70083 13

C++ 3

Submitters

sf23	Shihan Fu

Emarking

```
Chess Exercise Output: Summary for sf23 of v5
---------------------------------------------

  Execution Comparison:
    Test for invalid inputs:          0 / 1
    Test for static board state:      1 / 1
    Test pawn moves:                  0 / 1
    Test rook moves:                  1 / 1
    Test knight moves:                1 / 1
    Test bishop moves:                1 / 1
    Test queen moves:                 0 / 1
    Test king moves:                  1 / 1
    Test match provided to students:  1 / 1
    Unseen test match:                0 / 1

Git Repo: git@gitlab.doc.ic.ac.uk:lab2324_autumn/msc_lab3_sf23.git
Commit ID: b27d8
```

61/100

Good object oriented design of the board and chess pieces, however there are some major issues with the functionality and design of the code. The main source of the issues is your ‘status’ approach, which seems not to be working properly (see results of failed tests). I am not completely sure what has gone wrong here, and it is very difficult to read and understand the update status function.

A better approach would have been to implement a series of checks to the board where necessary, rather than trying to keep a running status log (which is also somewhat inefficient). E.g. you should actively scan the board and check for potential attacks on the king every time you test for the game being in check. This would have been a more explicit approach and would have been easier to spot errors in

```
Detailed Output for test: Test for invalid inputs
-------------------------------------------------

Test for invalid inputs

  Compiled OK

  Compilation Standard Output:

g++ -Wall -g -c ChessBoard.cpp
g++ -Wall -g -c ChessPiece.cpp
g++ -Wall -g -c Color.cpp
g++ -Wall -g -c ChessMain.cpp
g++ -Wall -g ChessBoard.o ChessPiece.o ChessMain.o -o chess

  Compilation Standard Error:

ChessBoard.cpp: In destructor ???ChessBoard::~ChessBoard()???:
ChessBoard.cpp:29:13: warning: deleting object of abstract class type ???ChessPiece??? which has non-virtual destructor will cause undefined behavior [-Wdelete-non-virtual-dtor]
   29 |             delete cps[i][j];
      |             ^~~~~~~~~~~~~~~~
ChessBoard.cpp: In member function ???void ChessBoard::loadState(std::string)???:
ChessBoard.cpp:627:22: warning: comparison of integer expressions of different signedness: ???int??? and ???std::__cxx11::basic_string<char>::size_type??? {aka ???long unsigned int???} [-Wsign-compare]
  627 |         for(int j=0;j<tokens[i].length();j++){
      |                     ~^~~~~~~~~~~~~~~~~~~
ChessBoard.cpp: In member function ???int ChessBoard::castling_move(ChessPiece*, std::string, std::string, ChessPiece* (*)[8], int* (*)[8])???:
ChessBoard.cpp:692:30: warning: comparison of integer expressions of different signedness: ???std::__cxx11::basic_string<char>::size_type??? {aka ???long unsigned int???} and ???int??? [-Wsign-compare]
  692 |     if(castling.find("piece")==-1){
      |        ~~~~~~~~~~~~~~~~~~~~~~^~~~
ChessBoard.cpp: In member function ???int ChessBoard::submitMoveWithoutComments(std::string, std::string)???:
ChessBoard.cpp:564:1: warning: control reaches end of non-void function [-Wreturn-type]
  564 | }
      | ^
ChessBoard.cpp: In member function ???int ChessBoard::submitMove(std::string, std::string)???:
ChessBoard.cpp:616:1: warning: control reaches end of non-void function [-Wreturn-type]
  616 | }
      | ^

  Test failed because Unexpected Exit Code
    Model Answer exit code was 0, Student's exit code was

  Model Output (Left) vs Student's Output (Right):

==========================                                    ==========================
Testing the Chess Engine                                      Testing the Chess Engine
==========================                                    ==========================

loadState("rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq"): A new board state is lo    loadState("rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq"): A new board state is lo

Invalid square references                                     Invalid square references
cb.submitMove("", "D5"): Invalid square reference!          | cb.submitMove("", "D5"): invalid move input
cb.submitMove("E2", ""): Invalid square reference!          | cb.submitMove("E2", ""): invalid move input
cb.submitMove("e2", "e4"): Invalid square reference!        | cb.submitMove("e2", "e4"): It is not Black's turn to move!
cb.submitMove("B2", "Z89"): Invalid square reference!       | cb.submitMove("B2", "Z89"): invalid move input
cb.submitMove("G2", "Hello"): Invalid square reference!     | cb.submitMove("G2", "Hello"): invalid move input
cb.submitMove("W0", "D1"): Invalid square reference!        | cb.submitMove("W0", "D1"): It is not Black's turn to move!
cb.submitMove("World", "C3"): Invalid square reference!     | cb.submitMove("World", "C3"): invalid move input
```

```
   59: Wrong player tries to move                                                  Wrong player tries to move
   60: cb.submitMove("D7", "D6"): It is not Black's turn to move!                  cb.submitMove("D7", "D6"): It is not Black's turn to move!
   61:
   62: No piece at source square                                                  No piece at source square
   63: cb.submitMove("D4", "H6"): There is no piece at position D4!                cb.submitMove("D4", "H6"): There is no piece at position D4!
   64:
   65: White tries to capture one of its own pieces                               White tries to capture one of its own pieces
   66: cb.submitMove("D1", "C1"): White's Queen cannot move to C1!                 cb.submitMove("D1", "C1"): White's Queen cannot move to C1!
   67:
   68: Change player to Black:                                                    Change player to Black:
   69: loadState("rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR b KQkq"): A new board state is lo    loadState("rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR b KQkq"): A new board state is
lo
   70:
   71: Black tries to capture one of its own pieces                               Black tries to capture one of its own pieces
   72: cb.submitMove("F8", "E7"): Black's Bishop cannot move to E7!               cb.submitMove("F8", "E7"): Black's Bishop cannot move to E7!
   73:
   74: loadState("2bqkbnr/8/8/8/8/8/8/RNBQKBNR w KQkq"): A new board state is loaded!    loadState("2bqkbnr/8/8/8/8/8/8/RNBQKBNR w KQkq"): A new board state is loaded!
   75:
   76: Try to move off board                                                      Try to move off board
   77: cb.submitMove("A1", "A9"): Invalid square reference!                    <
   78: cb.submitMove("F1", "I4"): Invalid square reference!                    <
   79:                                                                         <
   80:
   81:   Model Answer (Left) vs Student's Error Ouput (Right):
   82:
   83:                                                                         >  /usr/bin/timeout: the monitored command dumped core
   84:
   85:
   86: Detailed Output for test: Test for static board state
   87: ------------------------------------------------------
   88:
   89: Test for static board state
   90:
   91:   Compiled OK
   92:
   93:   Compilation Standard Output:
   94:
   95: g++ -Wall -g -c ChessMain.cpp
   96: g++ -Wall -g ChessBoard.o ChessPiece.o ChessMain.o -o chess
   97:
   98:   Test Passed
   99:
  100:
  101:
  102:
  103: Detailed Output for test: Test pawn moves
  104: ----------------------------------------
  105:
  106: Test pawn moves
  107:
  108:   Compiled OK
  109:
  110:   Compilation Standard Output:
  111:
  112: g++ -Wall -g -c ChessMain.cpp
  113: g++ -Wall -g ChessBoard.o ChessPiece.o ChessMain.o -o chess
  114:
  115:   Test failed because Output differs
  116:
  117:   Model Output (Left) vs Student's Output (Right):
  118:
```

```
==========================
Testing the Pawn Piece
==========================

loadState("rnbqkbnr/8/8/8/8/8/8/RNBQKBNR w KQkq"): A new board state is loaded!

Invalid Moves for Pawns
cb.submitMove("A2", "A5"): White's Pawn cannot move to A5!
cb.submitMove("C2", "E6"): White's Pawn cannot move to E6!

Valid Moves for Pawns
cb.submitMove("F2", "F3"): White's Pawn moves from F2 to F3
cb.submitMove("B7", "B6"): Black's Pawn moves from B7 to B6

Invalid Move (White)
cb.submitMove("F3", "F5"): White's Pawn cannot move to F5!
Still White's Turn; Valid Move
cb.submitMove("F3", "F4"): White's Pawn moves from F3 to F4

Invalid Moves (Black)
cb.submitMove("B6", "B4"): Black's Pawn cannot move to B4!
cb.submitMove("D7", "A4"): Black's Pawn cannot move to A4!


Detailed Output for test: Test rook moves
-----------------------------------------

Test rook moves

  Compiled OK

  Compilation Standard Output:

g++ -Wall -g -c ChessMain.cpp
g++ -Wall -g ChessBoard.o ChessPiece.o ChessMain.o -o chess

  Test Passed



Detailed Output for test: Test knight moves
-------------------------------------------

Test knight moves

  Compiled OK

  Compilation Standard Output:

