Alexandria
University Faculty of
Engineering
Computer and Systems Engineering
Dept. CS121: Computer Programming *1*

Final Project
Chess

Names:

1. Ahmed Mahmoud elsa3ed gaballah 19015357
2. Ahmed Mahmoud abdelhay elemary 19015359

Problem statement

- **Two players:** Human vs. Human.
- **User Interface:**
  - ➔ You should represent black and white squares using different ASCII characters, e.g., “.” for black squares and “-” for white squares.
  - ➔ White pieces are represented by “p”,“r”,“n”,“b”,“q”,“k” for pawns, rooks, knights, bishops, queen and king respectively. Black pieces are represented by the same letters but capitalized.
  - ➔ You should display next to the game board the pieces that are currently taken out from both players.
  - ➔ You should display the characters from ‘A’ to ‘H’ to denote the columns above and below the board, along with the numbers from 1 to 8 to denote the rows to the right and left of the board.
- **Rules:**
  - ➔ Your program should support promotion and stalemate.
  - ➔ The game is won by checkmate.
  - ➔ The game may end in a draw by stalemate.

- **Movement:**

As shown in the Wikipedia page, each square is indexed by a character and a number (e.g. D1,C3,...etc.). The user specifies his next move by entering the index of the piece he wishes to move, followed by the index of the square he wishes to move it to.(e.g. A3B4 will move the piece at A3 to the square B4). The only exception is in the case of promotion, where the user must specify an additional character indicating the desired piece. (e.g. H7H8B will promote the pawn to a bishop).

- **Undo and Redo:** The game could be undone till the first move.

- **Save and Load:**
  - ➔ At any time, the game could be saved to a file
  - ➔ A saved game could be loaded and continued
  - ➔ No need to save the undo data.

Description

The white pieces are represented in lowercase letters and the black pieces are represented by capital letters. They are all represented by the first letter of their names, the only exception being the Knight, which is represented by an **N**, leaving the **K** for the king).

The white squares are represented by " . " and the black squares are represented by " - ".

The pieces that are taken out from both players are shown above the board.

If the white player tries to move black pieces or the opposite, the program will print "Not valid!".

If a player tries to move an empty square, the program will print "Not valid!".

At every move the program check first that this move does not make the king in check and if it does, the program will print "Not valid move!" and ask the user again.

The game ends with draw in case one of the following:

- King versus king
- King and bishop versus king
- King and knight versus king
- King and bishop versus king and bishop with the bishops with the same color.

Or when a player is not checked and there is no legal move to do, the program will print "The game ends with DRAW!".

Features:

1. Castling:
   You can castle right or left if there is the first move to each of the king or any of the two rooks, and the squares between them are empty, if the king is under attack it will not castle in this condition, and if the king destination of castling is under attack from opponent's piece it will not castle, and the program will print "you can't do this move".

2. Promotion:
   If the white pawn arrives to row 8 the program will ask you to the promoted piece and the same if the black pawn arrives to row 1.

3. En passant:
   It is a special pawn capture that can only occur immediately after a pawn makes a move of two squares from its starting square, and it could have been captured by an enemy pawn had it advanced only one square. The opponent captures the just-moved pawn "as it passes" through the first square.

4. Redo the promotion
   After doing the promotion and replacing the pawn by the promoted piece and undone this play then redo it, we are saving the move with its promoted piece like (a7a8q), and during the redo we replace that pawn with the promoted piece.

5. Undo the loaded game
   if the user saved the game and then he loaded it, he can undo the till the first move.

Overview of the design:

Firstly, we make a 2D array of characters and assign to it the initial state of the chess board, after scanning the input from the user we check if this valid move for this piece by making a function for every piece that check this move, then we do this move, and we check if this move make the king in danger, if true we undo this move, and ask the user for valid move, then we print the captured pieces array and display the board, if this move was a king or any of the rooks but not castling we prevent the user to castle after that, then we check if this move check the opponent king or not, if it check the king the other player will play the move that get the king out of check like trying to move the king, capture the attacking piece or defend the king by another piece, Our program count the number of available moves for the attacked player, if there is no available moves to get out of check then the other player has won.

At every move we count the number of valid moves, if there is no valid move for the player and his king is not under attack, then the game ends with draw.

At every valid move we save it to a temp array, if the user undo once, we read the moves from this array except the last one and do them, and so on.

If the user redo a move, we read the moves from the array and do moves higher than the number of undo by one.

If the user needs to save the game, we will ask him about the file name and copy the content of the temp array to this file, and when he loads the game, he will enter the file name which he saved it, and we will read the moves from this file and do it.

If the user wants a new game, we will reset the board and clear the temp array.

Assumptions

- In case of promotion, the user shouldn't specify an additional character for the promoted piece, but he enter the original move like (a7a8) then the program will ask for the promoted piece.
- In case of en-passant, the user doesn't lose the right to do the *en passant* capture if he didn't do it on the very next turn.
- If the king is under check and the castling is valid, he can't castle, and the program will print "you can't do this move".
- If the king or one of the rooks moved from his start position them the castling will not valid any more in the game.

Description of all used data structures.

We used a 2D array of characters to store the chess board elements, and a 2D temp array to store the elements also and replace it with the original board when the user starts a new game, and a single dimensional array to store both the white and black taken pieces and reset them also when starting a new game.

We used a 1D array during reading the file to store the file movements in it, then put this moves in a 2D array and use it to do the moves during the loading, undo, or redo.

Description of the important functions

Promotion (); this function check if the white pawn reached to row 8 or the black pawn reached to row 1 and scan the promoted piece from the user and it returns that piece to replace it with this pawn.

Enpassent (); this function check if the white pawn moves to an empty square from row 5 to row 6 and the piece that in his row and the column next to it is black pawn and put it in the taken array, and if the black pawn moves to an empty square from row 4 to row 3 and the piece that in his row and the column next to it is white pawn and put it in the taken array.

Castling (); we check first if this piece is a white or black king and in row 1 or 8, and if this piece is white king, we check if the rook in her place and the squares between them are empty, and the same with the black king, then we return true.

validRookMove (); this function checks firstly the horizontal move, if the rook moves from row to the same row, then check that the squares in his way are empty and return true if the destination cell is free. secondly, check the vertical move, if the rook moves from column to the same column, then check that the squares in his way are unoccupied, and return true if the destination cell is free.

validBishopMove (); this function checks the diagonal move for the bishop, if the index of his row plus his column equals the index of destination row plus destination column (right diagonal), or if the index of his column minus his row equals the index of destination column minus destination row (left diagonal), it returns true.

validQueenMove (); this function calls two function, for the horizontal move it calls validRookMove function, and for the diagonal move it calls validBishopMove function.

validKnightMove (); this function checks for the valid moves for the knight, if he moves two squares vertically and one square horizontally, or two squares horizontally and one square vertically, and the destination cell is free or occupied by the enemy piece.

validkingMove (); this function checks if the king can move one square in any direction (horizontally, vertically, or diagonally), and the destination cell is free or occupied by enemy piece, or if castling function returns true and the destination cell is not under attack from any of the enemy pieces by the help of global variable (notValidMove).

validCapturePawn (); this function checks the diagonal move for the pawn, if the white pawn moved from his row to row up and to column to his right or left and the destination cell is occupied by any black piece, and the same with the black pawn, then it returns true.

validPawnMove (); this function checks firstly the vertical pawn move, if the white pawn move one row up or the black pawn move one row down, and the destination cell is empty, or if white pawn moved from row 2 to row 4 or the black pawn moved from row 7 to row 5 and the destination cell is empty, secondly it calls the validCapturePawn function, then it calls the enpassent function.

Check (); this function knows if the king is in check from the attacker piece, we check if the king is under attack from every valid move to the attacker piece.

PieceAttackTheKing (); if a player moved a piece we check if his king is under attack after this move or not.

numberOfMoves (); this function counts the number of valid moves in current turn, if the turn is on the white it checks the valid moves for all white pieces and for every valid move the white king should not be in check.

numberOfPieces (); this function count how many pieces of same kind in the board, it helps us in draw function.

Draw (); it checks for the dead positions, firstly it checks if the white and black king only on the board. Secondly, it checks if a white king and one bishop with black king only on the board and the opposite. Then, it checks if a white king and one knight with black king only on the board and the opposite. Then it checks if a white king and bishop with black king and bishop only on the board and the bishops are on the same type of squares.

readMoves (); this function reads the moves from the txt file and store them in a temp array.

Load (); this function calls readmoves() function and do all this moves and print the captured pieces.

Undo (); this function calls the load() function but with different parameters depending on the number of undo, e.g. when number of undo equal one the load function do all the moves except the last one.

Redo (); this function calls also the load() function but it minus the number of undo, e.g. when number of undo equal one he becomes zero and the load function do all the moves.

afterUndo (); if the a player undo then moved a piece this function remove all the moves after this play.

Save (); it saves the moves in the temp array to a txt file.

Pseudocode

This program will loop until any player win or the game ends with draw or the user press q

Function check (char attacker,int torow,int tocol){

If "board [torow][to col] is white" then the attacked king is the black

Else the attacked king is the white

We check all the attacker pieces after valid move for it if the opponent king is in his way if true underAttak=1

}

Function kingOutOfcheck(char c, int a_row, int a_col){

If the king is under attack //underAttack=2

We call check(c,int_a,int_col) function //to know if this piece attack the king

If (underAttack = 1) //if this piece already attack the king

Notvalid=1

}

Function pieceAttackTheKing(char attacker){ //when attacker piece check the opponent king

//we check for all the attacker pieces if also attack the king

We check every piece for the attacker and call the function kingOutOfCheck for this piece to know if it check also the king

}

```
Function uncheckedMove(char attacked){//check if the move doesn't make the king in danger
        underAttack=2
        if the attacked is white we call pieceAttackTheKing (black)
        if the attacked is black we call pieceAttackTheKing (white)
}
Function numberOfMoves(){//it counts the number of valid moves in current turn
        If the turn is even we check only the white pieces and the opposite
        Then We check for every valid move for every piece by doing it and raise moves by
        one and Undo this move on the board
}
Function load(int movement){//it do the stored moves in the temp array
        It assign the values of fromrow,fromcol,torow,tocol from the temp array
        Then do all this moves on the board from the first
        If  "there is no moves in the temp array "
                We call reset() function //it resets the board
        Else
                We call printCapturedPieces() function  //to print the taken pieces
        We call printBoardElements() function //it shows the board elements
}

Print command line and Print board elements
if "turn is even"
         print " white turn "
else
          print " black turn "
if  "underAttack=1"
        underAttack=2
Call function numberOfMoves(); //it counts the number of valid moves in current turn
If  "moves==0 and underAttack==2 "
        if "turn is odd"
```

```
                print " white turn "
        else
                print " black turn "
        break;
eles if “call function Draw()”  //it returns true if the moves=0 and the king is not in check
        print ”The game ends with draw! ”
        break;
scan “fromColChar”
if “fromColchar== n or N ”
        call newGame() function
        continue;
else if  “fromColchar==L or l ”
        loadFlag=1;
        call readMoves() function
        tempMoves=fileMoves; and   turn=tempMoves;
        call load() function      then continue
else if  “fromColchar = U or u”
        undoFlag=1;
        if  “tempMoves – numOfUndo >0”
                call reset() function then Undo() function
                turn--          then continue
        else
                print “you can’t undo more” and continue


else if  “fromColchar =R or r ”
        redoFlag=1;
        if “tempMoves – numOfUndo < tempMoves”
                call reset() function and Redo function
                turn++       then continue
```

```
        else
                print “you can’t undo more” and continue
else if “fromColchar = S or s”
        call save() function and break;
else if “fromColchar = Q or q”
        print “The game finished!” and break;
scan fromRow and toColchar and toRow
call fflush(stdin) function  //to handle input validation
if  “numOfUndo != 0”
        call afterUndo() function
        set numOfundo to zero
if (fromRow<1 or fromrow>8 or torow<1 or torow>8 or fromColchar< ‘a’ or fromColchar> ‘h’
        or toColchar<’a’ or toColchar > ‘h’ )
        print ”not valid input! ”   then continue
if (piece is small and the turn is odd)
        print “not valid!”   then continue
else if (piece is capital and the turn is even)
        print “not valid!”   then continue
else if (squre is EMPTY)
        print “not valid!”   then continue

if (call swithValidMove function)  //it returns true if this valid move for this piece
        if ( piece is pawn )
                call the promotion function
                if (call isTherePromotion() function ) //it returns true if the promotion valid
                        save the promoted piece to the temp array
        do the move by move() function
        call pieceAttackTheKing function(attacker) //check if his king is under attack or not.
```

call uncheckedMove(piece) function //check if this move make the kind in danger

if (notValidMove = 1)

print “your king is under attack and you can’t do this move”

if (underAttack =1)

undo this move

if (isCastling)

undo the castling move

continue;

call printCapturedPieces() function and printBoardElements() function

call rookCastle() function and rookcastle() function

//if this move was a king or rook it make the castling not valid any more

Call check() function //check if the opponent king is in check from attacker piece.

If (underAttack = 1)

Print “white/black king under attack”

Else

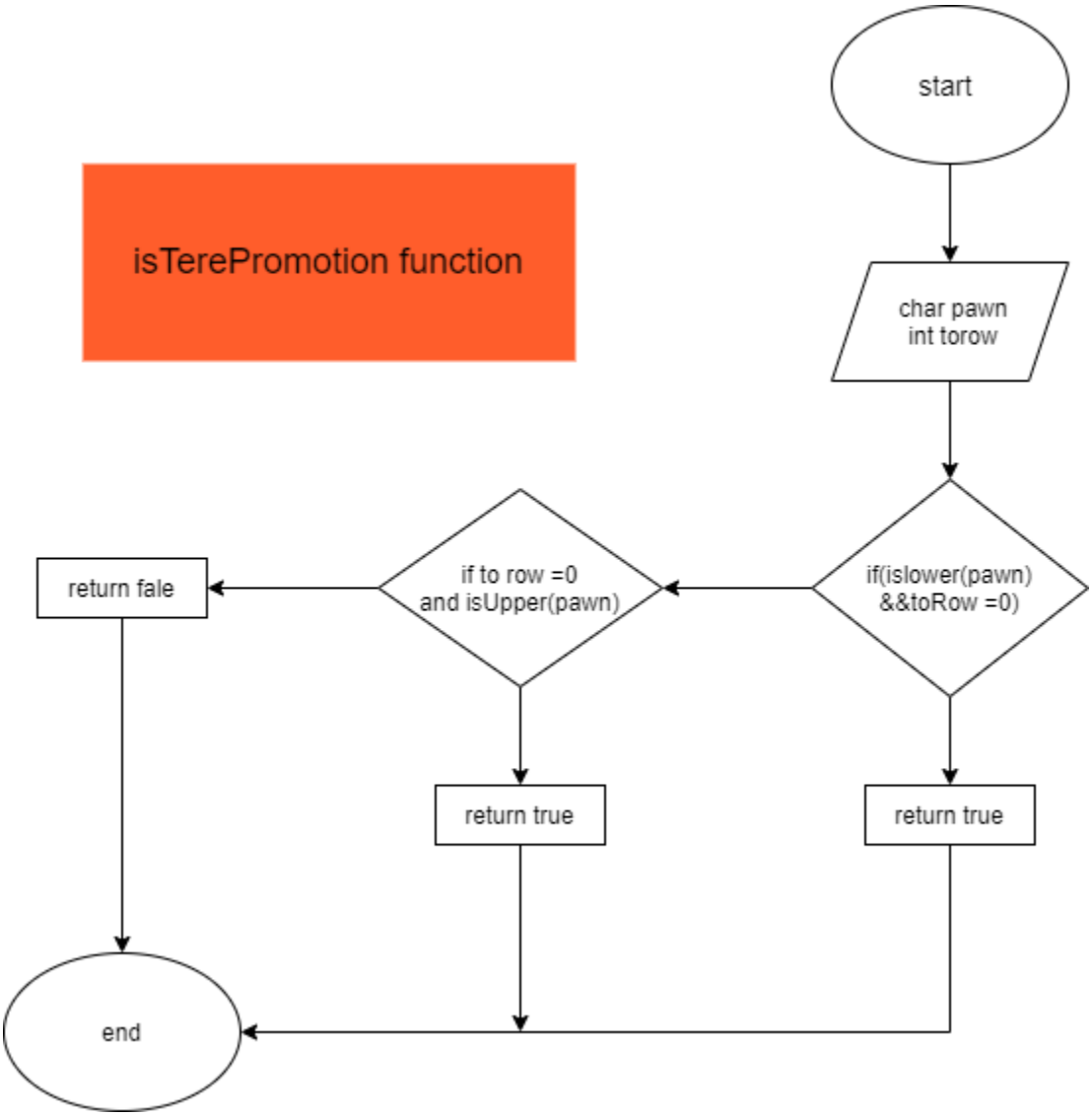
Print “you can’t do this move!” then continue

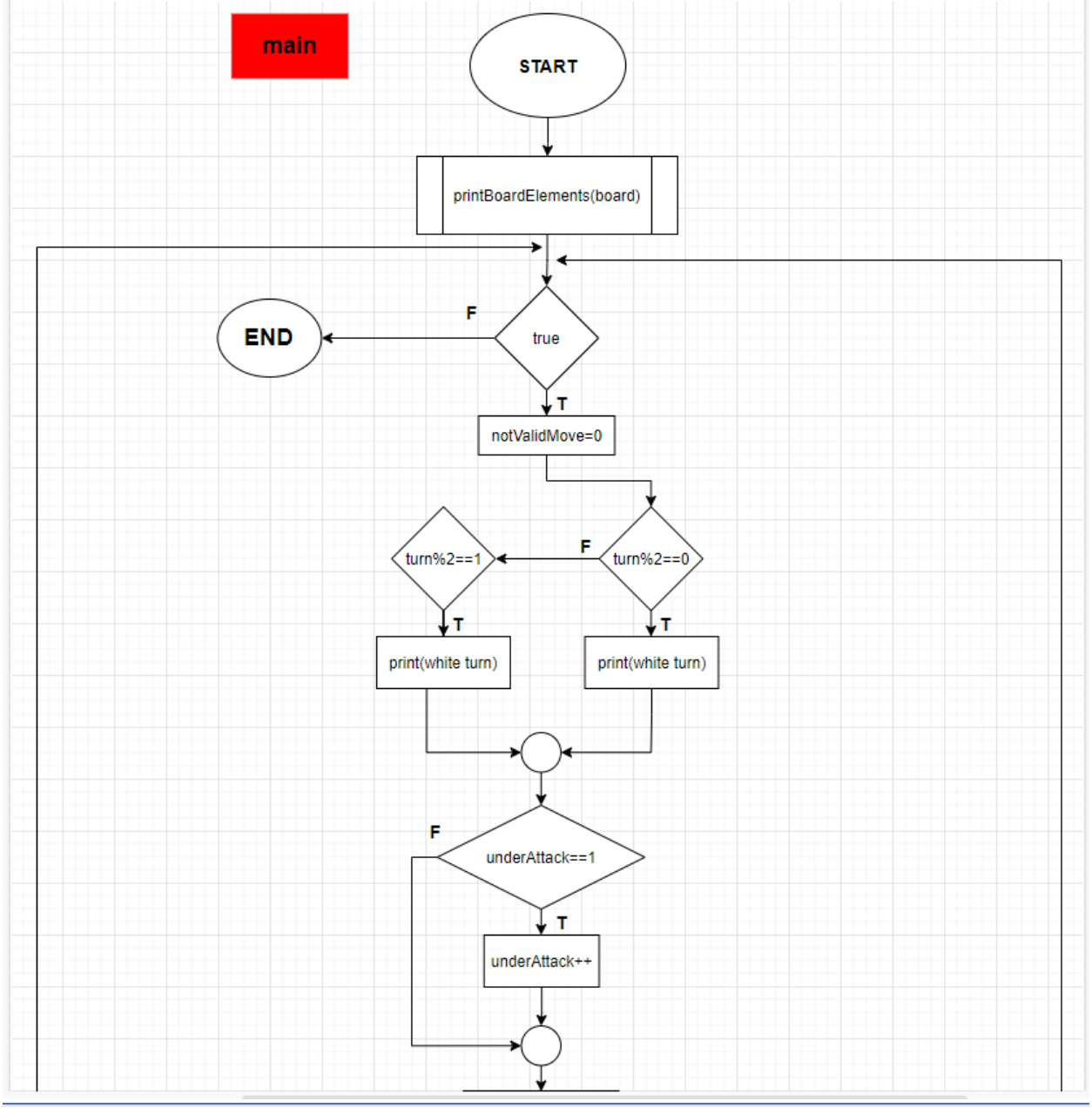
Call saveMoves() function //it save this move if it was valid

