

プログラミング A

2組

第二回レポート

学籍番号      09b20065

氏名            ブアマニー   タンピモン

提出日          9/8/2020

## プログラム全体の説明

### プログラム全体の動作説明

This program allowed player to play Othello with computer which is coded using greedy algorithm. The program firstly asked the player for the turn (First/Second) and then the game start. While the game is played, the program record each played and will later put in the txt. File called "history.txt".

### ソースコード

```
main()
{
    int board[BOARD_SIZE][BOARD_SIZE],user,place,cnt=0;
    int history[HISTORY_SIZE];
    history[0]=-2;
    init_board(board);
    char ans[100];
    printf("Is player first(y/n):");
    scanf("%s",ans);
    if(ans[0]=='y' || ans[0]=='Y') user = 1;
    else user = -1;
    for(int turn=1;turn<HISTORY_SIZE;turn++){
        int stone1 = count_stone(1, board);
        int stone2 = count_stone(-1, board);
        if(stone1==0 || stone2==0) break;
        else if(stone1+stone2==64) break;
        printf("Turn %d[%c]¥n",turn,(turn%2==1)?'o':'x');
        printf("Score: P1 | P2¥n");
        printf("      %d | %d ¥n",stone1,stone2);
        print_board(board);
        if((turn%2)*2-1 == user){
            place = input_place(user, board);
```

```

    }
    else{
        place= computer(-user,board);
    }
    update_history(place,history);
    if(place!=-1){
        cnt=0;
        place_stone((turn%2)*2-1, place/BOARD_SIZE, place%BOARD_SIZE,board);
        printf("PUT AT %d %d¥n",place/BOARD_SIZE, place%BOARD_SIZE);
    }
    else{
        cnt++;
        if(cnt==2){
            break;
        }
    }
    printf("-----¥n");
}
printf("=====¥n");
printf("Game Ended!!!¥n");
print_board(board);
printf("Score: P1 | P2¥n");
printf("    %d | %d ¥n",count_stone(1, board),count_stone(-1, board));
save_history(history);
}

```

## プログラムの説明

The program first assigned variable called “board” to collect the board’s state and history to collect each player turn. First the program ask the player whether he will start, and then the program will start. First the program will print out the amount of current stones for both players and printout the board. If its user turn, the program

will ask the user to input coordinate number using "input\_place" function, or else, it's the computer turn, the program will ask the "computer" function for the coordinate they prefer. Then the history will be updated with the new coordinate and the stone will be put there. If the coordinate is -1 it implies that the player pass that turn. The game will repeat this until the game ended. In one Othello game, the maximum turn will be 60 but it might ended beforehand if there's no possible move for both players anymore. That's how the first checkpoint work, if any of the player have zero stone left, it's an immediate win for the other player. The second one is that if both player decided to pass intentionally or rule-based, that means the game has already ended. After the program ended, the program will print out the board result with the score and will save the history to history.txt as mentioned above.

### **動作テスト内容と結果**

Tested by playing Othello with the program 7 times by different players with different skills while checking the results in every move to make sure that the overall result is correct.

Player1 2 4  
 Player2 4 5  
 Player1 5 4  
 Player2 2 3  
 Player1 4 2  
 Player2 4 1  
 Player1 5 1  
 Player2 6 5  
 Player1 6 4  
 Player2 2 5  
 Player1 4 0  
 Player2 7 4  
 Player1 4 6  
 Player2 5 6  
 Player1 1 5  
 Player2 0 5  
 Player1 3 5  
 Player2 4 7  
 Player1 5 7  
 Player2 1 4  
 Player1 3 7  
 Player2 3 6  
 Player1 0 3  
 Player2 0 4  
 Player1 0 6  
 Player2 2 7  
 Player1 5 5  
 Player2 6 7  
 Player1 7 5  
 Player2 7 6  
 Player1 7 3  
 Player2 7 2  
 Player1 1 3  
 Player2 0 2  
 Player1 0 1  
 Player2 5 3  
 Player1 6 3  
 Player2 6 2  
 Player1 2 6  
 Player2 5 2  
 Player1 6 1  
 Player2 7 0  
 Player1 7 1  
 Player2 5 0  
 Player1 7 7  
 Player2 3 0  
 Player1 1 7  
 Player2 3 2  
 Player1 3 1  
 Player2 1 2  
 Player1 2 1  
 Player2 2 0  
 Player1 6 6  
 Player2 6 0  
 Player1 1 1  
 Player2 1 6  
 Player1 0 7  
 Player2 0 1  
 Player1 0 0  
 Player2 1 0

```

Game Ended!!!

  0 1 2 3 4 5 6 7
0  o x x x x x x x
1  o x x x o x x o
2  o x o o x x x o
3  o o x o o o x o
4  o o o x o o o o
5  o o o o x o o o
6  o o o o o o o o
7  o o o o o o o o

Score: P1 | P2
      44 | 20
  
```

## Test1

This playtest is performed by a friend who is a beginner in Othello game. He played as the second player. The computer win with the score of 44:20. He stated that the program works correctly according to Othello's rule and is harder than the Easy level in Othello game online.

Player1 4 2  
Player2 5 2  
Player1 6 2  
Player2 2 3  
Player1 2 4  
Player2 6 1  
Player1 7 0  
Player2 5 3  
Player1 2 2  
Player2 1 3  
Player1 0 2  
Player2 0 3  
Player1 0 4  
Player2 4 5  
Player1 6 3  
Player2 4 1  
Player1 3 0  
Player2 7 2  
Player1 4 6  
Player2 5 0  
Player1 5 1  
Player2 5 4  
Player1 7 3  
Player2 3 2  
Player1 7 1  
Player2 1 2  
Player1 4 0  
Player2 2 0  
Player1 2 1  
Player2 3 1  
Player1 6 4  
Player2 4 7  
Player1 2 5  
Player2 2 6  
Player1 3 5  
Player2 7 5  
Player1 2 7  
Player2 1 5  
Player1 5 7  
Player2 6 7  
Player1 7 4  
Player2 3 6  
Player1 7 6  
Player2 5 6  
Player1 0 -1  
Player2 1 6  
Player1 0 7  
Player2 6 5  
Player1 3 7  
Player2 1 4  
Player1 7 7  
Player2 1 7  
Player1 0 5  
Player2 0 -1  
Player1 5 5  
Player2 0 6  
Player1 0 -1  
Player2 6 6  
Player1 0 -1  
Player2 0 -1  
Player1 0 -1

```
Game Ended!!!  
      0 1 2 3 4 5 6 7  
      0 . . x x x x . o  
      1 . . x x x x o o  
      2 o o x x x o o o  
      3 o o x x x o o o  
      4 o x x x x o o o  
      5 o x o o o x o o  
      6 x x x x x x x o  
      7 o x o o o o o o  
Score: P1 | P2  
      31 | 28
```

## Test2

This playtest is performed by my dad who is an intermediate player in Othello game. He played as the second player. The computer win with the score of 31:28. All the program works correctly according to law.

Player1 4 2  
 Player2 5 4  
 Player1 4 5  
 Player2 3 2  
 Player1 2 2  
 Player2 4 6  
 Player1 4 7  
 Player2 2 1  
 Player1 2 0  
 Player2 3 6  
 Player1 5 3  
 Player2 5 2  
 Player1 3 5  
 Player2 2 4  
 Player1 3 7  
 Player2 5 7  
 Player1 6 7  
 Player2 2 7  
 Player1 3 1  
 Player2 7 7  
 Player1 5 5  
 Player2 3 0  
 Player1 2 3  
 Player2 1 0  
 Player1 2 5  
 Player2 2 6  
 Player1 5 1  
 Player2 6 4  
 Player1 7 4  
 Player2 5 0  
 Player1 5 6  
 Player2 7 3  
 Player1 7 2  
 Player2 6 5  
 Player1 7 5  
 Player2 0 -1  
 Player1 6 3  
 Player2 0 -1  
 Player1 1 3  
 Player2 0 4  
 Player1 1 5  
 Player2 6 2  
 Player1 1 2  
 Player2 0 5  
 Player1 0 3  
 Player2 0 2  
 Player1 1 6  
 Player2 4 1  
 Player1 1 4  
 Player2 0 7  
 Player1 4 0  
 Player2 0 1  
 Player1 1 7  
 Player2 0 6  
 Player1 0 0  
 Player2 6 1  
 Player1 6 0  
 Player2 7 0  
 Player1 7 1  
 Player2 7 6  
 Player1 6 6  
 Player2 0 -1  
 Player1 1 1

```

Game Ended!!!
      0 1 2 3 4 5 6 7
      0 o o o o o o o x
      1 x o o o o o o x
      2 x o o o o x o x
      3 x o x o x x o x
      4 x o o x o x o x
      5 x o x o o o o x
      6 x x x x x x o x
      7 x o x x x x x x
Score: P1 | P2
      32 | 32
  
```

### Test3

This playtest is performed by my friend who is an intermediate player in Othello game. He played as the first player. It's a tie with the score of 32:32. All the program works correctly according to law.

