ANKARA UNIVERSITY
COM1002B, COM102A
2020-21 Spring
PA3 – Brick Shooter Game

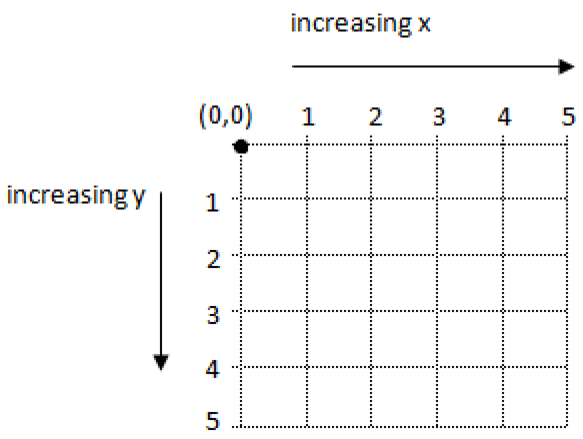
Due Date: 19/05/2021 11:59 p.m. (23:59)
Assoc. Prof. Dr. Hacer Yalım Keleş
Corresponding Assistant: Özge Mercanoğlu Sincan (omercanoglu@ankara.edu.tr)

Task: For this assignment, your job is going to create brick shooter game.

Bricks are colored blocks which have 1x1 unit dimension in space. The objective of this game is for the player to explode all bricks in the space by using a shooter which throws a brick in a particular direction. Player determines the shooter's shooting angle and the color of the shooted brick. The thrown brick hits to either a wall or another colored brick, in which case it is located to the closest empty space in its trajectory before hitting. At least 3 same colored bricks need to come together in an 8-neighborhood relation for explosion. After explosion, all the same colored bricks are removed from the game space.

These are the additional rules:

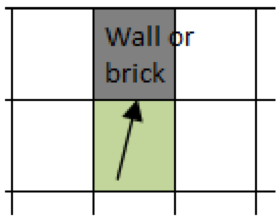
- You will use a Cartesian coordinate system in which x coordinates increase to the right and y coordinates increase to the down, as depicted below. On the coordinate system, the game will be played only in the region where x and y are positive, i.e. the depicted area in the figure below.

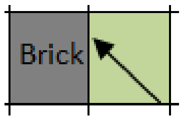


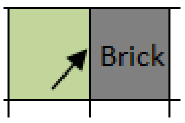
- The game starts by setting the max (x,y) coordinates where the game will be played. These x and y values will be provided at run time. The top, left and right outer edges will always be considered as wall bricks (depicted below using '|' and '~' symbols), that are fixed and never explode. (Please look at the output files for representation of the walls.)
- After the game area size, initial brick content will be provided. Start placing upper left corner of the first brick aligned to the point (1,1) in the space and continue placing the bricks from left to right and top to bottom order, until seeing the **"end"** keyword. Note that each brick occupies 1x1 unit square in the space. Brick colors can be 'R', 'G', 'B', 'Y', which means red, green, blue and yellow, respectively. The initial brick content can also contain ' ' which means

an empty brick size area. This representation enables us to configure the game environment without providing a position coordinate for each brick.

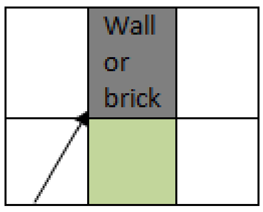
- After setting the initial set of bricks, the position of the shooter will be provided using (x_coordinate, y_coordinate) point of the shooter. Hence, the first given number is the x_coordinate and the second number is the y_coordinate of the shooter. You can assume that its exact shooting position is the given upper left coordinates of the shooter, but it occupies a cell (like bricks, as depicted using '_' symbol in the output file).
- After the given configuration, the game starts. The user provides a number of colored bricks to be shooted in the given order via the shooter, together with the shooting angle. The brick colors are provided using the same set of characters, i.e. 'R', 'G', 'B', 'Y'. The angle of the shooter is provided using two floating point numbers in x and y directions. The movement in the y direction will always be given in the upward direction (y_direction < 0). Hence, the bricks will always reside in the game area.
- When a brick is thrown, it hits either a wall or colored brick on its direction. Then it is placed in an empty area on its trajectory;
    - When the trajectory passes through a cell, it is placed to that cell. For example, full cells (wall or brick) are shown with gray background in the following figures. New bricks will be placed to the areas with green background.