g++ -Wall -g -c ChessMain.cpp
g++ -Wall -g ChessBoard.o ChessPiece.o ChessMain.o -o chess

  Test Passed



Detailed Output for test: Test bishop moves
-------------------------------------------
```

```
==========================
Testing the Pawn Piece
==========================

loadState("rnbqkbnr/8/8/8/8/8/8/RNBQKBNR w KQkq"): A new board state is loaded!

Invalid Moves for Pawns
cb.submitMove("A2", "A5"): White's Pawn cannot move to A5!
cb.submitMove("C2", "E6"): White's Pawn cannot move to E6!

Valid Moves for Pawns
cb.submitMove("F2", "F3"): White's Pawn moves from F2 to F3
cb.submitMove("B7", "B6"): Black's Pawn moves from B7 to B6

Invalid Move (White)
cb.submitMove("F3", "F5"): White's Pawn moves from F3 to F5
Still White's Turn; Valid Move
cb.submitMove("F3", "F4"): There is no piece at position F3!

Invalid Moves (Black)
cb.submitMove("B6", "B4"): Black's Pawn moves from B6 to B4
cb.submitMove("D7", "A4"): It is not Black's turn to move!
```

```
180: Test bishop moves
181:
182:   Compiled OK
183:
184:   Compilation Standard Output:
185:
186: g++ -Wall -g -c ChessMain.cpp
187: g++ -Wall -g ChessBoard.o ChessPiece.o ChessMain.o -o chess
188:
189:   Test Passed
190:
191:
192:
193:
194: Detailed Output for test: Test queen moves
195: ----------------------------------------
196:
197: Test queen moves
198:
199:   Compiled OK
200:
201:   Compilation Standard Output:
202:
203: g++ -Wall -g -c ChessMain.cpp
204: g++ -Wall -g ChessBoard.o ChessPiece.o ChessMain.o -o chess
205:
206:   Test failed because Output differs
207:
208:   Model Output (Left) vs Student's Output (Right):
209:
210: ==========================                                    ==========================
211: Testing the Queen Piece                                       Testing the Queen Piece
212: ==========================                                    ==========================
213:
214: loadState("rnbqkbnr/8/8/3p4/8/8/8/RNBQKBNR w KQkq"): A new board state is loaded!          loadState("rnbqkbnr/8/8/3p4/8/8/8/RNBQKBNR w KQkq"): A new board state is loaded!
215:
216: Testing invalid moves for White's Queen                      Testing invalid moves for White's Queen
217: cb.submitMove("D1", "E3"): White's Queen cannot move to E3!  cb.submitMove("D1", "E3"): White's Queen cannot move to E3!
218: cb.submitMove("D1", "D6"): White's Queen cannot move to D6! |cb.submitMove("D1", "D6"): White's Queen moves from D1 to D6
219:
220: Change player to Black:                                     Change player to Black:
221: loadState("rnbqkbnr/2p5/8/8/8/8/8/RNBQKBNR b KQkq"): A new board state is loaded!        loadState("rnbqkbnr/2p5/8/8/8/8/8/RNBQKBNR b KQkq"): A new board state is loaded!
222:
223: Testing invalid moves for Black's Queen                     Testing invalid moves for Black's Queen
224: cb.submitMove("D8", "E6"): Black's Queen cannot move to E6!  cb.submitMove("D8", "E6"): Black's Queen cannot move to E6!
225: cb.submitMove("D8", "A5"): Black's Queen cannot move to A5! |cb.submitMove("D8", "A5"): Black's Queen moves from D8 to A5
226:                                                            > White is in check
227:
228:
229: Detailed Output for test: Test king moves
230: ----------------------------------------
231:
232: Test king moves
233:
234:   Compiled OK
235:
236:   Compilation Standard Output:
237:
238: g++ -Wall -g -c ChessMain.cpp
239: g++ -Wall -g ChessBoard.o ChessPiece.o ChessMain.o -o chess
240:
```

```
  Test Passed



Detailed Output for test: Test match provided to students
---------------------------------------------------------

Test match provided to students

  Compiled OK

  Compilation Standard Output:

g++ -Wall -g -c ChessMain.cpp
g++ -Wall -g ChessBoard.o ChessPiece.o ChessMain.o -o chess

  Test Passed



Detailed Output for test: Unseen test match
-------------------------------------------

Unseen test match

  Compiled OK

  Compilation Standard Output:

g++ -Wall -g -c ChessMain.cpp
g++ -Wall -g ChessBoard.o ChessPiece.o ChessMain.o -o chess

  Test failed because Output differs

  Model Output (Left) vs Student's Output (Right):

==========================                                ==========================
Alekhine vs. Bruce (1938)                                 Alekhine vs. Bruce (1938)
==========================                                ==========================

A new board state is loaded!                              A new board state is loaded!

White's Pawn moves from E2 to E4                          White's Pawn moves from E2 to E4
Black's Pawn moves from C7 to C6                          Black's Pawn moves from C7 to C6

White's Knight moves from B1 to C3                        White's Knight moves from B1 to C3
Black's Pawn moves from D7 to D5                          Black's Pawn moves from D7 to D5

White's Knight moves from G1 to F3                        White's Knight moves from G1 to F3
Black's Pawn moves from D5 to E4 taking White's Pawn      Black's Pawn moves from D5 to E4 taking White's Pawn

White's Knight moves from C3 to E4 taking Black's Pawn    White's Knight moves from C3 to E4 taking Black's Pawn
Black's Bishop moves from C8 to F5                        Black's Bishop moves from C8 to F5

White's Knight moves from E4 to G3                        White's Knight moves from E4 to G3
Black's Bishop moves from F5 to G6                        Black's Bishop moves from F5 to G6

White's Pawn moves from H2 to H4                          White's Pawn moves from H2 to H4
Black's Pawn moves from H7 to H6                          Black's Pawn moves from H7 to H6
```

```
302:
303: White's Knight moves from F3 to E5                    White's Knight moves from F3 to E5
304: Black's Bishop moves from G6 to H7                    Black's Bishop moves from G6 to H7
305:
306: White's Queen moves from D1 to H5                     White's Queen moves from D1 to H5
307: Black's Pawn moves from G7 to G6                      Black's Pawn moves from G7 to G6
308:
309: White's Bishop moves from F1 to C4                    White's Bishop moves from F1 to C4
310: Black's Pawn moves from E7 to E6                      Black's Pawn moves from E7 to E6
311:
312: White's Queen moves from H5 to E2                     White's Queen moves from H5 to E2
313: Black's Knight moves from G8 to F6                    Black's Knight moves from G8 to F6
314:
315: White's Knight moves from E5 to F7 taking Black's Pawn  White's Knight moves from E5 to F7 taking Black's Pawn
316: Black's King moves from E8 to F7 taking White's Knight |  Black's King cannot move to F7!
317:
318: White's Queen moves from E2 to E6 taking Black's Pawn |  It is not White's turn to move!
319: Black is in check                                     |  It is not White's turn to move!
320: Black's King moves from F7 to G7                      <
321:
322: White's Queen moves from E6 to F7                     |  Black's Pawn cannot move to F7!
323: Black is in checkmate                                 <
324:
325: Trying to make a move after the game has ended        Trying to make a move after the game has ended
326: The game is over!                                     |  Black's Knight cannot move to E8!
```

```
/*
 * @Author: shihan
 * @Date: 2023-11-22 21:38:52
 * @version: 1.0
 * @description: definition of the ChessBoard
 */
#ifndef CHESSBOARD_H
#define CHESSBOARD_H

#include <iostream>
#include "ChessPiece.h"
#include <string>
#include <vector>
#include "Color.h"

using namespace std;

// defined string split function
vector<string> splitString(const string& input, char delimiter);

class ChessBoard{
    private:
        // each piece on the chess board
        ChessPiece* cps[8][8];
        // the "being attacked" status of each square
        //2: under attack by Black,
        //-1: under attack by White,
        //1: under attack by both Black and White,
        //0: not under attack
        int* status[8][8];
        // whose turn to move the next piece
        char turn;
        // castling status
        string castling;
        // the last "eaten" chess piece
        ChessPiece* temp;

    public:
        // initial constructor
        ChessBoard();
        // deconstructor
        ~ChessBoard();

        // load the state of the chess board
        void loadState(string FEN);
        // update the status of each square
        void updateStatus();

        // the user submit move
        int submitMove(string from, string to);
        // the background logic try to move
        int submitMoveWithoutComments(string from, string to);
        // redo the last move
        void cancelMove(string origin_from, string origin_to);

        // castling move
        int castling_move(ChessPiece* piece, string from, string to, ChessPiece* cb[8][8], int* status[8][8]);

        // whether there is a "in check" status, does not check checkmate
        bool isCheck();
        // whether the check is a checkmate
        bool isCheckmate(int x, int y);
        int checkState();
        // whether there is a "in check" status, also check checkmate
        bool isStatusCheck();
```

