CS 201, Spring 2022

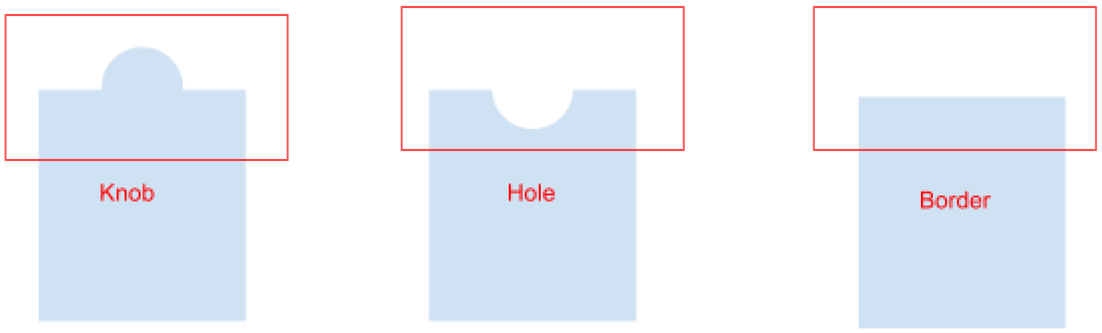
Homework Assignment 3

Due: 23:59, April 27, 2022

Puzzle enthusiasts need a software tool to see possible piece configurations on a puzzle container. You will develop a puzzle container to achieve this and you are expected to use the linked lists for the implementation. Below we give some definitions which you will use while implementing your homework.

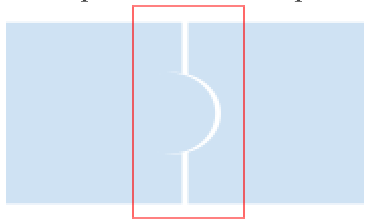
<u>Pattern Types and Fitting Conditions</u>

Pattern types are knob, border and hole which are illustrated as following in rectangles:

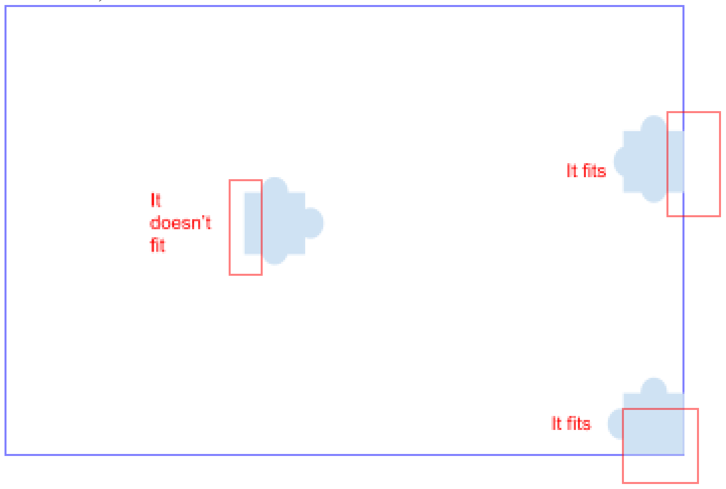


Fitting conditions are as following:

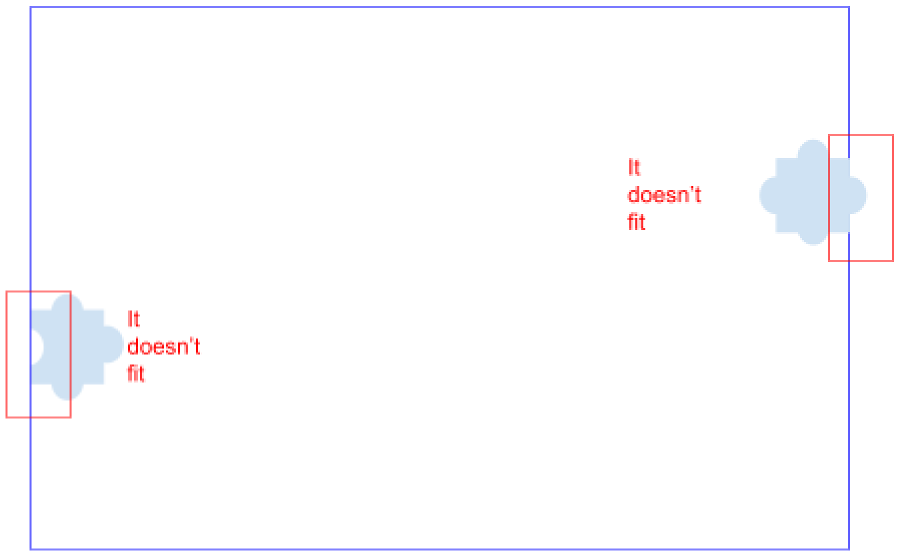
1. Knob pattern and hole pattern fit each other.



