Practical

Focus/Domination Game

What You Need to Do in this LAB

1. Download the CLion project provided in *Week 8 >> Practical March 27 >> Focus.zip*
2. Import the project in CLion and familiarize with the code
   - **TO UNDERSTAND THE CODE YOU WILL NEED TO READ THE REST of the slides AND the COMMENTS PROVIDED IN THE SOURCE FILES**
3. Implement the data structure necessary to define the players in *game_init.h*
   - Players should be characterized by: name, color, number of own pieces accumulated, number of adversary's pieces captured.
4. Implement method *initialize_players*
5. *Implement a new method to visualize the size of the stacks in each square of the board.*
6. Between 12 and 1pm you should fill the QUIZ in Week 8 >> Practical March 27. This will be used to take note of your attendance.

Contents of the Focus Project (1/2)

The project includes the following files:

- **main.c**: declares the variables representing the players and the board. It invokes the functionalities to initialize the players and the board.

- **game_init.h**: it provides constants to represent the color of the pieces (GREEN/RED) and the type of squares (VALID/INVALID). It also provides the data structures to represent players, pieces and the squares of the board. Finally, it includes the prototypes of the methods necessary to create the players and the board.

- **game_init.c:** it implements the methods necessary to create the players and the board.

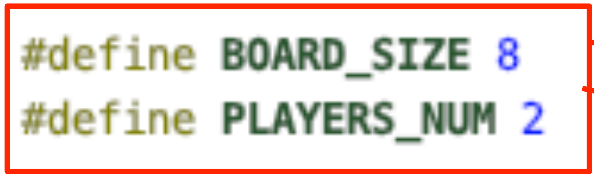
Contents of the Focus Project (2/2)

- **input_output.h**: It is the library that contains the methods to print on the standard output and require inputs from the users. At the moment, it only includes a method to print the board. You can add more method prototypes to ask the users' inputs during the game.

- **input_output.c**: It implements the methods declared in *input_output.h*. At the moment, it only implements the method necessary to print the board.

game_init.h

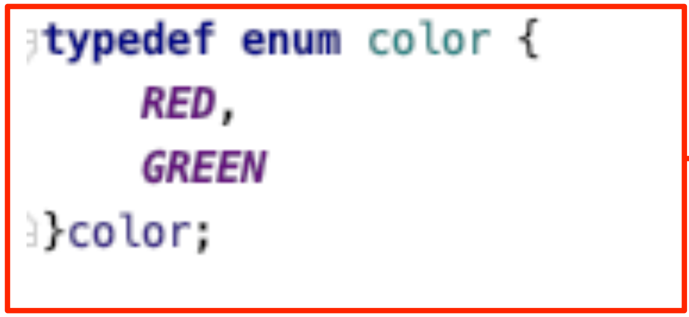
Defines basic constants and pre-defined values



```
#define BOARD_SIZE 8
#define PLAYERS_NUM 2
```

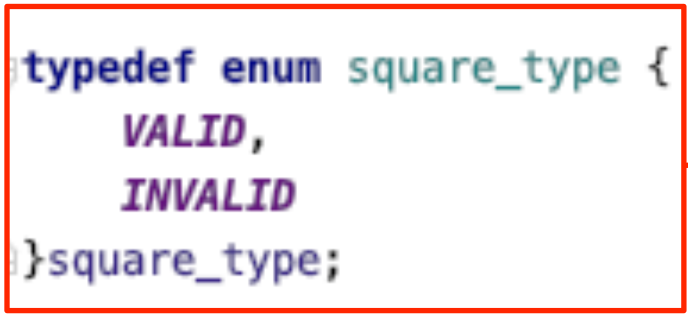
The size of the board (8x8)

The number of players (2)



```
typedef enum color {
    RED,
    GREEN
}color;
```

The color of the pieces (e.g., RED or GREEN)



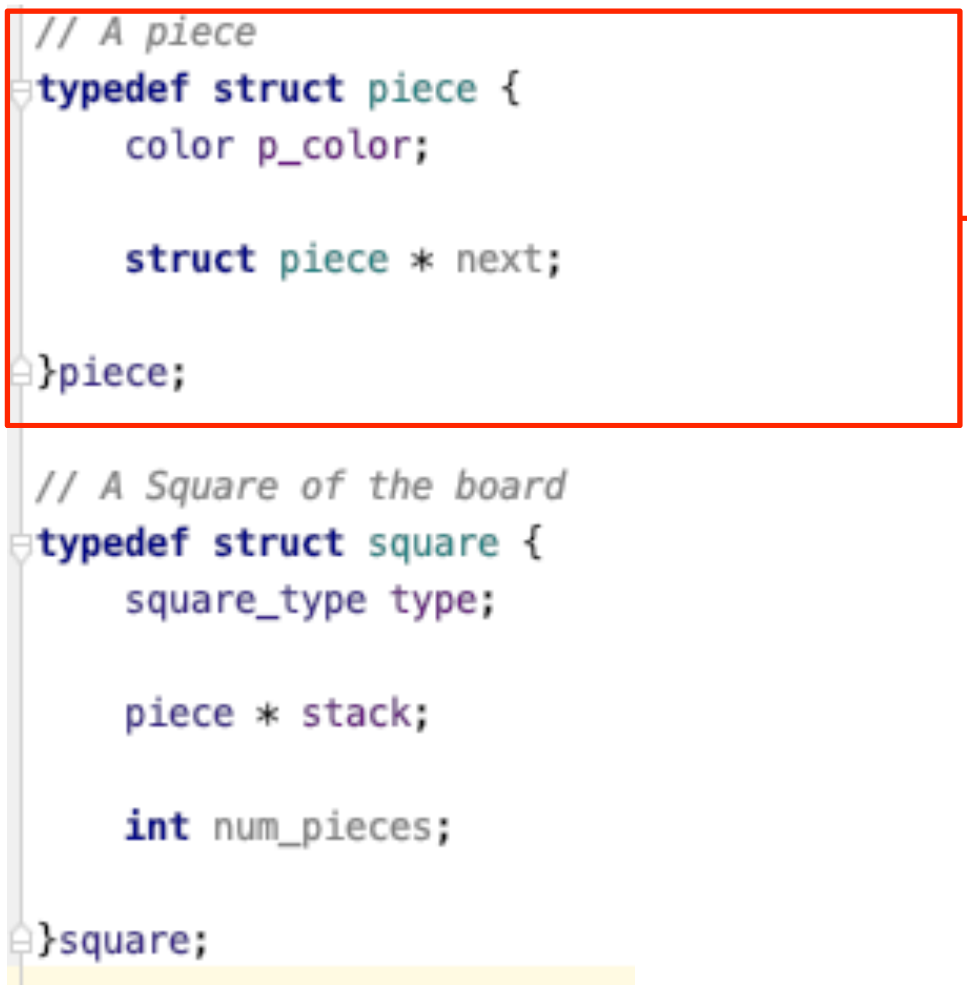
```
typedef enum square_type {
    VALID,
    INVALID
}square_type;
```

A square of the board can be

- *INVALID:* squares that are on the *sides and where no piece can be placed*
- *VALID*: squares where it is possible to place a piece or a stack

game_init.h

It also defines the data structures necessary to represent players, the squares of the board and the pieces composing stacks.



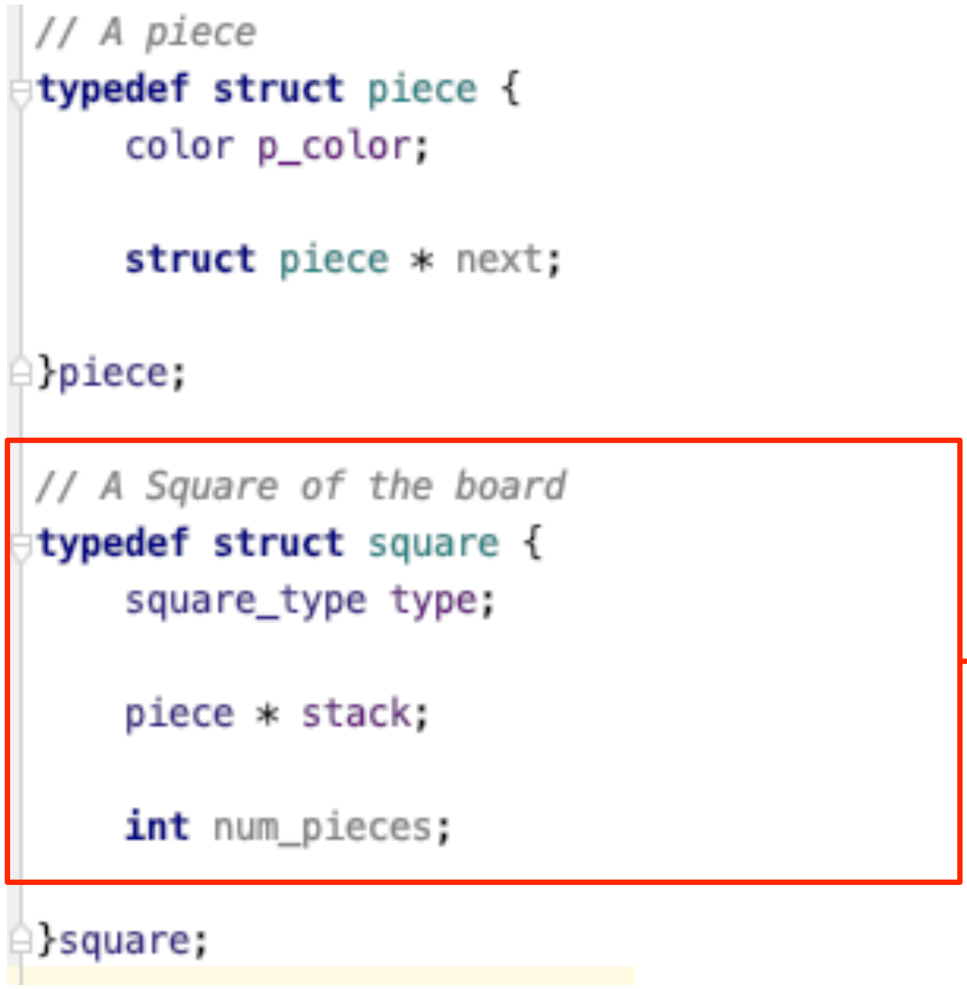
```
// A piece
typedef struct piece {
    color p_color;

    struct piece * next;

}piece;

// A Square of the board
typedef struct square {
    square_type type;

    piece * stack;

    int num_pieces;

}square;
```