Would be better to use an enum to store this attack info instead of int -1


```
        // try every move to see if the check can be removed
        bool tryRemoveCheck(Color color, string FEN);

        // print the status of each square
        void printStatus();
        // print each piece of the chess board
        void printBoard();
        // transfer the chess board to the FEN format
        string transferFEN();
        // transfer the loaction to FEN format
        string getLocation(int x, int y);


};
#endif
```

```
/*
 * @Author: shihan
 * @Date: 2023-12-02 18:30:46
 * @version: 1.0
 * @description: definition of the ChessPiece
 */
#ifndef CHESSPIECE_H
#define CHESSPIECE_H

#include <iostream>
#include <string>
#include "Color.h"

using namespace std;

class ChessPiece{
    protected:
        Color color;    // the color of the piece
        char symbol;    // the symbol of the piece
        bool first_move;    // whether it is the first move of a piece


    public:
        ChessPiece();   // initial constructor
        ~ChessPiece();  // deconstructor
        ChessPiece(Color _color);  // constructor gith color

        // virtual function of move
        virtual int move(int *from_index, int *to_index, ChessPiece* cb[8][8], int* status[8][8]);
        // virtual function of get the tyoe of a chess piece
        virtual string getType() const = 0;

        // setter and getter fucntion of ChessPiece
        void setSymbol(char _symbol);
        void setColor(Color _color);
        void setFirstMove(bool _firstMove);
        Color getColor();
        char getSymbol();
        bool getFirstMove();

        // define the cout of each chess piece
        friend ostream &operator<<(ostream &out, const ChessPiece &object) {
            out << object.symbol;
            return out;
        }

};

class King: public ChessPiece{
    private:
        int move(int *from_index, int *to_index, ChessPiece* cb[8][8], int* status[8][8]) override;
    public:
        King(Color _color);
        string getType() const override;
};

class Rook: public ChessPiece{
    private:
        int move(int *from_index, int *to_index, ChessPiece* cb[8][8], int* status[8][8]) override;
    public:
        Rook(Color _color);
        string getType() const override;
};
```