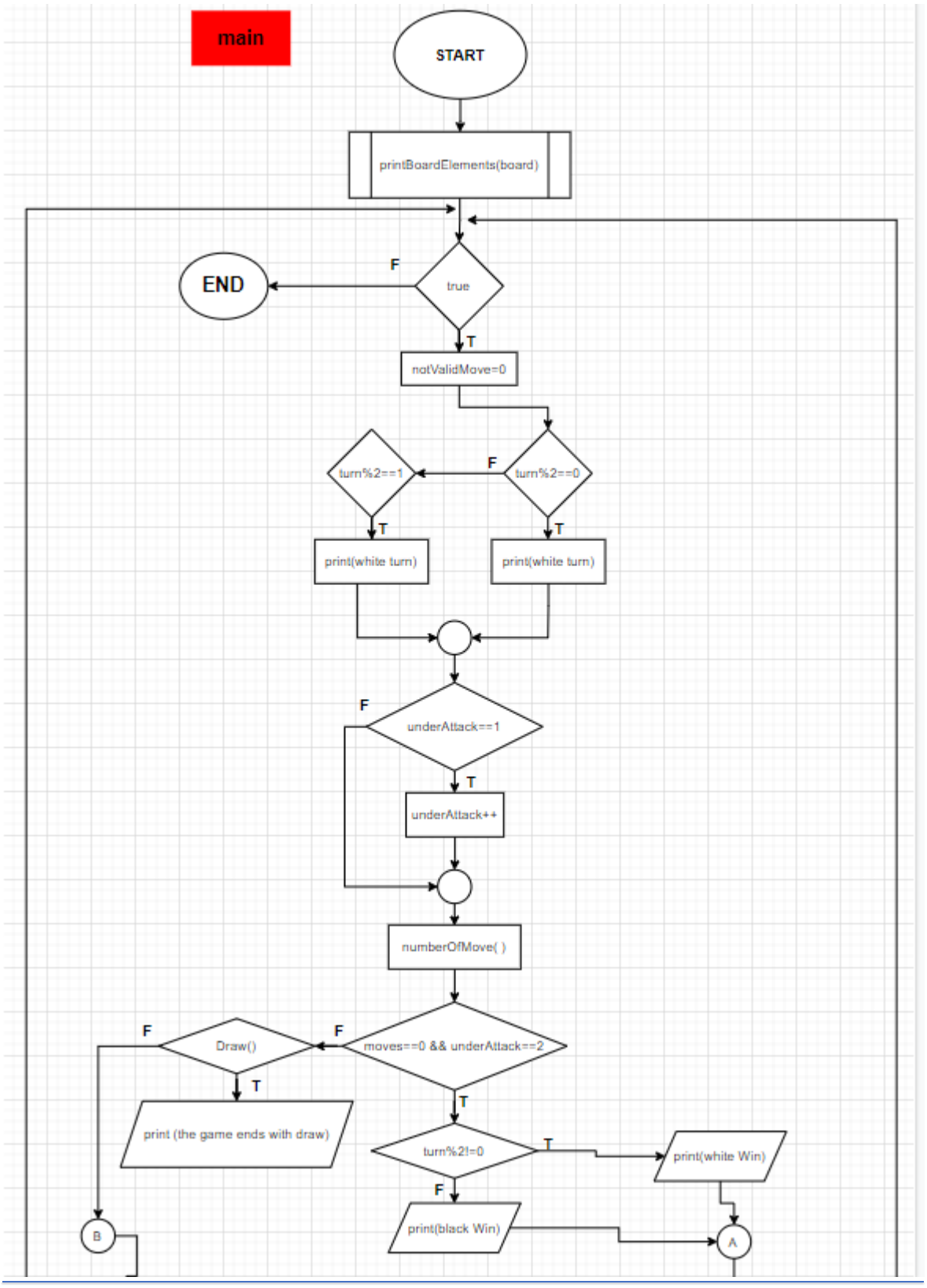
Trurn++;tempMoves++;isCastling=0;

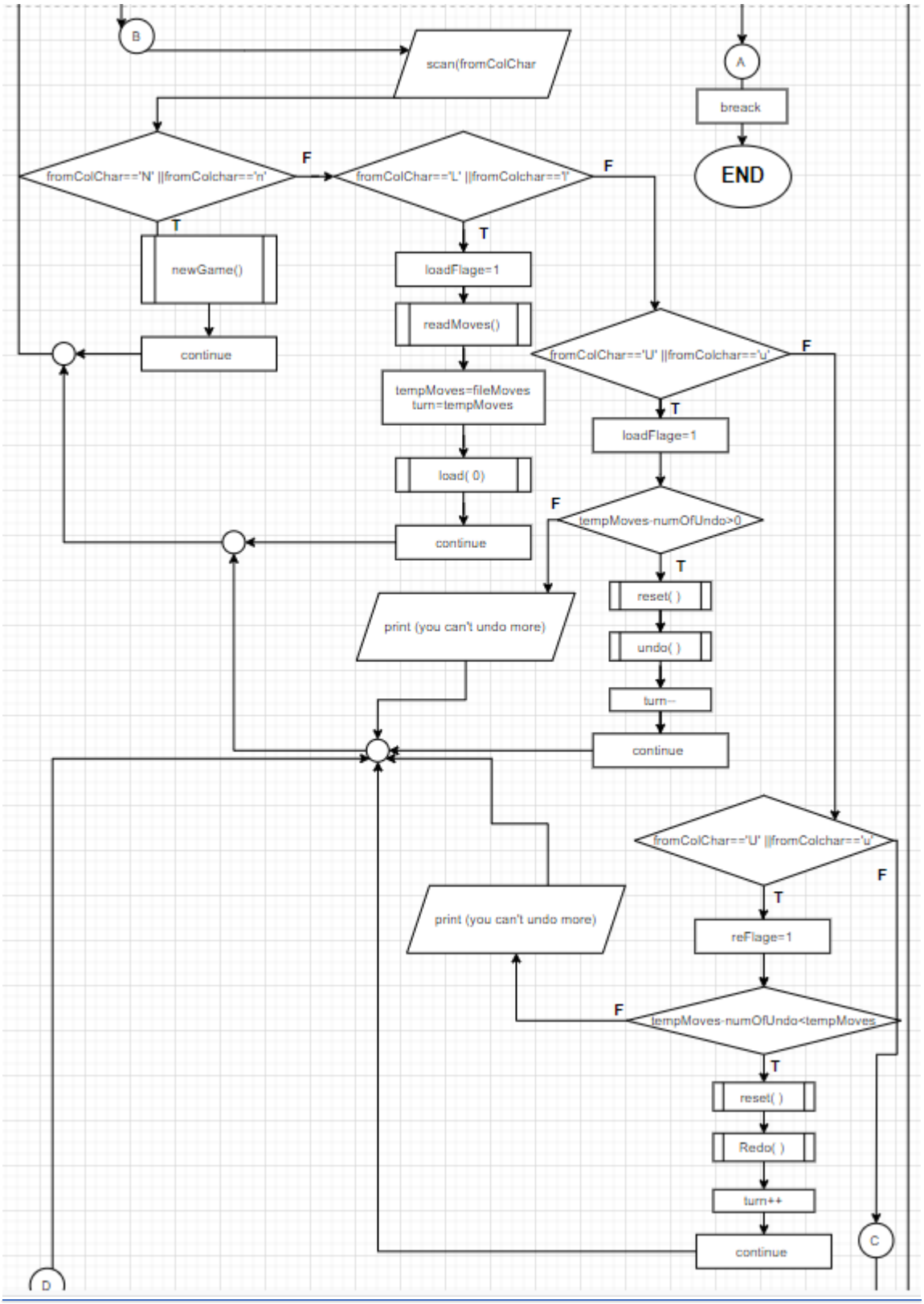
End

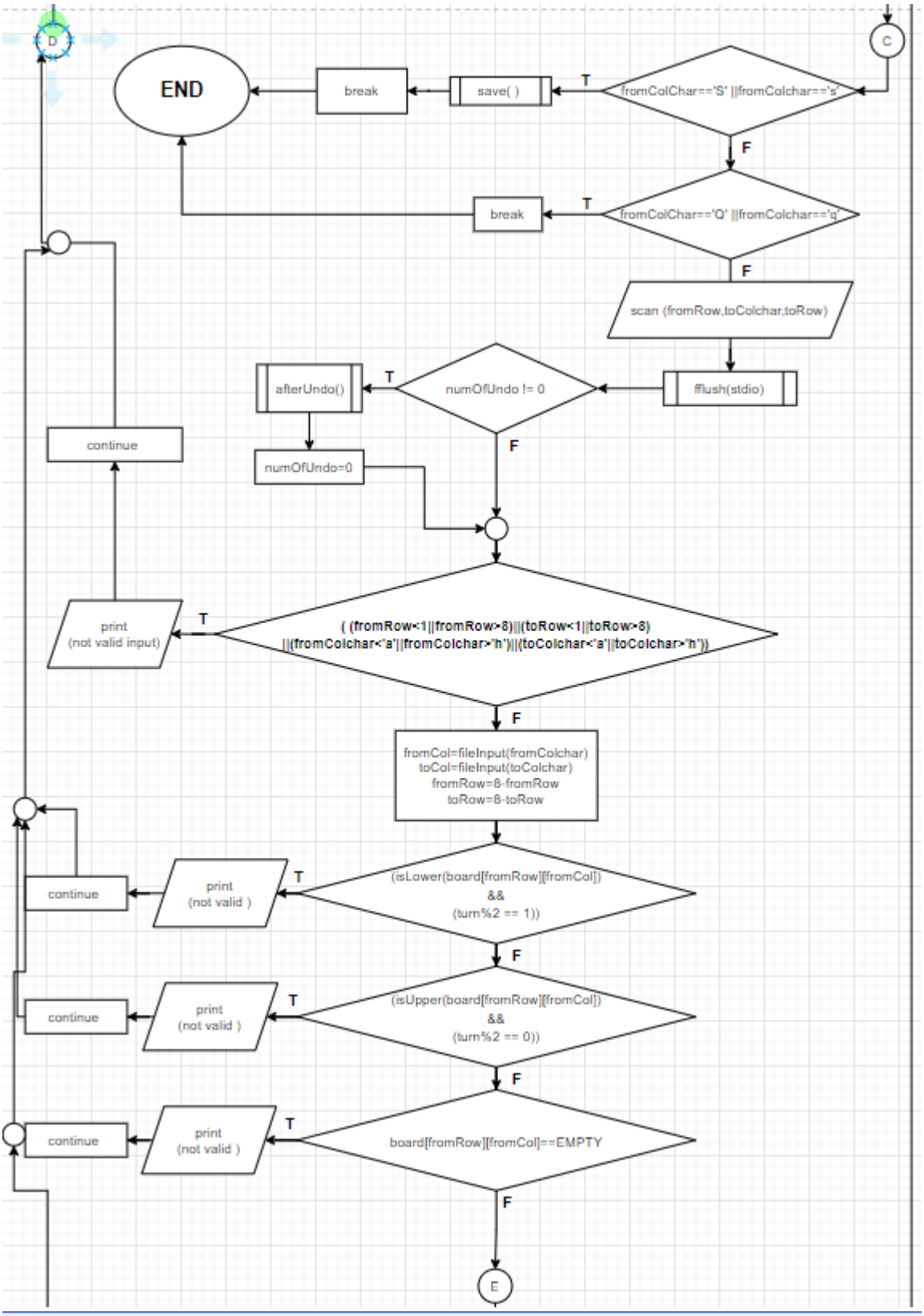
Flow charts

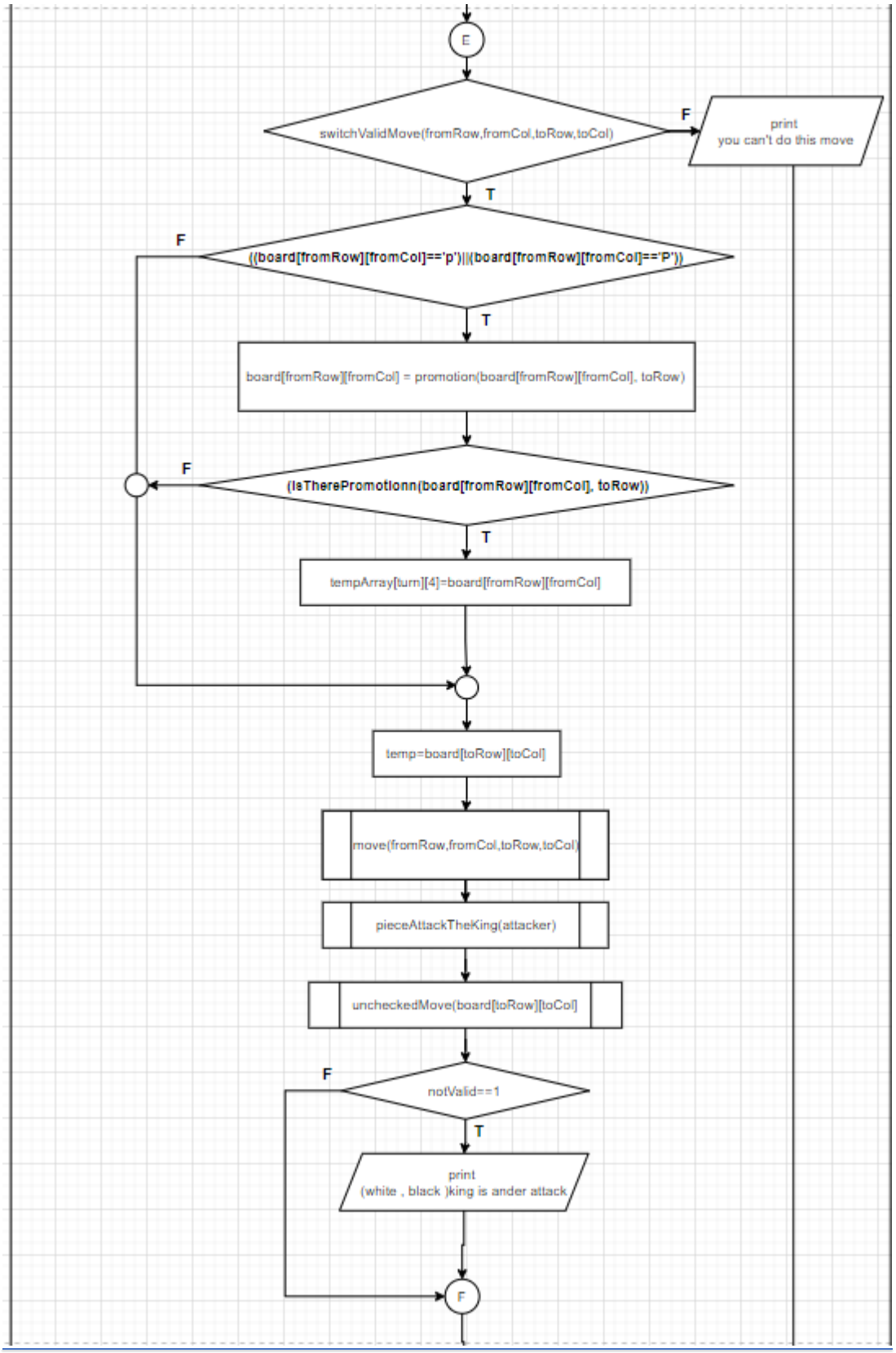

isTerePromotion function
start
char pawn
int torow
if(islower(pawn)
&&toRow =0)
if to row =0
and isUpper(pawn)
return fale
return true
return true
end

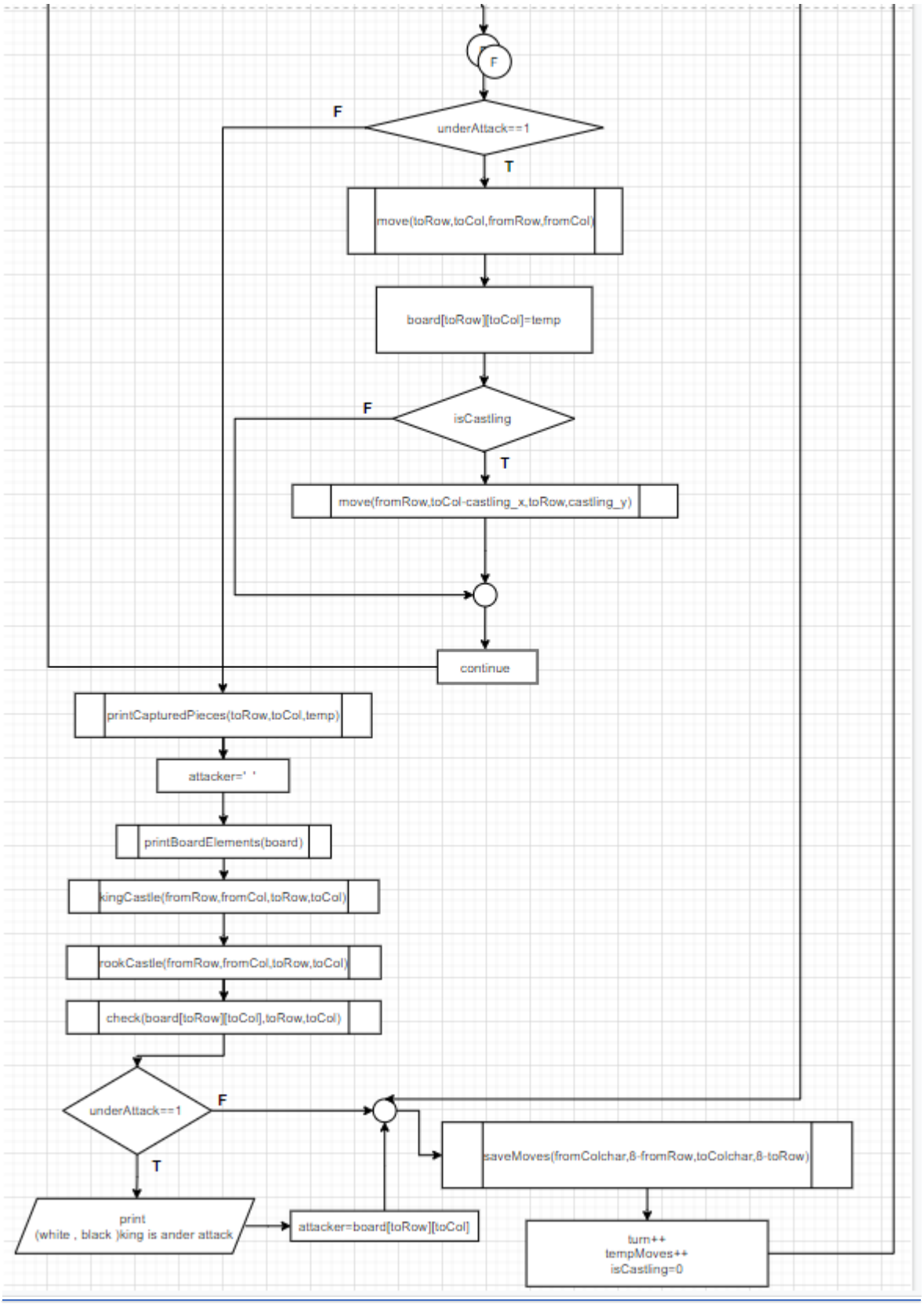

main
START
printBoardElements(board)
F
END
true
T
notValidMove=0
turn%2==1
F
turn%2==0
T
T
print(white turn)
print(white turn)
F
underAttack==1
T
underAttack++

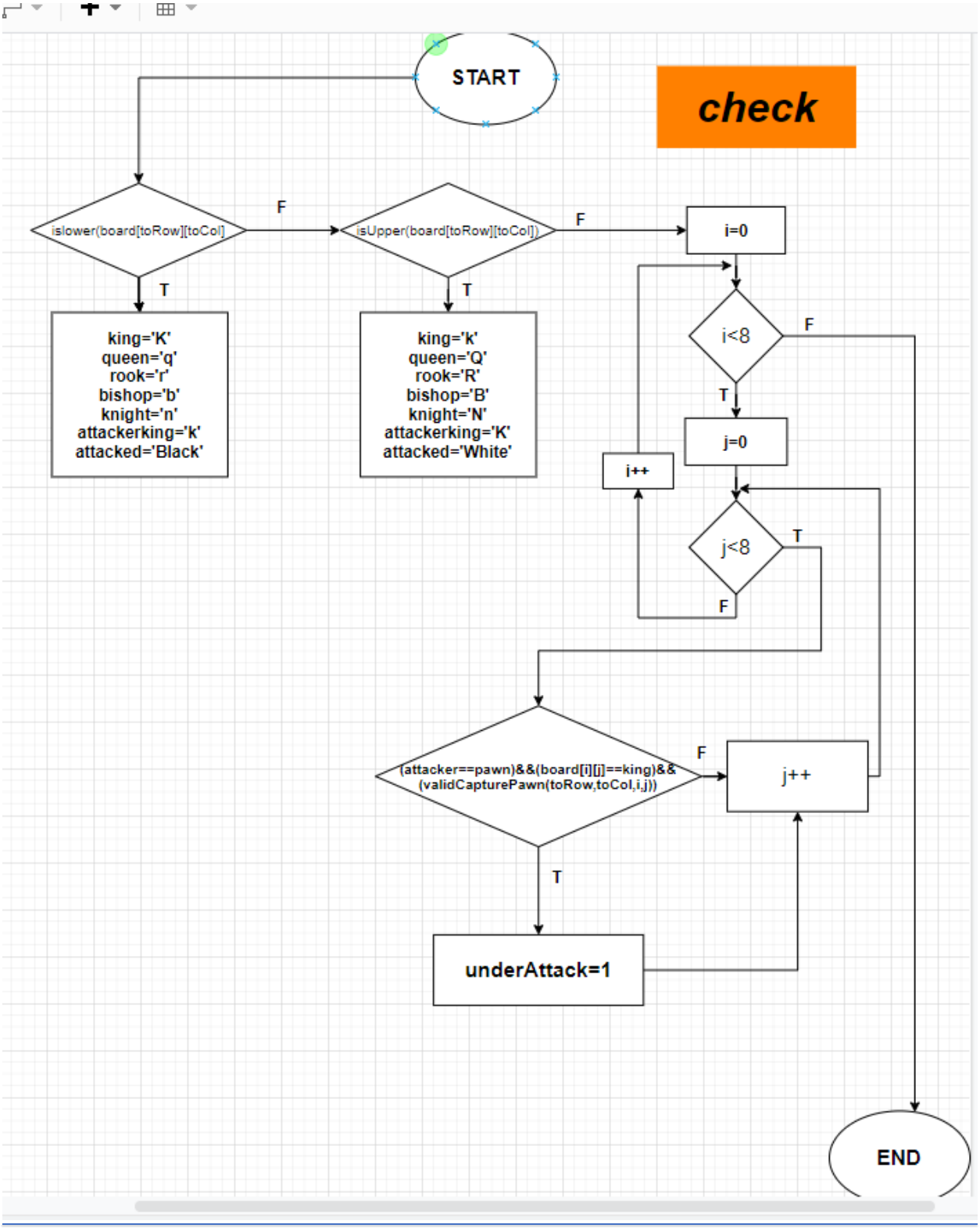

main
START
printBoardElements(board)
true
F
END
T
notValidMove=0
turn%2==1
F
turn%2==0
T
T
print(white turn)
print(white turn)
F
underAttack==1
T
underAttack++
numberOfMove( )
F
Draw()
F
moves==0 && underAttack==2
T
print (the game ends with draw)
T
turn%2!=0
T
print(white Win)
F
print(black Win)
B
A

