Lab 6

Please hand in your code on Sakai-Assignment part. You only need to upload a .c file which includes all of your code and don’t pack it.
You should run your program to TA during Lab time and upload your program (both are required), or you won’t get the point.

Question 1

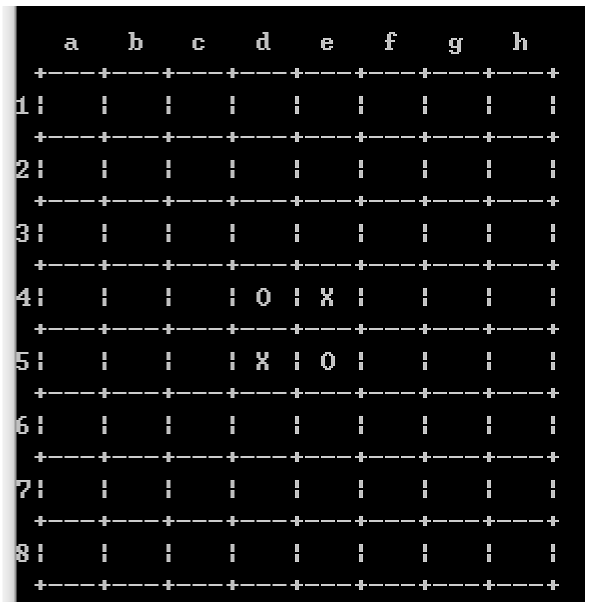
Tasks:

You task is writing a game named Reversi and an AI of it.

One of attachment is my program of the game Reversi. You can take it as sample.

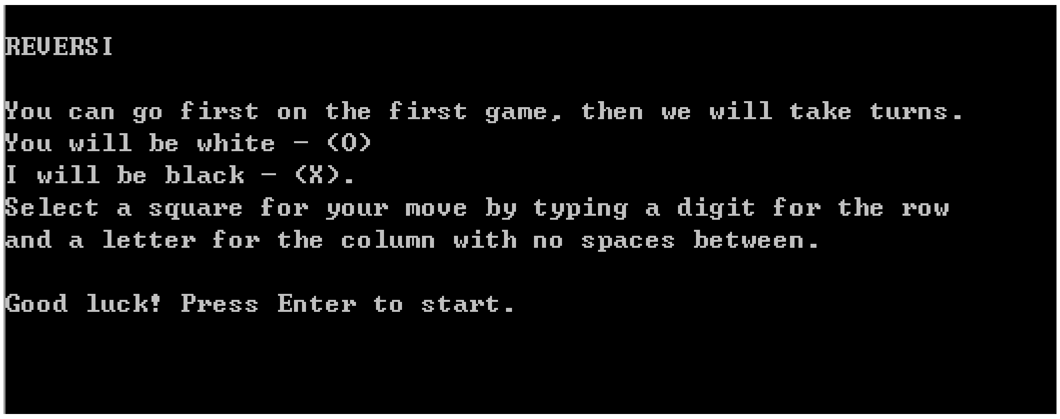
The input and output of your project should be exactly the same with mine, for example,

Chessboard notation:



```
   a   b   c   d   e   f   g   h
 +---+---+---+---+---+---+---+---+
1¦   ¦   ¦   ¦   ¦   ¦   ¦   ¦   ¦
 +---+---+---+---+---+---+---+---+
2¦   ¦   ¦   ¦   ¦   ¦   ¦   ¦   ¦
 +---+---+---+---+---+---+---+---+
3¦   ¦   ¦   ¦   ¦   ¦   ¦   ¦   ¦
 +---+---+---+---+---+---+---+---+
4¦   ¦   ¦   ¦ O ¦ X ¦   ¦   ¦   ¦
 +---+---+---+---+---+---+---+---+
5¦   ¦   ¦   ¦ X ¦ O ¦   ¦   ¦   ¦
 +---+---+---+---+---+---+---+---+
6¦   ¦   ¦   ¦   ¦   ¦   ¦   ¦   ¦
 +---+---+---+---+---+---+---+---+
7¦   ¦   ¦   ¦   ¦   ¦   ¦   ¦   ¦
 +---+---+---+---+---+---+---+---+
8¦   ¦   ¦   ¦   ¦   ¦   ¦   ¦   ¦
 +---+---+---+---+---+---+---+---+
```

