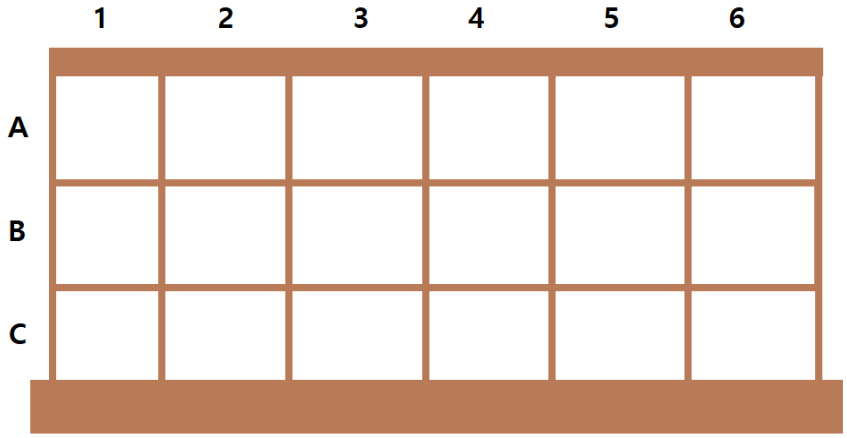
CS 201, Fall 2021

Homework Assignment 1

Due: 23:59, November 19, 2021

Scientists in the chemistry lab need a software tool to keep track of combustible and retardant chemicals that are stored in the lab cabinets. In ordinary circumstances this software would simply be required to keep the location of each chemical in the cabinets. However, since some of the chemicals are combustible, scientists need to employ a specific strategy to minimize the risk of fire. That is since combustive materials are likely to ignite if in contact, no two combustive chemicals can be placed next to each other. Therefore the software designed for the lab needs to account for this rule.

The cabinets in the lab have varying heights and widths which are measured with the number of individual storage slots. The rows in the cabinet are marked with the English alphabet, while columns are numbered with integers starting from 1. Below is an illustration of a cabinet with three rows and six columns.



The tracking system that you will implement should have the following functionalities. The details are given below.

- Add a cabinet
- Remove a cabinet
- Show the list of cabinets
- Show the contents of a cabinet
- Place a chemical
- Find a chemical
- Remove the chemical

Add a cabinet (addCabinet): The tracking system will allow the user to add a new cabinet indicating a cabinet ID (which should be an integer), the number of rows and the number of columns. Each cabinet needs to have a unique ID number and you need to check the collection of existing cabinets to make sure that the entered ID for the new cabinet 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 height and width of the cabinet are not larger than 9. If any of the entered dimensions is larger than 9, you need to display a warning message and not allow the operation to proceed. If ID, width, and height are all valid, you should add the cabinet and display a message indicating that the operation was a success. At the time of adding a cabined, you should ONLY allocate a necessary amount of memory for that specific cabinet. For example, DO NOT allocate memory necessary for a 9x9 cabinet when a 4x5 cabined is created. You also cannot make any assumption about the number of cabinets in 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] for rows and numbers [1, 2, 3, 4, 5, 6, 7, 8, 9] for columns. (Please see the code given below and its output for the full signature of the method and format of the output.)