Player1 5 3  
 Player2 3 2  
 Player1 2 1  
 Player2 6 3  
 Player1 3 5  
 Player2 2 6  
 Player1 7 3  
 Player2 3 1  
 Player1 2 3  
 Player2 1 2  
 Player1 3 0  
 Player2 4 1  
 Player1 5 2  
 Player2 5 0  
 Player1 2 5  
 Player2 7 4  
 Player1 7 5  
 Player2 2 4  
 Player1 5 1  
 Player2 5 4  
 Player1 5 5  
 Player2 4 2  
 Player1 6 4  
 Player2 4 0  
 Player1 6 2  
 Player2 2 0  
 Player1 2 2  
 Player2 5 6  
 Player1 2 7  
 Player2 7 2  
 Player1 7 1  
 Player2 1 5  
 Player1 3 6  
 Player2 3 7  
 Player1 4 7  
 Player2 1 3  
 Player1 0 2  
 Player2 0 3  
 Player1 0 4  
 Player2 1 4  
 Player1 0 5  
 Player2 4 5  
 Player1 6 5  
 Player2 5 7  
 Player1 6 7  
 Player2 4 6  
 Player1 6 6  
 Player2 0 -1  
 Player1 0 1  
 Player2 0 -1  
 Player1 0 -1  
 Player2 0 -1

```

Game Ended!!!
      0 1 2 3 4 5 6 7
0  .  .  x x x x .  .
1  o  .  x x x x .  o
2  o  o  o o o x o o
3  o  x  o o o x x o
4  o  o  x x o x o o
5  o  o  o x x o o o
6  .  .  o x x o o .
7  .  .  o o o o o .
Score: P1 | P2
      33 | 19
  
```

## Test 4

This playtest is performed by my brother who is an advanced player in Othello game. He played as the first player. The computer loses with the score of 33:19. He stated that the program works correctly according to Othello's rule. He suggested the program not to give up half way of PASSing too much. Also, he suggested giving score to each coordinate and add some random variable to it so that the program will be less rigid.



Player1 4 2  
 Player2 3 2  
 Player1 2 2  
 Player2 3 1  
 Player1 4 0  
 Player2 3 0  
 Player1 2 0  
 Player2 5 2  
 Player1 6 2  
 Player2 2 1  
 Player1 3 5  
 Player2 4 1  
 Player1 5 3  
 Player2 4 5  
 Player1 4 6  
 Player2 5 1  
 Player1 5 0  
 Player2 3 6  
 Player1 3 7  
 Player2 5 4  
 Player1 5 5  
 Player2 7 2  
 Player1 1 0  
 Player2 4 7  
 Player1 5 7  
 Player2 2 7  
 Player1 1 7  
 Player2 5 6  
 Player1 6 4  
 Player2 1 2  
 Player1 6 7  
 Player2 7 5  
 Player1 7 4  
 Player2 7 3  
 Player1 2 3  
 Player2 2 4  
 Player1 0 2  
 Player2 1 3  
 Player1 6 3  
 Player2 6 5  
 Player1 0 3  
 Player2 2 5  
 Player1 2 6  
 Player2 1 5  
 Player1 1 4  
 Player2 0 -1  
 Player1 0 4  
 Player2 0 5  
 Player1 0 6  
 Player2 6 1  
 Player1 0 -1  
 Player2 0 -1

```

=====
Game Ended!!!
      0 1 2 3 4 5 6 7
      0 . o o o o o . .
      1 . . o o o o x .
      2 o o o o x x x x
      3 o o o x x o o x
      4 o o o x o x x x
      5 o o x x x x x x
      6 o . o o o o . .
      7 . o o o o o o .
Score: P1 | P2
      35 | 18
  
```

## Test 5

This playtest is performed by the creator, Thanpimon Buamanee. The player win with the score of 35:18. Player 1 is me and player 2 is the computer. The program works correctly according to Othello's rule. Still, the computer's operation is rigid and didn't work well on endgame.

Player1 4 2  
Player2 3 2  
Player1 2 2  
Player2 3 1  
Player1 2 4  
Player2 3 5  
Player1 2 5  
Player2 5 3  
Player1 5 2  
Player2 4 1  
Player1 3 0  
Player2 2 0  
Player1 1 0  
Player2 4 0  
Player1 5 0  
Player2 5 1  
Player1 4 5  
Player2 1 2  
Player1 5 4  
Player2 4 6  
Player1 4 7  
Player2 6 2  
Player1 2 3  
Player2 2 6  
Player1 0 2  
Player2 5 7  
Player1 6 7  
Player2 5 5  
Player1 7 2  
Player2 2 1  
Player1 5 6  
Player2 0 3  
Player1 0 4  
Player2 6 4  
Player1 1 3  
Player2 3 7  
Player1 6 3  
Player2 7 7  
Player1 3 6  
Player2 1 4  
Player1 2 7  
Player2 1 7  
Player1 6 5  
Player2 7 5  
Player1 7 6  
Player2 7 4  
Player1 6 6  
Player2 1 5  
Player1 0 5  
Player2 6 1  
Player1 6 0  
Player2 1 6  
Player1 0 -1  
Player2 0 -1

```
Game Ended!!!
  0 1 2 3 4 5 6 7
  0 . o o o o o o .
  1 . . x x x o x .
  2 o x o x o x x o
  3 o x o o x x x .
  4 o x o x o o x x
  5 o x x o o o x x
  6 . x x o o o x o
  7 . x x x x x x x
Score: P1 | P2
      26 | 30
```

## Test 6

This playtest is performed by the creator, Thanpimon Buamane. The computer win with the score of 26:30. The program work correctly according to Othello's rule. Still, the program focus too much on the side and edge that it didn't manage to get much stone from the opponent. Without the mistake I've made, allowing the computer to get the (7, 7) position, the program would have lost.

Player1 4 2  
 Player2 5 2  
 Player1 6 2  
 Player2 5 4  
 Player1 2 5  
 Player2 3 2  
 Player1 6 4  
 Player2 3 5  
 Player1 2 1  
 Player2 5 3  
 Player1 2 4  
 Player2 2 3  
 Player1 2 2  
 Player2 3 1  
 Player1 2 0  
 Player2 1 3  
 Player1 0 3  
 Player2 7 4  
 Player1 7 5  
 Player2 7 6  
 Player1 1 4  
 Player2 3 0  
 Player1 4 0  
 Player2 6 5  
 Player1 4 1  
 Player2 5 1  
 Player1 6 3  
 Player2 1 5  
 Player1 5 0  
 Player2 1 2  
 Player1 0 2  
 Player2 7 3  
 Player1 0 5  
 Player2 4 5  
 Player1 0 4  
 Player2 0 -1  
 Player1 5 5  
 Player2 4 6  
 Player1 4 7  
 Player2 5 7  
 Player1 6 7  
 Player2 2 6  
 Player1 2 7  
 Player2 1 1  
 Player1 7 2  
 Player2 7 1  
 Player1 3 6  
 Player2 3 7  
 Player1 5 6  
 Player2 7 7  
 Player1 0 0  
 Player2 1 7  
 Player1 6 6  
 Player2 1 6  
 Player1 0 7  
 Player2 0 6  
 Player1 0 -1  
 Player2 0 1  
 Player1 0 -1  
 Player2 1 0  
 Player1 0 -1  
 Player2 6 0  
 Player1 7 0  
 Player2 6 1

```

=====
Game Ended!!!
      0 1 2 3 4 5 6 7
    0 o o o o o o o o
    1 x x x o o x x x
    2 x x o x x x x x
    3 x o x x x x o x
    4 x o x x o x o x
    5 x x o x x o x x
    6 x x x o o x o x
    7 o x x x x x x x
  
```

## Test 7

This playtest is performed by the creator, Thanpimon Buamanee. With computer being player 1 and creator being the second player. The program work correctly according to Othello's rule. Even though the result ended with the computer losing with the score of 22:42, during the competition the computer is always in the lead and ended up losing with the endgame's flaw. This flaw is created by the lacking of the intuition, the skill to predict other's moves and purely play by the order given.

## 関数 print\_board(int board[][BOARD\_SIZE]) の説明

### ソースコード

```
void print_board(int board[][BOARD_SIZE]){
    printf("¥t ");
    for(int i=0;i<BOARD_SIZE;i++){
        printf("%2d",i);
    }
    printf("¥n");
    for(int j=0;j<BOARD_SIZE;j++) {
        printf("¥t%2d ",j);
        for(int i=0;i<BOARD_SIZE;i++) {
            switch(board[i][j]) {
                case 1: printf("o ");break;
                case -1: printf("x ");break;
                default : printf(". ");break;
            }
        }
        printf("¥n");
    }
}
```

### プログラムの説明

This function will print out the current board status from the inputted board. Data in board will be consisted of 0, 1, -1 in which 0 is the blank space, 1 is the first player's stone and -1 is the second player's stone. The program will first write the x-coordinate at the upper part of the program so the user will be able to perceive the board easily.

When start printing each line, the first thing printed in the line will be y-coordinate. If the number in board's location is 0, the program will print out "." which represent that that coordinate is still blank. On the other hand, the symbol "o" and "x" represent player1 and player2's symbol sequentially.

## 動作テスト内容と結果

Tested by putting this function in check program as shown below. To create the board, the init\_board function is checked at the same time

```
#include<stdio.h>
#define BOARD_SIZE 8
void print_board(int board[][BOARD_SIZE]);
void init_board(int board[][BOARD_SIZE]);
main()
{
    int board[BOARD_SIZE][BOARD_SIZE],user,place,cnt=0;
    init_board(board);
    print_board(board);
    board[0][0] = -1;
    board[1][1] = 1;
    print_board(board);
}
```

The first print\_board print the initial shape of the board. We tried changing each element of the board to see if how the board print change accordingly

(        o if board[][]=1,  
         x if board[][]=-1  
         . if board[][]=0        )

```

0 1 2 3 4 5 6 7
0 . . . . . . .
1 . . . . . . .
2 . . . . . . .
3 . . . o x . .
4 . . . x o . .
5 . . . . . . .
6 . . . . . . .
7 . . . . . . .
0 1 2 3 4 5 6 7
0 x . . . . . .
1 . o . . . . .
2 . . . . . . .
3 . . . o x . .
4 . . . x o . .
5 . . . . . . .
6 . . . . . . .
7 . . . . . . .
```

## 関数 init\_board(int board[][BOARD\_SIZE]) の説明

### ソースコード

```
void init_board(int board[][BOARD_SIZE]){
    for(int i=0;i<BOARD_SIZE;i++){
        for(int j=0;j<BOARD_SIZE;j++){
            board[j][i]=0; //x y
        }
    }
    board[(BOARD_SIZE/2)][(BOARD_SIZE/2)] = 1;
    board[(BOARD_SIZE/2)-1][(BOARD_SIZE/2)] = -1;
    board[(BOARD_SIZE/2)][(BOARD_SIZE/2)-1] = -1;
    board[(BOARD_SIZE/2)-1][(BOARD_SIZE/2)-1] = 1;
}
```

### プログラムの説明

This function help set the board's starting shape with 4 stones on port, stone of each player are positioned in horizontal shape as shown in image. The program first set all the numbers in the array to be 0 , representing the blank space. Then the program set the middle part of the board to the shape mentioned.