Good use of Color enum


```

class Bishop: public ChessPiece{
    private:
        int move(int *from_index, int *to_index, ChessPiece* cb[8][8], int* status[8][8]) override;
    public:
        Bishop(Color _color);
        string getType() const override;
};

class Queen: public ChessPiece{
    private:
        int move(int *from_index, int *to_index, ChessPiece* cb[8][8], int* status[8][8]) override;
    public:
        Queen(Color _color);
        string getType() const override;
};

class Knight: public ChessPiece{
    private:
        int move(int *from_index, int *to_index, ChessPiece* cb[8][8], int* status[8][8]) override;
    public:
        Knight(Color _color);
        string getType() const override;
};

class Pawn: public ChessPiece{
    private:
        int move(int *from_index, int *to_index, ChessPiece* cb[8][8], int* status[8][8]) override;
    public:
        Pawn(Color _color);
        string getType() const override;
};
#endif
```

```h
#ifndef COLORS_H
#define COLORS_H
#include <string>
using namespace std;
// Declare the enum in the header file
enum Color {
    Black,
    White
};

string colorToString(Color c);

#endif
```



```cpp
/*
 * @Author: shihan
 * @Date: 2023-11-22 21:40:28
 * @version: 1.0
 * @description: implementation of ChessBoard
 */
#include "ChessBoard.h"
#include "ChessPiece.h"
#include <iostream>
#include <string>
#include <vector>
#include "Color.h"

using namespace std;

// initial constructor
ChessBoard::ChessBoard(){
    for (int i = 0; i < 8; ++i) {
        for (int j = 0; j < 8; ++j) {
            cps[i][j] = nullptr;
        }
    }
}

// deconstructor
ChessBoard::~ChessBoard(){
    for (int i = 0; i < 8; ++i) {
        for (int j = 0; j < 8; ++j) {
            delete cps[i][j];
        }
    }
}

// update the status of each square
void ChessBoard::updateStatus(){
    // Initialize the array with some values (for demonstration purposes)
    for (int i = 0; i < 8; ++i) {
        for (int j = 0; j < 8; ++j) {
            status[i][j] = new int(0);
        }
    }
    //update the status
    for(int i=0;i<8;i++){
        for(int j=0;j<8;j++){
            if(cps[i][j]!=nullptr){
                if(cps[i][j]->getSymbol()=='p'){
                    // for Black's Pawn
                    // the next one to pawn is being attacked by Black
                    if(i+1<8){
                        if(*status[i+1][j]==-1 ||*status[i+1][j]==0){*status[i+1][j]+=2;}
                    }
                    // if the next squae is empty, the second next is also attcked
                    if( i+2 <8 && cps[i+1][j]==nullptr){
                        if(*status[i+2][j]==-1 ||*status[i+2][j]==0){*status[i+2][j]+=2;}
                    }

                }
                if(cps[i][j]->getSymbol()=='k'){
                    // for Black's king
                    // every square next to it is attacked
                    if(i-1>=0){ // each left square
                        if(*status[i-1][j]==-1 || *status[i-1][j] ==0){*status[i-1][j]+=2;}
```

It is completely unclear what this function is attempting to do, and in any case it looks very duplicative and inefficient. Try to split your functions out in to sub functions with meaningful names rather than having a giant set of nested if statements -5

Having now read through the rest of your code, I assume that this function is updating the status of every square on the board after every move. This is quite inefficient in terms of time and memory. It is time inefficient because you are updating the whole board rather than just checking the necessary pieces at each turn, and it is memory inefficient because you are storing and passing around two chessboards instead of one. It also appears to have caused incorrect functionality (see the test results above), but it is very difficult to know where something has gone wrong due to the structure -10