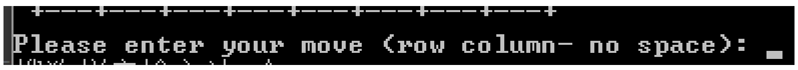
Instructions:


```
REVERSI

You can go first on the first game, then we will take turns.
You will be white - (O)
I will be black - (X).
Select a square for your move by typing a digit for the row
and a letter for the column with no spaces between.

Good luck! Press Enter to start.
```

Tips:


```
 +---+---+---+---+---+---+---+---+
Please enter your move (row column- no space): _
```

Be careful about the other format of output which I did not list here.

If you're not clear about the format of the output, take a close look on my program.

You need to use macro definition to decide SIZE (the length of the chessboard) as follow:

```
#define SIZE 8
```

Write an AI of this game in your code. Your AI should not be too stupid, and the algorithm at least should be reasonable. You can finish the other part first. Some possible algorithm about AI will be uploaded as soon as possible,

You're also encouraged to use your own algorithm. Good AI algorithm will get bonus.

BTW, I played with my AI many times and I only won one time. (To be honest, I am not good at this kind of game.)

Introduction:

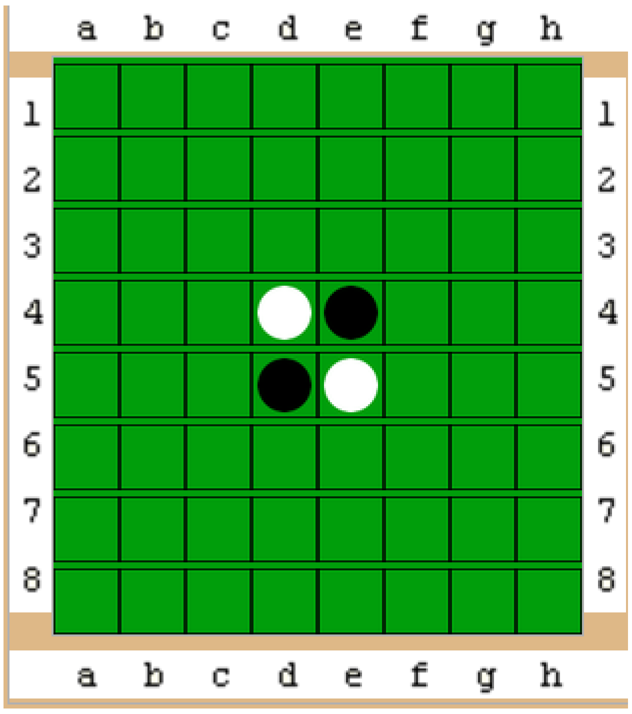
Reversi is a strategy board game for two players, played on a board. Players take turns placing disks on the board with their assigned piece. During a play, any disks of the opponent's piece that are in a straight line and bounded by the disk just placed and another disk of the current player's piece are turned over to the current player's piece.

The object of the game is to have the majority of disks turned to display your piece when the last playable empty square is filled.

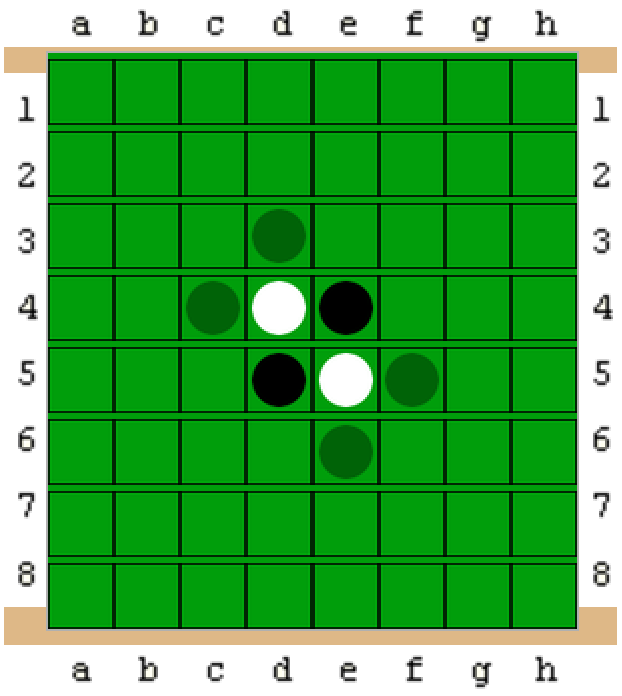
Rules:

The first four pieces go in the center, but in a standard diagonal pattern, rather than being placed by players

The rules state that the game begins with four disks placed in a square in the middle of the grid, two facing white side up, two pieces with the dark side up, with same-colored disks on a diagonal with each other. Convention has initial board position such that the disks with dark side up are to the north-east and south-west (from both players' perspectives), though this is only marginally meaningful to play (where opening memorization is an issue, some players may benefit from consistency on this).

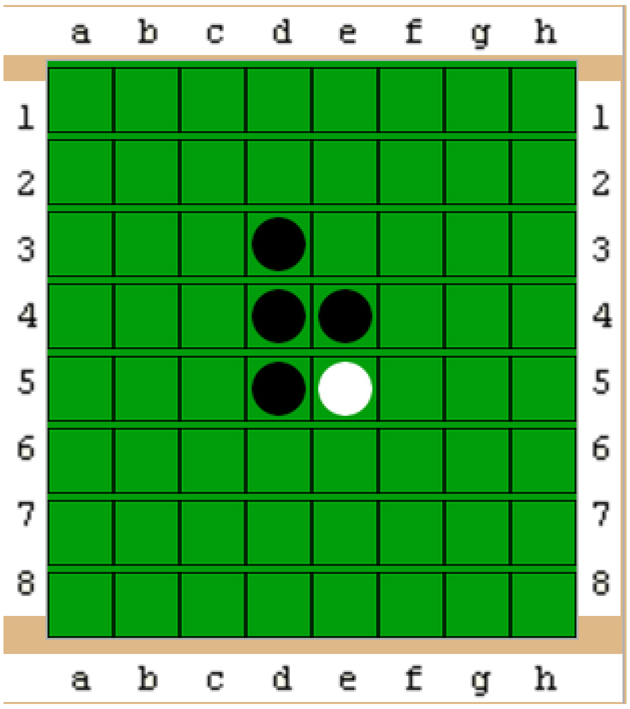


Dark must place a piece with the dark side up on the board, in such a position that there exists at least one straight (horizontal, vertical, or diagonal) occupied line between the new piece and another dark piece, with one or more contiguous light pieces between them. In the below situation, dark has the following options indicated by translucent pieces:

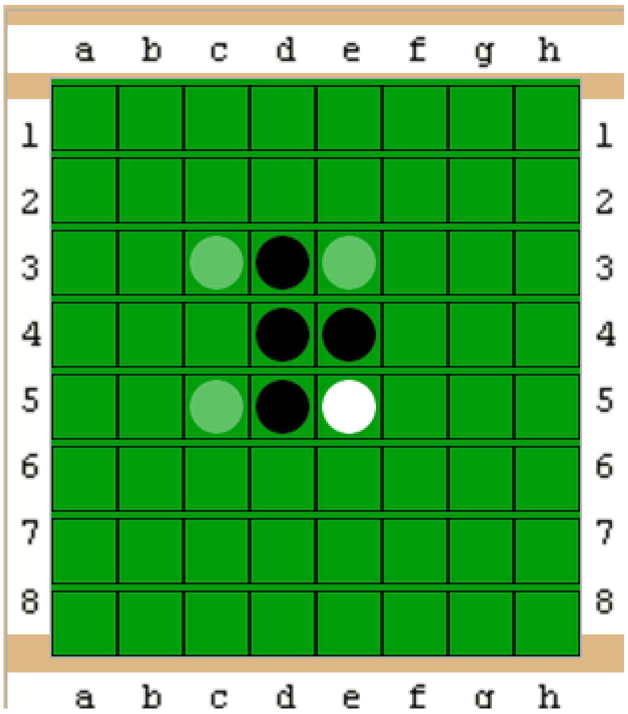


After placing the piece, dark turns over (flips, captures) all light pieces lying on a straight line between the new piece and any anchoring dark pieces. All reversed pieces now show the dark side, and dark can use them in later moves—unless light has reversed them back in the meantime. In other words, a valid move is one where at least one piece is reversed.