## 動作テスト内容と結果

```
#include<stdio.h>

#define BOARD_SIZE 8

void print_board(int board[][BOARD_SIZE]);

void init_board(int board[][BOARD_SIZE]);

main()

{
    int board[BOARD_SIZE][BOARD_SIZE],user,place,cnt=0;

    init_board(board);

    print_board(board);

}
```

This function is checked together with `print_board` to print out the result using the main program as shown below. We tried in different board size to make sure that it'll work in any even numbers less than 100.

BOARD\_SIZE = 8

|   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | . | . | . | . | . | . | . | . |
| 1 | . | . | . | . | . | . | . | . |
| 2 | . | . | . | . | . | . | . | . |
| 3 | . | . | . | o | x | . | . | . |
| 4 | . | . | . | x | o | . | . | . |
| 5 | . | . | . | . | . | . | . | . |
| 6 | . | . | . | . | . | . | . | . |
| 7 | . | . | . | . | . | . | . | . |

BOARD\_SIZE = 12

[illegible]

BOARD\_SIZE = 20

[illegible]

|  |
|--|
| 関数 count_stone(int player_id, int board[][BOARD_SIZE]) の説明 . |
|--|

### ソースコード

```
int count_stone(int player_id, int board[][BOARD_SIZE]){
    int sum=0;
    for(int i=0;i<BOARD_SIZE;i++){
        for(int j=0;j<BOARD_SIZE;j++){
            if(board[j][i]==player_id) sum++; //x y
        }
    }
    return sum;
}
```

### プログラムの説明

This function count the number of the stone of player player\_id in the current board and return the amount of it.



## 動作テスト内容と結果

The function is tested using the following program, together with print\_board function and init\_board function.

```
#include<stdio.h>

#define BOARD_SIZE 8

int count_stone(int player_id, int board[][BOARD_SIZE]);

void print_board(int board[][BOARD_SIZE]);

void init_board(int board[][BOARD_SIZE]);

main(){

    int board[BOARD_SIZE][BOARD_SIZE],user,place,cnt=0;

    init_board(board);print_board(board);

    printf("o is: %d\n x is: %d\n ",count_stone(1,board),count_stone(-1,board));

    board[0][0]=1;

    print_board(board);

    printf("o is: %d\n x is: %d\n ",count_stone(1,board),count_stone(-1,board));

    board[1][0]=-1;

    print_board(board);

    printf("o is: %d\n x is: %d\n ",count_stone(1,board),count_stone(-1,board));

}
```

At first we let the program count the stone for the starting board. Then we tried adding 1 stone from player one and see if the new counting is correct or not. Then we tried adding more stone of the second player and check if the counting is correct compare to the image.

```

  0 1 2 3 4 5 6 7
0 . . . . . . .
1 . . . . . . .
2 . . . . . . .
3 . . o x . . .
4 . . x o . . .
5 . . . . . . .
6 . . . . . . .
7 . . . . . . .
o is: 2
x is: 2

  0 1 2 3 4 5 6 7
0 o . . . . . .
1 . . . . . . .
2 . . . . . . .
3 . . o x . . .
4 . . x o . . .
5 . . . . . . .
6 . . . . . . .
7 . . . . . . .
o is: 3
x is: 2

  0 1 2 3 4 5 6 7
0 o x . . . . .
1 . . . . . . .
2 . . . . . . .
3 . . o x . . .
4 . . x o . . .
5 . . . . . . .
6 . . . . . . .
7 . . . . . . .
o is: 3
x is: 3
```

関数 place\_stone(int player\_id,int x,int y,int board[][BOARD\_SIZE])の説明

## ソースコード

```
void place_stone(int player_id, int x, int y, int board[][BOARD_SIZE]){
    int i;
    board[x][y]=player_id;
    //check up
    for(i=x-1;i>=0 && board[i][y]==-player_id;i--);
    if(i>=0 && board[i][y]==player_id){
        for(;i!=x;i++) board[i][y]=player_id;
    }
    //check down
    for(i=x+1;i<BOARD_SIZE && board[i][y]==-player_id;i++);
    if(i<BOARD_SIZE && board[i][y]==player_id){
        for(;i!=x;i--) board[i][y]=player_id;
    }
    //check right
    for(i=y+1;i<BOARD_SIZE && board[x][i]==-player_id;i++);
    if(i<BOARD_SIZE && board[x][i]==player_id){
        for(;i!=y;i--)board[x][i]=player_id;
    }
    //check left
    for(i=y-1;i>=0 && board[x][i]==-player_id;i--);
    if(i>=0 && board[x][i]==player_id){
        for(;i!=y;i++)board[x][i]=player_id;
    }
    //check ㄥup
    for(i=1;x-i>=0 && y-i>=0 &&board[x-i][y-i]==-player_id;i++);
```

```

if(x-i>=0 && y-i>=0 && board[x-i][y-i]==player_id){
    for(;i!=0;i--)board[x-i][y-i]=player_id;
}
//check /up
for(i=1;x+i<BOARD_SIZE && y-i>=0 &&board[x+i][y-i]==-player_id;i++);
if(x+i<BOARD_SIZE && y-i>=0 && board[x+i][y-i]==player_id){
    for(;i!=0;i--)board[x+i][y-i]=player_id;
}
//check /down
for(i=1;x-i>=0 && y+i<BOARD_SIZE &&board[x-i][x+i]==-player_id;i++);
if(x-i>=0 && y+i<BOARD_SIZE && board[x-i][y+i]==player_id){
    for(;i!=0;i--)board[x-i][y+i]=player_id;
}
//check ¥down
for(i=1;x+i<BOARD_SIZE && y+i<BOARD_SIZE &&board[x+i][y+i]==-
player_id;i++);
if(x+i<BOARD_SIZE && y+i<BOARD_SIZE && board[x+i][y+i]==player_id){
    for(;i!=0;i--)board[x+i][y+i]=player_id;
}
}

```

## プログラムの説明

This function place the stone in the mentioned area and swap the opponent stone according to Othello's rule. First, this function received the place the stone is put, and change the number of that coordinate in the array to the number of the player (player1: 1, player2: -1). Then, the program search for swappable stone in 8 direction. In each direction, first the program go to the stone next to the lastly put-in coordinate, then check if its opponent stone, and keep on repeating until there's no stone or it's the player's stone. If its player's stone, it means that all the stone that we

have come across that are opponent's color are to be eaten and converted to our stone. If not, we won't be able to move it.

### 動作テスト内容と結果 (1 direction)

The code downward will show how the "o" is eaten given "x" in different position circling around the "o" square of position (2,2) (6,6) and one x in the middle (4,4) position. With that it will be able to check how eating pattern work in all 8 directions.

```
#include<stdio.h>
#define BOARD_SIZE 8
void print_board(int board[][BOARD_SIZE]);
void init_board(int board[][BOARD_SIZE]);
void place_stone(int player_id, int x, int y, int board[][BOARD_SIZE]);
main() {
    int board[BOARD_SIZE][BOARD_SIZE],user,place,cnt=0;
    int y[] = {1,7};
    for(int u=0;u<2;u++){
        for(int p=0;p<7;p++){
            init_board(board);
            for(int i=2;i<7;i++)
                for(int j=2;j<7;j++) board[i][j]=1;
            board[4][4]=-1;
            printf("PUT AT %d %d\n",1+p,y[u]);
            place_stone(-1,1+p,y[u],board);
            print_board(board);
        }
        for(int p=0;p<7;p++){
            init_board(board);
            for(int i=2;i<7;i++)
                for(int j=2;j<7;j++) board[i][j]=1;
            board[4][4]=-1;
            printf("PUT AT %d %d\n",y[u],1+p);
            place_stone(-1,y[u],1+p,board);
            print_board(board); } }
}
```

```
PUT AT 1 1
 0 1 2 3 4 5 6 7
0 . . . . .
1 . x . . . . .
2 . . x o o o o .
3 . . o x o o o .
4 . . o o x o o .
5 . . o o o o o .
6 . . o o o o o .
7 . . . . .

PUT AT 2 1
 0 1 2 3 4 5 6 7
0 . . . . .
1 . . x . . . .
2 . . o o o o o .
3 . . o o o o o .
4 . . o o x o o .
5 . . o o o o o .
6 . . o o o o o .
7 . . . . .

PUT AT 3 1
 0 1 2 3 4 5 6 7
0 . . . . .
1 . . . x . . .
2 . . o o o o o .
3 . . o o o o o .
4 . . o o x o o .
5 . . o o o o o .
6 . . o o o o o .
7 . . . . .

PUT AT 4 1
 0 1 2 3 4 5 6 7
0 . . . . .
1 . . . . x . .
2 . . o o x o o .
3 . . o o x o o .
4 . . o o x o o .
5 . . o o o o o .
6 . . o o o o o .
7 . . . . .

PUT AT 5 1
 0 1 2 3 4 5 6 7
0 . . . . .
1 . . . . . x .
2 . . o o o o o .
3 . . o o o o o .
4 . . o o x o o .
5 . . o o o o o .
6 . . o o o o o .
7 . . . . .

PUT AT 6 1
 0 1 2 3 4 5 6 7
0 . . . . .
```