A piece is characterized by a color and a pointer to the next piece on the stack

For this lab you do not have to worry too much about the next pointer.

game_init.h

It also defines the data structures necessary to represent players, the squares of the board and the pieces composing stacks.



```
// A piece
typedef struct piece {
    color p_color;

    struct piece * next;

}piece;

// A Square of the board
typedef struct square {
    square_type type;

    piece * stack;

    int num_pieces;

}square;
```

A square is characterized by:

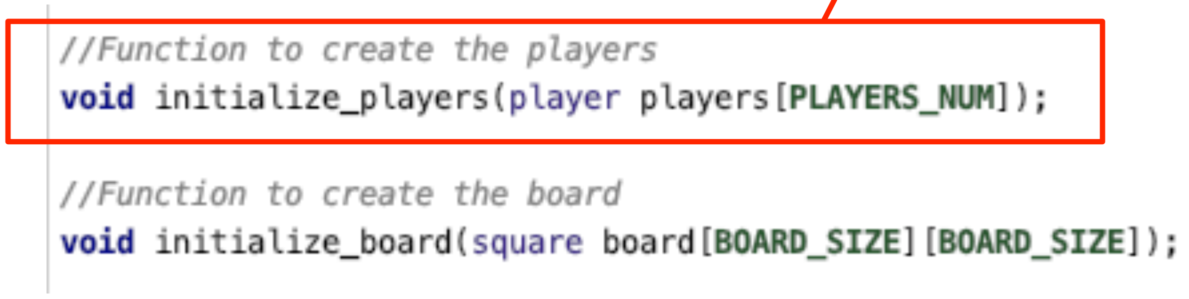
- a type (VALID/INVALID);
- a pointer to the piece on top of the stack;
- the number of pieces that are stacked on the square

game_init.h

It also declares the prototype of the methods necessary to initialize the players and the board.

This is the prototype of the function that creates the players for the first time.
It takes as input an array of two players

You will have to implement this function inside game_init.c



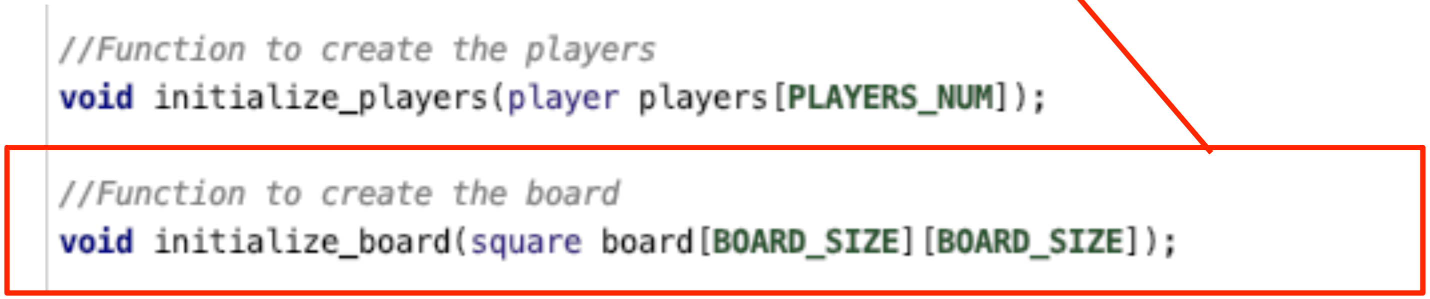
```c
//Function to create the players
void initialize_players(player players[PLAYERS_NUM]);

//Function to create the board
void initialize_board(square board[BOARD_SIZE][BOARD_SIZE]);
```

game_init.h

It also declares the prototype of the methods necessary to initialize the players and the board.

This is the prototype of the function that creates the board for the first time. This function is implemented inside game_init.c

It takes as input a 2-Dimensional (8x8) array of squares representing the board to be initialized.

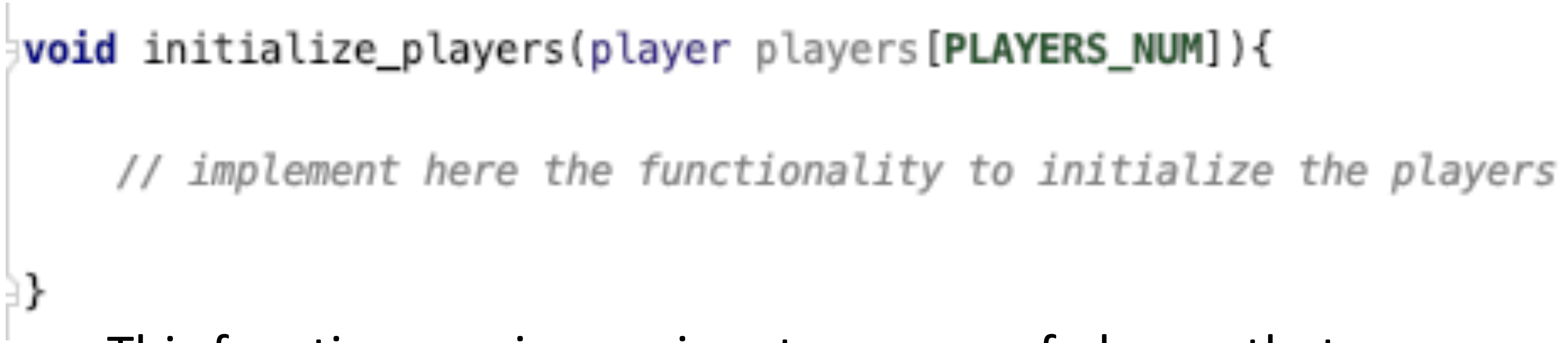


```c
//Function to create the players
void initialize_players(player players[PLAYERS_NUM]);

//Function to create the board
void initialize_board(square board[BOARD_SIZE][BOARD_SIZE]);
```

game_init.c

- Contains the implementation of the function necessary to initialize the board and the players.

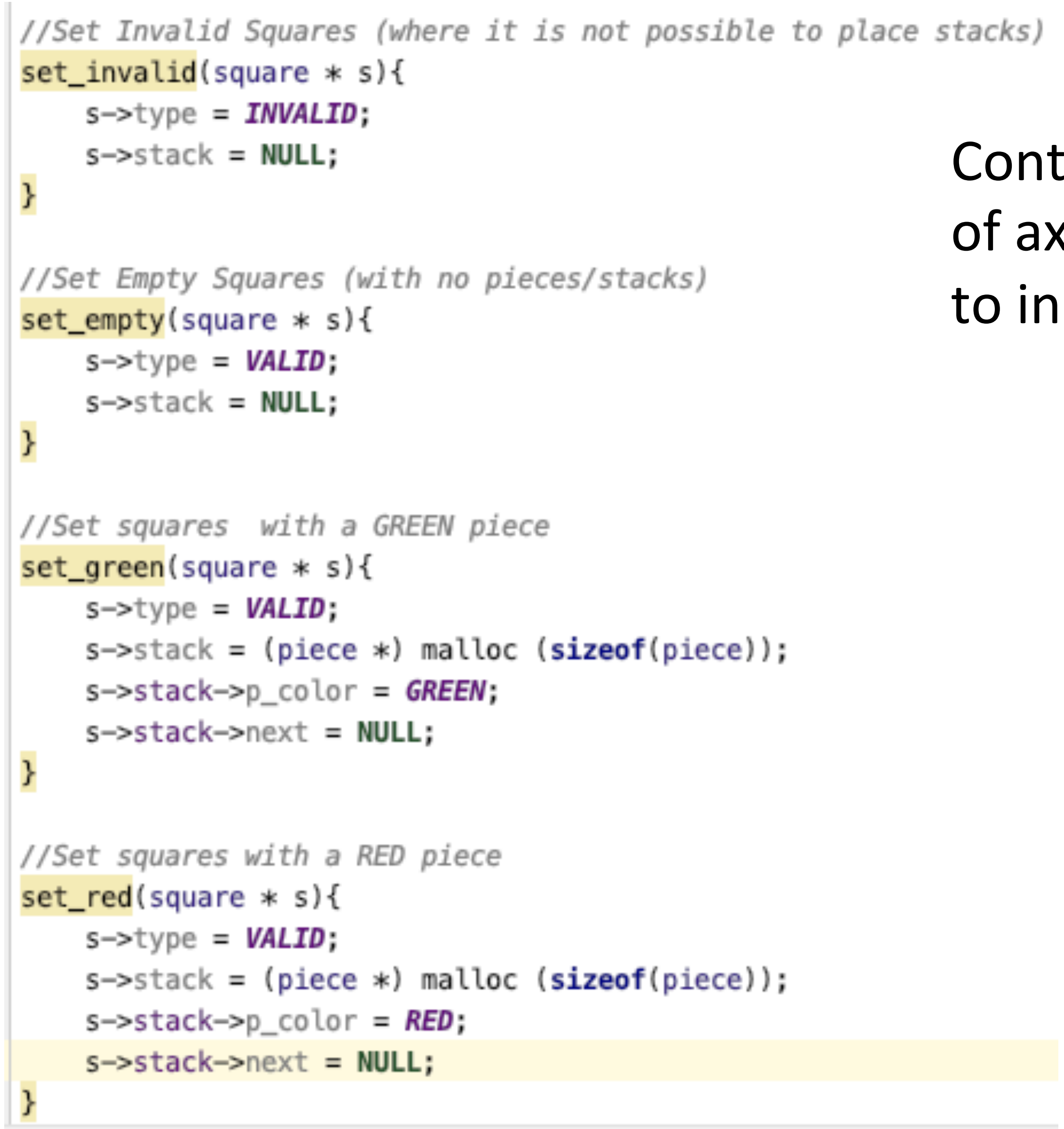
In this LAB, you will have to implement the following function necessary to initialize the players.



```c
void initialize_players(player players[PLAYERS_NUM]){

    // implement here the functionality to initialize the players

}
```

- This function receives as input an array of players that you should fill with the information about the players that the users provide as input.