Your code lines are too wide. In general, code should not be more than 80 characters wide (industry standard). -3 for readability


```
    63:                     }
    64:                     if(i+1<8){  // each right square
    65:                         if(*status[i+1][j]==-1 ↙
||*status[i+1][j]==0){*status[i+1][j]+=2;}
    66:                     }
    67:                     if(j-1>=0){ // each upper square
    68:                         if(i-1>=0){
    69:                             if(*status[i-1][j-1]==-1 || *status[i-1][j-1] ↙
==0){*status[i-1][j-1]+=2;}
    70:                         }
    71:                         if(i+1<8){
    72:                             if(*status[i+1][j-1]==-1 || ↙
*status[i+1][j-1]==0){*status[i+1][j-1]+=2;}
    73:                         }
    74:                         if↙
(*status[i][j-1]==-1||*status[i][j-1]==0){*status[i][j-1]+=2;}
    75:                     }
    76:                     if(j+1<8){  // each lower square
    77:                         if(i-1>=0){
    78:                             if(*status[i-1][j+1]==-1 || *status[i-1][j+1] ↙
==0){*status[i-1][j+1]+=2;}
    79:                         }
    80:                         if(i+1<8){
    81:                             if(*status[i+1][j+1]==-1 || ↙
*status[i+1][j+1]==0){*status[i+1][j+1]+=2;}
    82:                         }
    83:                         if↙
(*status[i][j+1]==-1||*status[i][j+1]==0){*status[i][j+1]+=2;}
    84:
    85:                     }
    86:
    87:                 }
    88:                 if(cps[i][j]->getSymbol()=='r'||cps[i][j]->getSymbol()=='q'){
    89:                     // for black's Rook and Queen
    90:                     int ii=i-1;
    91:                     while(ii>=0 && cps[ii][j]==nullptr){    // for each left ↙
square
    92:                         if↙
(*status[ii][j]==-1||*status[ii][j]==0){*status[ii][j]+=2;}
    93:                         ii--;
    94:                     }
    95:                     if(ii>=0){if↙
(*status[ii][j]==-1||*status[ii][j]==0){*status[ii][j]+=2;}}
    96:                     ii=i+1;
    97:                     while(ii<8 && cps[ii][j]==nullptr){     // for each right ↙
square
    98:                         if↙
(*status[ii][j]==-1||*status[ii][j]==0){*status[ii][j]+=2;}
    99:                         ii++;
   100:                     }
   101:                     if(ii<8){if↙
(*status[ii][j]==-1||*status[ii][j]==0){*status[ii][j]+=2;}}
   102:
   103:                     int jj = j-1;
   104:                     while(jj>=0 && cps[i][jj]==nullptr){    // for each upper ↙
square
   105:                         if↙
(*status[i][jj]==-1||*status[i][jj]==0){*status[i][jj]+=2;}
   106:                         jj--;
   107:
   108:                     }
   109:                     if(jj>=0){if↙
(*status[i][jj]==-1||*status[i][jj]==0){*status[i][jj]+=2;}}
   110:
   111:                     jj=j+1;
   112:                     while(jj<8 && cps[i][jj]==nullptr){     // for each lower ↙
```



```
square
   113:                         if↙
(*status[i][jj]==-1||*status[i][jj]==0){*status[i][jj]+=2;}
   114:                         jj++;
   115:                     }
   116:                     if(jj<=8){if↙
(*status[i][jj]==-1||*status[i][jj]==0){*status[i][jj]+=2;}}
   117:
   118:                 }
   119:                 if(cps[i][j]->getSymbol()=='b'||cps[i][j]->getSymbol()=='q'){
   120:                     // for black's Bishop and Queen
   121:                     int ii=i-1, jj = j-1;
   122:                     while(ii>=0 && jj>=0 && cps[ii][jj]==nullptr){
   123:                         if↙
(*status[ii][jj]==-1||*status[ii][jj]==0){*status[ii][jj]+=2;}
   124:                         ii--;
   125:                         jj--;
   126:                     }
   127:                     if(ii>=0&&jj>=0){if↙
(*status[ii][jj]==-1||*status[ii][jj]==0){*status[ii][jj]+=2;}}
   128:                     ii=i-1;
   129:                     jj=j+1;
   130:                     while(ii>=0 && jj<8 && cps[ii][jj]==nullptr){
   131:                         if↙
(*status[ii][jj]==-1||*status[ii][jj]==0){*status[ii][jj]+=2;}
   132:                         ii--;
   133:                         jj++;
   134:                     }
   135:                     if(ii>=0&&jj<8){if↙
(*status[ii][jj]==-1||*status[ii][jj]==0){*status[ii][jj]+=2;}}
   136:                     ii=i+1;
   137:                     jj=j-1;
   138:                     while(ii<8 && jj>=0 && cps[ii][jj]==nullptr){
   139:                         if↙
(*status[ii][jj]==-1||*status[ii][jj]==0){*status[ii][jj]+=2;}
   140:                         ii++;
   141:                         jj--;
   142:                     }
   143:                     if(ii<8&&jj>=0){if↙
(*status[ii][jj]==-1||*status[ii][jj]==0){*status[ii][jj]+=2;}}
   144:                     ii=i+1;
   145:                     jj=j+1;
   146:                     while(ii<8 && jj<8 && cps[ii][jj]==nullptr){
   147:                         if↙
(*status[ii][jj]==-1||*status[ii][jj]==0){*status[ii][jj]+=2;}
   148:                         ii++;
   149:                         jj++;
   150:                     }
   151:                     if(ii<8&&jj<8){if↙
(*status[ii][jj]==-1||*status[ii][jj]==0){*status[ii][jj]+=2;}}
   152:                 }
   153:                 if(cps[i][j]->getSymbol()=='n'){
   154:                     // for Black's Knight
   155:                     int ii=i+1;
   156:                     int jj = j+2;
   157:                     if(ii>=0 && ii<8 && jj>=0 && jj<8 ){
   158:                         if↙
(*status[ii][jj]==-1||*status[ii][jj]==0){*status[ii][jj]+=2;}
   159:                     }
   160:
   161:                     jj = j-2;
   162:                     if(ii>=0 && ii<8 && jj>=0 && jj<8 ){
   163:                         if↙
(*status[ii][jj]==-1||*status[ii][jj]==0){*status[ii][jj]+=2;}
   164:                     }
   165:
```