## 動作テスト内容と結果 (2 direction)

To check 2 similar pattern, we create board shaped mentioned on right upper image and put stone on (2,2),(2,6),(6,2),(6,6) to see how it is eaten and if it's eaten correctly.

```
#include<stdio.h>
#define BOARD_SIZE 8
void print_board(int board[][BOARD_SIZE]);
void init_board(int board[][BOARD_SIZE]);
void place_stone(int player_id, int x, int y, int
board[][BOARD_SIZE]);
main()
{
    int board[BOARD_SIZE][BOARD_SIZE],user,place,cnt=0;
    init_board(board);
    for(int i=2;i<7;i++)
    for(int j=2;j<7;j++)
    {
        board[i][j]=1;
    }
    board[4][4]=-1;
    board[2][2]=0,board[2][6]=0,board[6][2]=0,board[6][6]=0;
    board[4][2]=-1,board[2][4]=-1,board[6][4]=-1,board[4][6]=-1;
    print_board(board);
    place_stone(-1,2,2,board);
    print_board(board);
}
```

|   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | . | . | . | . | . | . | . | . |
| 1 | . | . | . | . | . | . | . | . |
| 2 | . | . | . | o | x | o | . | . |
| 3 | . | . | o | o | o | o | o | . |
| 4 | . | . | x | o | x | o | x | . |
| 5 | . | . | o | o | o | o | o | . |
| 6 | . | . | . | o | x | o | . | . |
| 7 | . | . | . | . | . | . | . | . |
|   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 | . | . | . | . | . | . | . | . |
| 1 | . | . | . | . | . | . | . | . |
| 2 | . | . | x | x | x | o | . | . |
| 3 | . | . | x | x | o | o | o | . |
| 4 | . | . | x | o | x | o | x | . |
| 5 | . | . | o | o | o | o | o | . |
| 6 | . | . | . | o | x | o | . | . |
| 7 | . | . | . | . | . | . | . | . |

|  |
|--|
| 関数 num_obtained_stone(int player_id,int x,int y,int board[][BOARD_SIZE]) |
|--|

|     |
|-----|
| の説明 |
|-----|

## ソースコード

```
int num_obtained_stone(int player_id, int x, int y, int board[][BOARD_SIZE])
{
    int i,sum=0;
    //check up
    for(i=x-1;i>=0 && board[i][y]==-player_id;i--); //printf("UP");
    if(i>=0 && board[i][y]==player_id){
        sum+=(x-i-1);
    }
    //check down
    for(i=x+1;i<BOARD_SIZE && board[i][y]==-player_id;i++); //printf("DOWN");
    if(i<BOARD_SIZE && board[i][y]==player_id){
        sum+=(i-x-1);
    }
    //check right
    for(i=y+1;i<BOARD_SIZE && board[x][i]==-player_id;i++); //printf("RIGHT");
    if(i<BOARD_SIZE && board[x][i]==player_id){
        sum+=(i-y-1);
    }
    //check left
    for(i=y-1;i>=0 && board[x][i]==-player_id;i--); //printf("LEFT");
    if(i>=0 && board[x][i]==player_id){
        sum+=(y-i-1);
    }
    //check 斜up
    for(i=1;x-i>=0 && y-i>=0 && board[x-i][y-i]==-player_id;i++);
    if(x-i>=0 && y-i>=0 && board[x-i][y-i]==player_id){
        sum+=(i-1);
    }
}
```

```

//check /up
for(i=1;x+i<BOARD_SIZE && y-i>=0 &&board[x+i][y-i]==-player_id;i++);
if(x+i<BOARD_SIZE && y-i>=0 && board[x+i][y-i]==player_id){
    sum+=(i-1);
}
//check /down
for(i=1;x-i>=0 && y+i<BOARD_SIZE &&board[x-i][y+i]==-player_id;i++);
if(x-i>=0 && y+i<BOARD_SIZE && board[x-i][y+i]==player_id){
    sum+=(i-1);
}
//check ¥down
for(i=1;x+i<BOARD_SIZE && y+i<BOARD_SIZE &&board[x+i][y+i]==-player_id;i++);
if(x+i<BOARD_SIZE && y+i<BOARD_SIZE && board[x+i][y+i]==player_id){
    sum+=(i-1);
}
return sum;
}

```

## プログラムの説明

This program print how many of opponent stone can be obtained by putting the stone in the mentioned position. The function work in similar to place\_stone function, by searching in 8 different directions. First, we assigned the integer variable "sum" as 0, and then in each direction, we started at the stone next to that mentioned position in one of the right direction. If the stone is the opponent stone, we continue moving until we find the one that's not the opponent stone. If the non-opponent stone are our stone, we get all the opponent stone in that direction but else (no stone), we gain nothing. After summing up stones we obtain, we return that number.

## 動作テスト内容と結果 (1 direction)

Similar to place\_stone function, the code downward will show how the "o" is eaten given "x" in different position circling around the "o" square of position (2,2) (6,6) and one x in the middle (4,4) position. With that it will be able to check how eating pattern work in all 8 directions.

```
#include<stdio.h>
#define BOARD_SIZE 8
int num_obtained_stone(int player_id, int x, int y, int board[][BOARD_SIZE]);
void print_board(int board[][BOARD_SIZE]);
void place_stone(int player_id, int x, int y, int board[][BOARD_SIZE]);

main() {
    int board[BOARD_SIZE][BOARD_SIZE],user,place,cnt=0;
    int y[] = {1,7};
    for(int u=0;u<2;u++){
        for(int p=0;p<7;p++){
            init_board(board);
            for(int i=2;i<7;i++)
                for(int j=2;j<7;j++) board[i][j]=1;
            board[4][4]=-1;
            printf("PUT AT %d %d\n",1+p,y[u]);
            printf("obtain %d stone\n",num_obtained_stone(-1,1+p,y[u],board));
            place_stone(-1,1+p,y[u],board);
            print_board(board);
        }
    }
    for(int p=0;p<7;p++){
        init_board(board);
        for(int i=2;i<7;i++)
            for(int j=2;j<7;j++) board[i][j]=1;
        board[4][4]=-1;
        printf("PUT AT %d %d\n",y[u],1+p);
        printf("obtain %d stone\n",num_obtained_stone(-1,1+p,y[u],board));
        place_stone(-1,y[u],1+p,board);
        print_board(board); } }
```

```

      7 . . . . . . . .
PUT AT 7 6
obtain 0 stone
      0 1 2 3 4 5 6 7
0 . . . . . . . .
1 . . . . . . . .
2 . . o o o o o .
3 . . o o o o o .
4 . . o o x o o .
5 . . o o o o o .
6 . . o o o o o x
7 . . . . . . . .

PUT AT 7 7
obtain 2 stone
      0 1 2 3 4 5 6 7
0 . . . . . . . .
1 . . . . . . . .
2 . . o o o o o .
3 . . o o o o o .
4 . . o o x o o .
5 . . o o o x o .
6 . . o o o o x .
7 . . . . . . . x
```



## 動作テスト内容と結果 (2 direction)

In addition to how we check num\_obtained\_stone function in 2 or more direction, we create board shaped mentioned on right upper image and put stone on (2,2),(2,6),(6,2),(6,6) to see how it is eaten and if it's eaten correctly and the number or stone that is outputted is correct.

```
#include<stdio.h>
#define BOARD_SIZE 8
void print_board(int board[][BOARD_SIZE]);
void init_board(int board[][BOARD_SIZE]);
void place_stone(int player_id, int x, int y, int
board[][BOARD_SIZE]);
main()
{
    int board[BOARD_SIZE][BOARD_SIZE],user,place,cnt=0;
    init_board(board);
    for(int i=2;i<7;i++)
    for(int j=2;j<7;j++)
    {
        board[j][i]=1;
    }
    board[4][4]=-1;
    board[2][2]=0,board[2][6]=0,board[6][2]=0,board[6][6]=0;
    board[4][2]=-1,board[2][4]=-1,board[6][4]=-1,board[4][6]=-1;
    print_board(board);
    int stone = num_obtained_stone(-1,2,2,board);
    printf("\nwe gained %d stone\n",stone);
    place_stone(-1,2,2,board);
    print_board(board);
    //printf("PUT AT %d %d: Get %d stones\n",1+p,y[u],stone);
}
```

```
    0 1 2 3 4 5 6 7
0 . . . . . . .
1 . . . . . . .
2 . . . o x o . .
3 . . o o o o o .
4 . . x o x o x .
5 . . o o o o o .
6 . . . o x o . .
7 . . . . . . .

we gained 3 stone
    0 1 2 3 4 5 6 7
0 . . . . . . .
1 . . . . . . .
2 . . x x x o . .
3 . . x x o o o .
4 . . x o x o x .
5 . . o o o o o .
6 . . . o x o . .
7 . . . . . . .
```

## 関数 `ascii_to_int(char string[])` の説明

### ソースコード

```
int ascii_to_int(char string[]){
    int sum = 0,i=0;
    while(string[i]!='\0'){
        if(string[i]-'0'>=0 && string[i]-'0'<=9){
            sum*=10;
            sum+=string[i]-'0';
            i++;
        }
        else return -1;
    }
    return sum;
}
```

### プログラムの説明

This program convert string into int. First the program input string and we assigned one integer variable called sum as 0. First we start at string position0 (string [0]) if it's between '0'-'9' we add it into sum, else we know that there's non-number character in the list so we have to return -1. After we finish with position 0 we move on to position one, and repeat the process. However, before we add the next number to sum, we have to time 10 to the current sum as to change the digit to one furthermore. After we finish with all of the string, we return the number we got in sum.