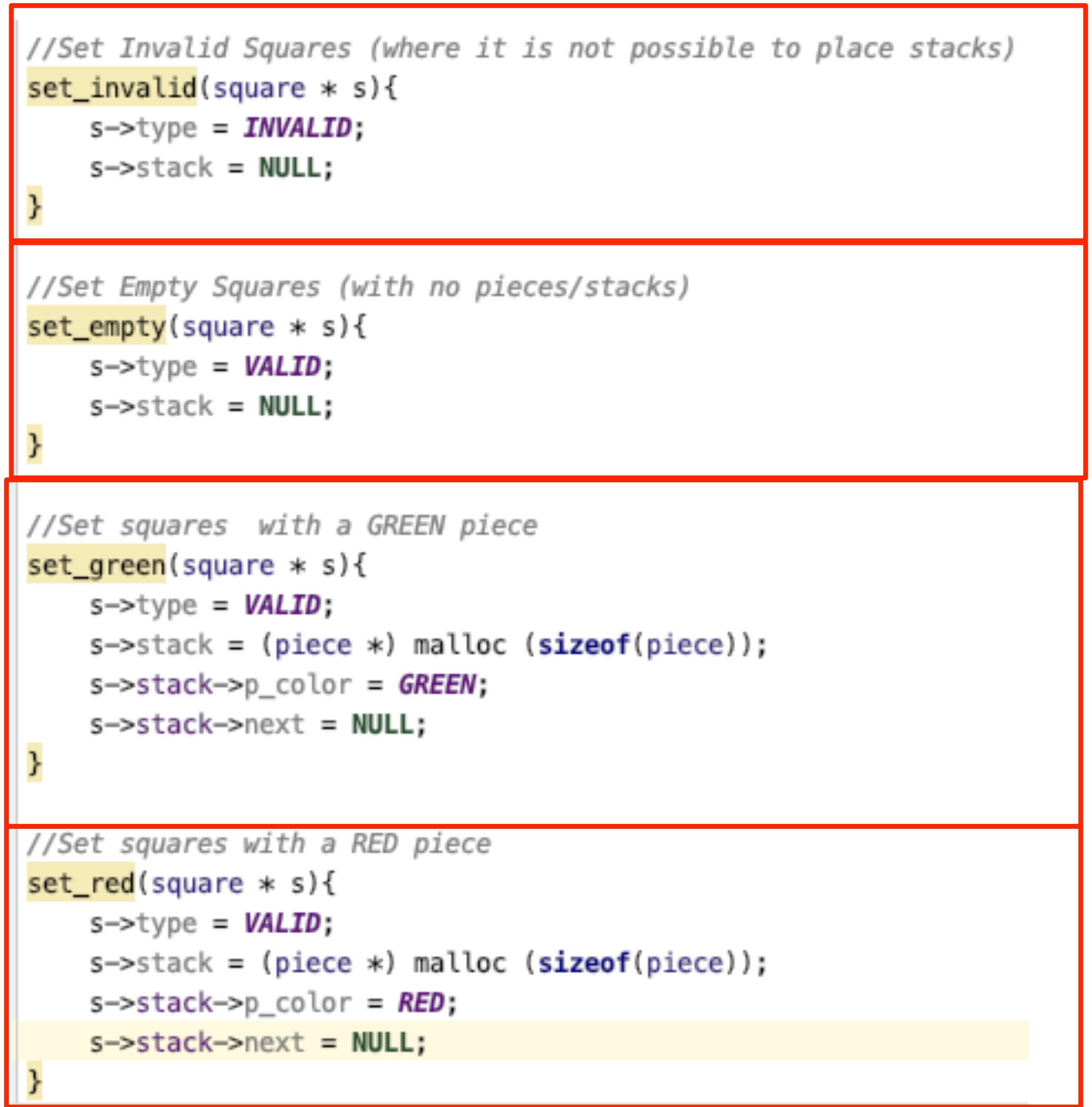
game_init.c



```
//Set Invalid Squares (where it is not possible to place stacks)
set_invalid(square * s){
    s->type = INVALID;
    s->stack = NULL;
}

//Set Empty Squares (with no pieces/stacks)
set_empty(square * s){
    s->type = VALID;
    s->stack = NULL;
}

//Set squares  with a GREEN piece
set_green(square * s){
    s->type = VALID;
    s->stack = (piece *) malloc (sizeof(piece));
    s->stack->p_color = GREEN;
    s->stack->next = NULL;
}

//Set squares with a RED piece
set_red(square * s){
    s->type = VALID;
    s->stack = (piece *) malloc (sizeof(piece));
    s->stack->p_color = RED;
    s->stack->next = NULL;
}
```

Contains the implementation of axuliary functions necessary to initialize the board.

game_init.c



```c
//Set Invalid Squares (where it is not possible to place stacks)
set_invalid(square * s){
    s->type = INVALID;
    s->stack = NULL;
}
```

Sets invalid squares where no pieces can be placed

```c
//Set Empty Squares (with no pieces/stacks)
set_empty(square * s){
    s->type = VALID;
    s->stack = NULL;
}
```

Sets valid squares with no pieces

```c
//Set squares  with a GREEN piece
set_green(square * s){
    s->type = VALID;
    s->stack = (piece *) malloc (sizeof(piece));
    s->stack->p_color = GREEN;
    s->stack->next = NULL;
}
```

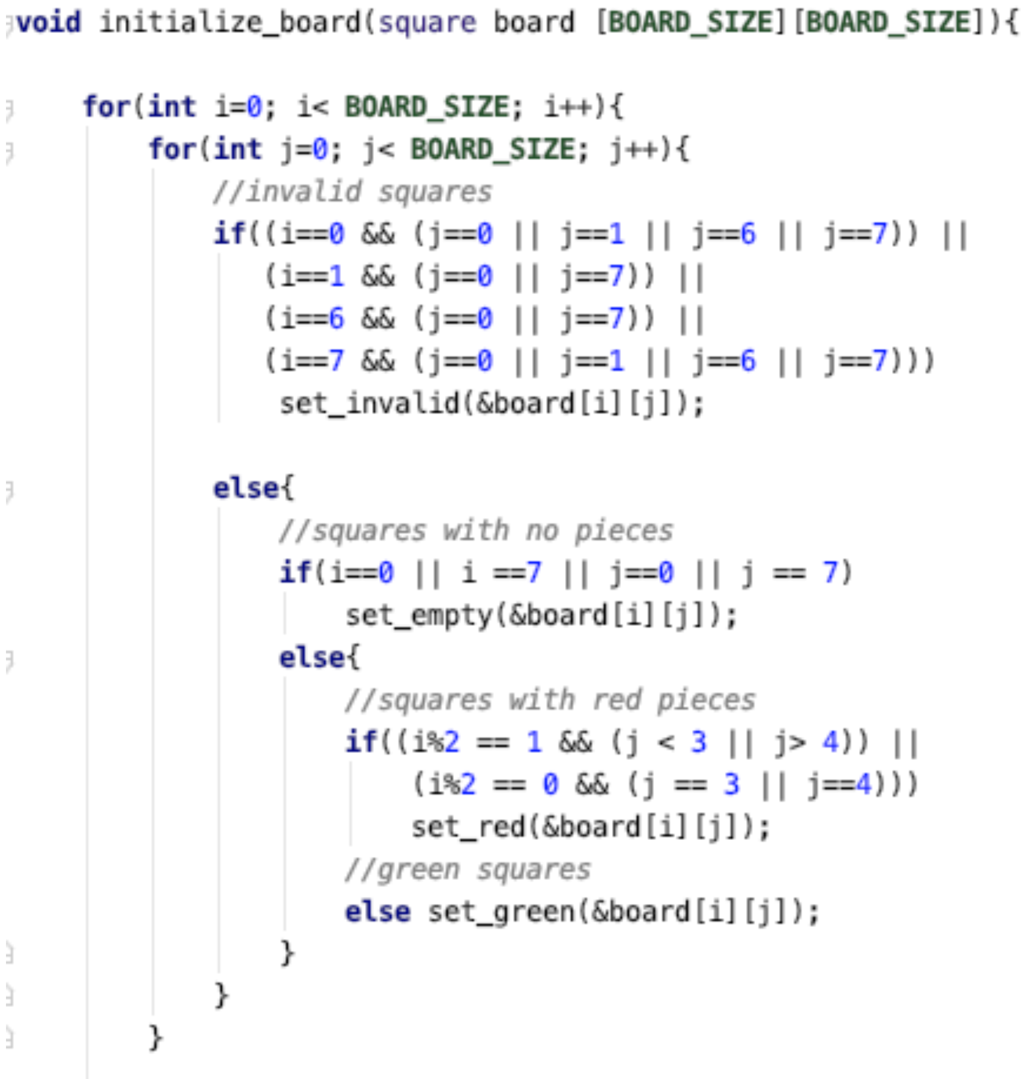
Sets valid squares with GREEN pieces

```c
//Set squares with a RED piece
set_red(square * s){
    s->type = VALID;
    s->stack = (piece *) malloc (sizeof(piece));
    s->stack->p_color = RED;
    s->stack->next = NULL;
}
```