Remove a cabinet (removeCabinet): The tracking system should allow the user to remove a certain cabinet from the collection using the cabinet ID. If a cabinet with the specified ID does not exist, you should display a warning message. If it exists, you should remove all the chemicals from its shelves and then remove the cabinet itself. You should display a message indicating that the cabinet has been removed and list the names of chemicals that were disposed of in the process. (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 cabinets (listCabinets): The tracking system should allow the user to see the list of all existing cabinets. For each cabinet you should display the ID, dimensions and the number of empty slots in the cabinet. The order in which you display (and store) the cabinets is arbitrary, i.e., you can choose any order you please, as long as you show all the cabinets. (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 cabinet (cabinetContents): The tracking system should allow the user to see the contents and detailed information about a specific cabinet by specifying the ID of that cabinet. If a cabinet 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 chemicals, and a printout that looks like 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
```

Here, ‘c’ stands for combustive chemical and ‘r’ stands for retardant chemical, while ‘+’ is used to indicate an empty slot. (Please see the code given below and its output for the full signature of the method and format of the output.)

Place a chemical (placeChemical): The system should allow the user to place a chemical in the cabinet of their choosing by specifying the cabinet ID, location in the cabinet (e.g. 'C5'), type of chemical, and unique chemical ID. Note that just like every cabinet has a unique ID, so does every chemical has a unique ID in the system, meaning that the ID should be unique not only to that closet but to all the closets in the system. The program should check if the corresponding cabinet ID exists in the system, if corresponding space in the cabinet is empty, if that type of chemical can be placed at that location, and if the chemical ID is unique. If any of these checks fail, the system should display a corresponding warning message and not proceed with the operation.

The general rule for placing chemicals is as follows: retardant chemicals can be placed at any free location. However, combustive chemicals cannot be next to each other in horizontal, vertical or diagonal directions. So for the available slots of the cabinet shown below, C4 or D5 are two example slots where a combustive chemical can be placed, while slot C6 cannot be used since it contains a combustive chemical next to it in a diagonal direction.

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

If the combustive chemical cannot be placed at chosen location, in addition to printing a warning message, you should also provide the names of closest suitable locations. If more than one such slot exists, you should provide all their names. For example, if the user wants to place a combustive chemical at C6, then you should print a warning message indicating that such operation is not possible and that locations C5 and D5 are closest (both at a distance 1) available locations for a combustive chemical. 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 a chemical (findChemical): The system should allow a user to find the location of a chemical using its ID. The program should display the cabinet ID and chemical location if the chemical 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.)

Remove a chemical (removeChemical): The system should allow the user to remove a certain chemical using its ID and display a warning message if that ID 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.)

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

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

        void addCabinet(int id, int rows, int columns);
        void removeCabinet(int id);
        void listCabinets();
        void cabinetContents(int id);
        void placeChemical(int cabinetId, string location, string chemType, int chemID);
        void findChemical(int id);
        void removeChemical(int id);
        // ...
        //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 as this one. So the code you submit needs to work with that configuration.

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


int main(){
        LabOrganizer L;

        L.listCabinets();

        // Testing add cabinet
        cout << "Testing add cabinet" << endl;
        cout << endl;

        L.addCabinet(101, 3, 4);
        L.addCabinet(102, 5, 3);
        L.addCabinet(103, 8, 8);
        cout << endl;

        L.addCabinet(103, 3, 3);
        L.addCabinet(104, 9, 19);
        L.addCabinet(201, 9, 9);
        cout << endl;

        L.addCabinet(203, 9, 3);
        L.addCabinet(205, 4, 4);
        cout << endl;

        L.listCabinets();
        cout << endl;
```

```
        // Testing remove cabinet
        cout << endl;
        cout << "Testing remove cabinet" << endl;
        cout << endl;

        L.removeCabinet(452);
        L.removeCabinet(101);
        L.removeCabinet(205);
        cout << endl;
        L.removeCabinet(203);
        L.removeCabinet(347);
        L.removeCabinet(201);
        cout << endl;
        L.addCabinet(101, 7, 7);
        cout << endl;

        L.listCabinets();
        cout << endl;

        // Testing place Chemical
        cout << endl;
        cout << "Testing place chemical" << endl;
        cout << endl;

        L.placeChemical( 101, "C1", "combustive", 200);
        L.placeChemical( 101, "C1", "retardant", 139);
        L.placeChemical( 101, "D3", "retardant", 139);
        cout << endl;

        L.placeChemical( 101, "E3", "combustive", 188);
        L.placeChemical( 101, "D2", "combustive", 888);
        L.placeChemical( 101, "E1", "combustive", 888);
        L.placeChemical( 102, "B2", "combustive", 200);
        L.placeChemical( 102, "B2", "combustive", 156);
        cout << endl;

        L.placeChemical( 102, "C3", "retardant", 190);
        L.placeChemical( 102, "A3", "combustive", 171);
        L.placeChemical( 102, "E1", "combustive", 171);
        cout << endl;

        L.placeChemical( 103, "B2", "combustive", 139);
        L.placeChemical( 103, "B2", "combustive", 144);
        L.placeChemical( 103, "C3", "retardant", 120);
        cout << endl;
        L.placeChemical( 103, "A1", "retardant", 127);
        L.placeChemical( 103, "G8", "combustive", 191);
        cout << endl;

        L.listCabinets();
        cout << endl;

        //  Testing find chemical
        cout << endl;
        cout << "Testing find chemical" << endl;
        cout << endl;


        L.findChemical(191);
```

```
        L.findChemical(156);
        L.findChemical(171);
        cout << endl;
        L.findChemical(155);
        L.findChemical(100);
        L.findChemical(200);
        cout << endl;
        L.findChemical(139);
        cout << endl;


        // Testing show cabinet contents
        cout << endl;
        cout << "Testing show cabinet contents" << endl;
        cout << endl;

        L.cabinetContents(101);
        cout << endl;
        L.cabinetContents(102);
        cout << endl;
        L.cabinetContents(103);
        cout << endl;

        L.cabinetContents(107);
        cout << endl;

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

        L.removeChemical(156);
        L.removeChemical(177);
        L.removeChemical(188);
        cout << endl;
        L.removeChemical(189);
        L.removeChemical(777);
        L.removeChemical(127);
        cout << endl;

        L.removeCabinet(103);
        cout << endl;
        L.removeCabinet(102);
        cout << endl;

        return 0;
}
```