This looks like it would be super hard to debug or update

```
166:                     ii=i+2;
167:                     jj = j+1;
168:                     if(ii>=0 && ii<8 && jj>=0 && jj<8){
169:                         if(*status[ii][jj]==-1||*status[ii][jj]==0){*status[ii][jj]+=2;}
170:                     }
171:                     jj = j-1;
172:                     if(ii>=0 && ii<8 && jj>=0 && jj<8 ){
173:                         if(*status[ii][jj]==-1||*status[ii][jj]==0){*status[ii][jj]+=2;}
174:                     }
175:
176:                     ii=i-1;
177:                     jj=j+2;
178:                     if(ii>=0 && ii<8 && jj>=0 && jj<8 ){
179:                         if(*status[ii][jj]==-1||*status[ii][jj]==0){*status[ii][jj]+=2;}
180:                     }
181:                     jj=j-2;
182:                     if(ii>=0 && ii<8 && jj>=0 && jj<8 ){
183:                         if(*status[ii][jj]==-1||*status[ii][jj]==0){*status[ii][jj]+=2;}
184:                     }
185:
186:                     ii=i-2;
187:                     jj=j+1;
188:                     if(ii>=0 && ii<8 && jj>=0 && jj<8 ){
189:                         if(*status[ii][jj]==-1||*status[ii][jj]==0){*status[ii][jj]+=2;}
190:                     }
191:                     jj=j-1;
192:                     if(ii>=0 && ii<8 && jj>=0 && jj<8 ){
193:                         if(*status[ii][jj]==-1||*status[ii][jj]==0){*status[ii][jj]+=2;}
194:                     }
195:                 }
196:
197:                 // white piece
198:                 if(cps[i][j]->getSymbol()=='P'){
199:                     // for White's Pawn
200:                     // the next one to pawn is being attacked by White
201:                     int ii = i-1;
202:                     if(ii>=0){
203:                         if(*status[ii][j]==2 ||*status[ii][j]==0){*status[ii][j]+=-1;}
204:                     }
205:                     // if the next squae is empty, the second next is also attcked
206:                     if( ii-1>=0 && cps[ii][j]==nullptr){
207:                         if(*status[ii-1][j]==2 ||*status[ii-1][j]==0){*status[ii-1][j]+=-1;}
208:                     }
209:
210:                 }
211:                 if(cps[i][j]->getSymbol()=='K'){
212:                     // for white's's king
213:                     // every square next to it is attacked
214:                     if(i-1>=0){
215:                         if(*status[i-1][j]==2 || *status[i-1][j] ==0){*status[i-1][j]+=-1;}
216:                     }
217:                     if(i+1<8){
218:                         if(*status[i+1][j]==2 ||*status[i+1][j]==0){*status[i+1][j]+=-1;}
219:                     }
220:                     if(j-1>=0){
```



```
221:                         if(i-1>=0){
222:                             if(*status[i-1][j-1]==2 || *status[i-1][j-1] ==0){*status[i-1][j-1]+=-1;}
223:                         }
224:                         if(i+1<8){
225:                             if(*status[i+1][j-1]==2 || *status[i+1][j-1]==0){*status[i+1][j-1]+=-1;}
226:                         }
227:                         if(*status[i][j-1]==2||*status[i][j-1]==0){*status[i][j-1]+=-1;}
228:                     }
229:                     if(j+1<8){
230:                         if(i-1>=0){
231:                             if(*status[i-1][j+1]==2 || *status[i-1][j+1] ==0){*status[i-1][j+1]+=-1;}
232:                         }
233:                         if(i+1<8){
234:                             if(*status[i+1][j+1]==2 || *status[i+1][j+1]==0){*status[i+1][j+1]+=-1;}
235:                         }
236:                         if(*status[i][j+1]==2||*status[i][j+1]==0){*status[i][j+1]+=-1;}
237:                     }
238:                 }
239:
240:                 if(cps[i][j]->getSymbol()=='R'||cps[i][j]->getSymbol()=='Q'){
241:                     // for white's Rook and Queen
242:                     int ii=i-1;
243:
244:                     while(ii>=0 && cps[ii][j]==nullptr){
245:                         if(*status[ii][j]==2||*status[ii][j]==0){*status[ii][j]+=-1;}
246:                         ii--;
247:                     }
248:                     if(ii>=0){if(*status[ii][j]==2||*status[ii][j]==0){*status[ii][j]+=-1;}}
249:                     ii=i+1;
250:                     while(ii<8 && cps[ii][j]==nullptr){
251:                         if(*status[ii][j]==2||*status[ii][j]==0){*status[ii][j]+=-1;}
252:                         ii++;
253:                     }
254:                     if(ii<8){if(*status[ii][j]==2||*status[ii][j]==0){*status[ii][j]+=-1;}}
255:
256:                     int jj = j-1;
257:                     while(jj>=0 && cps[i][jj]==nullptr){
258:                         if(*status[i][jj]==2||*status[i][jj]==0){*status[i][jj]+=-1;}
259:                         jj--;
260:                     }
261:                     if(jj>=0){if(*status[i][jj]==2||*status[i][jj]==0){*status[i][jj]+=-1;}}
262:                     jj=j+1;
263:                     while(jj<8 && cps[i][jj]==nullptr){
264:                         if(*status[i][jj]==2||*status[i][jj]==0){*status[i][jj]+=-1;}
265:                         jj++;
266:                     }
267:                     if(jj<8){if(*status[i][jj]==2||*status[i][jj]==0){*status[i][jj]+=-1;}}
268:
269:                 }
270:                 if(cps[i][j]->getSymbol()=='B'||cps[i][j]->getSymbol()=='Q'){
271:                     // for white's Bishop and Queen
272:                     int ii=i-1, jj = j-1;
```

```
273:                     while(ii>=0 && jj>=0 && cps[ii][jj]==nullptr){
274:                         if
(*status[ii][jj]==2||*status[ii][jj]==0){*status[ii][jj]+=-1;}
275:                         ii--;
276:                         jj--;
277:                     }
278:                     if(ii>=0&&jj>=0){if
(*status[ii][jj]==2||*status[ii][jj]==0){*status[ii][jj]+=-1;}}
279:                     ii=i-1;
280:                     jj=j+1;
281:                     while(ii>=0 && jj<8 && cps[ii][jj]==nullptr){
282:                         if
(*status[ii][jj]==2||*status[ii][jj]==0){*status[ii][jj]+=-1;}
283:                         ii--;
284:                         jj++;
285:                     }
286:                     if(ii>=0&&jj<8){if
(*status[ii][jj]==2||*status[ii][jj]==0){*status[ii][jj]+=-1;}}
287:                     ii=i+1;
288:                     jj=j-1;
289:                     while(ii<8 && jj>=0 && cps[ii][jj]==nullptr){
290:                         if
(*status[ii][jj]==2||*status[ii][jj]==0){*status[ii][jj]+=-1;}
291:                         ii++;
292:                         jj--;
293:                     }
294:                     if(ii<8&&jj>=0){if
(*status[ii][jj]==2||*status[ii][jj]==0){*status[ii][jj]+=-1;}}
295:                     ii=i+1;
296:                     jj=j+1;
297:                     while(ii<8 && jj<8 && cps[ii][jj]==nullptr){
298:                         if
(*status[ii][jj]==2||*status[ii][jj]==0){*status[ii][jj]+=-1;}
299:                         ii++;
300:                         jj++;
301:                     }
302:                     if(ii<8&&jj<8){if
(*status[ii][jj]==2||*status[ii][jj]==0){*status[ii][jj]+=-1;}}
303:                 }
304:                 if(cps[i][j]->getSymbol()=='N'){
305:                     // for Black's Knight
306:                     int ii=i+1;
307:                     int jj = j+2;
308:                     if(ii>=0 && ii<8 && jj>=0 && jj<8){
309:                         if
(*status[ii][jj]==2||*status[ii][jj]==0){*status[ii][jj]+=-1;}
310:                     }
311:                     jj = j-2;
312:                     if(ii>=0 && ii<8 && jj>=0 && jj<8){
313:                         if
(*status[ii][jj]==2||*status[ii][jj]==0){*status[ii][jj]+=-1;}
314:                     }
315:
316:                     ii=i+2;
317:                     jj = j+1;
318:                     if(ii>=0 && ii<8 && jj>=0 && jj<8){
319:                         if
(*status[ii][jj]==2||*status[ii][jj]==0){*status[ii][jj]+=-1;}
320:                     }
321:                     jj = j-1;
322:                     if(ii>=0 && ii<8 && jj>=0 && jj<8){
323:                         if
(*status[ii][jj]==2||*status[ii][jj]==0){*status[ii][jj]+=-1;}
324:                     }
325:
326:                     ii=i-1;
```



```
327:                     jj=j+2;
328:                     if(ii>=0 && ii<8 && jj>=0 && jj<8 ){
329:                         if
(*status[ii][jj]==2||*status[ii][jj]==0){*status[ii][jj]+=-1;}
330:                     }
331:                     jj=j-2;
332:                     if(ii>=0 && ii<8 && jj>=0 && jj<8 ){
333:                         if
(*status[ii][jj]==2||*status[ii][jj]==0){*status[ii][jj]+=-1;}
334:                     }
335:
336:                     ii=i-2;
337:                     jj=j+1;
338:                     if(ii>=0 && ii<8 && jj>=0 && jj<8){
339:                         if
(*status[ii][jj]==2||*status[ii][jj]==0){*status[ii][jj]+=-1;}
340:                     }
341:                     jj=j-1;
342:                     if(ii>=0 && ii<8 && jj>=0 && jj<8){
343:                         if
(*status[ii][jj]==2||*status[ii][jj]==0){*status[ii][jj]+=-1;}
344:                     }
345:                 }
346:             }
347:         }   // end for loop of j
348:     }   // end for loop of i
349: }
350:
351: // redo the last move
352: void ChessBoard::cancelMove(string origin_from, string origin_to){
353:     // if only the origin move is success, can we do the cancel move
354:     // since this is done by force, no need to check the rule
355:     int from_index[2]={'8'-origin_from[1],origin_from[0]-'A'};
356:     int to_index[2]={'8'-origin_to[1],origin_to[0]-'A'};
357:     cps[from_index[0]][from_index[1]]=cps[to_index[0]][to_index[1]];
358:     cps[to_index[0]][to_index[1]]=temp;
359:     temp=nullptr;
360: }
361:
362: // whether there is a "in check" status, does not check checkmate
363: bool ChessBoard::isCheck(){
364:     // only check if the king is in check
365:     for(int i=0;i<8;i++){
366:         for(int j=0;j<8;j++){
367:             if(cps[i][j]!=nullptr){
368:                 if(cps[i][j]->getSymbol()=='k'){
369:                     if(*status[i][j]==-1 || *status[i][j]==1){
370:                         return true;
371:                     }
372:                 }
373:                 if(cps[i][j]->getSymbol()=='K'){
374:                     if(*status[i][j]==2 || *status[i][j]==1){
375:                         return true;
376:                     }
377:                 }
378:             }
379:         }   // end for loop of j
380:     }   // end for loop of i
381:     return false;
382: }
383:
384: // whether there is a "in check" status, also check checkmate
385: bool ChessBoard::isStatusCheck(){
386:     for(int i=0;i<8;i++){
387:         for(int j=0;j<8;j++){
388:             if(cps[i][j]!=nullptr){
```