Sets valid squares with RED pieces

game_init.c

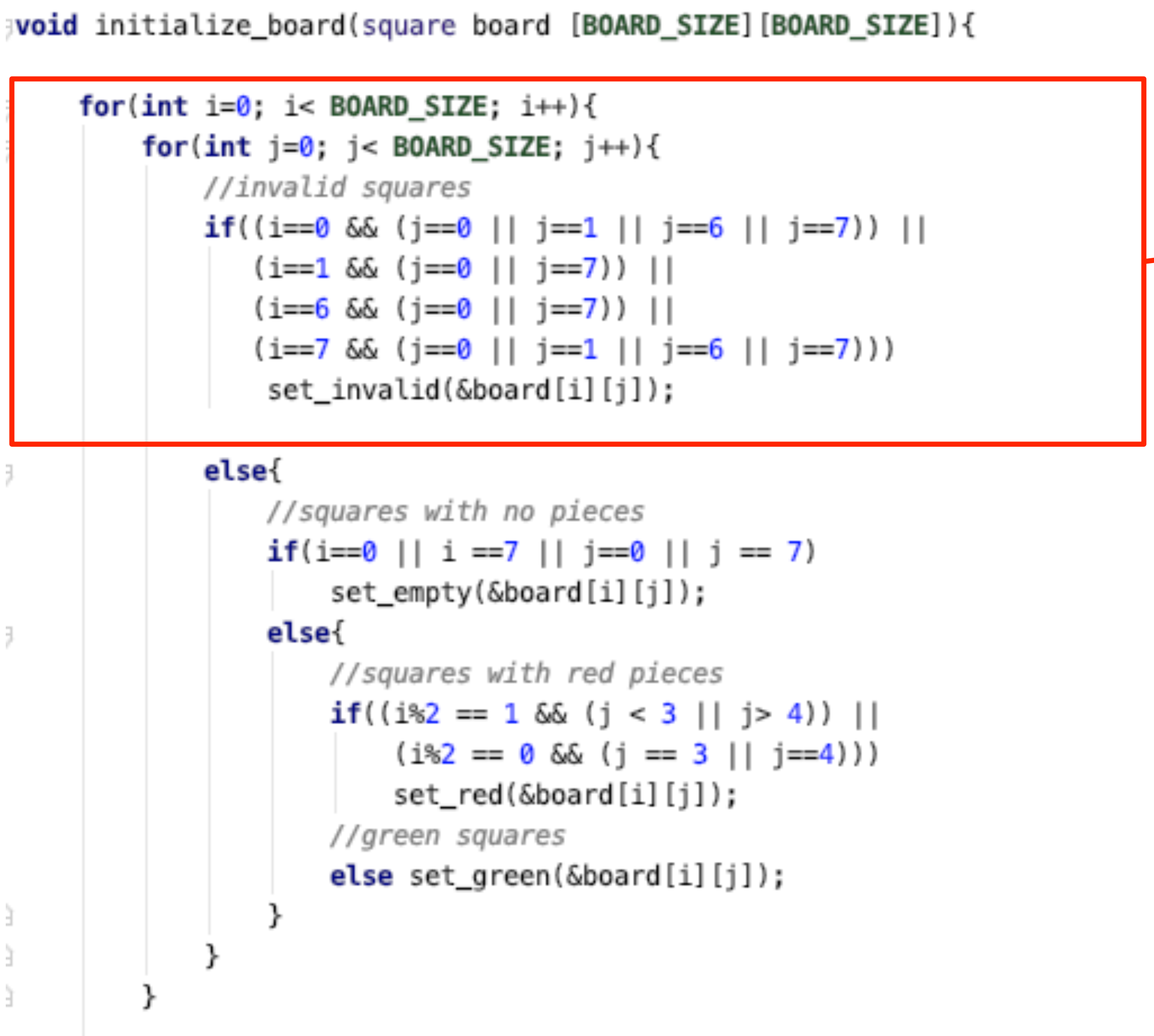
- Contains the implementation of the method necessary to initialize the board.



```c
void initialize_board(square board [BOARD_SIZE][BOARD_SIZE]){

    for(int i=0; i< BOARD_SIZE; i++){
        for(int j=0; j< BOARD_SIZE; j++){
            //invalid squares
            if((i==0 && (j==0 || j==1 || j==6 || j==7)) ||
               (i==1 && (j==0 || j==7)) ||
               (i==6 && (j==0 || j==7)) ||
               (i==7 && (j==0 || j==1 || j==6 || j==7)))
                set_invalid(&board[i][j]);

            else{
                //squares with no pieces
                if(i==0 || i ==7 || j==0 || j == 7)
                    set_empty(&board[i][j]);
                else{
                    //squares with red pieces
                    if((i%2 == 1 && (j < 3 || j> 4)) ||
                        (i%2 == 0 && (j == 3 || j==4)))
                        set_red(&board[i][j]);
                    //green squares
                    else set_green(&board[i][j]);
                }
            }
        }
```

game_init.c

- Contains the implementation of the method necessary to initialize the board.

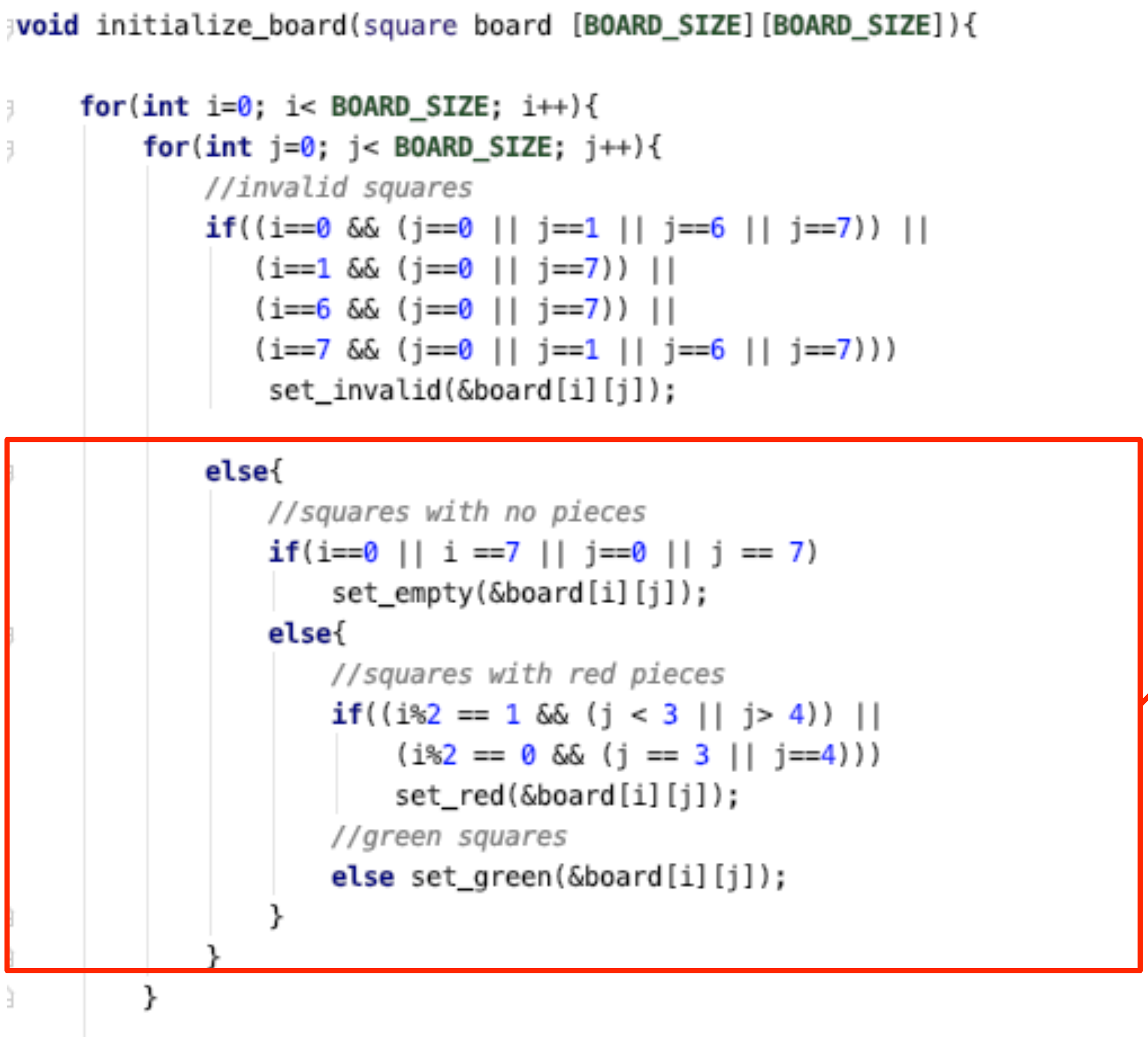


```c
void initialize_board(square board [BOARD_SIZE][BOARD_SIZE]){

    for(int i=0; i< BOARD_SIZE; i++){
        for(int j=0; j< BOARD_SIZE; j++){
            //invalid squares
            if((i==0 && (j==0 || j==1 || j==6 || j==7)) ||
               (i==1 && (j==0 || j==7)) ||
               (i==6 && (j==0 || j==7)) ||
               (i==7 && (j==0 || j==1 || j==6 || j==7)))
                set_invalid(&board[i][j]);

            else{
                //squares with no pieces
                if(i==0 || i ==7 || j==0 || j == 7)
                    set_empty(&board[i][j]);
                else{
                    //squares with red pieces
                    if((i%2 == 1 && (j < 3 || j> 4)) ||
                        (i%2 == 0 && (j == 3 || j==4)))
                        set_red(&board[i][j]);
                    //green squares
                    else set_green(&board[i][j]);
                }
            }
        }
```

Sets invalid squares (0,0), (0,1), (1,0), (0,7), (0,6), (1,7), (7,0). (6,0), (7,1), (7,7), (7,6), (6,7)

game_init.c

- Contains the implementation of the method necessary to initialize the board.