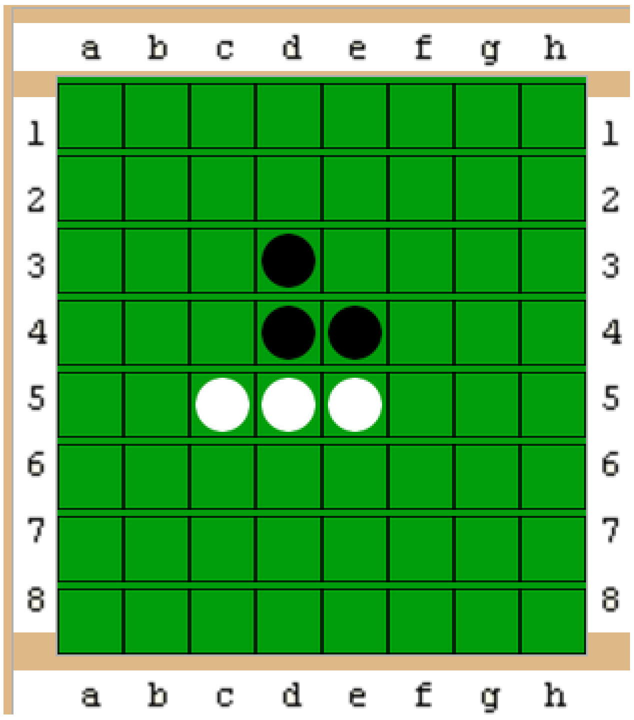
If dark decided to put a piece in the topmost location (all choices are strategically equivalent at this time), one piece gets turned over, so that the board appears thus:



Now light plays. This player operates under the same rules, with the roles reversed: light lays down a light piece, causing a dark piece to flip. Possibilities at this time appear thus (indicated by transparent pieces):

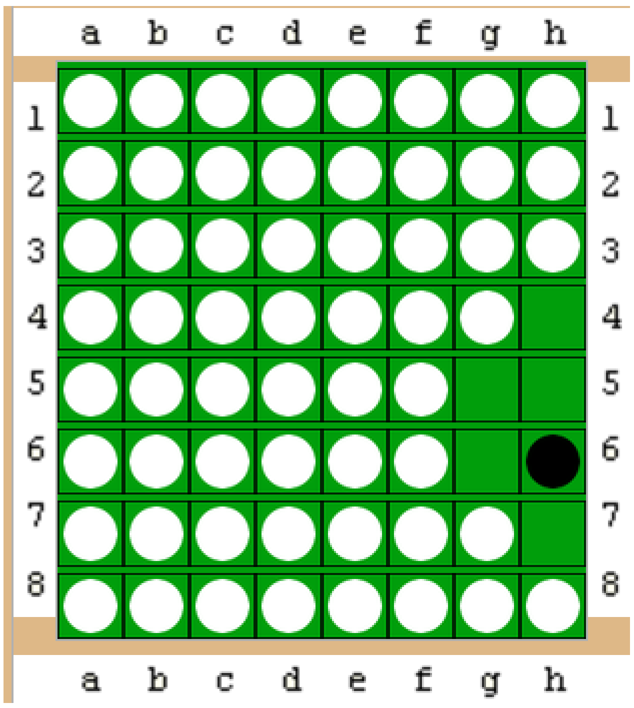


Light takes the bottom left option and reverses one piece:



Players take alternate turns. If one player can not make a valid move, play passes back to the other player. When neither player can move, the game ends. This occurs when the grid has filled up or when neither player can legally place a piece in any of the remaining squares. This means the game may end before the grid is completely filled. This possibility may occur because one player has no pieces remaining on the board in that player's color. In over−the−board play this is generally scored as if the board were full (64 - 0).