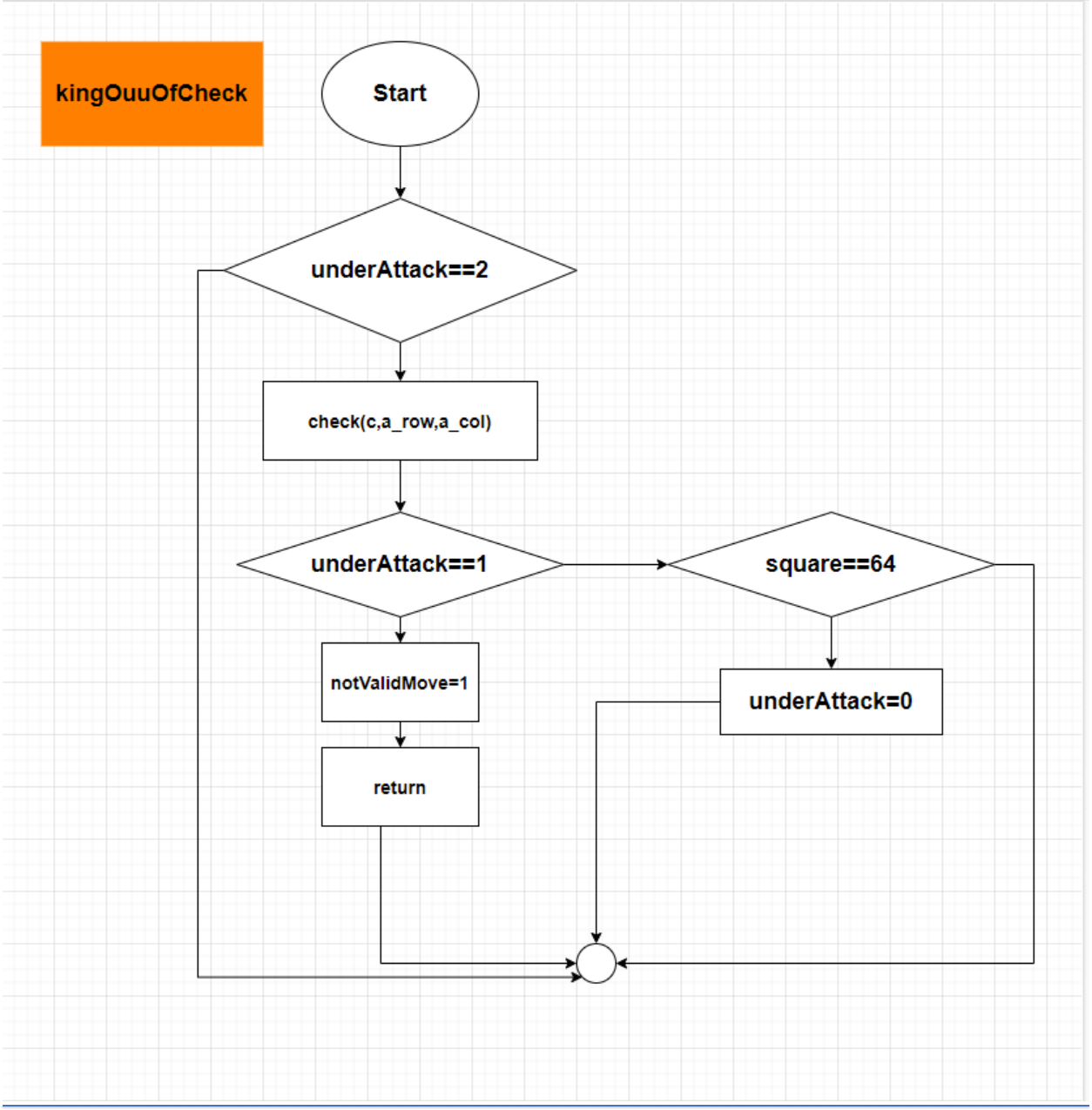

B
scan(fromColChar
A
breack
END
fromColChar=='N' ||fromColchar=='n'
F
fromColChar=='L' ||fromColchar=='l'
F
T
T
newGame()
loadFlage=1
readMoves()
continue
fromColChar=='U' ||fromColchar=='u'
F
tempMoves=fileMoves
turn=tempMoves
T
loadFlage=1
load(0)
F
tempMoves-numOfUndo>0
continue
T
reset( )
print (you can't undo more)
undo( )
turn--
continue
fromColChar=='U' ||fromColchar=='u'
F
T
print (you can't undo more)
reFlage=1
F
tempMoves-numOfUndo<tempMoves
T
reset( )
Redo( )
turn++
continue
C
D

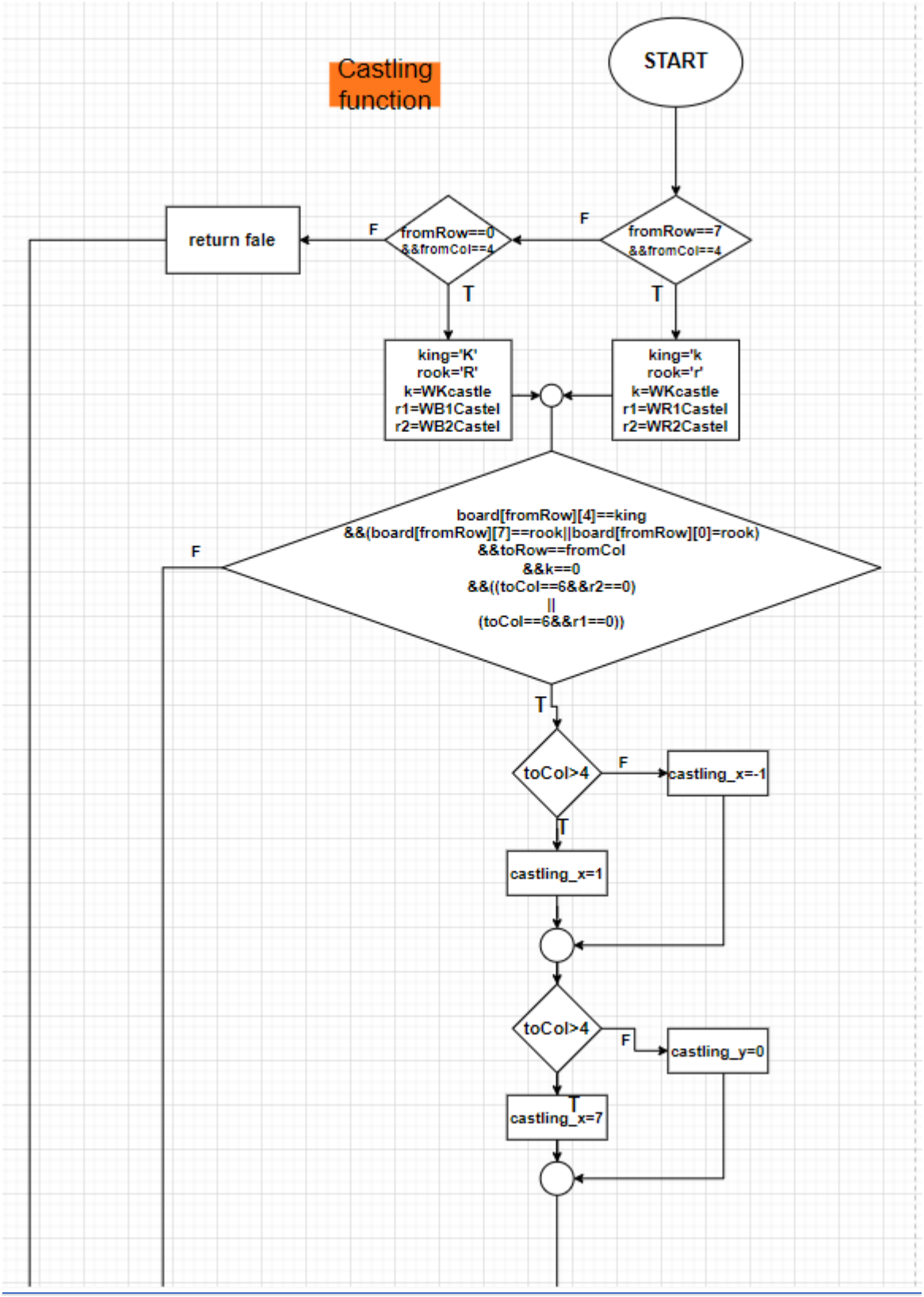

D
END
break
save( )
T
fromColChar=='S' ||fromColchar=='s'
C
F
break
T
fromColChar=='Q' ||fromColchar=='q'
F
scan (fromRow,toColchar,toRow)
fflush(stdio)
numOfUndo != 0
T
afterUndo()
numOfUndo=0
F
continue
print
(not valid input)
T
((fromRow<1||fromRow>8)||(toRow<1||toRow>8)
||(fromColchar<'a'||fromColchar>'h')||(toColchar<'a'||toColchar>'h'))
F
fromCol=fileInput(fromColchar)
toCol=fileInput(toColchar)
fromRow=8-fromRow
toRow=8-toRow
continue
print
(not valid)
T
(isLower(board[fromRow][fromCol])
&&
(turn%2 == 1))
F
continue
print
(not valid)
T
(isUpper(board[fromRow][fromCol])
&&
(turn%2 == 0))
F
continue
print
(not valid)
T
board[fromRow][fromCol]==EMPTY
F
E

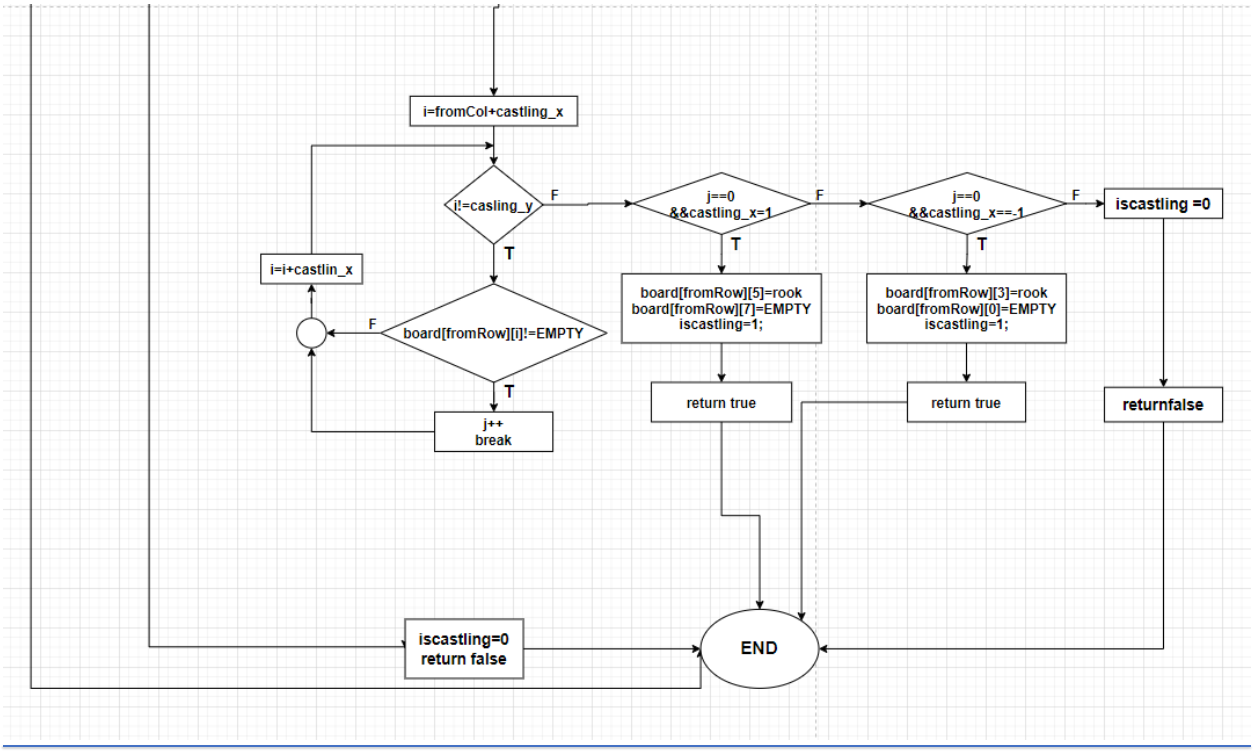

E
switchValidMove(fromRow,fromCol,toRow,toCol)
F
print
you can't do this move
T
F
((board[fromRow][fromCol]=='p')||(board[fromRow][fromCol]=='P'))
T
board[fromRow][fromCol] = promotion(board[fromRow][fromCol], toRow)
F
(IsTherePromotionn(board[fromRow][fromCol], toRow))
T
tempArray[turn][4]=board[fromRow][fromCol]
temp=board[toRow][toCol]
move(fromRow,fromCol,toRow,toCol)
pieceAttackTheKing(attacker)
uncheckedMove(board[toRow][toCol]
F
notValid==1
T
print
(white , black)king is ander attack
F

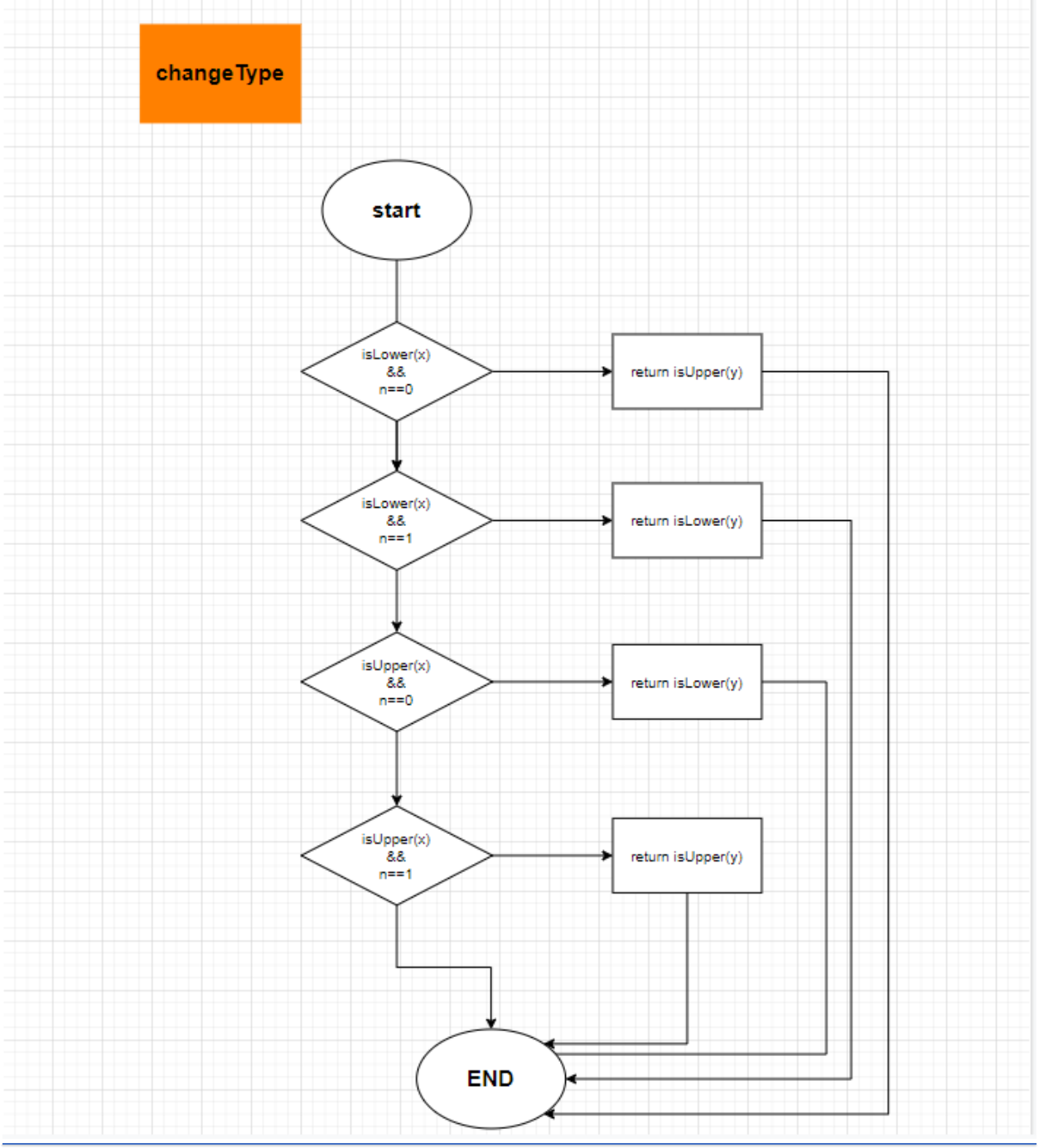

F
underAttack==1
F
T
move(toRow,toCol,fromRow,fromCol)
board[toRow][toCol]=temp
isCastling
F
T
move(fromRow,toCol-castling_x,toRow,castling_y)
continue
printCapturedPieces(toRow,toCol,temp)
attacker=' '
printBoardElements(board)
kingCastle(fromRow,fromCol,toRow,toCol)
rookCastle(fromRow,fromCol,toRow,toCol)
check(board[toRow][toCol],toRow,toCol)
underAttack==1
F
T
saveMoves(fromColchar,8-fromRow,toColchar,8-toRow)
print
(white , black)king is ander attack
attacker=board[toRow][toCol]
turn++
tempMoves++
isCastling=0

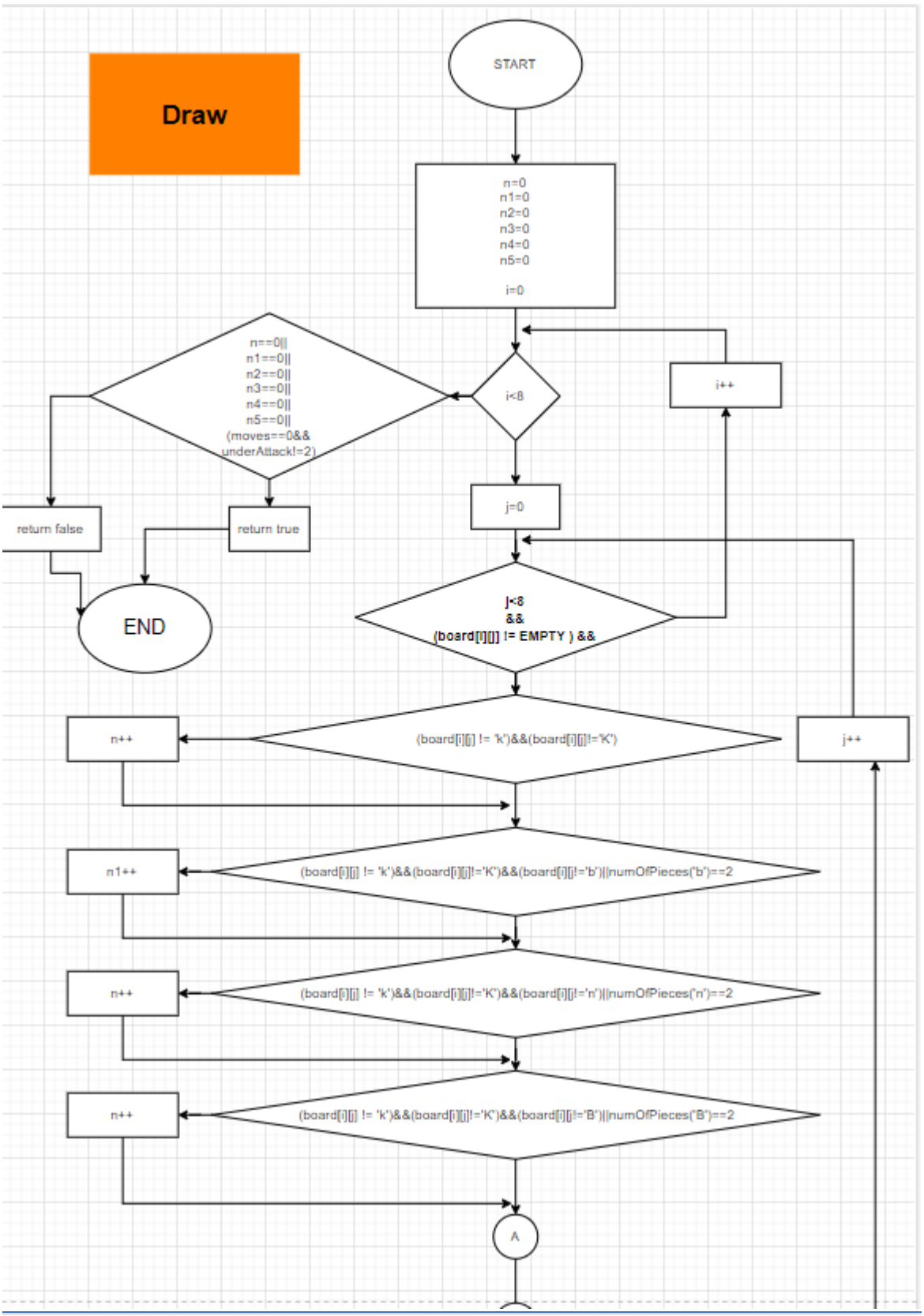

START
check
islower(board[toRow][toCol]
F
isUpper(board[toRow][toCol])
F
i=0
T
T
king='K'
queen='q'
rook='r'
bishop='b'
knight='n'
attackerking='k'
attacked='Black'
king='k'
queen='Q'
rook='R'
bishop='B'
knight='N'
attackerking='K'
attacked='White'
i<8
F
T
j=0
i++
j<8
T
F
(attacker==pawn)&&(board[i][j]==king)&&
(validCapturePawn(toRow,toCol,i,j))
F
j++
T
underAttack=1
END