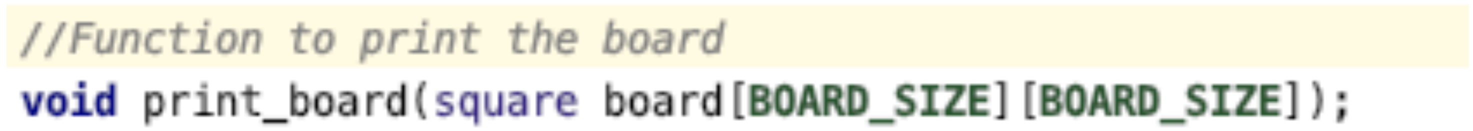
```c
void initialize_board(square board [BOARD_SIZE][BOARD_SIZE]){

    for(int i=0; i< BOARD_SIZE; i++){
        for(int j=0; j< BOARD_SIZE; j++){
            //invalid squares
            if((i==0 && (j==0 || j==1 || j==6 || j==7)) ||
               (i==1 && (j==0 || j==7)) ||
               (i==6 && (j==0 || j==7)) ||
               (i==7 && (j==0 || j==1 || j==6 || j==7)))
                set_invalid(&board[i][j]);

            else{
                //squares with no pieces
                if(i==0 || i ==7 || j==0 || j == 7)
                    set_empty(&board[i][j]);
                else{
                    //squares with red pieces
                    if((i%2 == 1 && (j < 3 || j> 4)) ||
                       (i%2 == 0 && (j == 3 || j==4)))
                        set_red(&board[i][j]);
                    //green squares
                    else set_green(&board[i][j]);
                }
            }
        }
```

Sets empty squares and those that have RED and GREEN pieces.

input_output.h

- This header file contains the prototypes of the methods to print on the standard output and to require input from the user.

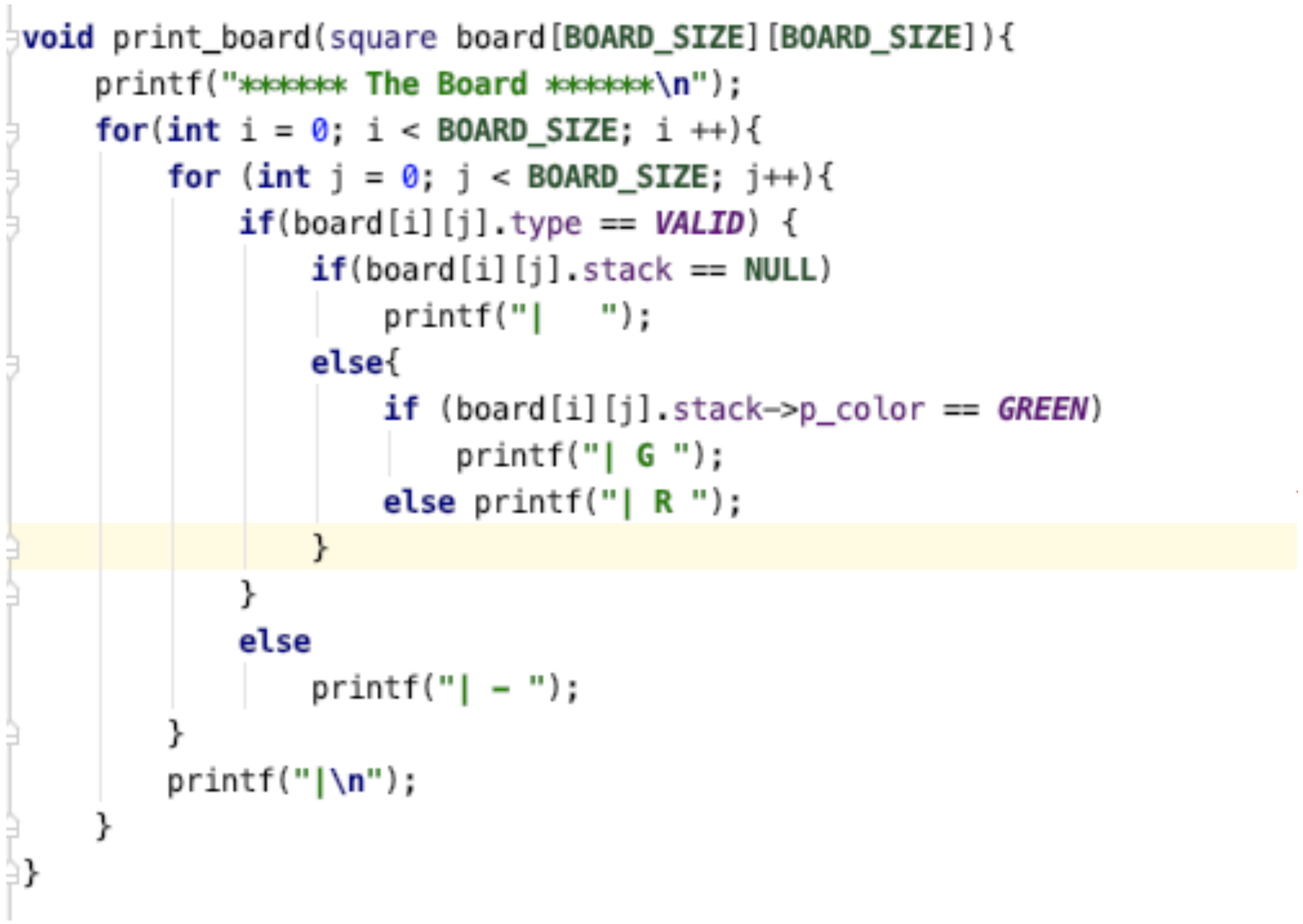
- At the moment it only contains the prototype of the method necessary to print the board



```c
//Function to print the board
void print_board(square board[BOARD_SIZE][BOARD_SIZE]);
```

input_output.c

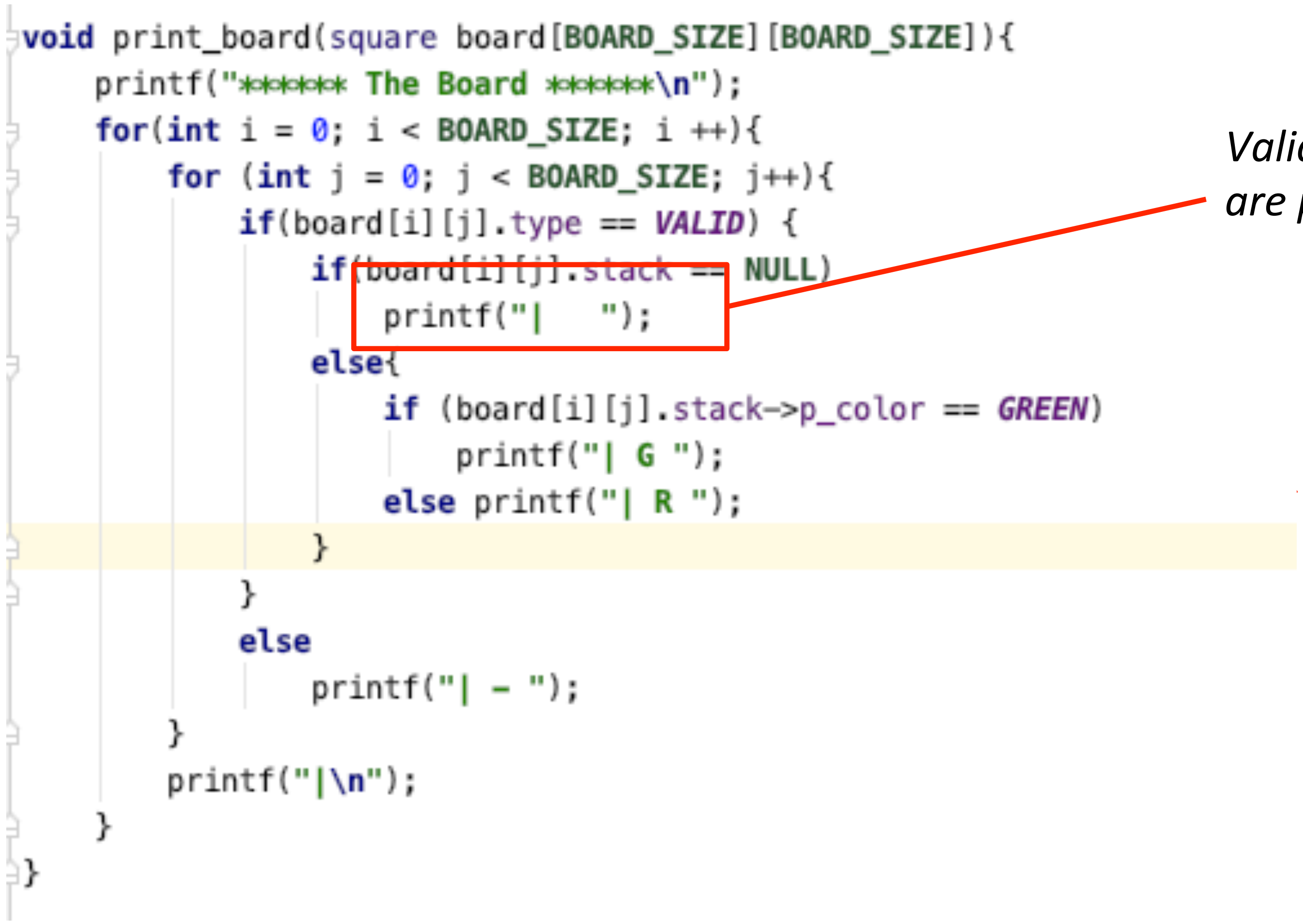
- This source file contains the implementation of the methods declared in *input_output.h*.



```c
void print_board(square board[BOARD_SIZE][BOARD_SIZE]){
    printf("****** The Board ******\n");
    for(int i = 0; i < BOARD_SIZE; i ++){
        for (int j = 0; j < BOARD_SIZE; j++){
            if(board[i][j].type == VALID) {
                if(board[i][j].stack == NULL)
                    printf("|   ");
                else{
                    if (board[i][j].stack->p_color == GREEN)
                        printf("| G ");
                    else printf("| R ");
                }
            }
            else
                printf("| - ");
        }
        printf("|\n");
    }
}
```

input_output.c

- This source file contains the implementation of the methods declared in *input_output.h*.