## 動作テスト内容と結果

This program below is written to check `ascii_to_int` function. First we check if the program can convert numbers in form of string to number from 1-digit to tens and hundreds. Then we check if the program will return -1 if the input have letters other than 0-9 in it.

```
#include<stdio.h>

int ascii_to_int(char string[]);

main(){
    char a[20];
    while(1){
        scanf("%s",a);
        printf("Output: %d\n",ascii_to_int(a));
    }
}
```

```
0
Output: 0
6
Output: 6
48
Output: 48
146
Output: 146
5t6
Output: -1
-6
Output: -1
```

## 関数 input\_place(int player\_id, int board[][BOARD\_SIZE]) の説明

### ソースコード

```
int input_place(int player_id, int board[][BOARD_SIZE]){
    char x[100],y[100];
    while(1){
        printf("Input x: ");
        scanf("%s",x);
        if(x[0]=='P' && x[1]=='A' && x[2]=='S' && x[3]=='S' && x[4]!='\0') return -1;
        printf("Input y: ");
        scanf("%s",y);
        if(y[0]=='P' && y[1]=='A' && y[2]=='S' && y[3]=='S' && y[4]!='\0') return -1;
        int intx = asci_to_int(x);
        int inty = asci_to_int(y);
        if(board[intx][inty]==0 && num_obtained_stone(player_id,intx,inty,board)!=0) return
intx*BOARD_SIZE+inty;
    }
}
```

### プログラムの説明

This function will tell the user to input number of coordination in the order of x, y respectively and check if that position is valid according to rule and if the user is typing it correctly or not. Note that if the player type "PASS" in any of them it means that he want to pass on that turn. Else, if the input is incorrect, repeat asking for numbers until it's correct. The program will converted x and y which is in string form to intx,inty with integer form by using asci\_to\_int function. Then, the program will return the coordinate in the form of intx\*BOARD\_SIZE+inty.

## 動作テスト内容と結果

Using program below, we check if the function `input_place` is working properly by trying to input many coordinate to the starting board. As first turn is "O", the only available options are (4,2) , (2,4) , (3,5) , (5,3) so if other coordinate and/or number others than coordinate is being input, the program should ask for a new input. If it's correct coordinate, the program will print out the coordinate in the form of `intx*BOARD_SIZE+inty`. It can be seen when (2,4) (4,2) (3,5) (5,3) is inputted, the output is 20,34,29,43 respectively.z

```
#include<stdio.h>

#define BOARD_SIZE 8

#define HISTORY_SIZE 100

void init_board(int board[][BOARD_SIZE]);

int input_place(int player_id, int board[][BOARD_SIZE]);

void print_board(int board[][BOARD_SIZE]);

int num_obtained_stone(int player_id, int x, int y, int board[][BOARD_SIZE]);

int asci_to_int(char string[]);

main(){

    int board[BOARD_SIZE][BOARD_SIZE],k,player_id=1;

    init_board(board);

    print_board(board);

    while(1){

        k = input_place(player_id,board);

        printf("        place = %d\n",k);

    }

}
```

```
    0 1 2 3 4 5 6 7
0 . . . . . . .
1 . . . . . . .
2 . . . . . . .
3 . . o x . . .
4 . . x o . . .
5 . . . . . . .
6 . . . . . . .
7 . . . . . . .

Input x: 0
Input y: 0
Input x: -1
Input y: 2
Input x: 3
Input y: 2
Input x: 3
Input y: 6
Input x: 2
Input y: 3
Input x: 2
Input y: 4
        place = 20
Input x: 4
Input y: 2
        place = 34
Input x: 3
Input y: 5
        place = 29
Input x: 5
Input y: 3
        place = 43
Input x:
```

## 関数 computer(int player\_id, int board[][BOARD\_SIZE])の説明

### ソースコード

```
int computer(int player_id, int board[][BOARD_SIZE]){
    srand(time(0));
    int current_id = -1, current_max = 0, k, i, j, p, q;
    //the most important part
    int special[] = {0, BOARD_SIZE-1};
    for(int p=0; p<2; p++){
        for(int q=0; q<2; q++){
            j= special[p];
            i = special[q];
            if(board[j][i]==0){
                k= num_obtained_stone(player_id, j, i, board);
                if(k>current_max) {
                    current_max = k;
                    current_id = j*BOARD_SIZE+i;
                }
                else if(k!=0 && k==current_max){
                    if(rand()%2==0){
                        current_max = k;
                        current_id = j*BOARD_SIZE+i;
                    }
                }
            }
        }
    }
    if(current_id!=-1) {return current_id;}
    //8 vulnerable spot
    current_id = eightspot(player_id, board, 0);
    if(current_id!=-1) {return current_id;}
```

```

int side[] = {0,BOARD_SIZE-1};
//the side(not include 8 spot)
for(i=2;i<BOARD_SIZE-2;i++){
    for(p=0;p<2;p++){
        j= side[p];
        if(board[j][i] ==0){
            k=num_obtained_stone(player_id,j,i,board);
            if(k>current_max) {
                current_max =k;
                current_id = j*BOARD_SIZE+i;
            }
            else if(k!=0 && k==current_max){
                if(rand()%2==0){
                    current_max =k;
                    current_id = j*BOARD_SIZE+i;
                }
            }
        }
    }
    if(board[i][j] == 0){
        k=num_obtained_stone(player_id,i,j,board);
        if(k>current_max) {
            current_max =k;
            current_id = i*BOARD_SIZE+j;
        }
        else if(k!=0 && k==current_max){
            if(rand()%2==0){
                current_max =k;
                current_id = i*BOARD_SIZE+j;
            }
        }
    }
}
}

```

```

if(current_id!=-1) {return current_id;}
//printf("CANNOT side¥n");
//the rest
for(i=1;i<BOARD_SIZE-1;i++){
    for(j=1;j<BOARD_SIZE-1;j++){
        if((i==1 && j==1)|| (i==1 && j==BOARD_SIZE-2)|| (i==BOARD_SIZE-2 &&
j==1)|| (i==BOARD_SIZE-2 && j==BOARD_SIZE-2));
        else{
            if(board[j][i]==0){
                k= num_obtained_stone(player_id,j,i,board);
                if(k>current_max){
                    current_max=k;
                    current_id = j*BOARD_SIZE+i;
                }
                else if(k!=0 && k==current_max){
                    if(rand()%2==0){
                        current_max =k;
                        current_id = j*BOARD_SIZE+i;
                    }
                }
            }
        }
    }
}
}

if(current_id!=-1) {return current_id;}
//8 vulnerable spot
current_id = eightspot(player_id,board,1);
if(current_id!=-1) {return current_id;}
//(1,1)(1,6)(6,1)(6,6)
if(count_stone(1,board)+count_stone(-1,board)>=48){
    int xx[] = {1,BOARD_SIZE-2,BOARD_SIZE-2,1};
    int yy[] = {1,BOARD_SIZE-2,1,BOARD_SIZE-2};
    int save = -1;

```



```

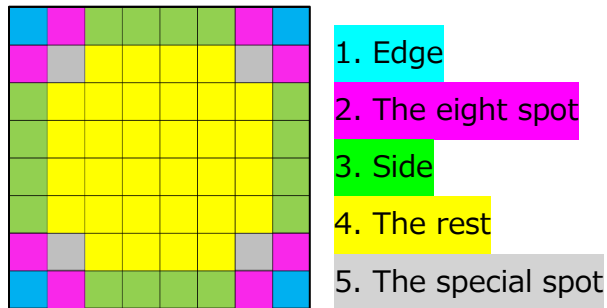
int plus,ending;
for(int p=0;p<4;p++){
    int x =xx[p];
    int y= yy[p];
    if(board[x][y]==0 && num_obtained_stone(player_id,x,y,board)!=0){
        if(xx[i]==1) plus = 1;
        else plus = -1;
        while(x<BOARD_SIZE && y<BOARD_SIZE && x>=0 && y>=0){
            if(xx[p]==yy[p]) {x+=plus;y+=plus;}
            else {x+=plus;y-=plus;}
            if(board[x][y]!=0)ending = board[x][y];
        }
        if(ending==player_id) {return xx[p]*BOARD_SIZE+yy[p];}
        else save = xx[p]*BOARD_SIZE+yy[p];
    }
}
if(save!=-1) return save;
}
for(int i=0;i<BOARD_SIZE;i++){
    for(int j=0;j<BOARD_SIZE;j++){
        if(board[j][i]==0 && num_obtained_stone(player_id,j,i,board)!=0)
            return j*BOARD_SIZE+i;
    }
}
}

```

## プログラムの説明

### Function used in this function: eightspot

We separated the board into 5 different parts as shown below.



In each turn, the stone will be put on the more prioritized spot mentioned below.

Except for the eight spot and the special spot, the program will check in each group in the order below if it's possible to put the stone on the edge. If there's more than one spot, the program will choose the spot that captured the most stone. If more than one spot catch the same amount of the stone, it will random one of them using time random function. If there's a possible case to put on among each part of the board, the program will immediately put on that part. (return the edge location) Else, they'll look at other spot in the other part of the board.