2. Border pattern fits only on the container border. (In figure, dark blue is container border.)



3. Knobs and holes cannot be on the container border



Data representation of a shape and its realization procedure is given below.

Consider the shape represented as *0x58*. It's binary form as following:

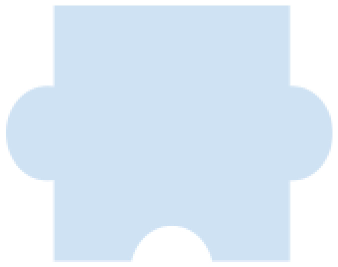
01011000

By division of the binary form we get patterns as following:

```
Left / Right / Up / Down
01   /   01   / 10  /    00
 1    /    1    /  2  /     0
```

From integer representation(see below), we know that shape is
Down : Hole
Up : Border
Right: Knob
Left: Knob

The shape is illustrated in the following figure:



Functionalities

The puzzle container that you will implement will have the following functionalities. The details are given below.

- Add a piece
- Remove a piece
- Show the occupied places in the container
- Show a column
- Show a row
- Show the shape of the piece in the given coordinate
- Find specific shaped puzzle

Add a piece (addPiece): The puzzle container system will allow the user to add a piece if coordinates of the piece are in the frame, suitable and empty. A place is suitable for a piece if neighbors are either empty or shapes fit each other (it will be explained later). If the place is out of the frame, occupied or not suitable, you should not add the item and display a warning message indicating the problem accordingly. First coming message overrides others; for example you do not need to display the occupied message if you already have displayed an out of frame message.
(Please see the code given below and its output for the full signature of the method and format of the output.)

Remove a piece (removePiece): The puzzle container system will allow the user to remove a certain piece from the collection using its coordinates. If a piece with the specified coordinate does not exist, you should display a warning message. If it exists, you should remove it. You should also display a message indicating that the piece has been removed.
(Please see the code given below and its output for the full signature of the method and format of the output.)

Show the occupied places in the container (displayContainer): The puzzle container system will allow the user to see the coordinate system which shows occupied places with 'X' and empty places with 'O'. Coordinates are numerals which are started with 1.
(Please see the code given below and its output for the full signature of the method and format of the output.)

Show a column (showColumn): The puzzle container system will allow the user to display a certain column. If the column is out of the border, you should display a warning message. If it is in the container, you should display that column in a similar manner to displayContainer function.
(Please see the code given below and its output for the full signature of the method and format of the output.)

Show a row (showRow): The puzzle container system will allow the user to display a certain row. If the column is out of the border, you should display a warning message. If it

is in the container, you should display that row in a similar manner to displayContainer function.
(Please see the code given below and its output for the full signature of the method and format of the output.)

Show the shape of the piece in the given coordinate (showPiece): The puzzle container system should allow the user to see the shape of a piece. If a piece in a given coordinate does not exist, you should display a warning message. Otherwise, you should display the coordinate and piece shape. There are four directions: down, up, right and left. For each direction there are possible patterns: knob, border and hole. For piece shape you should display the pattern of each direction.
(Please see the code given below and its output for the full signature of the method and format of the output.)