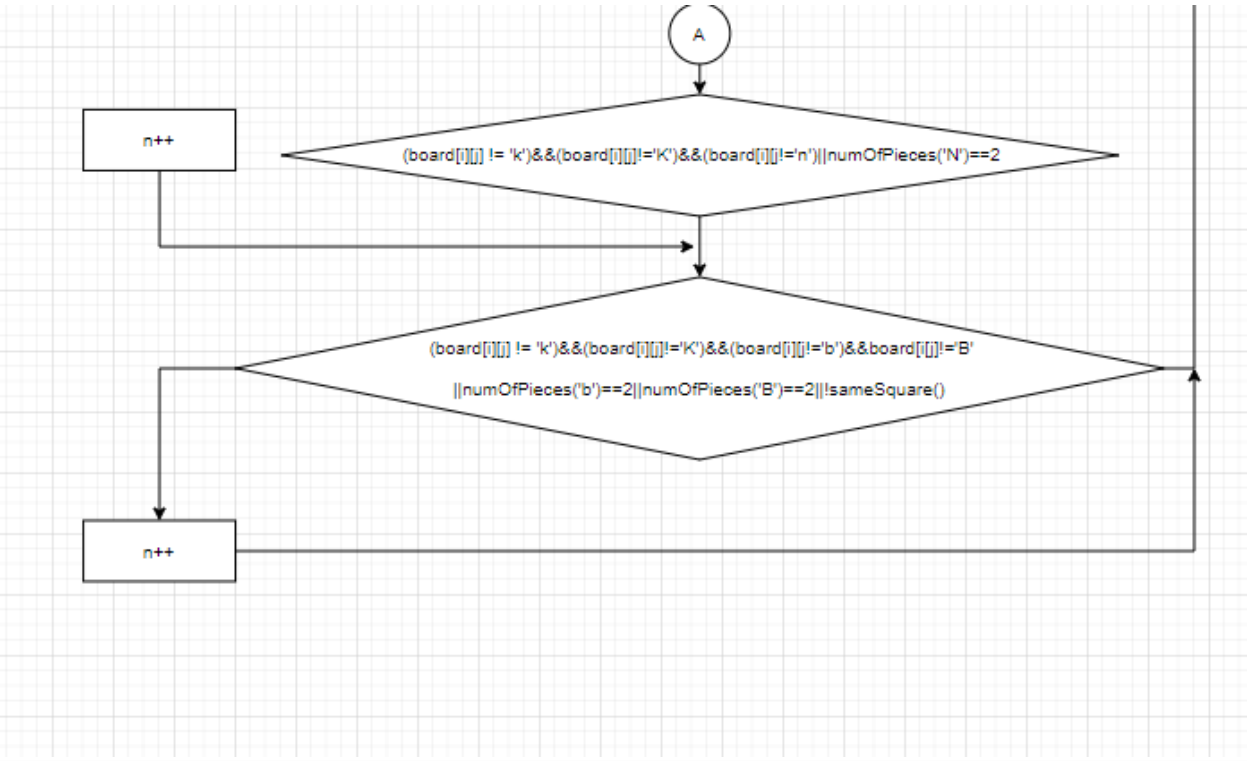

kingOuuOfCheck
Start
underAttack==2
check(c,a_row,a_col)
underAttack==1
square==64
notValidMove=1
underAttack=0
return

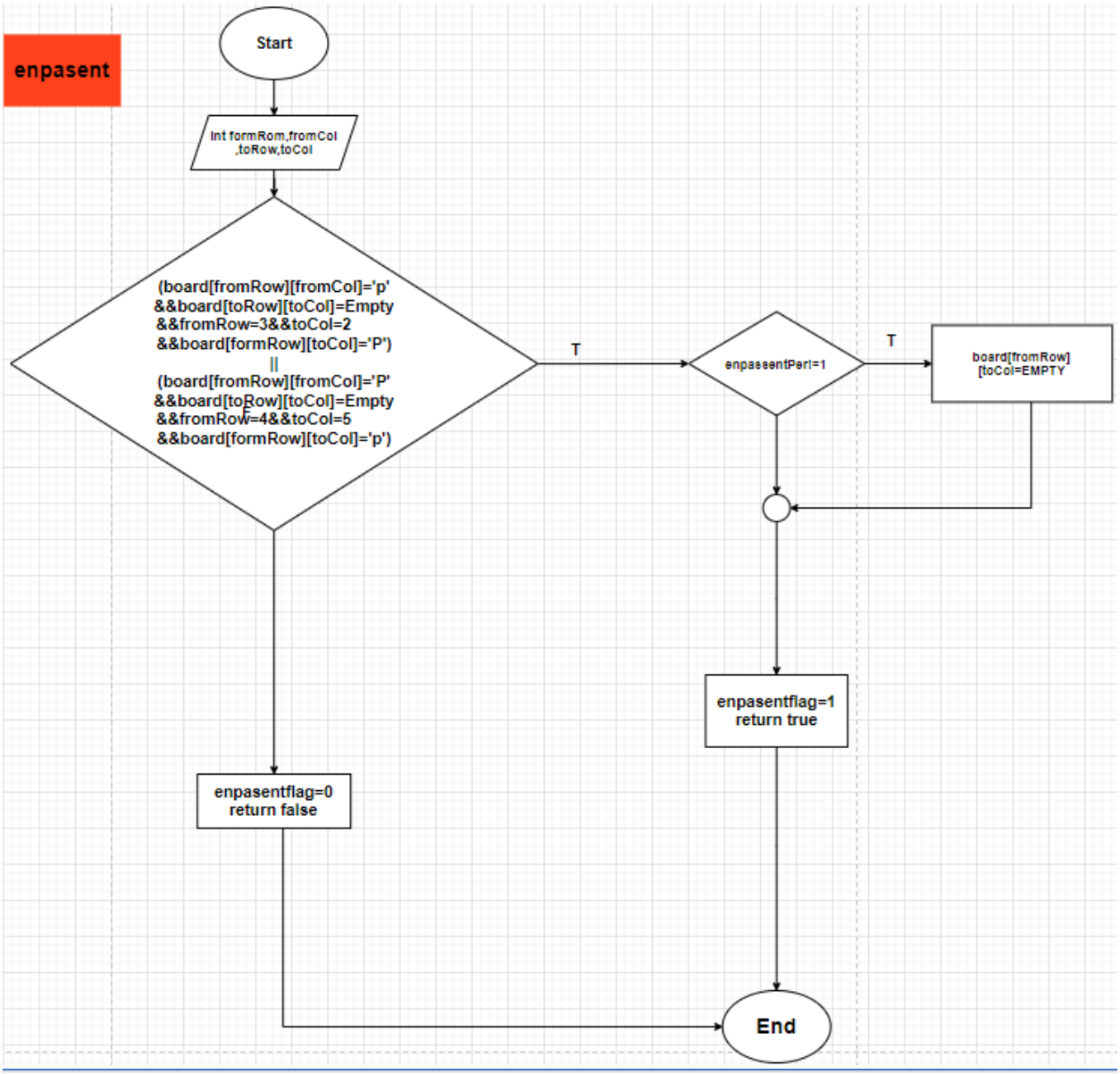

Castling function
START
fromRow==7 &&fromCol==4
F
fromRow==0 &&fromCol==4
F
return fale
T
T
king='k' rook='r' k=WKcastle r1=WR1Castel r2=WR2Castel
king='K' rook='R' k=WKcastle r1=WB1Castel r2=WB2Castel
board[fromRow][4]==king &&(board[fromRow][7]==rook||board[fromRow][0]=rook) &&toRow==fromCol &&k==0 &&((toCol==6&&r2==0) || (toCol==6&&r1==0))
F
T
toCol>4
F
castling_x=-1
T
castling_x=1
toCol>4
F
castling_y=0
T
castling_x=7

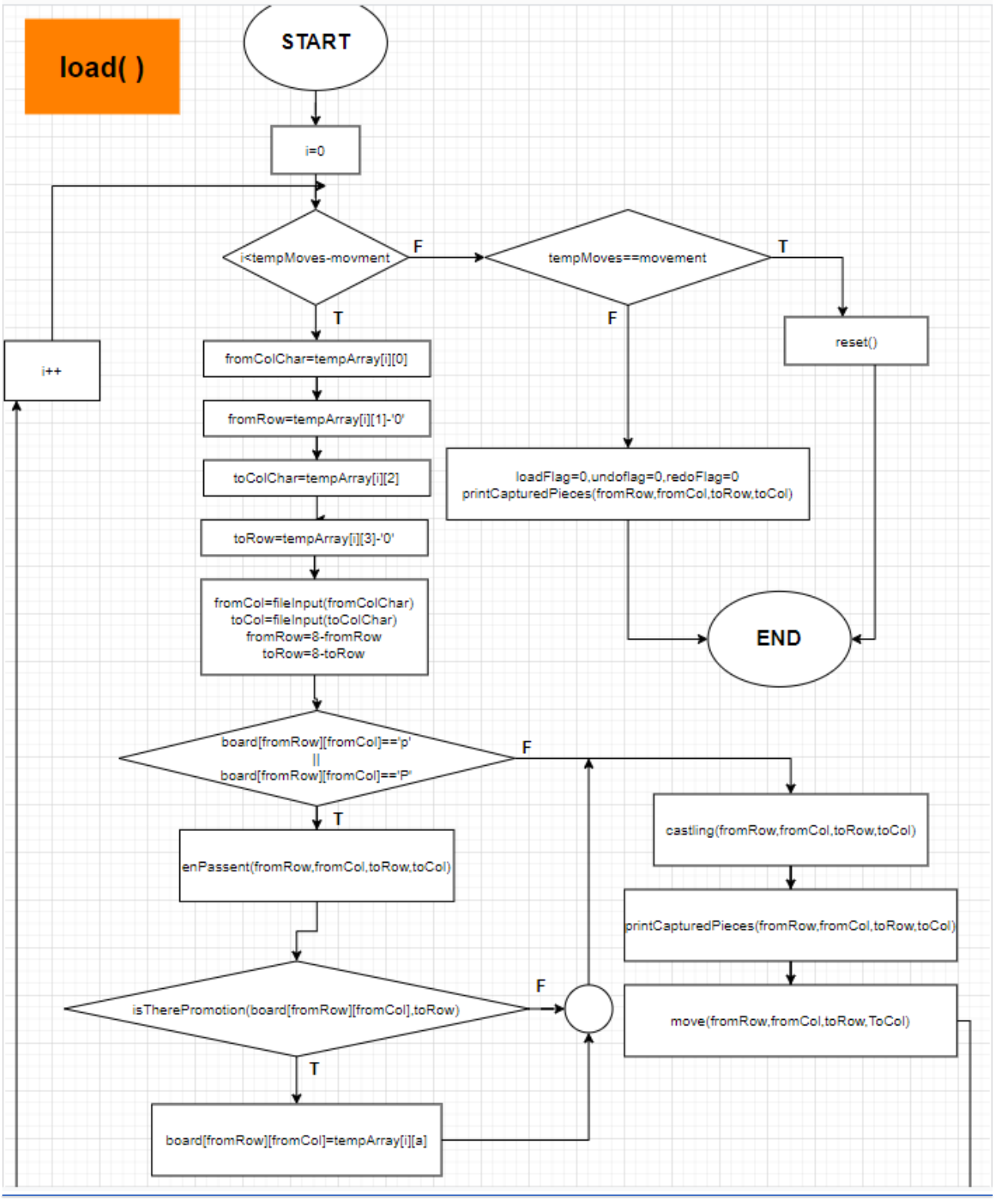

i=fromCol+castling_x
i!=casling_y
F
T
j==0
&&castling_x=1
F
T
j==0
&&castling_x==-1
F
T
iscastling =0
i=i+castlin_x
F
board[fromRow][i]!=EMPTY
T
j++
break
board[fromRow][5]=rook
board[fromRow][7]=EMPTY
iscastling=1;
board[fromRow][3]=rook
board[fromRow][0]=EMPTY
iscastling=1;
return true
return true
returnfalse
iscastling=0
return false
END

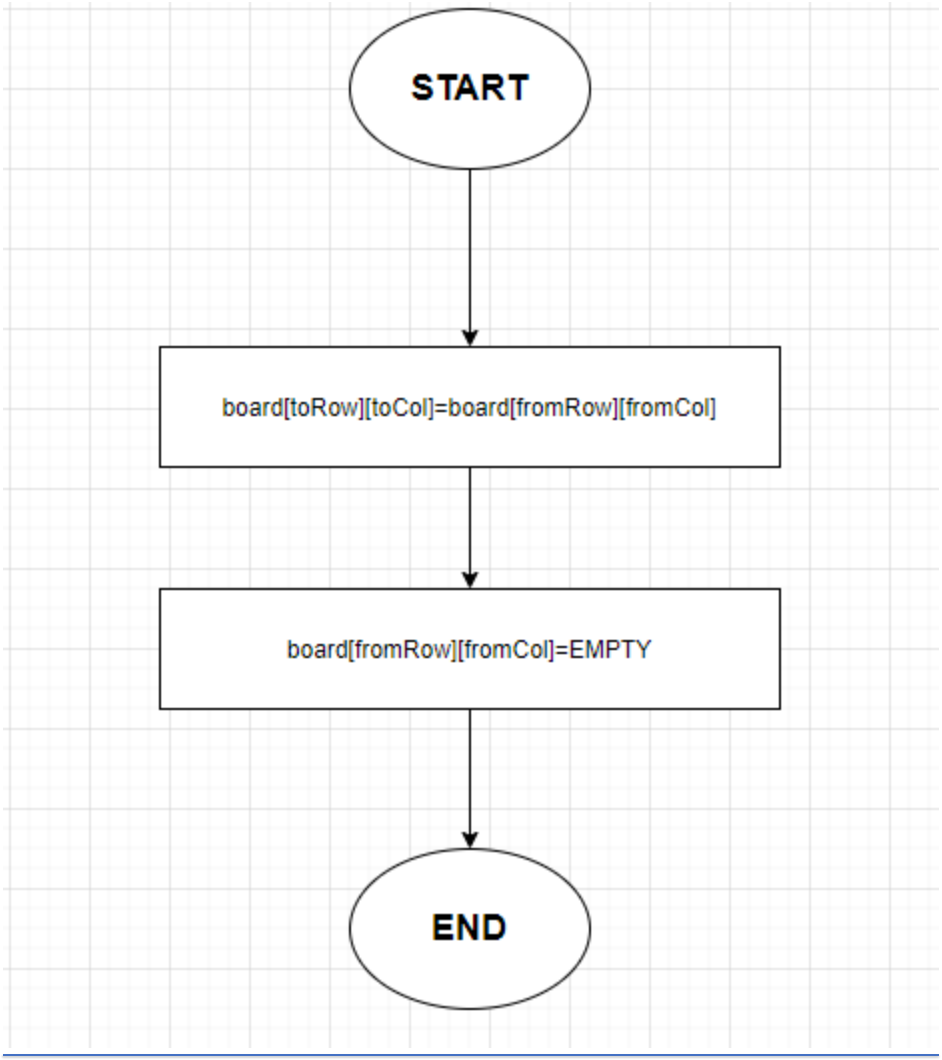

changeType
start
isLower(x)
&&
n==0
return isUpper(y)
isLower(x)
&&
n==1
return isLower(y)
isUpper(x)
&&
n==0
return isLower(y)
isUpper(x)
&&
n==1
return isUpper(y)
END

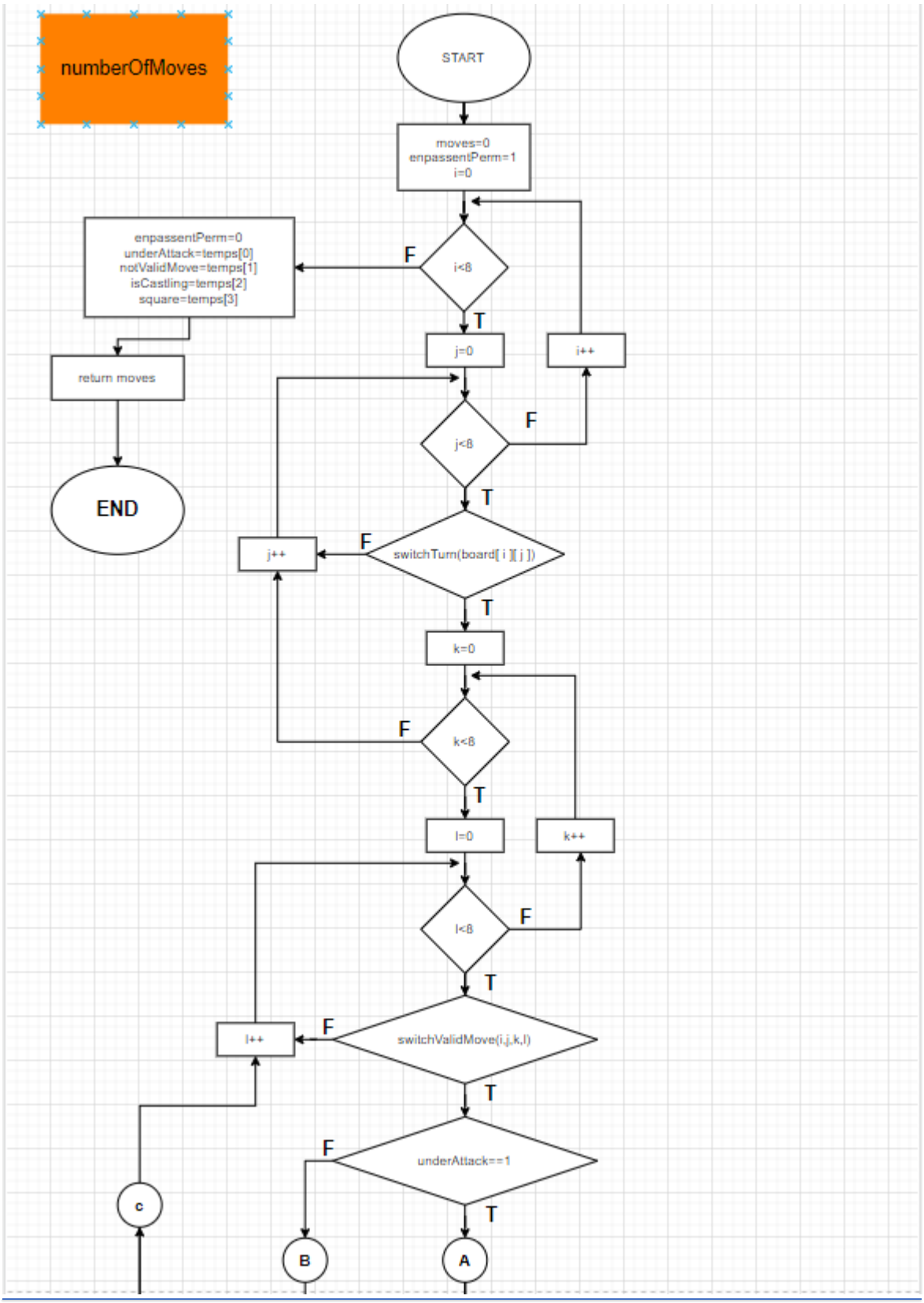

Draw
START
n=0
n1=0
n2=0
n3=0
n4=0
n5=0
i=0
i<8
i++
n==0||
n1==0||
n2==0||
n3==0||
n4==0||
n5==0||
(moves==0&&
underAttack!=2)
return false
return true
END
j=0
j<8
&&
(board[i][j] != EMPTY) &&
j++
(board[i][j] != 'k')&&(board[i][j]!='K')
n++
(board[i][j] != 'k')&&(board[i][j]!='K')&&(board[i][j!='b')||numOfPieces('b')==2
n1++
(board[i][j] != 'k')&&(board[i][j]!='K')&&(board[i][j!='n')||numOfPieces('n')==2
n++
(board[i][j] != 'k')&&(board[i][j]!='K')&&(board[i][j!='B')||numOfPieces('B')==2
n++
A