Example where the game ends before the grid is completely filled:



The player with the most pieces on the board at the end of the game wins.

Cite Wiki: https://en.wikipedia.org/wiki/Reversi

Question 2

Search on the Internet and explain to TA what the following expression mean:

```
int (*(*fun(int))[4])(double);
char*(*(*fp1)(int))[10];
```

Extra question

The other attachment is the code for the following problem.

It is about bitwise operation priority level of operators in C.

You task is to understand it and rewrite it in a different way but in the

same or similar algorithm.

Try to understand it~

D - Digital Clock

Electronic equipment commonly uses 7-segment elements to show numbers. A 7-segment element gets its name because it uses 7 line segments to show a shape: 3 horizontal segments and 4 vertical segments. Each segment can be independently switched on and off, making it possible to show any digit from 0 to 9.

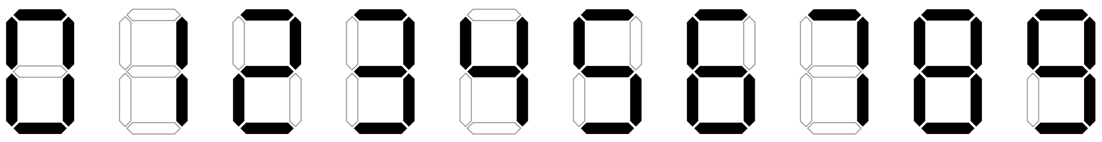


Figure 1: The 7-segment patterns of the decimal digits.

To show more than 1 digit at the same time, a larger display is constructed by putting multiple 7-segment elements next to each other. In this case, each segment in each digit can be independently switched on and off. For example, electronic alarm clocks typically use four 7-segment elements to show the time of day: two digits for the hour (00 to 23) and two digits for the minutes (00 to 59).

You managed to find an old alarm clock in a dusty corner of the basement. Unfortunately, it does not seem to work very well: it does not always show the correct time. You suspect that this is due to faulty wires in the 7-segment display.

As a result of the faults, some of the segments may not be working at all; they never light up, no matter which digits the clock tries to put on its display. On the other hand, the segments that do sometimes light up should all be working correctly. You thus assume that each of the 28 segments on the clock is either completely broken (never lights up) or is working perfectly fine (lights up precisely when it should).

Problem

You want to know what time it is, of course, but the silly clock is not making it very easy. You have been watching the clock for some time, writing down the pattern of digits on its display every minute. (Note that it is possible that the display stays the same for a few minutes. When that happens, you just write down the same pattern several times.)

Your task is to find out what time the clock was really trying to show when you wrote down the first pattern of the sequence. The answer must be consistent with all observations of the clock display, for the first minute as well as for the following minutes, under the assumption that each display segment is either completely broken (never lights up) or is working perfectly fine.

There may be multiple possible answers. In that case, you should make a list of all possible answers, in order of increasing time of day.

It is still possible that the clock is really broken in some other way than just faulty segments. In that case, it may happen that there is no possible answer which is consistent with all observations.

Input

For each test case, there is one line of input containing the following items:

- A positive integer N ($1 \leq N \leq 50$), the number of minutes you have been watching the clock.
- N items, each representing a pattern of digits that was observed on the clock.
  Each pattern is formatted as two decimal digits, followed by a '`:`' character, followed by two more decimal digits.

  The patterns are listed in the order in which they were seen on the clock.

It is theoretically possible for a faulty 7-segment display to show a shape which does not correspond to any of the digits from 0 to 9. However, for some mysterious reason, this never happened during the time you were watching the clock.

Output

For each test case, give one line of output.

If there is at least one possible answer, print a list of all possible answers separated by spaces. Each possible answer must be a valid 24-hour clock time, formatted as two digits (00 to 23), followed by a '`:`' character, followed by two digits (00 to 59).
The list of possible answers must be printed in order of increasing time of day.

If there is no possible answer, print the word '`none`'.

Example

input	output

`1 88:88`
`2 23:25 23:26`
`3 71:57 71:57 71:07`
	`none`
	`23:25`
	`00:58 03:58 07:58 08:58`