Find a specific shaped puzzle piece (findPiece): The puzzle container system will allow the user to display a list of the coordinates that has a specified shaped puzzle piece. Only input parameter is the shape which is represented as a char. If there is no puzzle piece that has that shape, you 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.)

Note: You may divide the header file to multiple header files but you have to upload all of them. Also you may add new member functions and variables. But a new global function or variable cannot be added.

Below is the header file for the all class (PuzzleContainer.h)

```
void shapeConverter(char shape, int& down, int& up, int& right, int& left){
    down = shape%4;
    up = (shape/4)%4;
    right= (shape/16)%4;
    left= shape/64;
}




class PuzzleRow{
private:
    struct PuzzlePiece{
        char shape;
        int rowindex;
        int colindex;
        PuzzlePiece* next;
};
    PuzzlePiece* head;
    PuzzleRow* next;
    int rowindex;
    int maxSize; //equals to puzzleContainer width

public:
    PuzzleRow( int size, int rowindex);
    ~PuzzleRow();
    int getRowIndex();
    void setNext( PuzzleRow* next );
    PuzzleRow* getNext();
    bool insert( int index, char shape );
    bool remove( int index );
    bool isEmpty();
    char getPiece(int index );
    void printRow();
};
class PuzzleContainer{
private:
    PuzzleRow* rowLists;
    int height;
    int width;
public:
    PuzzleContainer( int h, int w);
    ~PuzzleContainer();
    PuzzleRow* getRow( int index );
    bool insertRow( int index );
    bool removeRow( int index );
    bool addPiece( int row, int col , char shape );
    bool removePiece(int row, int col);
    int getHeight();
    int getWidth();
    void showPiece(int row, int col);
    void displayContainer();
    void showColumn(int col);
    void showRow(int row);
    void findPiece(char shape);
};
```

Below you will find some implementation details.

- Puzzle container has a height and a width which are immutable.
- Puzzle container has a linked list of rows. It only contains non-empty rows. If a row becomes empty, it should be removed from the list and deallocated.
- Each puzzle row contains a linked list of puzzle pieces.
- Shape of a piece is represented using a char variable. It can be unfolded by using the "shapeConverter" function. An integer represent pattern of a direction (down,up,right,left) as following:
    - Hole if 0
    - Knob if 1
    - Border if 2
- If there are two neighboring pieces, the patterns of pieces in direction towards each other must be complementing: One is a knob and the other is a hole.
- `getPiece` is an auxiliary function to get the shape of a piece in that row. It returns 255 if the index is empty.
- `printRow` is an auxiliary function to display 'X' and 'O's in that row.
- `findPiece` function needs an auxiliary function in `PuzzleRow` class which is not given in the header file.

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 "FlowerLibrary.h"


int main(){
        PuzzleContainer P(8,4);

        P.displayContainer();

        // Testing add piece
        cout << endl;
        cout << "Testing Add Piece" << endl;
        cout << endl;

        P.addPiece( 1,2, 0x58);
        P.addPiece( 2,4, 0x20);
        P.addPiece( 5,2, 0x05);
        P.addPiece( 5,3, 0x54);
        P.addPiece( 6,3, 0x55);
        P.addPiece( 7,2, 0x00);
        P.addPiece( 7,4, 0x65);
        P.addPiece( 8,3, 0x42);

        cout << endl;
        P.displayContainer();
        cout << endl;

        P.addPiece( 0,0, 0x58);
        P.addPiece( 1,2, 0x58);
        P.addPiece( 2,2, 0x58);
        P.addPiece( 7,3, 0x55);
        P.addPiece( 4,4, 0x55);
        P.addPiece( 5,5, 0x55);

        cout << endl;
        P.displayContainer();
        cout << endl;

        // Testing remove piece
        cout << endl;
        cout << "Testing Remove Piece" << endl;
        cout << endl;

        P.removePiece(5,9);
        P.removePiece(5,3);
        P.removePiece(4,4);
        P.removePiece(1,2);

        cout << endl;
        P.displayContainer();
        cout << endl;
        P.addPiece( 1,2, 0x58);

        cout << endl;
        P.displayContainer();
        cout << endl;