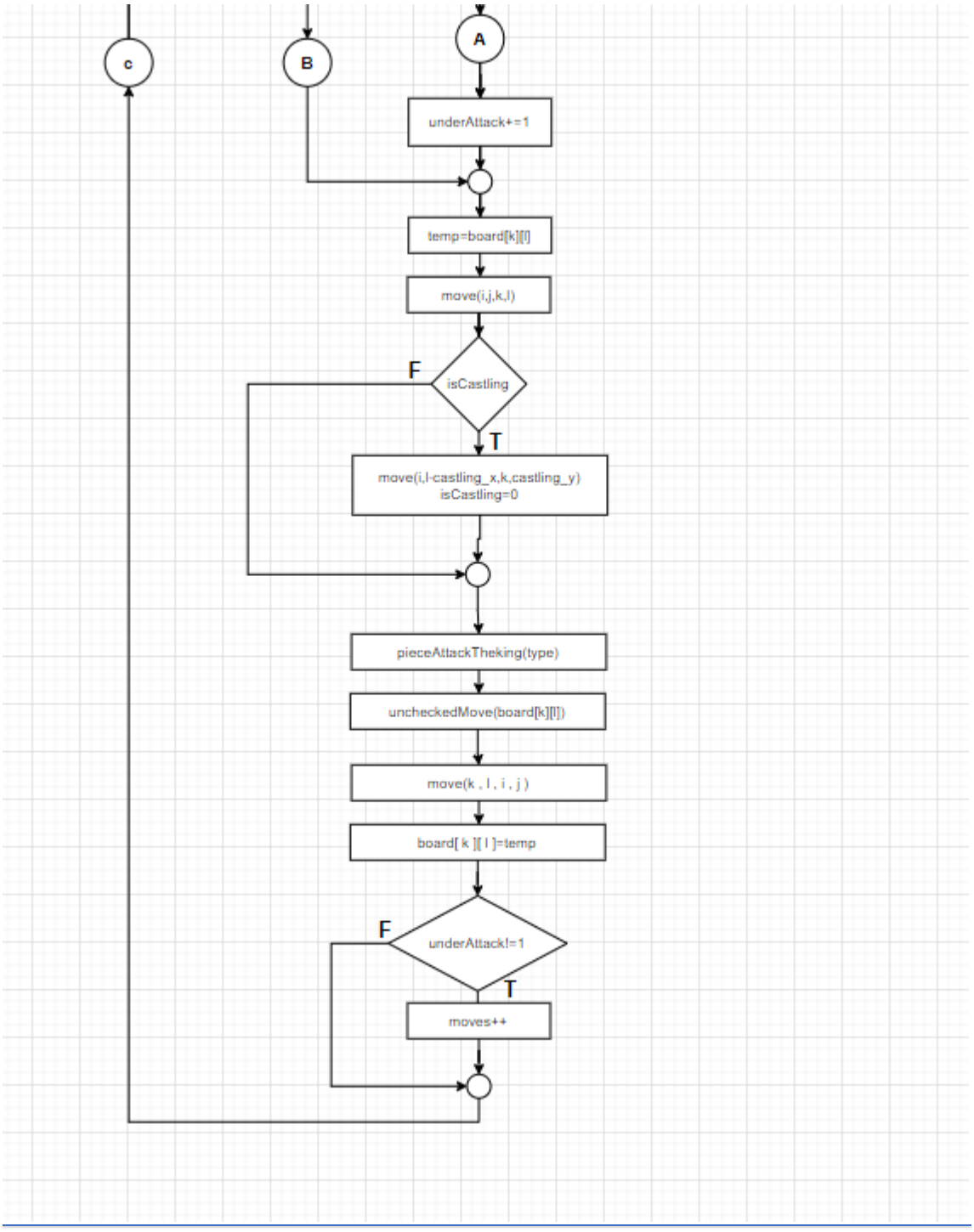

A
n++
(board[i][j] != 'k')&&(board[i][j]!='K')&&(board[i][j]!='n')||numOfPieces('N')==2
(board[i][j] != 'k')&&(board[i][j]!='K')&&(board[i][j]!='b')&&board[i[j]!='B'
||numOfPieces('b')==2||numOfPieces('B')==2||!sameSquare()
n++

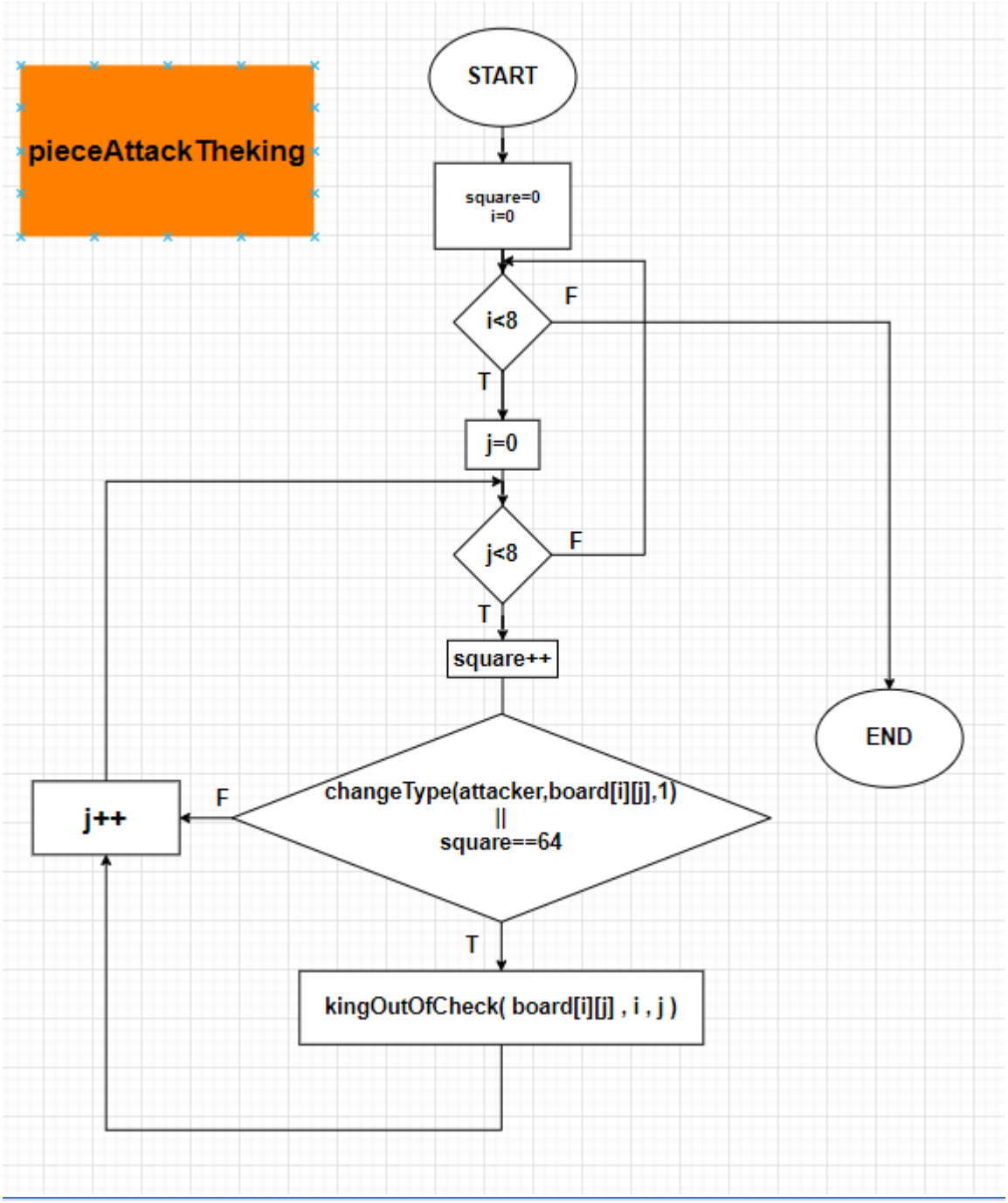

enpasent
Start
Int formRom,fromCol
,toRow,toCol
(board[fromRow][fromCol]='p'
&&board[toRow][toCol]=Empty
&&fromRow=3&&toCol=2
&&board[formRow][toCol]='P')
||
(board[fromRow][fromCol]='P'
&&board[toRow][toCol]=Empty
&&fromRow=4&&toCol=5
&&board[formRow][toCol]='p')
T
enpassentPerI=1
T
board[fromRow]
[toCol=EMPTY
enpasentflag=1
return true
enpasentflag=0
return false
End

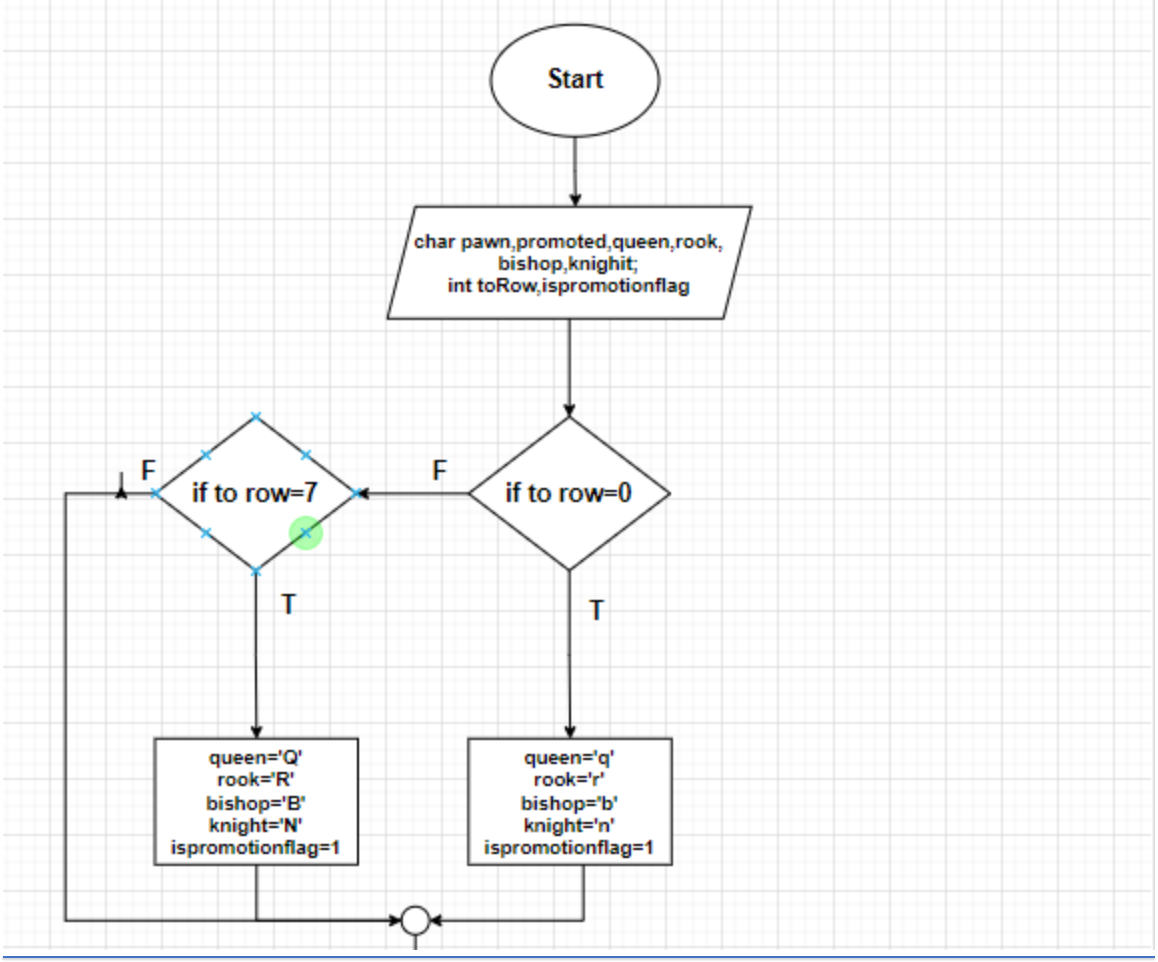

load( )
START
i=0
i<tempMoves-movment
F
T
tempMoves==movement
T
F
reset()
i++
fromColChar=tempArray[i][0]
fromRow=tempArray[i][1]-'0'
toColChar=tempArray[i][2]
toRow=tempArray[i][3]-'0'
fromCol=fileInput(fromColChar)
toCol=fileInput(toColChar)
fromRow=8-fromRow
toRow=8-toRow
loadFlag=0,undoflag=0,redoFlag=0
printCapturedPieces(fromRow,fromCol,toRow,toCol)
END
board[fromRow][fromCol]=='p'
||
board[fromRow][fromCol]=='P'
F
T
enPassent(fromRow,fromCol,toRow,toCol)
castling(fromRow,fromCol,toRow,toCol)
printCapturedPieces(fromRow,fromCol,toRow,toCol)
isTherePromotion(board[fromRow][fromCol],toRow)
F
T
move(fromRow,fromCol,toRow,ToCol)
board[fromRow][fromCol]=tempArray[i][a]

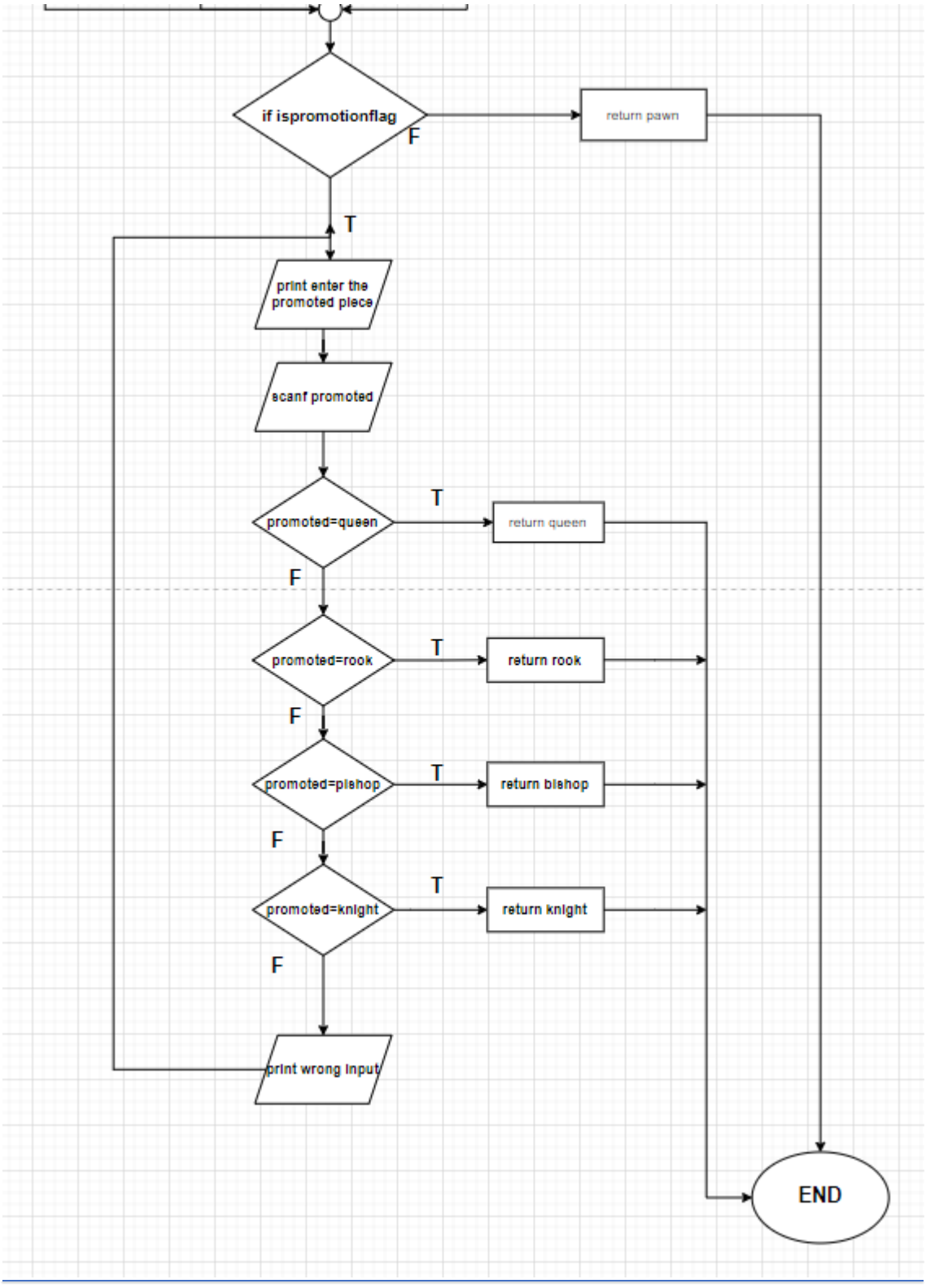

START
board[toRow][toCol]=board[fromRow][fromCol]
board[fromRow][fromCol]=EMPTY
END

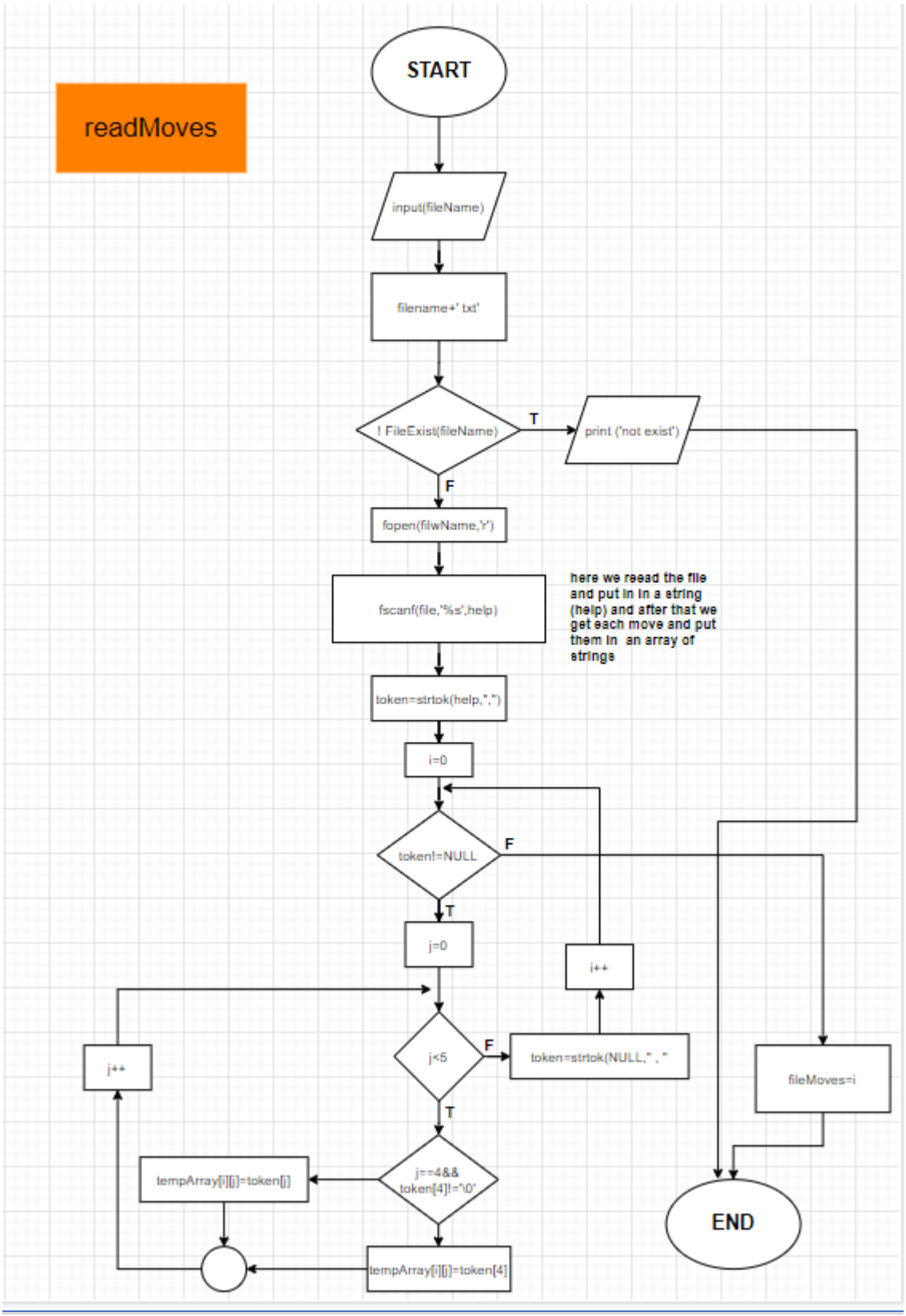

numberOfMoves
START
moves=0
enpassentPerm=1
i=0
i<8
F
T
enpassentPerm=0
underAttack=temps[0]
notValidMove=temps[1]
isCastling=temps[2]
square=temps[3]
return moves
END
j=0
i++
j<8
F
T
j++
F
switchTurn(board[i][j])
T
k=0
k<8
F
T
l=0
k++
l<8
F
T
l++
F
switchValidMove(i,j,k,l)
T
underAttack==1
F
T
c
B
A