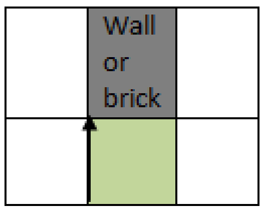


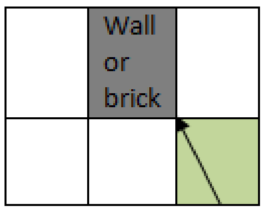




    - When the trajectory passes through a corner, it is placed on the right empty brick.







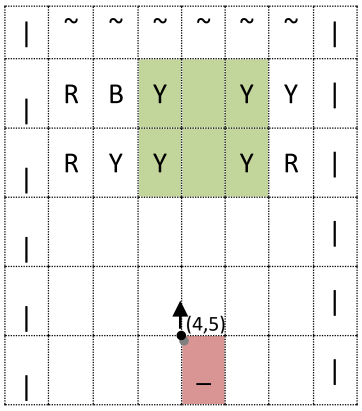
Then, it explodes the bricks with the same color that are located in the 8-neighbourhood of the inserted brick; for an explosion there must be at least three same-colored bricks in that area. The 8-neighbourhood of a brick includes the neighbors that touch one of their edges or corners to this brick.

- If 3 bricks are exploded, 30 points are collected, for 4 bricks, 40 points, and for more than 4 bricks, 100 points are collected. After brick explosions, your program prompts collected points and the total points got until that time using: **"You got %d points. Score: %d\n"** format. Then, the updated board content is printed ending with a **'\n'**.

For example;

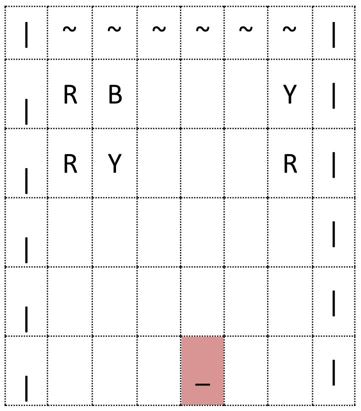
Input:

```
8 6
RBY YY
RYY YR
end
4 5
Y 0 -0.5
```



When the brick in color Y is thrown in (0, -0.5) direction, in the above case, it hits the brick in the (4,1) area. Its 8-neighbourhood is shown with green background. There are 5 green bricks in total, therefore all of them explode and the program prints "You got 100 points. Score: 100" and the updated board state as follows:

```
You got 100 points. Score: 100
```



- If the thrown brick hits the left or right wall, it turns back and continues to move; only its x_direction changes, the movement in the y_direction remains the same.
- 'q' is the command to quit from the game.
- Game finishes with either the 'q' command or when all bricks are exploded. If the user quits the game or bursts all bricks, the game is over and your program prompts: **"Game is over. Your score is %d!"**

- **Input format:**
  ```
  <width: integer>[S]<height: integer>
  (<R | G | B | Y | [S]>)*
  end
  <shooter_x: integer>[S]<shooter_y: integer>
  (<R | G | B | Y >[S]<x_direction: double>[S]<y_direction : double>)* | q
  ```

  where [S] is the space character, *: zero or more, |: or

Warning: Any form of code copying, including the copies from the internet, is strictly prohibited. If we determine similarities between your codes with any other students in the class, it will be treated as cheating and will be punished. So, you would increase the risk of cheating when you see somebody else's code directly or see a solution in the internet (i.e. somebody else might have also copied the same code from the internet like you, so both of these codes will be evaluated as copies, since they both copy from an external source. Such attempts will always be considered as cheating). You are supposed to write the program by yourselves.

Testing:

As usual, use *input redirection* mechanism of your operating system to test your programs. For example, if your executable is called as PA3, redirect the input.txt file to standard input using **<** operator and redirect your outputs to a file using **>** operator such as:

> ./PA3<input1.txt>myOutput1.txt

Then compare your outputs file with the given output files such as:

>diff myOutput1.txt output1.txt

Repeat these steps for all the given input and output files.

This kind of execution enables your programs to read inputs from a file without writing any file related functions (e.g. fopen(), fscanf() etc.). In other words, the getchar() or scanf() functions in your code reads data from the redirected files instead of the std. input in this way (e.g. keyboard).

Submission:

Before submission, rename your source file name as **StudentNumber.c.**

All submissions will be made using Moodle.