You should have a chessboard attribute keeping track of the kings position instead of looping over the board repeatedly to look for it, which is inefficient -2

Also here your use of ints for status instead of enum makes the code very unclear - another developer looking at this would have a hard time understanding it

```
389:                 if(cps[i][j]->getSymbol()=='k'){
390:                     if(*status[i][j]==-1 || *status[i][j]==1){
391:                         string str = isCheckmate(i,j)?"mate":"";
392:                         cout << "Black is in check" << str << endl;
393:                         return true;
394:                     }
395:                 }
396:                 if(cps[i][j]->getSymbol()=='K'){
397:                     if(*status[i][j]==2 || *status[i][j]==1){
398:                         string str = isCheckmate(i,j)?"mate":"";
399:                         cout << "White is in check" << str << endl;
400:                         return true;
401:                     }
402:                 }
403:             }
404:         }   // end for loop pf j
405:     }   // end for loop for i
406:     return false;
407: }
408:
409: // transfer the loaction to FEN format
410: string ChessBoard::getLocation(int x, int y){
411:     string str="  ";
412:     str[0]=(char)(y+'A');
413:     str[1]=(char)('8'-x);
414:     return str;
415: }
416:
417: // try every move to see if the check can be removed
418: bool ChessBoard::tryRemoveCheck(Color color, string FEN){
419:     for(int i=0;i<8;i++){
420:         for(int j=0;j<8;j++){
421:             if(cps[i][j]!=nullptr && cps[i][j]->getColor()==color){
422:                 // try to move to every square
423:                 char file = 'A';
424:                 char rank = '1';
425:                 for(int x=0;x<8;x++){
426:                     for (int y=0;y<8;y++){
427:                         string str_to = "  ";
428:                         str_to[0]=char(file+x);
429:                         str_to[1]=char(rank+y);
430:                         string str_from = getLocation(i,j);
431:                         if(submitMoveWithoutComments(str_from,str_to)==0){
432:                             if(isCheck()==false){   // after move, we can reduce
the check
433:
434:                                 cancelMove(str_from,str_to);
435:                                 return true; // remove the check
436:                             }
437:                             else{
                                    cancelMove(str_from,str_to);    // cancel the
move
438:
439:                             }
440:                         }   // end for loop of y
441:                     }// end for loop of x
442:                 }// end for loop of j
443:             }// end for loop of i
444:         }
445:         return false;   // cannot remove the check
446: }
447:
448: // transfer the chess board to the FEN format
449: string ChessBoard::transferFEN(){
450:     string str = "";
451:     for(int i=0;i<8;i++){
452:         for(int j=0;j<8;j++){
```