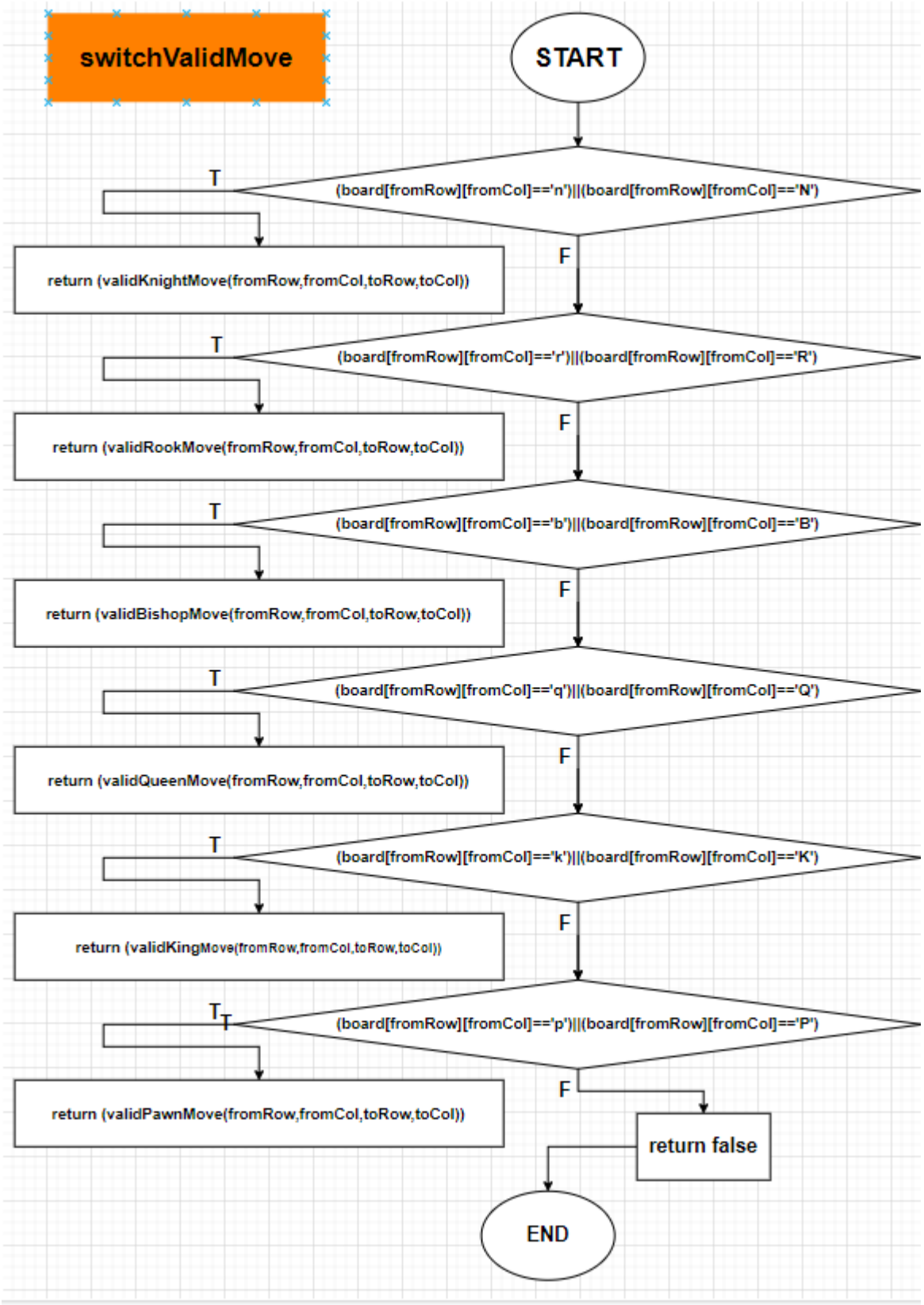

c
B
A
underAttack+=1
temp=board[k][l]
move(i,j,k,l)
F
isCastling
T
move(i,l-castling_x,k,castling_y)
isCastling=0
pieceAttackTheking(type)
uncheckedMove(board[k][l])
move(k , l , i , j)
board[k][l]=temp
F
underAttack!=1
T
moves++

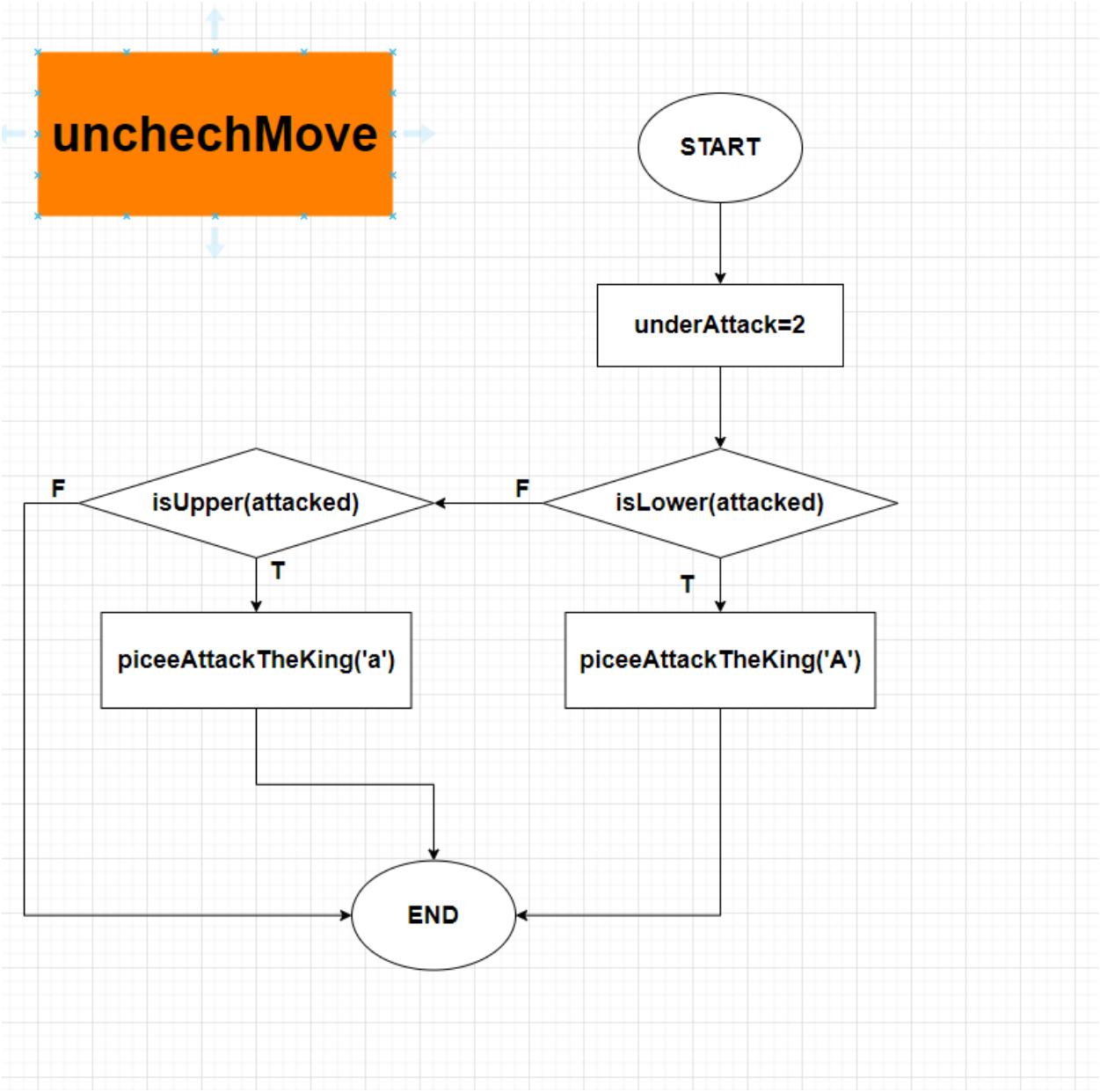

pieceAttackTheking
START
square=0
i=0
i<8
F
T
j=0
j<8
F
T
square++
END
changeType(attacker,board[i][j],1)
||
square==64
F
j++
T
kingOutOfCheck(board[i][j] , i , j)

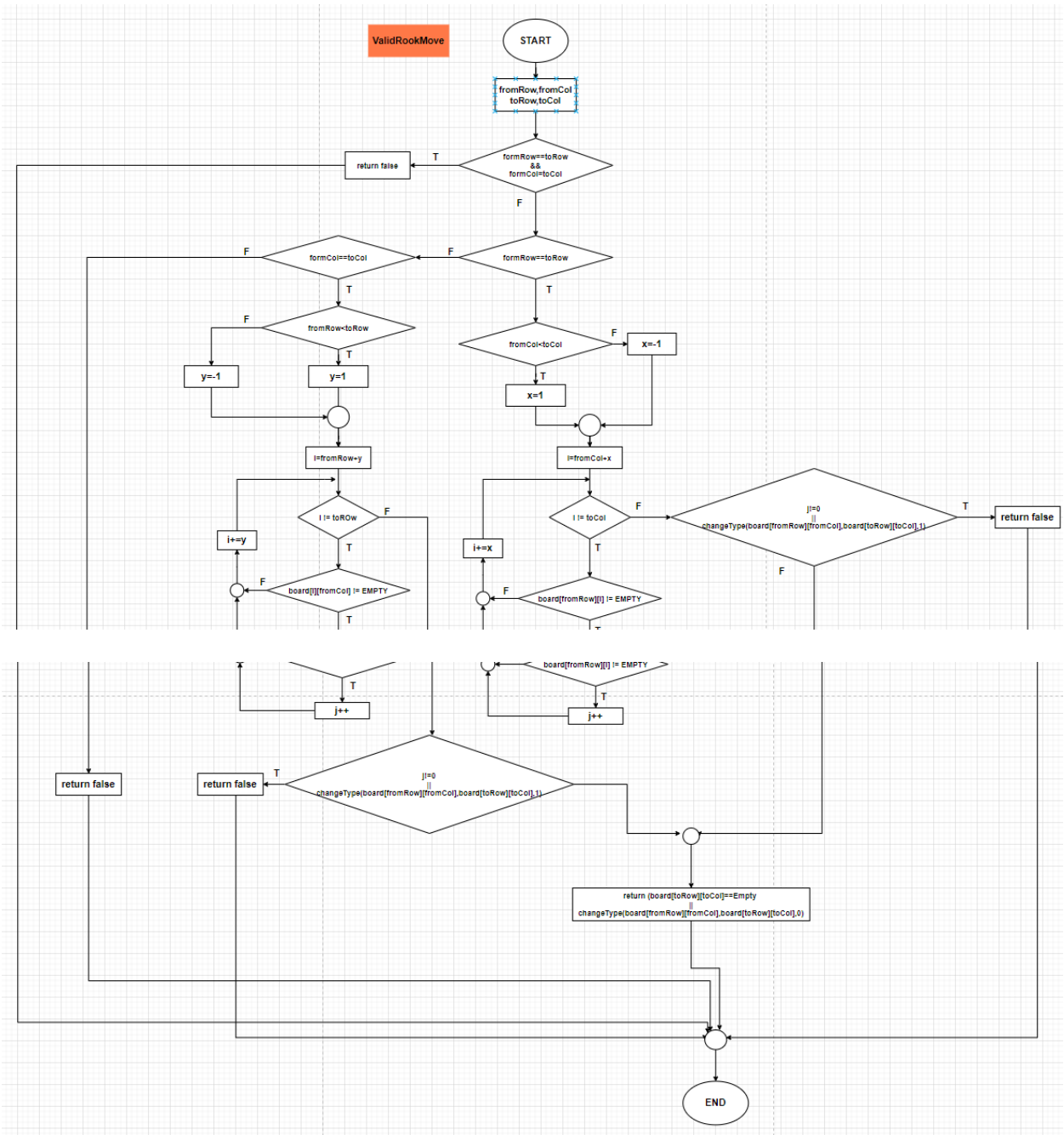

Start
char pawn,promoted,queen,rook,
bishop,knighit;
int toRow,ispromotionflag
if to row=0
F
if to row=7
F
T
T
queen='q'
rook='r'
bishop='b'
knight='n'
ispromotionflag=1
queen='Q'
rook='R'
bishop='B'
knight='N'
ispromotionflag=1

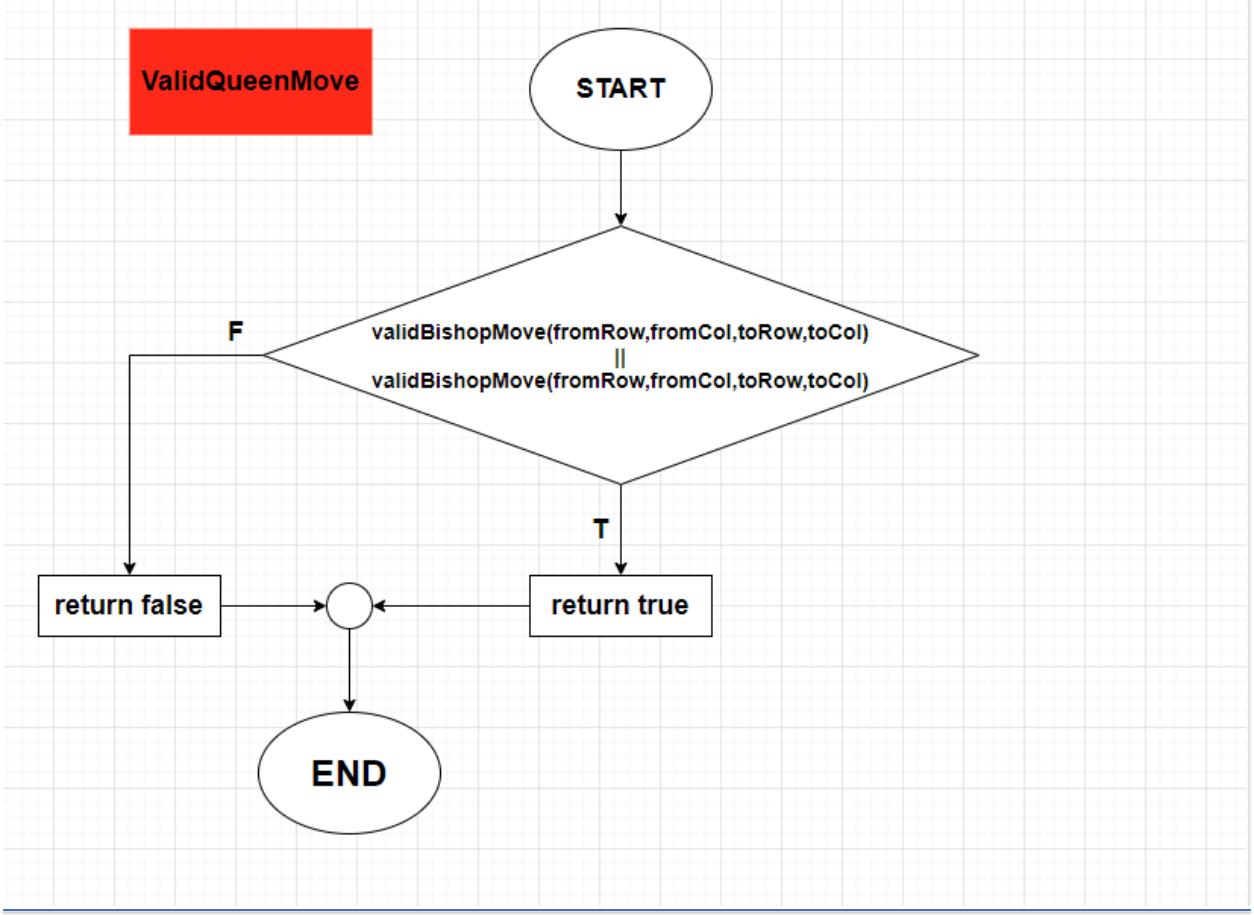

if ispromotionflag
F
return pawn
T
print enter the promoted piece
scanf promoted
promoted=queen
T
return queen
F
promoted=rook
T
return rook
F
promoted=pishop
T
return bishop
F
promoted=knight
T
return knight
F
print wrong input
END

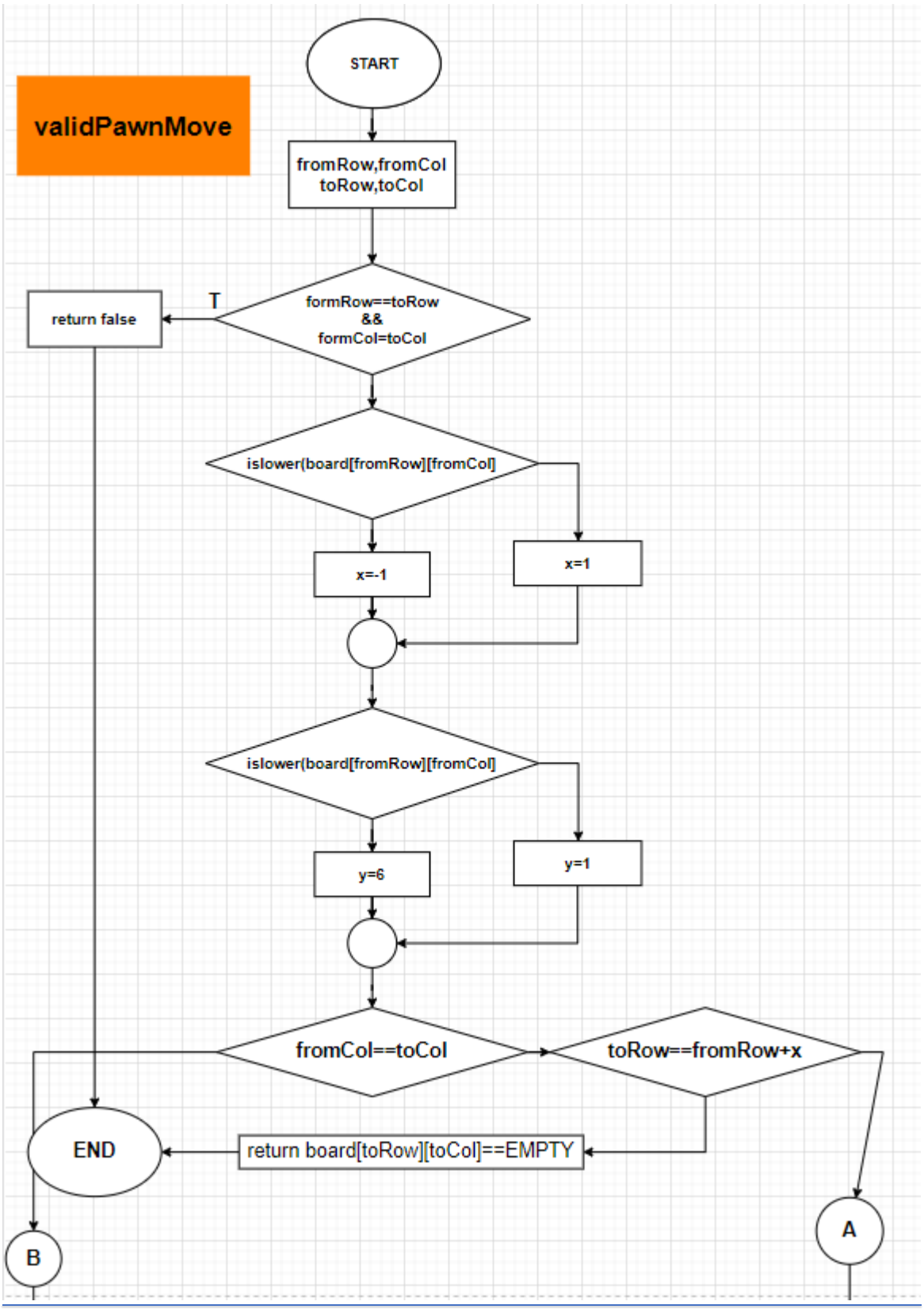

readMoves
START
input(fileName)
filename+' txt'
! FileExist(fileName)
T
print ('not exist')
F
fopen(filwName,'r')
fscanf(file,'%s',help)
here we reead the file and put in in a string (help) and after that we get each move and put them in an array of stringe
token=strtok(help,",")
i=0
token!=NULL
F
T
j=0
j++
j<5
F
token=strtok(NULL," , "
fileMoves=i
T
j++
tempArray[i][j]=token[j]
j==4&& token[4]!='\0'
tempArray[i][j]=token[4]
END