### Checking order

1. The edge
2. The eight spot (1<sup>st</sup> time)

Using the eightspot function, we will get if we should put stone on the eight spot or not. As we still haven't check the other spots the program will set the eightspot function's importance variable as 0, which state that it will immediately put on the stone on the eight spot if and only if the side next to that eight spot are opponent's stone.

If it surpass the mentioned conditions, the program will immediately put on the edge. (return the eightspot location) Else, they'll look at other spot in the board.

3. The side

4. The rest

5. The eight spot (2<sup>nd</sup> time)

Using the eightspot function, we will get the location of the spot that the stone can be put. As it's the second time checking, the importance variable will be set as 1 in which they will return the location immediately without checking the condition of the stone next to them on the side group.

6. The special spot

The special spot is the spot that is the key for the edge stone to be changed. To prevent the main stone from being gained by the opponent, the special spot will not be played until the very end of the game (after Turn 48). After checking that the stone can be put in that direction, the program will check the condition on the diagonal of the board if there'll be any opponent stone in that position after capturing. (See image1 for reference) If there's at least one of the opponent stone, it means that the edge stone will be able to be placed in the next opponent turn so the program will not put on that special spot. Else, it will return that special spot.

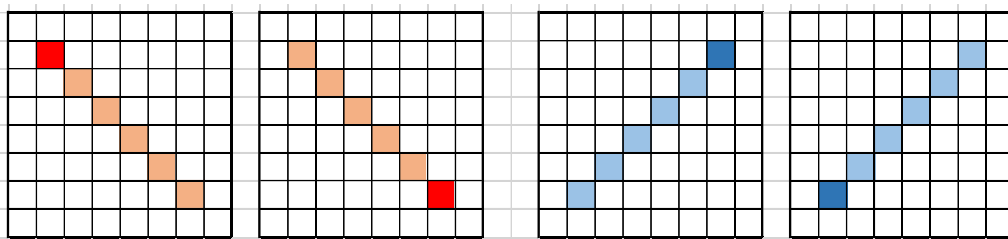


Image1: the corrdinate checked when the stone is put in special spot

If the board didn't exceed any of the 6 condition above, the program choose the first spot they found that can put the stone in and put it.

## 動作テスト内容と結果

Along with other function, this computer function is tested by showing the process of the decision using printf function in the code. That's how we can see how the program think and decide that move clearly.

Example:

```
-----
Turn 35[o]
Score: P1 | P2
      26 | 12
      0 1 2 3 4 5 6 7
      0 . . o o o o . .
      1 o . o o o o . .
      2 o o o o x x o x
      3 o o o x o o x x
      4 o . x x o x . x
      5 . . . o x . . .
      6 . . o x . . . .
      7 . o o o . . . .
#1 No possible move
#2 No possible move
#3 No possible move
#4 Current turn is 1 4
#4 Current turn is 6 4
#4 Current turn is 1 5
PUT AT 1 5
```

In turn 35 it can be seen that there's no possible moves in

1. Edge
2. The eight spot
3. Side

In 4. At first the computer found (1,4) as the best case, then it changes to (6,4) and (1,5) which had more stone gained.

```
-----
Turn 58[x]
Score: P1 | P2
      30 | 30
      0 1 2 3 4 5 6 7
      0 x x x x x x x .
      1 x x o o o o o x
      2 x o x o x o x x
      3 x o o x o o x x
      4 x o x o x o x x
      5 x o o o o x x x
      6 x o o o o o . x
      7 . o o o o o o .
#1 No possible move
#2 No possible move
#3 No possible move
#4 No possible move
#5 No possible move
#6 Current turn is 6 6
PUT AT 6 6
```

In turn 58, it could be seen that there's no possible moves in

1. Edge
2. The eight spot(importance=0)
3. Side
4. The rest
5. The eight spot(importance=1)

But there's one possible move in the spacial spot which the computer ended up putting the stone on (6,6)

## 関数 eightspot(int player\_id, int board[][BOARD\_SIZE],int importance)の説明

### ソースコード

```
int eightspot(int player_id, int board[][BOARD_SIZE],int importance){
    int yy[] = {0,0,1,1,BOARD_SIZE-2,BOARD_SIZE-2,BOARD_SIZE-1,BOARD_SIZE-1};
    int xx[] = {1,BOARD_SIZE-2,0,BOARD_SIZE-1,0,BOARD_SIZE-1,1,BOARD_SIZE-2};
    for(int i=0;i<8;i++){
        if(board[xx[i]][yy[i]]==0 && num_obtained_stone(player_id, xx[i], yy[i],board)>0){
            int aaa = sideisclear(player_id,xx[i],yy[i],board);
            if(aaa==1){
                if(importance==1) return  xx[i]*BOARD_SIZE+yy[i];
            }
            else{
                if(xx[i]==0 || xx[i]==BOARD_SIZE-1){
                    if(yy[i]==BOARD_SIZE-2 && board[xx[i]][yy[i]-1]!=-player_id) return -1;
                    if(yy[i]==1 && board[xx[i]][yy[i]+1]!=-player_id) return -1;
                }
                else if(yy[i]==0 || yy[i]==BOARD_SIZE-1){
                    if(xx[i]==BOARD_SIZE-2 && board[xx[i]-1][yy[i]]!=-player_id) return -1;
                    if(xx[i]==1 && board[xx[i]+1][yy[i]+1]!=-player_id) return -1;
                }
                return xx[i]*BOARD_SIZE+yy[i];
            }
        }
    }
    return -1;
}
```

## プログラムの説明

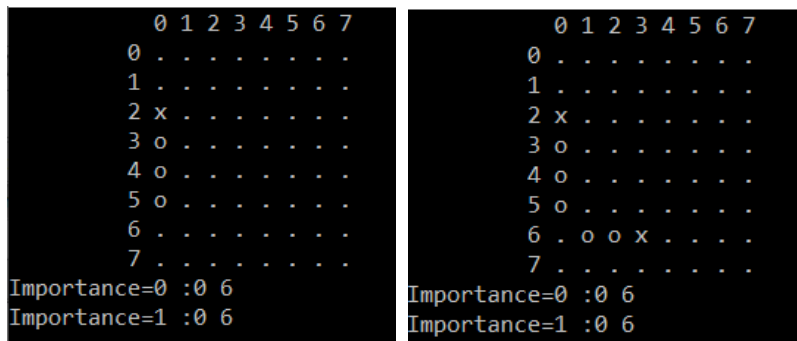
Sub-function: sideisclear

This function works on the 8 spot next to the edge. As they check each spot one-by-one, first they will check if the spot is blank and if they can be placed according to othello rule. Then the program will use “sideisclear” function to see if it’s possible for the opponent to gain the edge stone from us putting the current stone on the board. If the function return 1, it means that the opponent will not gain the edge and we are safe to place the stone there. As the eight spot checked twice, we set the difference checking time with “importance” variable. If the importance variable is 1 (2<sup>nd</sup> checking time) the program will immediately return the spot to computer function. In contrast, if the importance is 0, which means it still haven’t check the side, the rest and other board position, they will immediately place the stone on the board if the side is in risk of being taken by the opponent. Else, the program will return -1, indicating that the program hadn’t find any good spot and let the program find the appropriate spot from the side, the rest first.

## 動作テスト内容と結果

```
#include<stdio.h>
#define BOARD_SIZE 8
void print_board(int board[][BOARD_SIZE]);
int eightspot(int player_id, int board[][BOARD_SIZE],int importance);
int num_obtained_stone(int player_id, int x, int y, int board[][BOARD_SIZE]);
int sideisclear(int player_id, int x, int y, int board[][BOARD_SIZE]);
main(){
    int board[BOARD_SIZE][BOARD_SIZE]={0};
    for(int i=2;i<6;i++) board[i][0]=1;
    board[2][0]=-1;
    print_board(board);
    int im1 = eightspot(-1,board,1);
    int im0 = eightspot(-1,board,0);
    if(im0!=-1) printf("Importance=0 :%d %d\n",im0/BOARD_SIZE,im0%BOARD_SIZE);
    if(im1!=-1) printf("Importance=1 :%d %d\n",im1/BOARD_SIZE,im1%BOARD_SIZE);
}
```

We use this program above to check eightspot function. As we changed the blue part, board's shape change and we get to test the function in different board shape. We will see how the program will react as the 2<sup>nd</sup> player(x) in each situation.