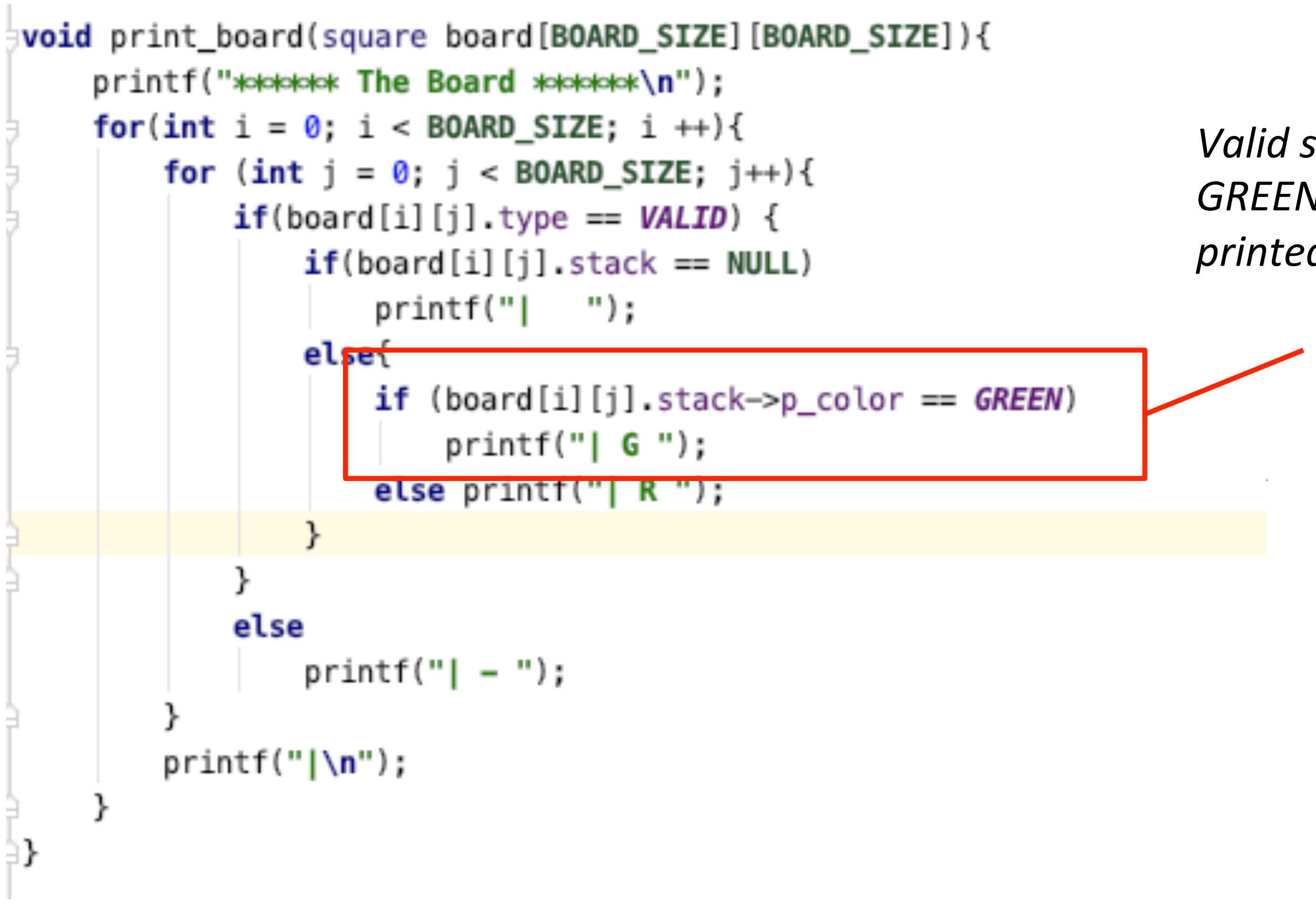


```c
void print_board(square board[BOARD_SIZE][BOARD_SIZE]){
    printf("****** The Board ******\n");
    for(int i = 0; i < BOARD_SIZE; i ++){
        for (int j = 0; j < BOARD_SIZE; j++){
            if(board[i][j].type == VALID) {
                if(board[i][j].stack == NULL)
                    printf("|   ");
                else{
                    if (board[i][j].stack->p_color == GREEN)
                        printf("| G ");
                    else printf("| R ");
                }
            }
            else
                printf("| - ");
        }
        printf("|\n");
    }
}
```

Valid empty squares are printed as | |

input_output.c

- This source file contains the implementation of the methods declared in *input_output.h*.

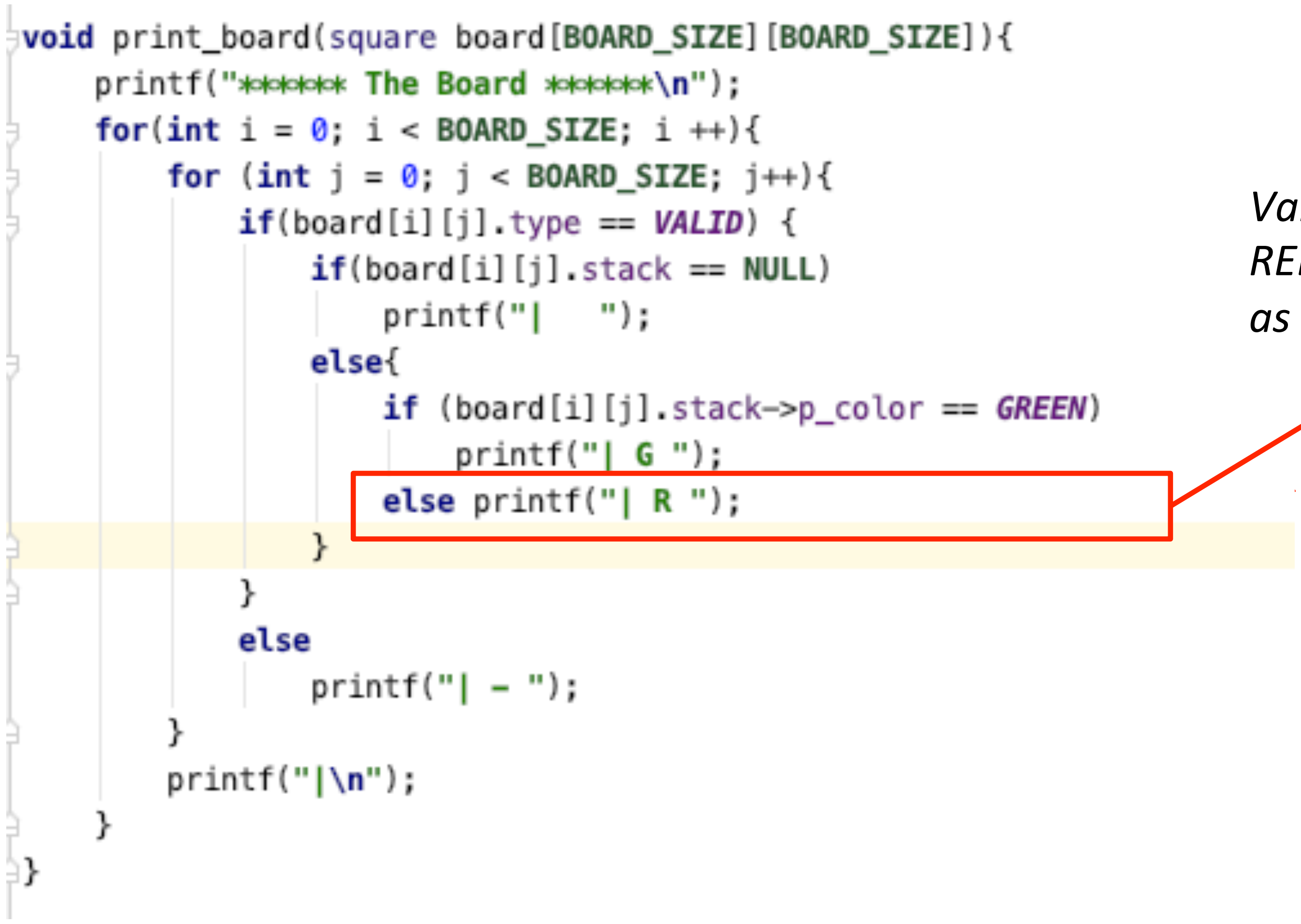


```c
void print_board(square board[BOARD_SIZE][BOARD_SIZE]){
    printf("****** The Board ******\n");
    for(int i = 0; i < BOARD_SIZE; i ++){
        for (int j = 0; j < BOARD_SIZE; j++){
            if(board[i][j].type == VALID) {
                if(board[i][j].stack == NULL)
                    printf("|   ");
                else{
                    if (board[i][j].stack->p_color == GREEN)
                        printf("| G ");
                    else printf("| R ");
                }
            }
            else
                printf("| - ");
        }
        printf("|\n");
    }
}
```

Valid squares with a GREEN piece are printed as | G |

input_output.c

- This source file contains the implementation of the methods declared in *input_output.h*.