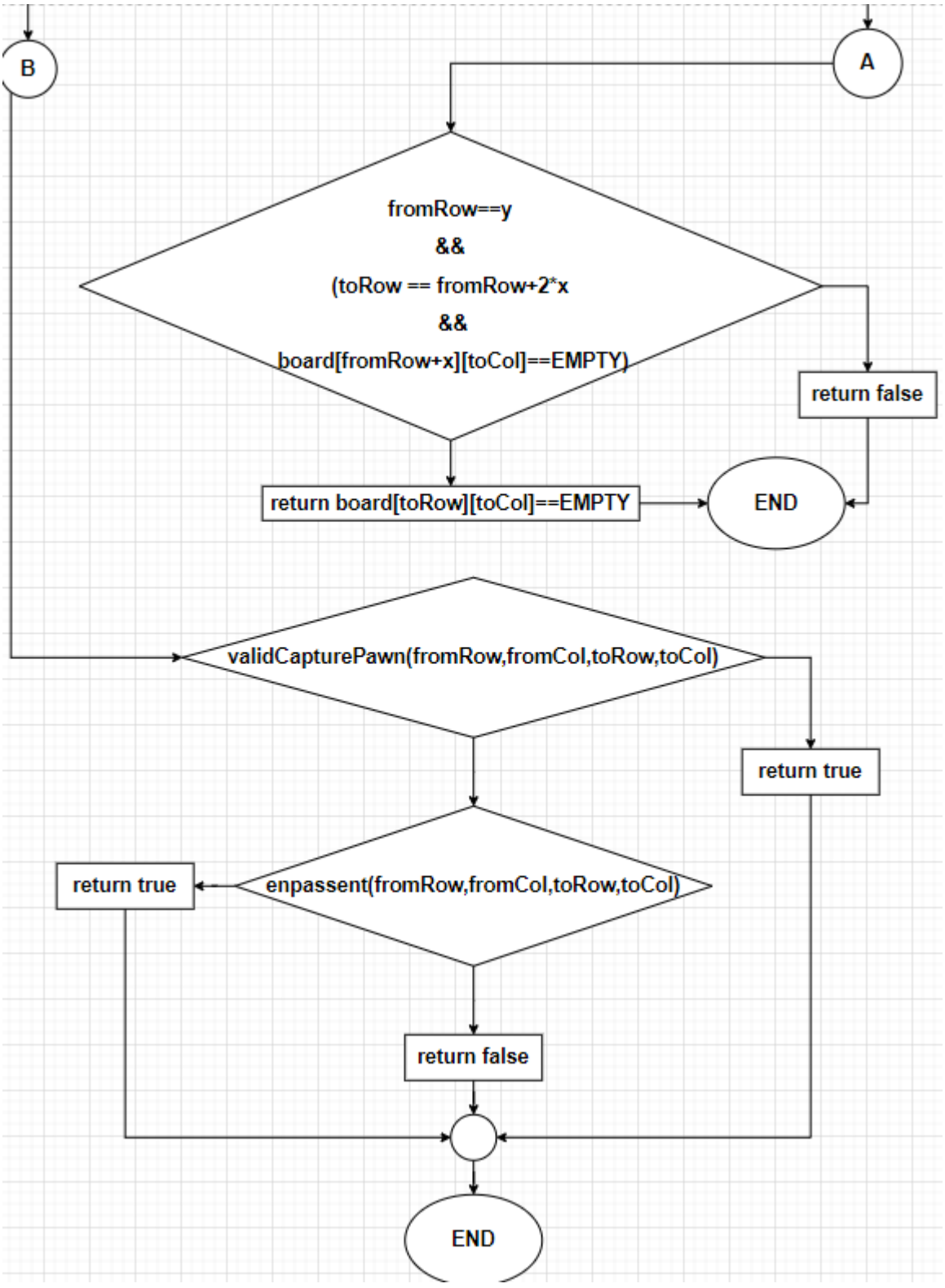

switchValidMove
START
(board[fromRow][fromCol]=='n')||(board[fromRow][fromCol]=='N')
T
F
return (validKnightMove(fromRow,fromCol,toRow,toCol))
(board[fromRow][fromCol]=='r')||(board[fromRow][fromCol]=='R')
T
F
return (validRookMove(fromRow,fromCol,toRow,toCol))
(board[fromRow][fromCol]=='b')||(board[fromRow][fromCol]=='B')
T
F
return (validBishopMove(fromRow,fromCol,toRow,toCol))
(board[fromRow][fromCol]=='q')||(board[fromRow][fromCol]=='Q')
T
F
return (validQueenMove(fromRow,fromCol,toRow,toCol))
(board[fromRow][fromCol]=='k')||(board[fromRow][fromCol]=='K')
T
F
return (validKingMove(fromRow,fromCol,toRow,toCol))
(board[fromRow][fromCol]=='p')||(board[fromRow][fromCol]=='P')
T
T
F
return (validPawnMove(fromRow,fromCol,toRow,toCol))
return false
END

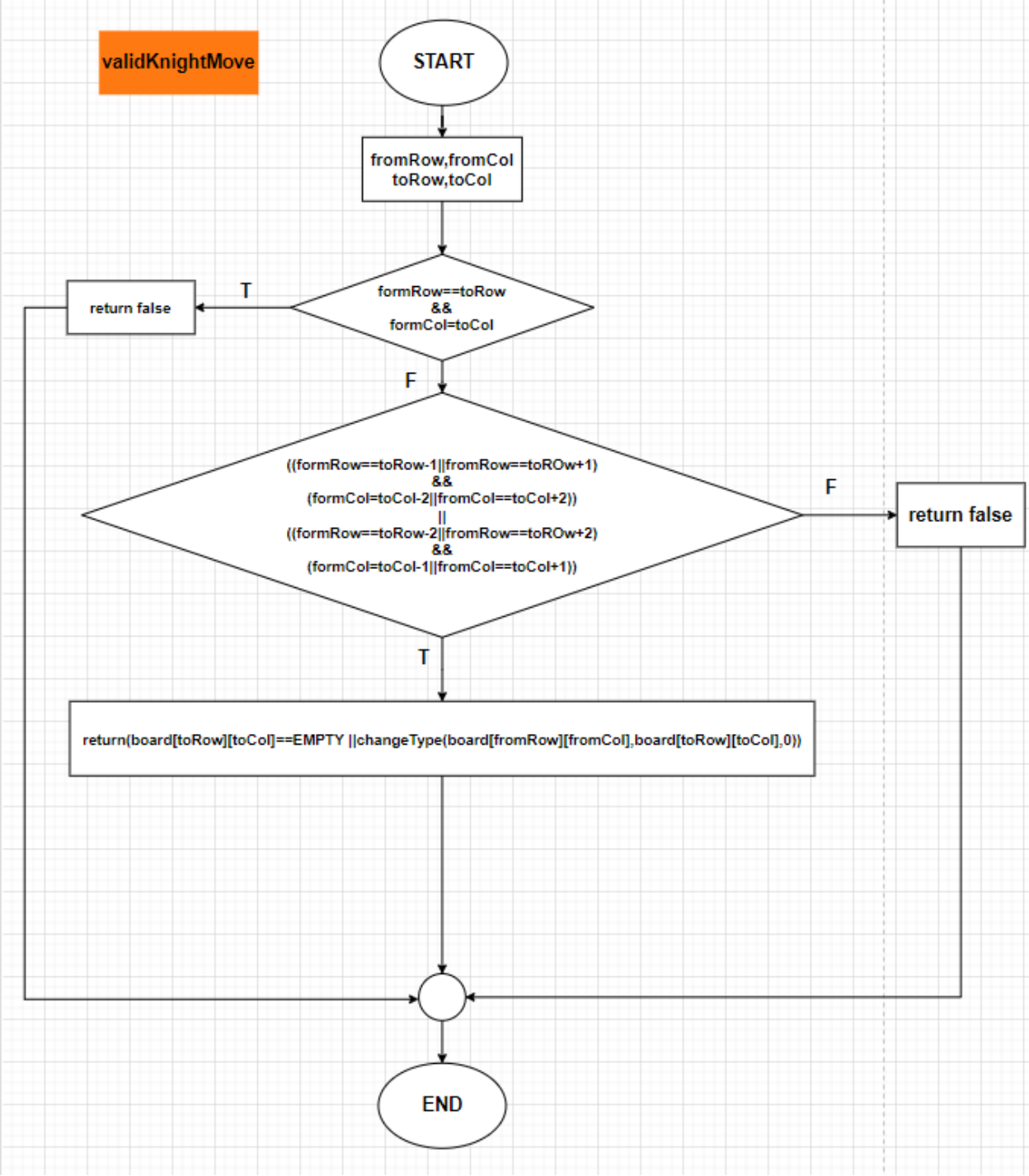

unchechMove
START
underAttack=2
isLower(attacked)
F
T
isUpper(attacked)
F
T
piceeAttackTheKing('A')
piceeAttackTheKing('a')
END


ValidRookMove
START
fromRow,fromCol
toRow,toCol
formRow==toRow
&&
formCol=toCol
T
return false
F
formRow==toRow
F
formCol==toCol
F
T
T
fromRow<toRow
fromCol<toCol
F
x=-1
T
T
y=-1
y=1
x=1
i=fromRow+y
j=fromCol+x
i != toROw
F
i != toCol
F
j!=0
||
changeType(board[fromRow][fromCol],board[toRow][toCol],1)
T
return false
T
i+=y
i+=x
T
T
F
board[i][fromCol] != EMPTY
F
board[fromRow][i] != EMPTY
T
T
j++
j++
return false
return false
T
j!=0
||
changeType(board[fromRow][fromCol],board[toRow][toCol],1)
return (board[toRow][toCol]==Empty
||
changeType(board[fromRow][fromCol],board[toRow][toCol],0)
END

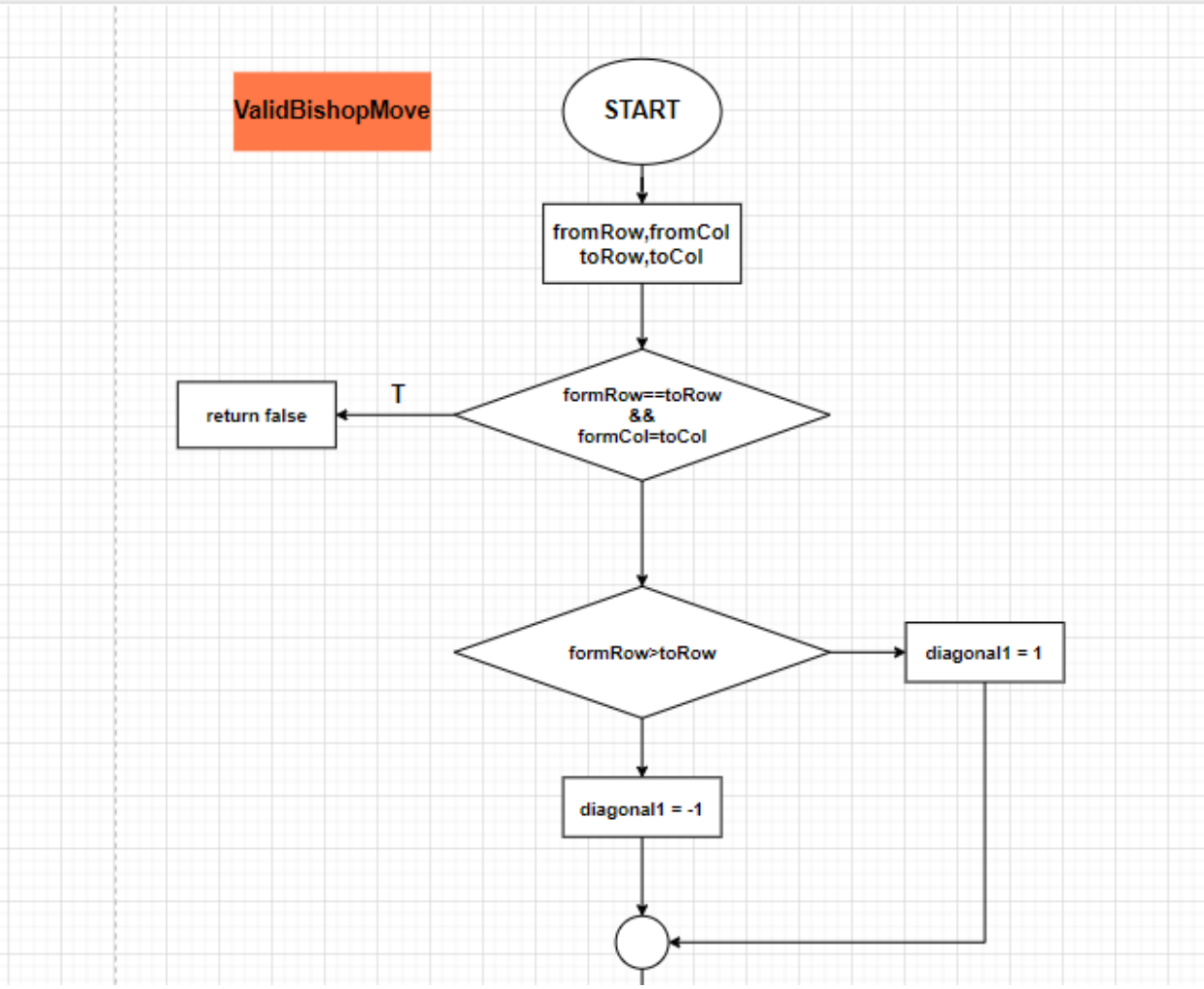

ValidQueenMove
START
validBishopMove(fromRow,fromCol,toRow,toCol)
||
validBishopMove(fromRow,fromCol,toRow,toCol)
F
T
return false
return true
END

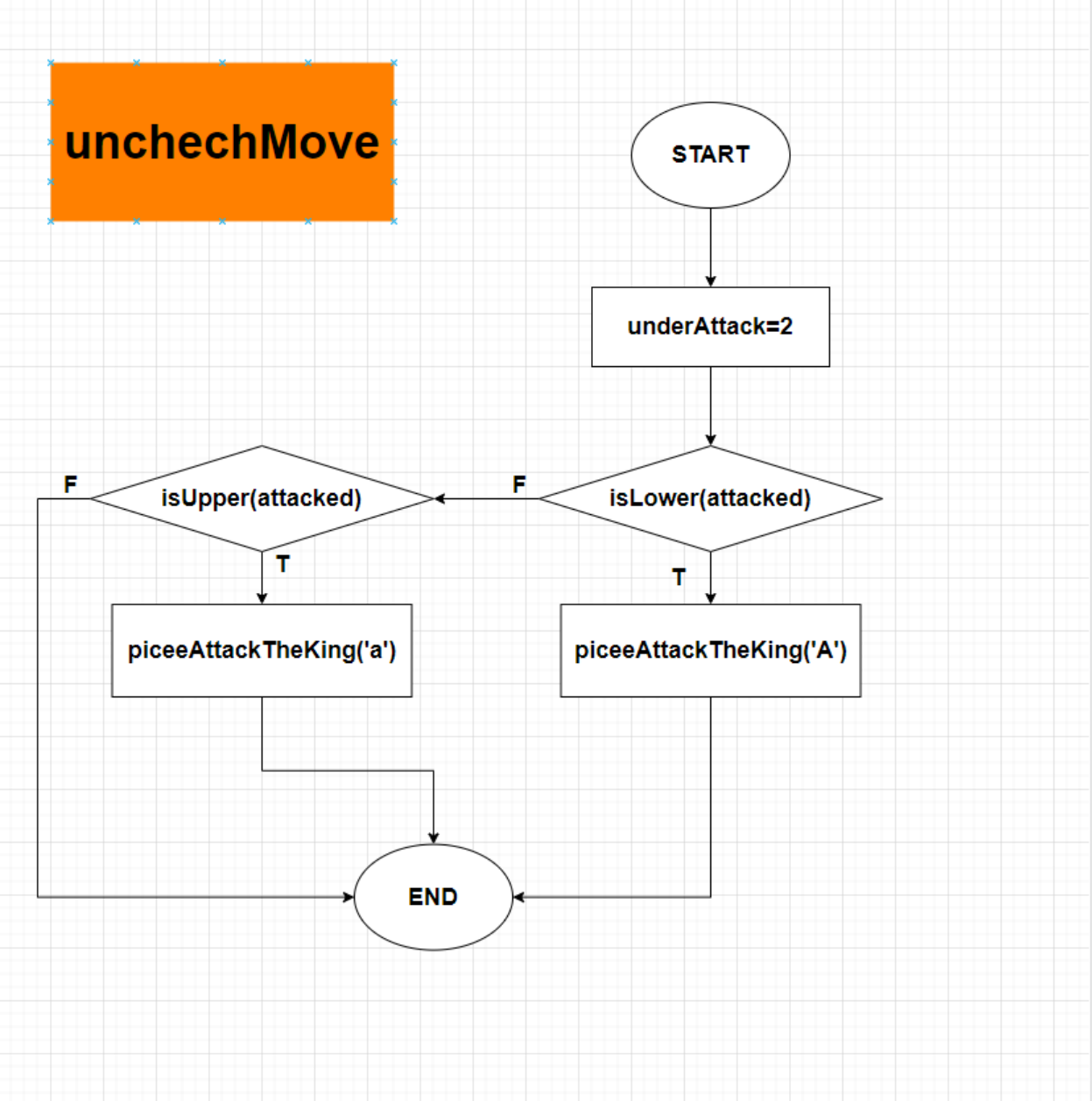

validPawnMove
START
fromRow,fromCol
toRow,toCol
formRow==toRow
&&
formCol=toCol
T
return false
islower(board[fromRow][fromCol]
x=-1
x=1
islower(board[fromRow][fromCol]
y=6
y=1
fromCol==toCol
toRow==fromRow+x
END
return board[toRow][toCol]==EMPTY
B
A


B
A
fromRow==y
&&
(toRow == fromRow+2*x
&&
board[fromRow+x][toCol]==EMPTY)
return false
return board[toRow][toCol]==EMPTY
END
validCapturePawn(fromRow,fromCol,toRow,toCol)
return true
return true
enpassent(fromRow,fromCol,toRow,toCol)
return false
END


validKnightMove
START
fromRow,fromCol
toRow,toCol
formRow==toRow
&&
formCol=toCol
T
return false
F
((formRow==toRow-1||fromRow==toROw+1)
&&
(formCol=toCol-2||fromCol==toCol+2))
||
((formRow==toRow-2||fromRow==toROw+2)
&&
(formCol=toCol-1||fromCol==toCol+1))
F
return false
T
return(board[toRow][toCol]==EMPTY ||changeType(board[fromRow][fromCol],board[toRow][toCol],0))
END


validCapturePawn
START
(((fromRow -1==toRow)&&(fromCol+1==toCol))&&
(IsUpper(board[toRow][toCol]))&&(IsLower(board[fromRow][fromCol])))
T
F
return true
(((fromRow -1==toRow)&&(fromCol-1==toCol))&&
(IsUpper(board[toRow][toCol]))&&(IsLower(board[fromRow][fromCol])))
T
F
return true
(((fromRow +1==toRow)&&(fromCol+1==toCol))&&
(IsLower(board[toRow][toCol]))&&IsUpper(board[fromRow][fromCol]))
T
F
return true
(((fromRow +1==toRow)&&(fromCol-1==toCol))&&
(IsLower(board[toRow][toCol]))&&(IsUpper(board[fromRow][fromCol])))
T
F
return true
return false
END


ValidBishopMove
START
fromRow,fromCol
toRow,toCol
formRow==toRow
&&
formCol=toCol
T
return false
formRow>toRow
diagonal1 = 1
diagonal1 = -1


unchechMove
START
underAttack=2
isLower(attacked)
F
T
isUpper(attacked)
F
T
piceeAttackTheKing('A')
piceeAttackTheKing('a')
END

User manual

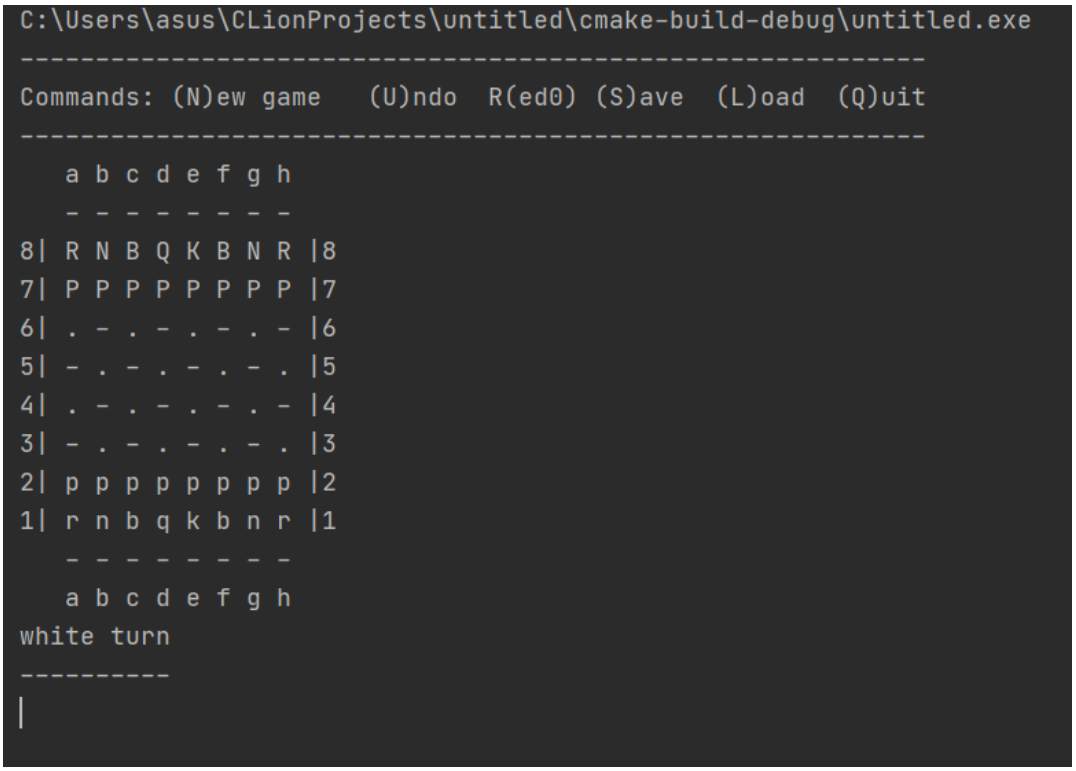
The Command line supports new game, loading game, saving, undo, redo, and the move.

