location B2, lot 222

Lot 221 is not in the system
Light-duty car with number plate 177 has been parked at location C3, lot 222
Lot 223 is not in the system

Light-duty car with number plate 256 has been parked at location A1, lot 743
Heavy-duty car with number plate 159 has been parked at location C4, lot 743
Cannot park heavy-duty car with number plate 120 at location B4, already occupied

List of lots:
ID: 980, Dim: (7,7), number of empty parking spaces: 43
ID: 222, Dim: (12,12), number of empty parking spaces: 140
ID: 301, Dim: (9,9), number of empty parking spaces: 81
ID: 743, Dim: (6,7), number of empty parking spaces: 39


Testing find a car
```

```
Light-duty car with number plate 199 is at location A3, lot 980
Light-duty car with number plate 156 is at location B2, lot 222
The car with number plate 198 is not in the system

Heavy-duty car with number plate 159 is at location C4, lot 743
The car with number plate 199 is not in the system
Light-duty car with number plate 177 is at location C3, lot 222


Testing show lot contents

ID: 980, (7,7), empty slots: 43, parked cars: 199, 211, 141, 888

   1 2 3 4 5 6 7
A  + + a + + + +
B  + + b + + + +
C  + + b + + + +
D  + + + + + + +
E  + + + + a + +
F  b + + + + + +
G  b + + + + + +

ID: 222, (12,12), empty slots: 140, parked cars: 156, 177, 200

   1 2 3 4 5 6 7 8 9 10 11 12
A  + + + + + + + + + +  +  +
B  + a + + + + + + + +  +  +
C  + + a + + + + + + +  +  +
D  + + + + + + + + + +  +  +
E  + + + + + + + + + +  +  +
F  + + + + + + b + + +  +  +
G  + + + + + + b + + +  +  +
H  + + + + + + + + + +  +  +
I  + + + + + + + + + +  +  +
J  + + + + + + + + + +  +  +
K  + + + + + + + + + +  +  +
L  + + + + + + + + + +  +  +

ID: 301, (9,9), empty slots: 81, parked cars: no cars yet

   1 2 3 4 5 6 7 8 9
A  + + + + + + + + +
B  + + + + + + + + +
C  + + + + + + + + +
D  + + + + + + + + +
E  + + + + + + + + +
F  + + + + + + + + +
G  + + + + + + + + +
H  + + + + + + + + +
I  + + + + + + + + +

ID: 743, (6,7), empty slots: 39, parked cars: 256, 159

   1 2 3 4 5 6 7
A  a + + + + + +
B  + + + b + + +
C  + + + b + + +
D  + + + + + + +
E  + + + + + + +
F  + + + + + + +


Testing remove car

Light-duty car with number plate 156 car has been removed from slot B2, lot 222
Light-duty car with number plate 177 car has been removed from slot C3, lot 222
The car with number plate 188 is not in the system

Heavy-duty car with number plate 159 car has been removed from slot C4, lot 743
Heavy-duty car with number plate 888 car has been removed from slot G1, lot 980
The car with number plate 127 is not in the system

Testing remove car, again, this time with cars in the lot
Notice how first cars are deleted

Light-duty car with number plate 199 car has been removed from slot A3, lot 980
Light-duty car with number plate 141 car has been removed from slot E5, lot 980
Heavy-duty car with number plate 211 car has been removed from slot C3, lot 980
Lot 980 has been successfully removed from the system
```

Please try to generate the output that is similar in formatting to the one provided here.

NOTES ABOUT IMPLEMENTATION:

1. You ARE NOT ALLOWED to modify the given parts of the header file. You MUST use dynamically allocated arrays in your implementation. You will get no points if you use fixed-sized arrays, linked lists, or any other data structures such as vectors/arrays from the standard library. However, if necessary, you may define additional data members and member functions.
2. Moreover, you ARE NOT ALLOWED to use any global variables or any global functions.

3. Your code must not have any memory leaks. You will lose points if you have memory leaks in your program even though the outputs of the operations are correct. To learn how to detect memory leaks you may refer to the course slides.
4. Unless otherwise stated in the description, you may assume that the inputs for the functions (e.g., the car location) are always valid so that you do not need to make any input checks.
5. Make sure that each file that you submit (each and every file in the archive) contains your name and student number at the top as comments.

NOTES ABOUT SUBMISSION:

This assignment is due by 23:59 on March 22, 2022. **This homework will be graded by your TA Aydamir Mirzayev (aydamir.mirzayev[at]bilkent.edu.tr). Please direct all your homework-related questions to him.**

1. In this assignment, you must have a separate interface and implementation files (i.e., separate .h and .cpp files) for your class. The file names should be "ParkingSystem.h" and "ParkingSystem.cpp". You should also submit other .h and .cpp files if you implement additional classes. We will test your implementation by writing our own main function. Thus, you should not submit any file that contains the main function. Although you are not going to submit it, we recommend you write your own driver file to test each of your functions. However, you SHOULD NOT submit this test code (we will use our own test code).

2. The code (main function) given above is just an example. We will test your implementation also using different main functions, which may contain different function calls. Thus, do not test your implementation only by using this example code. Write your own main functions to make extra tests (however, do not submit these test codes).

3. You should put your "ParkingSystem.h" and "ParkingSystem.cpp" (and additional .h and .cpp files if you implement additional classes) into a folder and zip the folder (in this zip file, there should not be any file containing the main function). The name of this zip file should conform to the following name convention: secX-Firstname-Lastname-StudentID.zip where X is your section number. The submissions that do not obey these rules will not be graded.

4. Then, before 23:59 on March 22nd, you need to upload this zipped file containing only your header and source codes (but not any file containing the main function) to Moodle.

No hardcopy submission is needed. The standard rules about late homework submissions apply. Please see the course syllabus for further discussion of the late homework policy as well as academic integrity.

5. You are free to write your programs in any environment (you may use Linux, Mac OS X, or Windows). On the other hand, we will test your programs on “dijkstra.ug.bcc.bilkent.edu.tr” and we will expect your programs to compile and run on the Dijkstra machine. If we could not get your program to work properly on the Dijkstra machine, you would lose a considerable amount of points. Therefore, we recommend you make sure that your program compiles and properly works on “dijkstra.ug.bcc.bilkent.edu.tr” before submitting your assignment.