        // Testing show piece
        cout << endl;
        cout << "Testing Show Piece" << endl;
        cout << endl;

        P.showPiece(5,9);
        P.showPiece(4,4);
        P.showPiece(1,2);

        // Testing show column
        cout << endl;
        cout << "Testing Show Column" << endl;
        cout << endl;

        P.showColumn(0);
        P.showColumn(1);

        // Testing show row
        cout << endl;
        cout << "Testing Show Row" << endl;
        cout << endl;

        P.showRow(9);
        P.showRow(8);

        // Testing find piece
        cout << endl;
        cout << "Testing Find Piece" << endl;
        cout << endl;

        P.findPiece(0x21);
        cout << endl;
        P.addPiece( 3,2, 0x05);
        P.findPiece(0x05);

        return 0;
}
```

Sample output of the program looks like:

```
Container 8x4:
  1 2 3 4
1 O O O O
2 O O O O
3 O O O O
4 O O O O
5 O O O O
6 O O O O
7 O O O O
8 O O O O

Testing Add Piece

A piece is added to (1,2)
A piece is added to (2,4)
A piece is added to (5,2)
A piece is added to (5,3)
A piece is added to (6,3)
A piece is added to (7,2)
A piece is added to (7,4)
A piece is added to (8,3)

Container 8x4:
  1 2 3 4
1 O X O O
2 O O O X
3 O O O O
4 O O O O
5 O X X O
6 O O X O
7 O X O X
8 O O X O

(0,0) is out of frame, it cannot be added
(1,2) has been already occupied, it cannot be added
Shape doesn't fit into (2,2), it cannot be added
Shape doesn't fit into (7,3), it cannot be added
Shape doesn't fit into (4,4), it cannot be added
(5,5) is out of frame, it cannot be added

Container 8x4:
  1 2 3 4
1 O X O O
2 O O O X
3 O O O O
4 O O O O
5 O X X O
6 O O X O
7 O X O X
8 O O X O
```

```
Testing Remove Piece

(5,9) is out of frame, it cannot be removed
The piece on (5,3) is removed
There is no piece on (4,4), it cannot be removed
The piece on (1,2) is removed

Container 8x4:
  1 2 3 4
1 O O O O
2 O O O X
3 O O O O
4 O O O O
5 O X O O
6 O O X O
7 O X O X
8 O O X O

A piece is added to (1,2)

Container 8x4:
  1 2 3 4
1 O X O O
2 O O O X
3 O O O O
4 O O O O
5 O X O O
6 O O X O
7 O X O X
8 O O X O


Testing Show Piece

(5,9) is out of frame, it cannot be shown
There is no piece on (4,4), it cannot be shown
Piece on (1,2) as following:
Down: Hole
Up: Border
Right: Knob
Left: Knob

Testing Show Column

Column 0 is out of border
Column 1:
O
O
O
O
O
O
O
O
```

```
Testing Show Row

Row 9 is out of border
Row 8:
O O X O

Testing Find Piece

There is no piece that has shape
Down: Knob
Up: Hole
Right: Border
Left: Hole

A piece is added to (3,2)
(3,2), (5,2) contain the piece that has shape
Down: Knob
Up: Knob
Right: Hole
Left: Hole
```

NOTES ABOUT IMPLEMENTATION:

1. You ARE NOT ALLOWED to modify the given parts of the header files. You MUST use linked-list in your implementation. You will get no points if you use fixed-sized arrays, dynamic arrays 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 except shapeConverter.
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. 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 April 27, 2022. **This homework will be graded by your TA Mahmud Sami Aydın (sami.aydin[at]bilkent.edu.tr). Please direct all your homework-related questions to him.**

1. We will test your implementation by writing our own main function.Thus, you should <u>not submit any file that contains the main function.</u>

   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 “PuzzleContainer.h” and “PuzzleContainer.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. **<u>Then, before 23:59 on April 27th, you need to upload this zipped file containing only your header and source codes (but not any file containing the main function) to Moodle.</u>**

   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.