Sample output of the program looks like:

```
Added a cabinet: ID 101 and dimensions 3 to 4
Added a cabinet: ID 102 and dimensions 5 to 3
Added a cabinet: ID 103 and dimensions 8 to 8

Cannot add the cabinet: ID 103 already in the system
Cannot add the cabinet: dimensions are out of bounds
Added a cabinet: ID 201 and dimensions 9 to 9

Added a cabinet: ID 203 and dimensions 9 to 3
Added a cabinet: ID 205 and dimensions 4 to 4

List of all cabinets:
ID: 101, Dim: 3x4, Number of empty slots: 12
ID: 102, Dim: 5x3, Number of empty slots: 15
ID: 103, Dim: 8x8, Number of empty slots: 64
ID: 201, Dim: 9x9, Number of empty slots: 81
ID: 203, Dim: 9x3, Number of empty slots: 27
ID: 205, Dim: 4x4, Number of empty slots: 16


Testing remove cabinet

Cabinet 452 does not exist in the system
Cabinet 101 has been removed
Cabinet 205 has been removed

Cabinet 203 has been removed
Cabinet 347 does not exist in the system
Cabinet 201 has been removed

Added a cabinet: ID 101 and dimensions 7 to 7

List of all cabinets:
ID: 101, Dim: 7x7, Number of empty slots: 49
ID: 102, Dim: 5x3, Number of empty slots: 15
ID: 103, Dim: 8x8, Number of empty slots: 64


Testing place chemical

Combustive chemical with ID 200 has been placed at location C1 in cabinet 101
Location C1 in cabinet 101 is already occupied. Nearest possible locations for this chemical: B1, B2, C2, D1, D2
Retardant chemical with ID 139 has been placed at location D3 in cabinet 101

Combustive chemical with ID 188 has been placed at location E3 in cabinet 101
Location D2 in cabinet 101 is not suitable for a combustive chemical. Nearest possible locations for this chemical: E1, C3
Combustive chemical with ID 888 has been placed at location E1 in cabinet 101
Chemical with ID 200 already exists in the system
Combustive chemical with ID 156 has been placed at location B2 in cabinet 102

Retardant chemical with ID 190 has been placed at location C3 in cabinet 102
Location A3 in cabinet 102 is not suitable for a combustive chemical. Nearest possible locations for this chemical: D3
Combustive chemical with ID 171 has been placed at location E1 in cabinet 102

Chemical with ID 139 already exists in the system
Combustive chemical with ID 144 has been placed at location B2 in cabinet 103
Retardant chemical with ID 120 has been placed at location C3 in cabinet 103

Retardant chemical with ID 127 has been placed at location A1 in cabinet 103
Combustive chemical with ID 191 has been placed at location G8 in cabinet 103

List of all cabinets:
ID: 101, Dim: 7x7, Number of empty slots: 45
ID: 102, Dim: 5x3, Number of empty slots: 12
ID: 103, Dim: 8x8, Number of empty slots: 60


Testing find chemical

Chemical 191 is at location G8 in cabinet 103
Chemical 156 is at location B2 in cabinet 102
Chemical 171 is at location E1 in cabinet 102

Chemical 155 is not in the system
Chemical 100 is not in the system
Chemical 200 is at location C1 in cabinet 101

Chemical 139 is at location D3 in cabinet 101
```

```
Testing show cabinet contents

ID: 101, 7x7, empty: 45. Chemicals: C1: 200, E1: 888, D3: 139, E3: 188
  1 2 3 4 5 6 7
A + + + + + + +
B + + + + + + +
C c + + + + + +
D + + r + + + +
E c + c + + + +
F + + + + + + +
G + + + + + + +

ID: 102, 5x3, empty: 12. Chemicals: B2: 156, C3: 190, E1: 171
  1 2 3
A + + +
B + c +
C + + r
D + + +
E c + +

ID: 103, 8x8, empty:  60. Chemicals: A1: 127, B3: 144, C3: 120, G8: 191
  1 2 3 4 5 6 7 8
A r + + + + + + +
B + + c + + + + +
C + + r + + + + +
D + + + + + + + +
E + + + + + + + +
F + + + + + + + +
G + + + + + + + c
H + + + + + + + +

Cabinet 107 is not is the system


Testing remove chemical

Chemical 156 removed from cabinet 102
Chemical 177 is not in the system
Chemical 188 removed from cabinet 101

Chemical 189 is not in the system
Chemical 777 is not in the system
Chemical 127 removed from cabinet 103

Chemical 144 removed from cabinet 103
Chemical 120 removed from cabinet 103
Chemical 191 removed from cabinet 103
Cabinet 103 has been removed

Chemical 190 removed from cabinet 102
Chemical 171 removed from cabinet 102
Cabinet 102 has been removed
```

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 detect memory leaks, you may want to use Valgrind which is available at http://valgrind.org.
4. Otherwise stated in the description, you may assume that the inputs for the functions (e.g., the chemical 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 November 19, 2021. **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 separate interface and implementation files (i.e., separate .h and .cpp files) for your class. The file names should be "LabOrganizer.h" and "LabOrganizer.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 to 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 "LabOrganizer.h" and "LabOrganizer.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 November 19th, 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 to make sure that your program compiles and properly works on "dijkstra.ug.bcc.bilkent.edu.tr" before submitting your assignment.