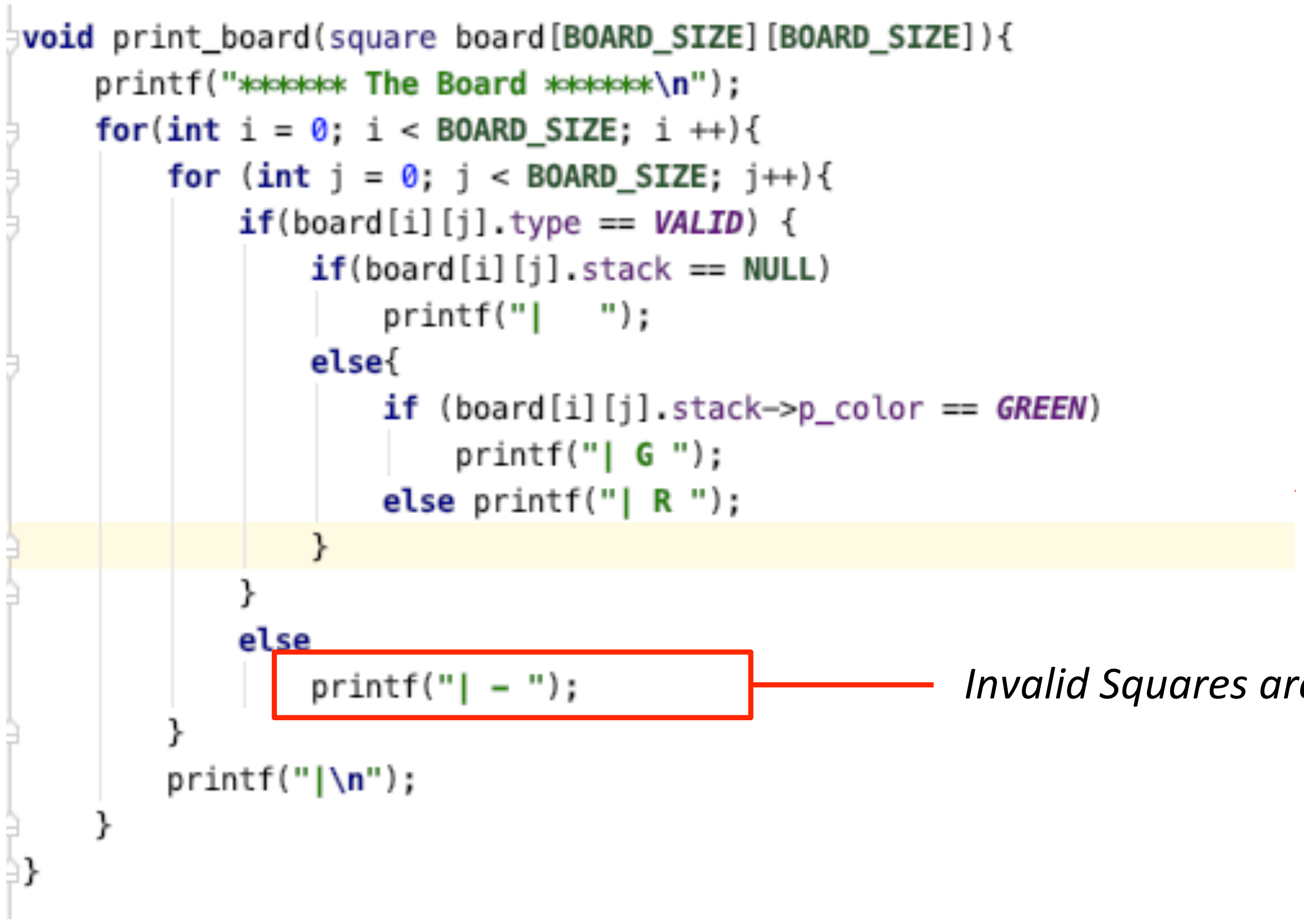


```c
void print_board(square board[BOARD_SIZE][BOARD_SIZE]){
    printf("****** The Board ******\n");
    for(int i = 0; i < BOARD_SIZE; i ++){
        for (int j = 0; j < BOARD_SIZE; j++){
            if(board[i][j].type == VALID) {
                if(board[i][j].stack == NULL)
                    printf("|   ");
                else{
                    if (board[i][j].stack->p_color == GREEN)
                        printf("| G ");
                    else printf("| R ");
                }
            }
            else
                printf("| - ");
        }
        printf("|\n");
    }
}
```

Valid squares with a RED piece are printed as | R |

input_output.c

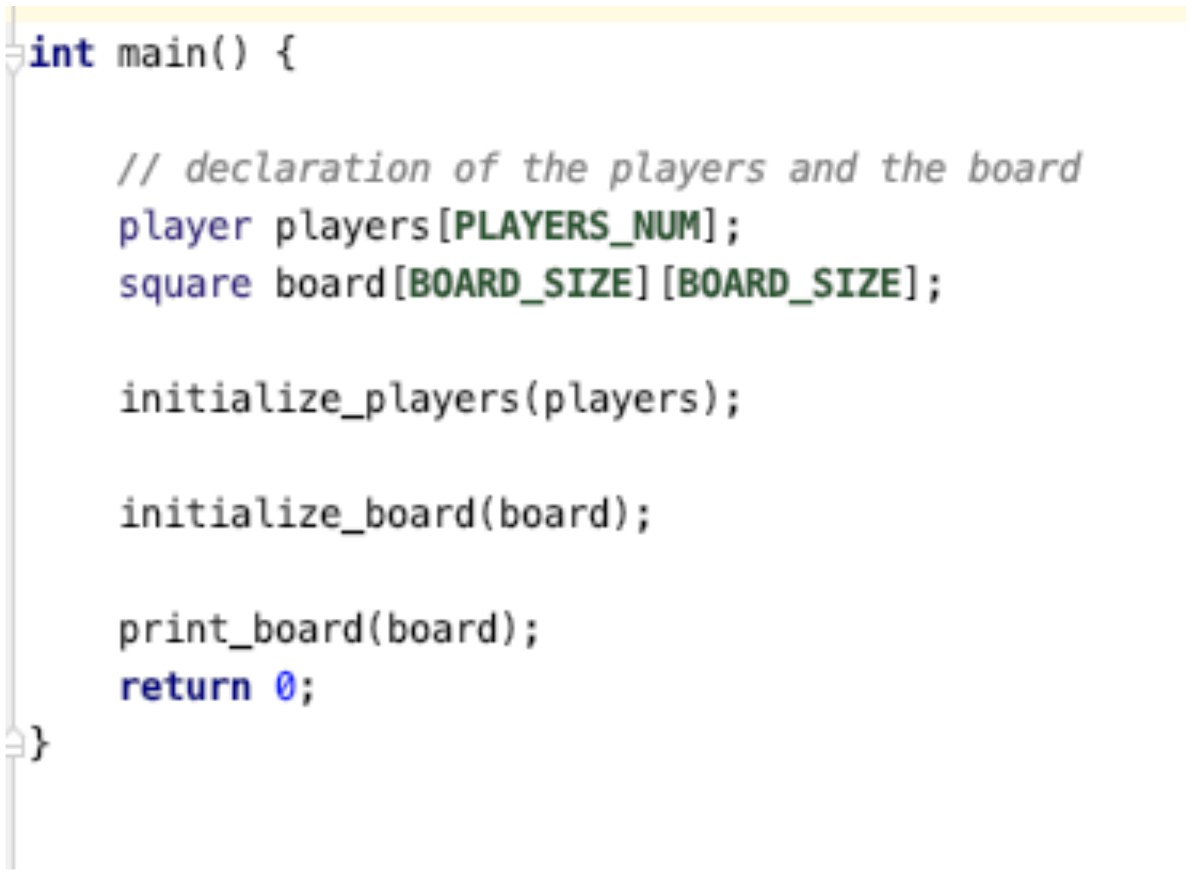
- This source file contains the implementation of the methods declared in *input_output.h*.



```c
void print_board(square board[BOARD_SIZE][BOARD_SIZE]){
    printf("****** The Board ******\n");
    for(int i = 0; i < BOARD_SIZE; i ++){
        for (int j = 0; j < BOARD_SIZE; j++){
            if(board[i][j].type == VALID) {
                if(board[i][j].stack == NULL)
                    printf("|   ");
                else{
                    if (board[i][j].stack->p_color == GREEN)
                        printf("| G ");
                    else printf("| R ");
                }
            }
            else
                printf("| - ");
        }
        printf("|\n");
    }
}
```