The image shows two side-by-side terminal screenshots. Each screenshot displays an 8x8 Go board with columns indexed 0-7 and rows indexed 0-7. In both, column 0 contains stones from row 2 to 5 (player 1, 'x') and row 2 contains a stone from player 0 ('o'). The left screenshot shows the board with no other stones, and the right screenshot shows an additional 'x' at (6,6) and an 'o' at (6,5). Below the board, both show the output of the eightspot function: Importance=0 : 0 6 and Importance=1 : 0 6.

|   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | . | . | . | . | . | . | . | . |
| 1 | . | . | . | . | . | . | . | . |
| 2 | x | . | . | . | . | . | . | . |
| 3 | o | . | . | . | . | . | . | . |
| 4 | o | . | . | . | . | . | . | . |
| 5 | o | . | . | . | . | . | . | . |
| 6 | . | . | . | . | . | . | . | . |
| 7 | . | . | . | . | . | . | . | . |

Importance=0 : 0 6  
Importance=1 : 0 6

|   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | . | . | . | . | . | . | . | . |
| 1 | . | . | . | . | . | . | . | . |
| 2 | x | . | . | . | . | . | . | . |
| 3 | o | . | . | . | . | . | . | . |
| 4 | o | . | . | . | . | . | . | . |
| 5 | o | . | . | . | . | . | . | . |
| 6 | . | o | o | x | . | . | . | . |
| 7 | . | . | . | . | . | . | . | . |

Importance=0 : 0 6  
Importance=1 : 0 6

In this 2 case, if we don't put the stone on (0,6) the opponent will gain the edge. That's why it's important to put the stone there. So, the program will put the stone on (0,6) from since the first checking.

|                   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------------------|---|---|---|---|---|---|---|---|
| 0                 | . | . | . | . | . | . | . | . |
| 1                 | . | . | . | . | . | . | . | . |
| 2                 | o | . | . | . | . | . | . | . |
| 3                 | o | . | . | . | . | . | . | . |
| 4                 | o | . | . | . | . | . | . | . |
| 5                 | o | . | . | . | . | . | . | . |
| 6                 | . | o | o | x | . | . | . | . |
| 7                 | . | . | . | . | . | . | . | . |
| Importance=0 : -1 |   |   |   |   |   |   |   |   |
| Importance=1 : -1 |   |   |   |   |   |   |   |   |

|                   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------------------|---|---|---|---|---|---|---|---|
| 0                 | . | . | . | . | . | . | . | . |
| 1                 | . | . | . | . | . | . | . | . |
| 2                 | o | . | . | . | . | . | . | . |
| 3                 | x | . | . | . | . | . | . | . |
| 4                 | o | . | . | . | . | . | . | . |
| 5                 | o | . | . | . | . | . | . | . |
| 6                 | . | o | o | x | . | . | . | . |
| 7                 | . | . | . | . | . | . | . | . |
| Importance=0 : -1 |   |   |   |   |   |   |   |   |
| Importance=1 : -1 |   |   |   |   |   |   |   |   |

In this case, even though it's possible to put the stone on (0,6) but as we are not able to capture all of the opponent stone on the left, side, we will lost the left-down position at the end. That's why we need to not put the stone on that position.

|                    | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|--------------------|---|---|---|---|---|---|---|---|
| 0                  | . | . | . | . | . | . | . | . |
| 1                  | . | . | . | . | . | . | . | . |
| 2                  | . | . | . | . | . | . | . | . |
| 3                  | . | . | x | . | . | . | . | . |
| 4                  | . | o | . | . | . | . | . | . |
| 5                  | . | o | . | . | . | . | . | . |
| 6                  | . | . | . | . | . | . | . | . |
| 7                  | . | . | . | . | . | . | . | . |
| Importance=0 : -1  |   |   |   |   |   |   |   |   |
| Importance=1 : 0 6 |   |   |   |   |   |   |   |   |

|                    | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|--------------------|---|---|---|---|---|---|---|---|
| 0                  | . | . | . | . | . | . | . | . |
| 1                  | . | . | . | . | . | . | . | . |
| 2                  | . | . | . | . | . | . | . | . |
| 3                  | . | . | . | . | . | . | . | . |
| 4                  | . | . | . | . | . | . | . | . |
| 5                  | . | . | . | . | . | . | . | . |
| 6                  | . | o | o | x | . | . | . | . |
| 7                  | . | . | . | . | . | . | . | . |
| Importance=0 : -1  |   |   |   |   |   |   |   |   |
| Importance=1 : 0 6 |   |   |   |   |   |   |   |   |

In this case, it's possible to put the stone on the position (0,6) according to the rule. Still as no opponent stone are on the edge, this case is not an urgent case and we can focus on other cases first. That's why we will decide to place (0,6) again if "the rest" position is not available.



関数 int sideisclear(int player\_id, int x, int y, int board[][BOARD\_SIZE])の説明

## ソースコード

```
int sideisclear(int player_id, int x, int y, int board[][BOARD_SIZE]){
    int ans = 1,opponent=0,same=0,add;
    if(x==0 || x==BOARD_SIZE-1){
        if(y==1) add = 1;
        else if(y==BOARD_SIZE-2) add=-1;
        if(board[x][y-add]==-player_id && board[x][y+add]==-player_id) return 1;
        y+=add;
        while(y<BOARD_SIZE && y>=0){
            if(board[x][y]==player_id){
                same++;
                if(same==1) opponent=0;
            }
            else if(board[x][y]==-player_id){
                opponent++;
            }
            else break;
            y+=add;
        }
        if(opponent>0) return 0;
        return 1;
    }
    if(y==0 || y==BOARD_SIZE-1){
        if(x==1) add = 1;
        else if(x==BOARD_SIZE-2) add=-1;
        x+=add;
        while(x<BOARD_SIZE && x>=0){
            if(board[x][y]==player_id){
                if(same==0) opponent=0;
```

```

        same++;
    }
    else if(board[x][y]==-player_id){
        if(same>0) return 0;
        opponent++;
    }
    else{
        if(opponent==0) return 1;
        else return 0;
    }
    x+=add;
}
if(opponent>0) return 0;
return 1;
}
}

```

## プログラムの説明

This function search across the side of the board if there'll be any other opponent's stone left on the (x,y) side after (x,y) eat all the eatable stone. As we can only eat the opponent stone until we meet our stone, we use "opponent" and "same" variable to count number of stones left after eating. By checking from (x,y) up to the end of the other side(shown on image 2). If the spot is opponent's stone "opponent" variable will increase by 1. If the spot is one's stone, "same" variable will increase by 1 along with "opponent" variable being converted to 0 if "same" is 0 before conversion as all the opponent stone beforehand will be eaten. If there's no stone in the spot or if the checking ended, we see the total amount of the opponent's stone and if it's not 0, it means that we are not able to clear all the opponent's stone and we shouldn't put the stone on that position as in the next turn, the opponent will be able to capture the

edge. So the program will return 0. Else, if the opponent at the end is 0, the (x,y) position is safe to be put so the program will return 1.

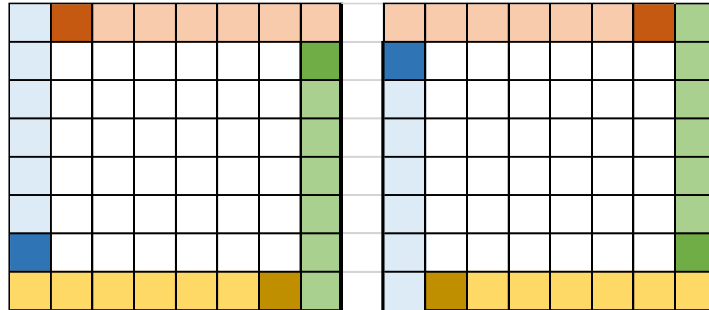


Image2: the image start searching from the thicker color to the end of the light color

### 動作テスト内容と結果

```
#include<stdio.h>
#define BOARD_SIZE 8
void print_board(int board[][BOARD_SIZE]);
void init_board(int board[][BOARD_SIZE]);
int sideisclear(int player_id, int x, int y, int board[][BOARD_SIZE]);
main(){
    int board[BOARD_SIZE][BOARD_SIZE]={0};
    board[0][1]=-1;board[0][2]=1;board[0][3]=-1;board[0][4]=1;
    print_board(board);
    int co[] = {0,6};
    int im1 = sideisclear(-1,co[0],co[1],board);
    printf("Putting stone on (%d %d) : %d\n",co[0],co[1],im1);
}
```

Using the code mentioned above, the program is tested in several cases and several directions by changing the board shape in the blue area. (0,6)'s test will be shown as an example below.

```

      0 1 2 3 4 5 6 7
0 . . . . . . .
1 . . . . . . .
2 . . . . . . .
3 x . . . . . .
4 o . . . . . .
5 o . . . . . .
6 . . . . . . .
7 . . . . . . .
Putting stone on (0 6) : 1

```

Case 1:

If we put the x in (0,6), all of the stone will be eaten. O wont be able to take (0,7) in the next round so putting x in (0,6) is possible.(return 1)

```

      0 1 2 3 4 5 6 7
0 . . . . . . .
1 . . . . . . .
2 o . . . . . .
3 x . . . . . .
4 o . . . . . .
5 o . . . . . .
6 . . . . . . .
7 . . . . . . .
Putting stone on (0 6) : 0

```

Case 2:

If we put x in (0,6), the stone from (0,5) to (0,4) will be converted to x but o in (0,2) wont. That's why we are not supposed to put x there.(return 0)

```

      0 1 2 3 4 5 6 7
0 . . . . . . .
1 x . . . . . .
2 o . . . . . .
3 x . . . . . .
4 o . . . . . .
5 o . . . . . .
6 . . . . . . .
7 . . . . . . .
Putting stone on (0 6) : 0

```

Case 3:

According to the rule, x at (0,1) cannot capture o in (0,2) so o in (0,2) will be left. If we put x in (0,6), (0,7) will be gained by the opponent next turn. (return 0)

```

      0 1 2 3 4 5 6 7
0 . . . . . . .
1 . . . . . . .
2 . . . . . . .
3 . . . x . . .
4 . . o . . . .
5 . o . . . . .
6 . . . . . . .
7 . . . . . . .
Putting stone on (0 6) : 1

```

Case 4:

As there's no stone on the side, the program is free to put the stone on (0,6). (return 1)

```

      0 1 2 3 4 5 6 7
0 . . . . . . .
1 x . . . . . .
2 o . . . . . .
3 x . . . . . .
4 o . . . . . .
5 . . . . . . .
6 . . . . . . .
7 . . . . . . .
Putting stone on (0 6) : 1

```

Case 5:

Even though there's a stone on the side, (0,5) is blank so the opponent wont be able to obtain (0,7) in the next round. (return 1)

## 関数 update\_history(int location, int history[]) の説明

### ソースコード

```
void update_history(int location, int history[]){  
    int i;  
    for(i=0;history[i]!=-2 && i<63;i++);  
    history[i] = location;  
    history[i+1]=-2;  
}
```

### プログラムの説明

This function add new data to the history array. Data array will have a lot of previous move recorded with -2 used to represent end of data. In case nothing has been recorded in function yet, -2 will be located at history[0]. First, we need to find the location that's -2 or also known as the first not-yet-added position. After we found it, we put the move coordinate in and then assign the next location to be -2 so the next time we come to the function we know where to put the net function.