- Press N or n for a new game.
- Press L or l for loading an existing game, the program will prompt you to Enter the file to be loaded.
- Press S or s for saving the game, the program will prompt you to Enter the file to be saved.
- Press U or u to undo one movement and you can undo till the first move.
- Press R or r to redo one movement, you can redo till the last game.

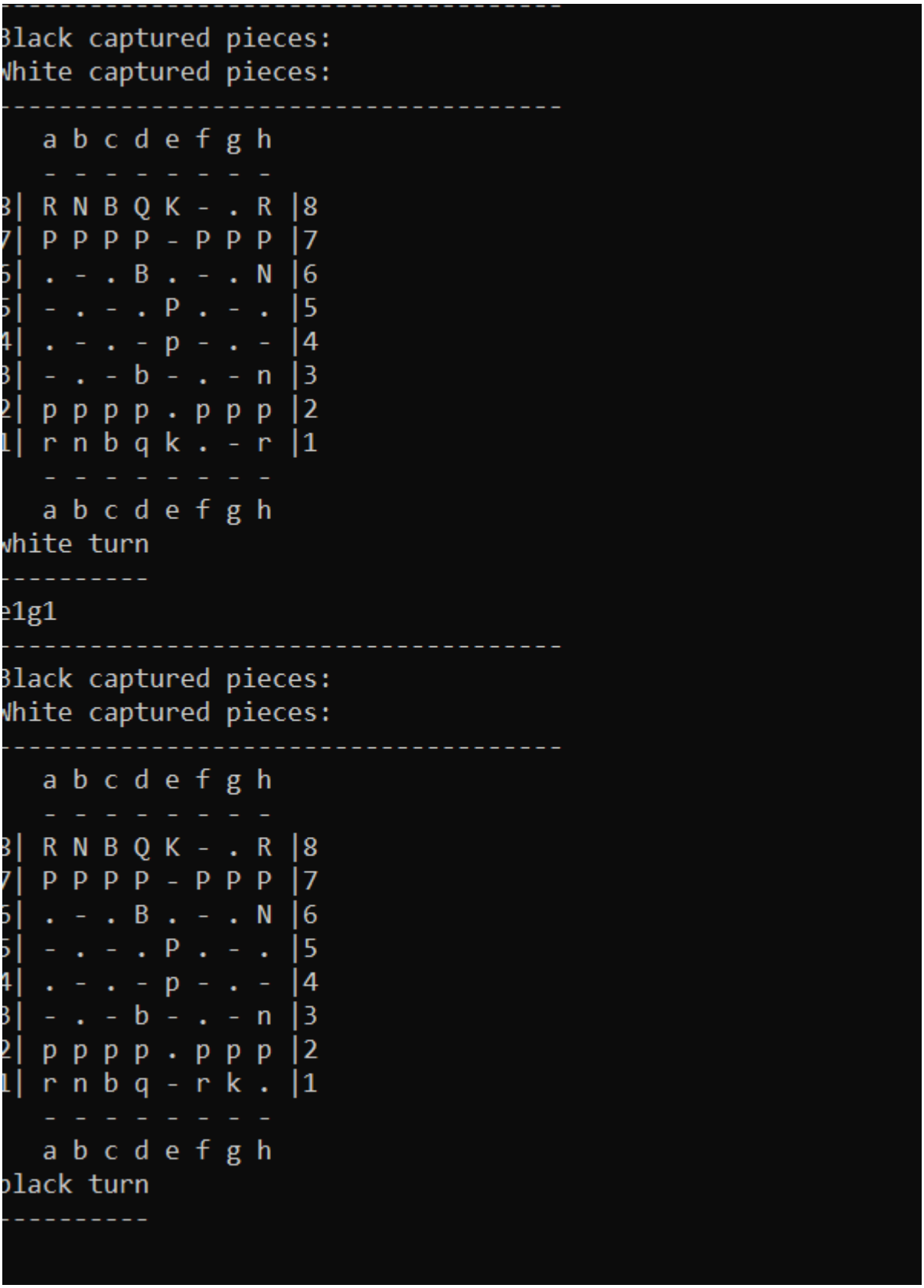
For moving the pieces Enter the index of the piece you want to move, followed by the index of the square you want to move it to.
(e.g., A3B4 or a3b4 will move the piece at A3 to the square B4).
Case of promotion the program will prompt you to enter the letter of the piece to be promoted, and if you enter wrong piece the program will ask you again.

Sample runs

The program start

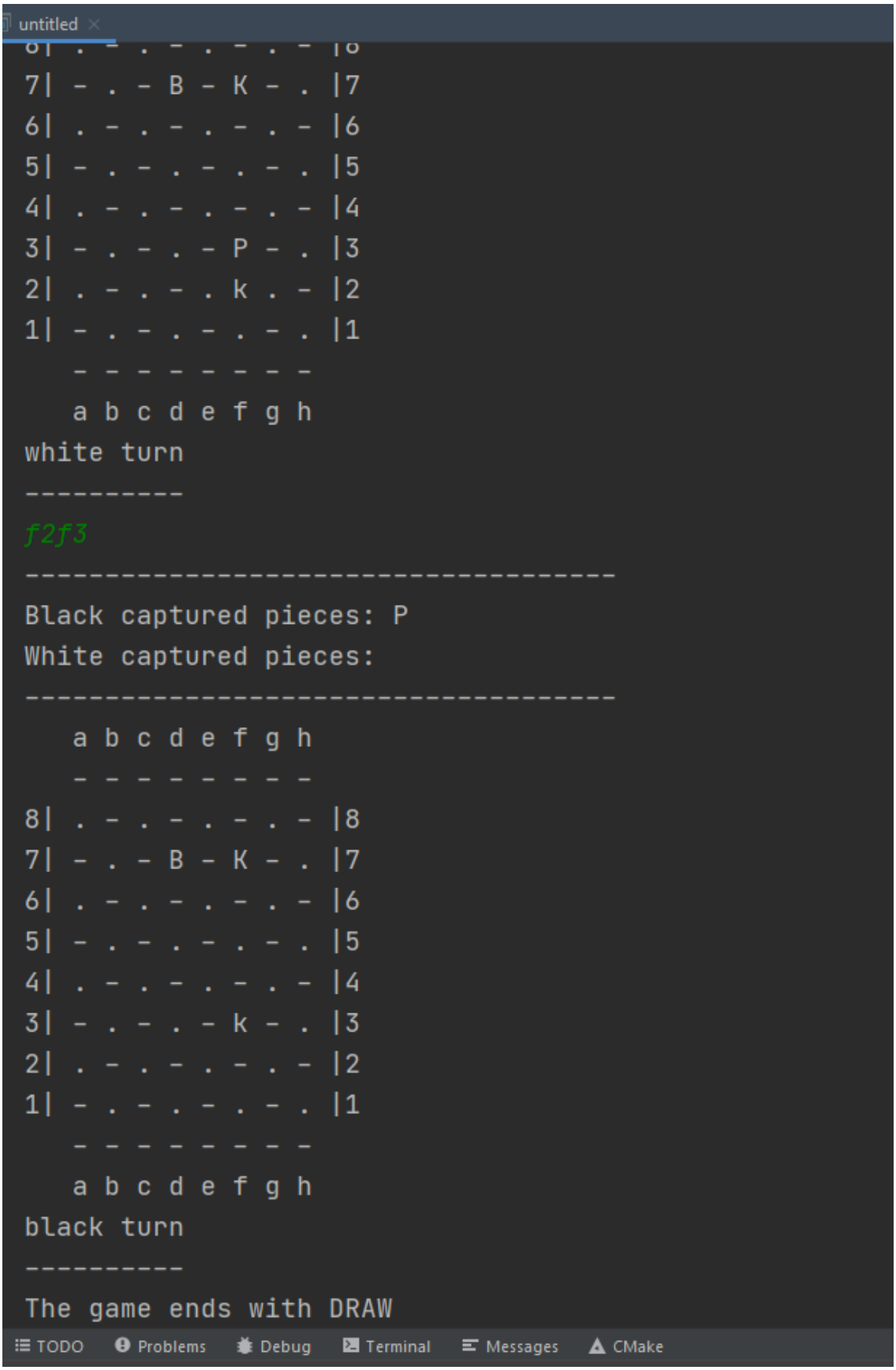


```
-------------------------------------
Black captured pieces: P
White captured pieces:
-------------------------------------
   a b c d e f g h
   - - - - - - - -
8| R N B Q K B . R |8
7| - P P P P P p P |7
6| . - . - . N . - |6
5| - . - . - . - . |5
4| P - . - . - p - |4
3| - . - . - . - . |3
2| p p p p p p . - |2
1| r n b q k b n r |1
   - - - - - - - -
   a b c d e f g h
white turn
----------
g7g8
Enter the required promoted piece: q
-------------------------------------
Black captured pieces: P
White captured pieces:
-------------------------------------
   a b c d e f g h
   - - - - - - - -
8| R N B Q K B q R |8
7| - P P P P P - P |7
6| . - . - . N . - |6
5| - . - . - . - . |5
4| P - . - . - p - |4
3| - . - . - . - . |3
2| p p p p p p . - |2
1| r n b q k b n r |1
   - - - - - - - -
   a b c d e f g h
black turn
----------
```


Black captured pieces:
White captured pieces:
a b c d e f g h
8| R N B Q K - . R |8
7| P P P P - P P P |7
6| . - . B . - . N |6
5| - . - . P . - . |5
4| . - . - p - . - |4
3| - . - b - . - n |3
2| p p p p . p p p |2
1| r n b q k . - r |1
a b c d e f g h
white turn
e1g1
Black captured pieces:
White captured pieces:
a b c d e f g h
8| R N B Q K - . R |8
7| P P P P - P P P |7
6| . - . B . - . N |6
5| - . - . P . - . |5
4| . - . - p - . - |4
3| - . - b - . - n |3
2| p p p p . p p p |2
1| r n b q - r k . |1
a b c d e f g h
black turn

```
------------------------------------
Black captured pieces:
White captured pieces:
------------------------------------
   a b c d e f g h
   - - - - - - - -
8| R N B Q K B N R |8
7| P P P P P . P P |7
6| . - . - . - . - |6
5| - . - . - P - . |5
4| . - . - p - . - |4
3| - . - . - . - . |3
2| p p p p . p p p |2
1| r n b q k b n r |1
   - - - - - - - -
   a b c d e f g h
white turn
----------
d1f5
you can't do this move
white turn
----------
d1h5
------------------------------------
Black captured pieces:
White captured pieces:
------------------------------------
   a b c d e f g h
   - - - - - - - -
8| R N B Q K B N R |8
7| P P P P P . P P |7
6| . - . - . - . - |6
5| - . - . - P - q |5
4| . - . - p - . - |4
3| - . - . - . - . |3
2| p p p p . p p p |2
1| r n b . k b n r |1
   - - - - - - - -
   a b c d e f g h
Black king under attack
black turn
----------
```

```
------------------------------------
   a b c d e f g h
   - - - - - - - -
8| R N B Q K B N R |8
7| P P P P P . P P |7
6| . - . - . - . - |6
5| - . - . - P - q |5
4| . - . - p - . - |4
3| - . - . - . - . |3
2| p p p p . p p p |2
1| r n b . k b n r |1
   - - - - - - - -
   a b c d e f g h
Black king under attack
black turn
----------
a7a6
Black king under attack
you can't do this move your king is under attack
play again
black turn
----------
g7g6
------------------------------------
Black captured pieces:
White captured pieces:
------------------------------------
   a b c d e f g h
   - - - - - - - -
8| R N B Q K B N R |8
7| P P P P P . - P |7
6| . - . - . - P - |6
5| - . - . - P - q |5
4| . - . - p - . - |4
3| - . - . - . - . |3
2| p p p p . p p p |2
1| r n b . k b n r |1
   - - - - - - - -
   a b c d e f g h
white turn
----------
```


untitled
8| . - . - . - . - |8
7| - . - B - K - . |7
6| . - . - . - . - |6
5| - . - . - . - . |5
4| . - . - . - . - |4
3| - . - . - P - . |3
2| . - . - . k . - |2
1| - . - . - . - . |1
a b c d e f g h
white turn
f2f3
Black captured pieces: P
White captured pieces:
a b c d e f g h
8| . - . - . - . - |8
7| - . - B - K - . |7
6| . - . - . - . - |6
5| - . - . - . - . |5
4| . - . - . - . - |4
3| - . - . - k - . |3
2| . - . - . - . - |2
1| - . - . - . - . |1
a b c d e f g h
black turn
The game ends with DRAW
TODO
Problems
Debug
Terminal
Messages
CMake

```
-------------------------------------
Black captured pieces: P
White captured pieces:
-------------------------------------
   a b c d e f g h
   - - - - - - - -
8| R N B Q K B N R |8
7| - P P P P . - . |7
6| . - . - . - P - |6
5| P . - . - P - q |5
4| . - . - p - . - |4
3| p . - . - . - . |3
2| . p p p . p p p |2
1| r n b . k b n r |1
   - - - - - - - -
   a b c d e f g h
white turn
----------
h5g6
-------------------------------------
```

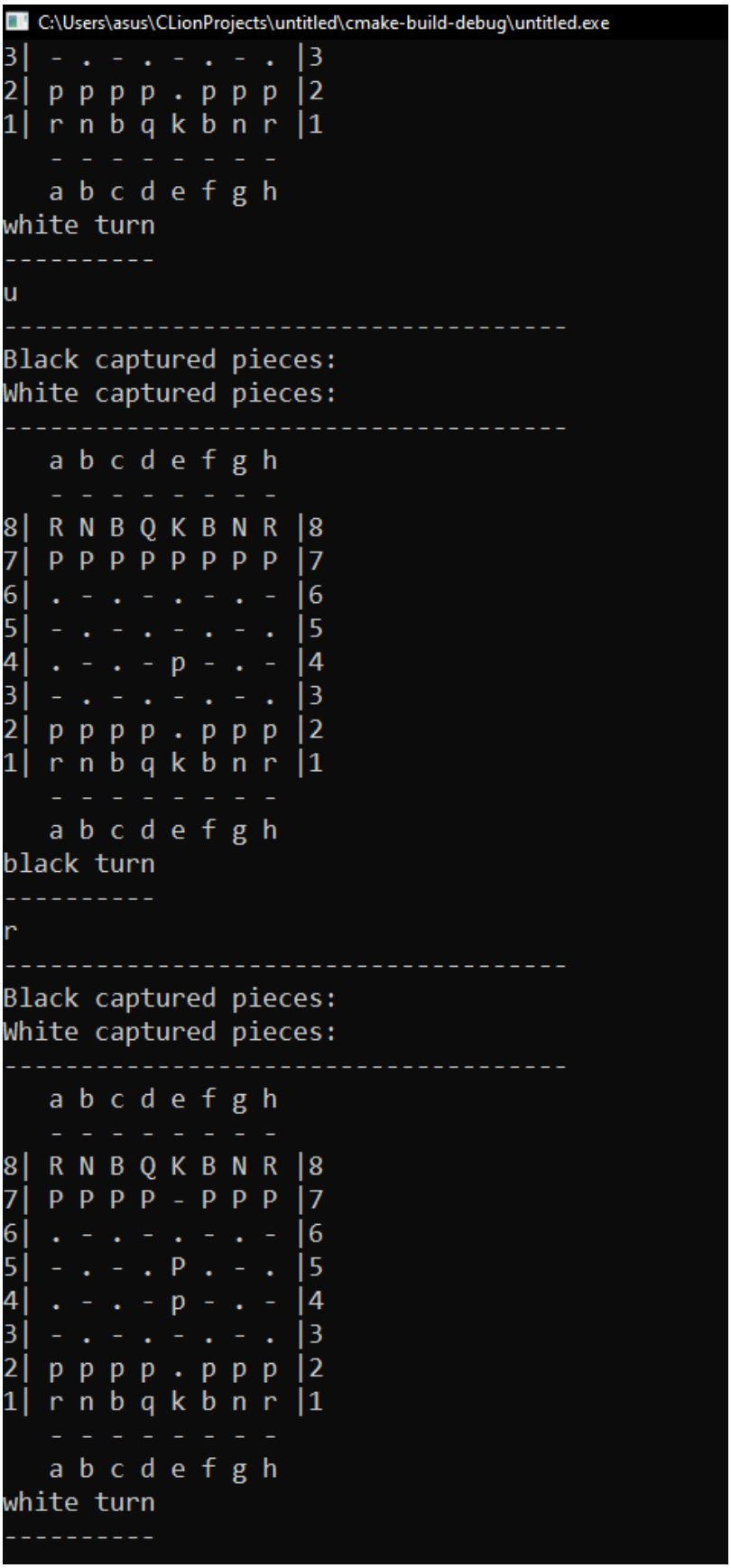
```
white turn
----------
h5g6
-------------------------------------
Black captured pieces: P P
White captured pieces:
-------------------------------------
   a b c d e f g h
   - - - - - - - -
8| R N B Q K B N R |8
7| - P P P P . - . |7
6| . - . - . - q - |6
5| P . - . - P - . |5
4| . - . - p - . - |4
3| p . - . - . - . |3
2| . p p p . p p p |2
1| r n b . k b n r |1
   - - - - - - - -
   a b c d e f g h
Black king under attack
black turn
----------
white Win

Process finished with exit code 0
```

```
--------------------------------------
Black captured pieces:
White captured pieces:
--------------------------------------
   a b c d e f g h
   - - - - - - - -
8| R N B Q K B N R |8
7| P P P P - P P P |7
6| . - . - . - . - |6
5| - . - . P . - . |5
4| . - . - p - . - |4
3| - . - . - . - . |3
2| p p p p . p p p |2
1| r n b q k b n r |1
   - - - - - - - -
   a b c d e f g h
white turn
----------
s
Enter name of the file: mygame
```

```
------------------------------------------------------------------
Commands: (N)ew game   (U)ndo  R(ed0) (S)ave  (L)oad  (Q)uit
------------------------------------------------------------------
   a b c d e f g h
   - - - - - - - -
8| R N B Q K B N R |8
7| P P P P P P P P |7
6| . - . - . - . - |6
5| - . - . - . - . |5
4| . - . - . - . - |4
3| - . - . - . - . |3
2| p p p p p p p p |2
1| r n b q k b n r |1
   - - - - - - - -
   a b c d e f g h
white turn
----------
1
Enter name of the file: mygame
-------------------------------------
Black captured pieces:
White captured pieces:
-------------------------------------
   a b c d e f g h
   - - - - - - - -
8| R N B Q K B N R |8
7| P P P P - P P P |7
6| . - . - . - . - |6
5| - . - . P . - . |5
4| . - . - p - . - |4
3| - . - . - . - . |3
2| p p p p . p p p |2
1| r n b q k b n r |1
   - - - - - - - -
   a b c d e f g h
white turn
----------
```

```
1| r n b q k b n r |1
   - - - - - - - -
   a b c d e f g h
black turn
----------
e7e5
-------------------------------------
Black captured pieces:
White captured pieces:
-------------------------------------
   a b c d e f g h
   - - - - - - - -
8| R N B Q K B N R |8
7| P P P P - P P P |7
6| . - . - . - . - |6
5| - . - . P . - . |5
4| . - . - p - . - |4
3| - . - . - . - . |3
2| p p p p . p p p |2
1| r n b q k b n r |1
   - - - - - - - -
   a b c d e f g h
white turn
----------
u
-------------------------------------
Black captured pieces:
White captured pieces:
-------------------------------------
   a b c d e f g h
   - - - - - - - -
8| R N B Q K B N R |8
7| P P P P P P P P |7
6| . - . - . - . - |6
5| - . - . - . - . |5
4| . - . - p - . - |4
3| - . - . - . - . |3
2| p p p p . p p p |2
1| r n b q k b n r |1
   - - - - - - - -
   a b c d e f g h
black turn
----------
```


C:\Users\asus\CLionProjects\untitled\cmake-build-debug\untitled.exe
3| - . - . - . - . |3
2| p p p p . p p p |2
1| r n b q k b n r |1
a b c d e f g h
white turn
u
Black captured pieces:
White captured pieces:
a b c d e f g h
8| R N B Q K B N R |8
7| P P P P P P P P |7
6| . - . - . - . - |6
5| - . - . - . - . |5
4| . - . - p - . - |4
3| - . - . - . - . |3
2| p p p p . p p p |2
1| r n b q k b n r |1
a b c d e f g h
black turn
r
Black captured pieces:
White captured pieces:
a b c d e f g h
8| R N B Q K B N R |8
7| P P P P - P P P |7
6| . - . - . - . - |6
5| - . - . P . - . |5
4| . - . - p - . - |4
3| - . - . - . - . |3
2| p p p p . p p p |2
1| r n b q k b n r |1
a b c d e f g h
white turn