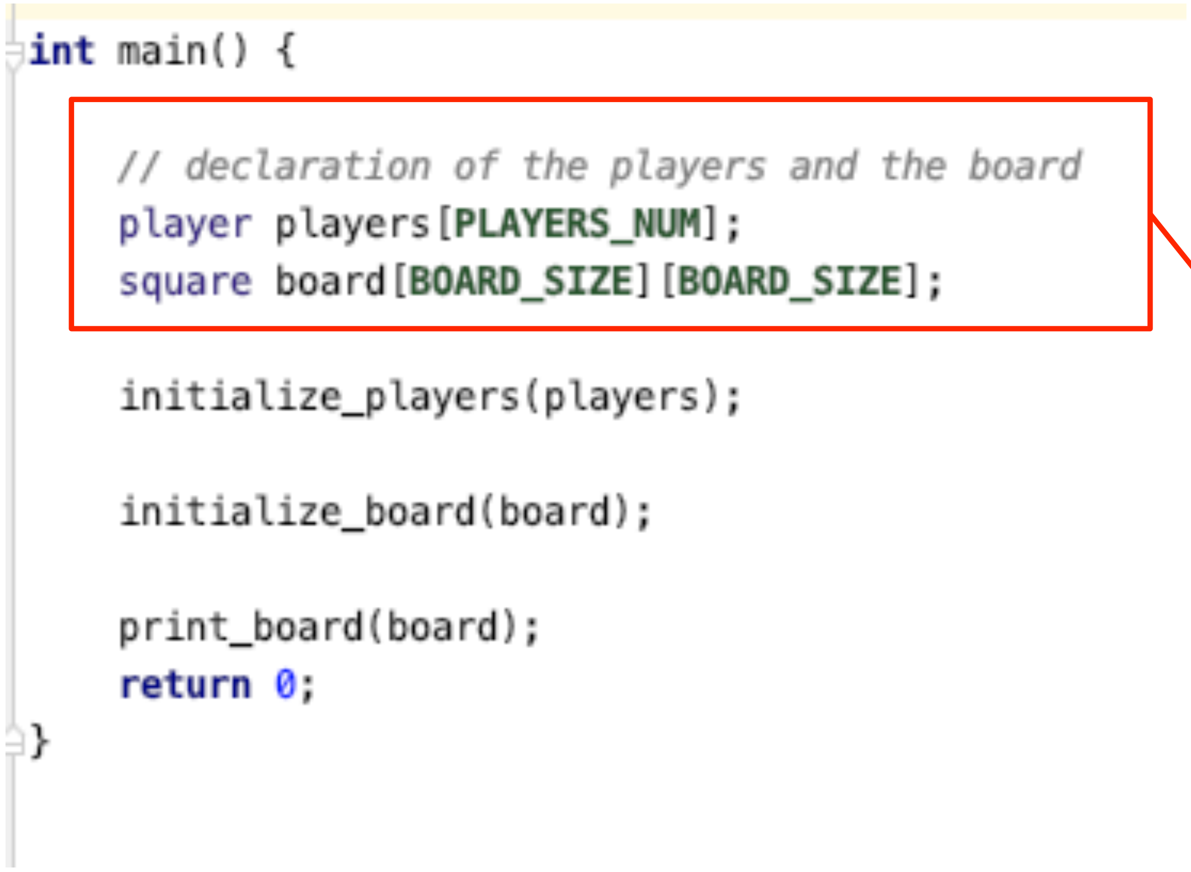
Invalid Squares are printed as | - |

main.c



```c
int main() {

    // declaration of the players and the board
    player players[PLAYERS_NUM];
    square board[BOARD_SIZE][BOARD_SIZE];

    initialize_players(players);

    initialize_board(board);

    print_board(board);
    return 0;
}
```

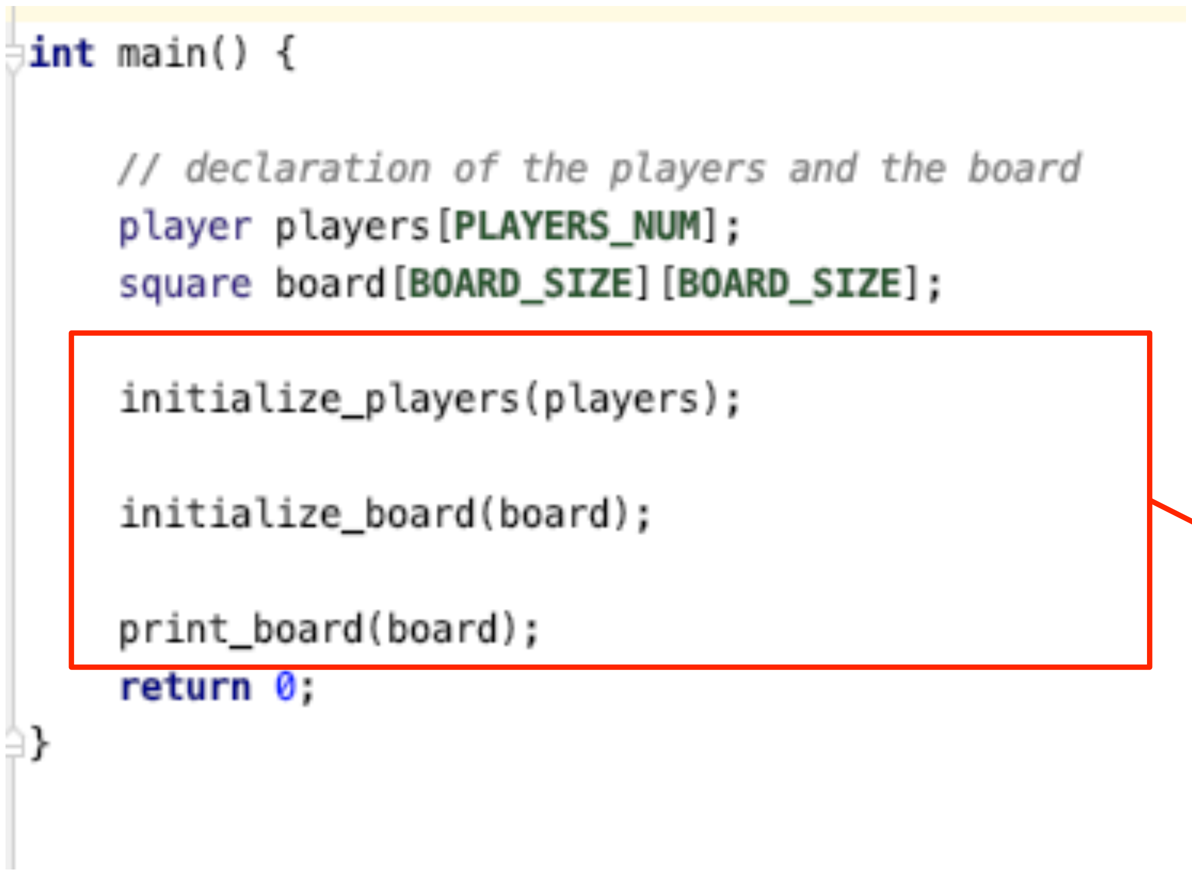
main.c



```
int main() {

    // declaration of the players and the board
    player players[PLAYERS_NUM];
    square board[BOARD_SIZE][BOARD_SIZE];

    initialize_players(players);

    initialize_board(board);

    print_board(board);
    return 0;
}
```

Declares the main entities of the game: the board, the players and the number of players.

main.c

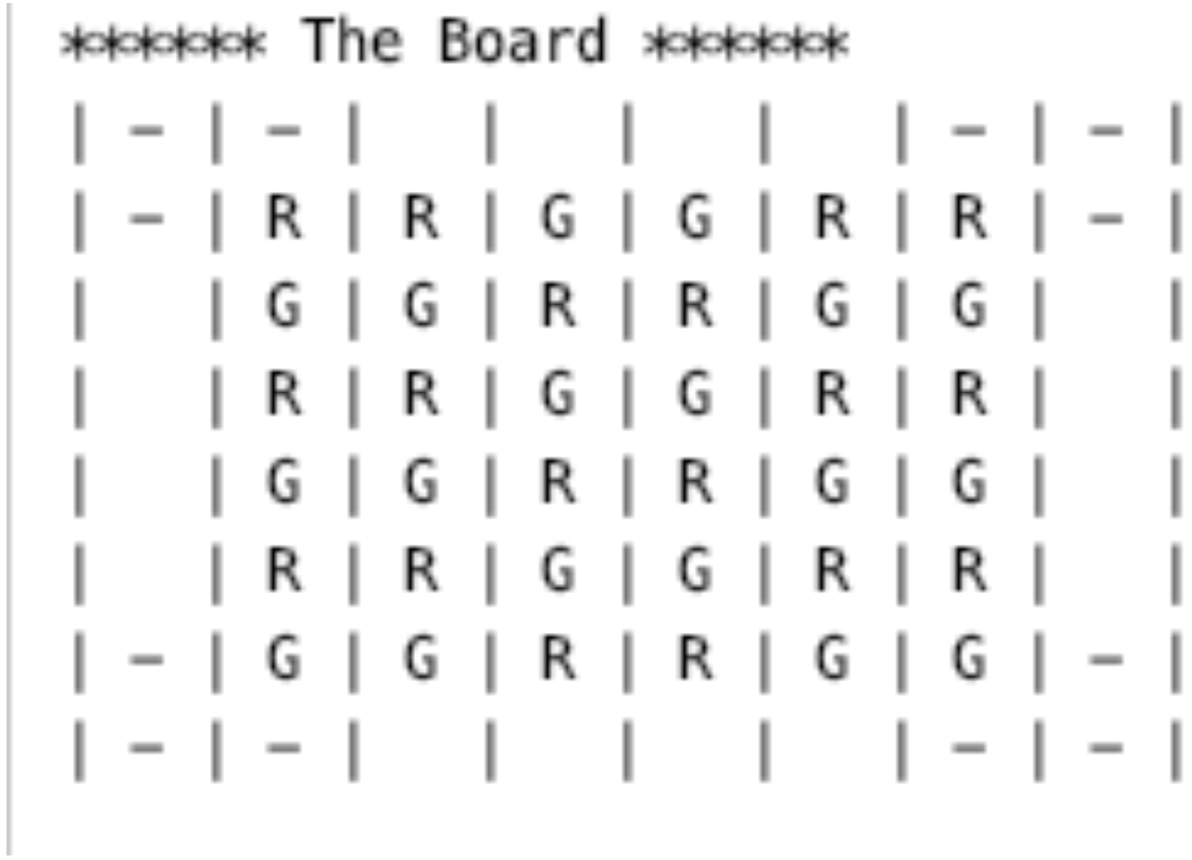


```
int main() {

    // declaration of the players and the board
    player players[PLAYERS_NUM];
    square board[BOARD_SIZE][BOARD_SIZE];

    initialize_players(players);

    initialize_board(board);

    print_board(board);
    return 0;
}
```

Initializes the players and the board and prints the board.

Output

- The program prints the board of the game as follows



```
****** The Board ******
| - | - |   |   |   |   | - | - |
| - | R | R | G | G | R | R | - |
|   | G | G | R | R | G | G |   |
|   | R | R | G | G | R | R |   |
|   | G | G | R | R | G | G |   |
|   | R | R | G | G | R | R |   |
| - | G | G | R | R | G | G | - |
| - | - |   |   |   |   | - | - |
```