## 動作テスト内容と結果

The program below is used to check the function by receiving the data from the input and then output the current history in the array. It could be seen that the program works perfectly fine but if the input is -2 the program will not record the number. However, all the location numbers in our program are whole number so we won't have trouble with that.

```
#include<stdio.h>
void update_history(int location, int history[]);
main(){
    int history[1000],a;
    history[0]=-2;
    while(1){
        printf("Add data:");
        scanf("%d",&a);
        update_history(a,history);
        for(int i=0;history[i]!=-2;i++) printf("%d ",history[i]);
        printf("\n");
    }
}
```

```
Add data:1
1
Add data:2
1 2
Add data:3
1 2 3
Add data:4
1 2 3 4
Add data:5
1 2 3 4 5
Add data:6
1 2 3 4 5 6
Add data:7
1 2 3 4 5 6 7
Add data:8
1 2 3 4 5 6 7 8
Add data:9
1 2 3 4 5 6 7 8 9
Add data:0
1 2 3 4 5 6 7 8 9 0
Add data:-1
1 2 3 4 5 6 7 8 9 0 -1
Add data:-2
1 2 3 4 5 6 7 8 9 0 -1
```

## 関数 save\_history(int history[]) の説明

### ソースコード

```
void save_history(int history[]){
    int a=0;
    int number;
    FILE *f;
    f = fopen("save_history.txt", "w");

    while(history[a]!=-2){

        fprintf(f,"Player%d %d %d¥n",(a%2)+1,history[a]/ BOARD_SIZE,history[a]%
BOARD_SIZE);
        a++;
    }
    fclose(f);
}
```

### プログラムの説明

This function change data of each moves in game to text file. First, the program will open new file called "history.txt" which we will write in. Then we have to convert the data in history [] to the form that we want in the text file. Data in history is in the form of  $x \times \text{BOARD\_SIZE} + y$ , that's why we need to convert it back to x and y eventually by divided by BOARD\_SIZE in terms of x and modulo by BOARD\_SIZE in terms of y. Then, we print each move in the file in the form of [Player (no. of player) (x) (y)¥n]. The number of player can be retrieved from the current location of the history file. If the location is even, it means it's player1. Else, it's player2. If history [location] is -1, it means that player pass that move, so print out [Player%d PASS¥n]


instead. We stated the starting number as -2, so if history [location] is -2, it means that it's the end of the string. The function will close the file and ended the function.

### 動作テスト内容と結果

```
#include<stdio.h>

save_history(int history[]);

main()
{
    int history[HISTORY_SIZE]={0};
    for(int i=0;i<HISTORY_SIZE;i++)
    {
        history[i]=i;
    }
    save_history(history);
}
```

 check\_history - Notepad

File Edit Format View Help

```
Player1 0 0
Player2 0 1
Player1 0 2
Player2 0 3
Player1 0 4
Player2 0 5
Player1 0 6
Player2 0 7
Player1 1 0
Player2 1 1
Player1 1 2
Player2 1 3
Player1 1 4
Player2 1 5
Player1 1 6
Player2 1 7
Player1 2 0
```

Using the program on the left, the function save the history into text file. History array in the program is set to contain number similar to array index "history[i] = i" so that we can see how program perform when number in the history array is converted to coordinate (x, y). The result on the right show Player 1 and 2 alternately, following with coordinate (x, y) converted from number in history [].



## 自由課題

### 作成したプログラムの概要

自分をもっとCがうめくプログラミングできるように、Codeforceで競争プロをやりはじめます。この問題は始めて1900レベルを解けるので、自慢で提出したいです。

これは英語のバージョンで、下に日本語に翻訳します。

<http://codeforces.com/contest/1360/problem/G>

問題：

$n, m, a, b$  の4つの整数から、下の3つの条件を満たす  $n \times m$  サイズの行列を作りなさい：

1. 各行は  $a$  つの “1” を持つ
2. 各列は  $b$  つの “1” を持つ
3. “1” 以外の数字は全部 “0” にする

例：

$n=3, m=6, a=2, b=1$  の場合

010100

001010

001100

行列を出力前に“YES”を出力する。しかし、3つの条件を満たす行列がない場合、“NO”を出力する。

入力：1行目は  $t$  ( $1 \leq t \leq 1000$ )、 $t$  はテストケースの数

次の  $t$  行で、 $n, m, a, b$  の順番の整数 ( $1 \leq b \leq n \leq 50; 1 \leq a \leq m \leq 50$ )

出力：テストケースごとに まず“YES”か“NO”を出力して、“YES”の場合なら  $n$  行の  $m$  つの “0”か“1”の数字を出力（スペースなし）

## ソースコード

```
#include<stdio.h>

main(){
    int a,h,w,ho,wo;
    scanf("%d",&a);
    for(int i=0;i<a;i++){
        int yn = 1;
        scanf("%d %d %d %d",&w,&h,&wo,&ho);
        int cc[55][55] = {0};
        int hchk[55] = {0},wchk[55] = {0};
        int current =0,newcurrent = 0;
        for(int j=0;j<w;j++){
            current = newcurrent+1;
            for(int k=0;k<h;k++){
                if(hchk[(k+current)%h]<ho && wchk[j]<wo){
                    cc[j][(k+current)%h] = 1;
                    hchk[(k+current)%h]++;
                    wchk[j]++;
                    newcurrent = (k+current)%h;
                    //printf("%d %d¥n",j,(k+current)%h);
                }
            }
        }
        for(int j=0;j<w;j++){
            for(int k=0;k<h;k++){
                if(hchk[k]!=ho || wchk[j]!=wo){
                    yn = 0;
                    break;
                }
            }
        }
    }
}
```

```

    }
    if(yn==0){
        break;
    }
}
if(yn==0){
    printf("NO¥n");
}
else{
    printf("YES¥n");
    for(int j=0;j<w;j++){
        for(int k=0;k<h;k++){
            printf("%d",cc[j][k]);
        }
        printf("¥n");
    }
}
}
}

```

## プログラムの説明

まず、プログラムは a 変数でテストケースの数字をもらう。次に w , h , wo , ho の数字の行数、列数、行での 1 の数、列での 1 の数もらう。cc のゼロ行列を作る。そして wchk , hchk の順番に行/列ずつに 1 の数を貯める。行ずつに 1 を入れる。一つの行を一回に入れすぎないように、1 が左の列にいと、つぎの右の行に移動する。例えば、現在 ( j , k ) 成分いと、行の 1 数が満たすまでに i=1,2,3,4.. ( j , k+i ) に 1 をいれて wchk[j] , hchk[k+i] に 1 を数える。もし、( j , m ) で列がもう満たすならつぎの ( j , m+1 ) に移動したが、もし ( j , m ) で行も数が満たすならつぎの ( j + 1 , m + 1 ) にチェックする。

例として、「3 6 2 1」の入力なら以下の形なる。

110000

001100

000011

プログラムが入れ終わると、もう一度列と行の1をチェックする、一つでも間違ったら、「NO」を出力する。しかし全部正しかったら、「YES」を出力して、行列を書き出す。

### 動作テスト内容と結果

```
5
3 6 2 1
YES
011000
000110
100001
2 2 2 1
NO
2 2 2 2
YES
11
11
4 4 2 2
YES
0110
1001
0110
1001
2 1 1 2
YES
1
4
```

これは課題のケースの例である。この例で、このプログラムが

1. 多いテストケースで動ける
2. No が使えること
3. 1 全員でも動ける。
4. ベクトル（行/列 = 1）でも動ける

[illegible]

このケースは一番大きい行列  
(50 \* 50) が動けるを証  
明する。

```
1
1 1 1 1
YES
1
```

```
1
1 1 0 0
YES
0
```

1 \* 1 サイズのケースを証明する。

## 感想

今年の授業、ありがとうございました！

私は少しCを書いたことありますが、授業で専門な言葉がたくさんあって、留学生としてはじめに覚えきれないが、よく読んで覚えるようになりました！授業で何回も課題を解けるとき、満足な気持ちを感じました。

スライドで勉強するので、マイペースができてよかったです。

最近、もっとプログラミングがうまくできるように Codeforce で競争プログラミングをやっています。夏休みでもたくさんやって、あたらしいレベルになりたいです！頑張ります。

この課題をやりながら、たくさん成長できました。私はさきに `place_stone` を書いたので、今このコードを見るとどうしてそんなに長く同じものをコピペコードを書いて、どうして `for` を使わないのと後悔して、自分が成長することを感じて、もう一度書き直したいです。もっとも、レポートを書き終わる途中で、友達やオセロをできる先輩や家族にコードを見せて、たくさんコメントをもらいました。今、たくさん新しいアイデアが思い浮かべて、夏休みではじめから書き直すと決めました。今を考えるアイデアで、もっと強くコンピューターを作れると信じます。実は私があたらしいコードを書き直して、締め切り内で先生に書き直したいが、この課題のほか、あと3つの他の授業の課題があります。もっと強くプログラムを作って、同じ専門の人のコードと競争したいなのに、今書き直すができなくて、悔しかったです

先生がいつまで学生同士のプログラムを競争ですか。締め切り外でも私があたらしいコードを提出してもいいですか。単位はこのコードとレポートで評価してもいいし、単純にあたらしいコードで競争したいです。後でメールで質問します。

よろしくお願いします。