Instead of using strings here you should use some kind of struct for position (e.g. struct pos {int rank; int file;}) or at least separate ints, which would be more readable -2


```
453:                 if(cps[i][j]==nullptr){
454:                     str+=".";
455:                 }
456:                 else{
457:                     str+=cps[i][j]->getSymbol();
458:                 }
459:             }
460:             if(i!=7){
461:                 str+="/";
462:             }
463:
464:         }
465:         string str2 = " ";
466:         str2[0]=turn;
467:         str+= " "+str2+" "+castling;
468:         return str;
469: }
470:
471: // whether the check is a checkmate
472: bool ChessBoard::isCheckmate(int x, int y){
473:     int checkmate[2] = {(cps[x][y]->getSymbol()=='K')?2:-1,1};
474:     // check every square around the king
475:     //upper row
476:     if(x-1>=0){
477:         if(y-1>=0 &&
*status[x-1][y-1]!=checkmate[0]&&*status[x-1][y-1]!=checkmate[1]&&cps[x-1][y-1]==nullptr)
{
478:             return false;
479:         }
480:         if
(*status[x-1][y]!=checkmate[0]&&*status[x-1][y]!=checkmate[1]&&cps[x-1][y]==nullptr){
481:             return false;
482:         }
483:         if(y+1<8 &&
*status[x-1][y+1]!=checkmate[0]&&*status[x-1][y+1]!=checkmate[1]&&cps[x-1][y+1]==nullptr)
{
484:             return false;
485:         }
486:     }
487:     // this row
488:     if(y-1>=0 &&
*status[x][y-1]!=checkmate[0]&&*status[x][y-1]!=checkmate[1]&&cps[x][y-1]==nullptr){
489:         return false;
490:     }
491:     if(y+1<8 &&
*status[x][y+1]!=checkmate[0]&&*status[x][y+1]!=checkmate[1]&&cps[x][y+1]==nullptr){
492:         return false;
493:     }
494:     //downer row
495:     if(x+1<8){
496:         if(y-1>=0 &&
*status[x+1][y-1]!=checkmate[0]&&*status[x+1][y-1]!=checkmate[1]&&cps[x+1][y-1]==nullptr)
{
497:             return false;
498:         }
499:         if
(*status[x+1][y]!=checkmate[0]&&*status[x+1][y]!=checkmate[1]&&cps[x+1][y]==nullptr){
500:             return false;
501:         }
502:         if(y+1<8 &&
*status[x+1][y+1]!=checkmate[0]&&*status[x+1][y+1]!=checkmate[1]&&cps[x+1][y+1]==nullptr)
{
503:             return false;
504:         }
505:     }
506:     // whether move a piece can change the status
```

```
507:     if(tryRemoveCheck(cps[x][y]->getColor(),transferFEN())){
508:         // can remove the check, return false ( is not a checkmate )
509:         return false;
510:     }
511:     // cannot remove the check, return true ( is a checkmate )
512:     return true;
513: }
514:
515: // print the status of each square
516: void ChessBoard::printStatus(){
517:     for(int i=0;i<8;i++){
518:         for(int j=0;j<8;j++){
519:             cout << *status[i][j] << "\t";
520:         }
521:         cout << endl;
522:     }
523: }
524:
525: // the background logic try to move
526: int ChessBoard::submitMoveWithoutComments(string from, string to){
527:     // 1. check input valid
528:     if(from.length()!=2 || to.length()!=2){
529:         return -1;
530:     }
531:
532:     int from_index[2]={'8'-from[1],from[0]-'A'};
533:     int to_index[2]={'8'-to[1],to[0]-'A'};
534:
535:     // 2. check if from has a chesspiece
536:     if(cps[from_index[0]][from_index[1]]==nullptr){
537:         return -1;
538:     }
539:
540:     // 4. check if has piece and with the same side
541:     if(cps[to_index[0]][to_index[1]]!=nullptr &&
cps[from_index[0]][from_index[1]]->getColor()==cps[to_index[0]][to_index[1]]->getColor())
{
542:         return -1;
543:     }
544:
545:     // has no chesspiece, check the move logic
546:     // has chesspiece, both check the move logic and if the chesspiece can be
eaten
547:     if
(cps[from_index[0]][from_index[1]]->move(from_index,to_index,cps,status)==-1){
548:         return -1;
549:     }
550:
551:     // if success
552:     if(cps[to_index[0]][to_index[1]]!=nullptr){
553:         temp = cps[to_index[0]][to_index[1]];
554:     }
555:     else{
556:         temp = nullptr;
557:     }
558:
559:     cps[to_index[0]][to_index[1]]=cps[from_index[0]][from_index[1]];
560:
561:     cps[from_index[0]][from_index[1]]=nullptr;
562:     // no need to switch turn
563:     updateStatus();
564: }
565:
566: // the user submit move
567: int ChessBoard::submitMove(string from, string to){
568:     // 1. check input valid
```



```
569:     if(from.length()!=2 || to.length()!=2){
570:         cout << "invalid move input" << endl;
571:         return -1;
572:     }
573:     int from_index[2]={'8'-from[1],from[0]-'A'};
574:     int to_index[2]={'8'-to[1],to[0]-'A'};
575:
576:     // 2. check if from has a chesspiece
577:     if(cps[from_index[0]][from_index[1]]==nullptr){
578:         cout << "There is no piece at position " << from <<"!" << endl;
579:         return -1;
580:     }
581:     // 3. check turn
582:     Color turn_color = (turn=='w')?White:Black;
583:     if((char)tolower(cps[from_index[0]][from_index[1]]->getColor())!=turn_color){
584:         string turn_color = turn=='w'?"Black":"White";
585:         cout <<"It is not "<< turn_color <<"'s turn to move!" << endl;
586:         return -1;
587:     }
588:
589:     string taking="";
590:     // 4. check if has piece and with the same side
591:     if(cps[to_index[0]][to_index[1]]!=nullptr &&
cps[from_index[0]][from_index[1]]->getColor()==cps[to_index[0]][to_index[1]]->getColor())
{
592:         cout << colorToString(cps[from_index[0]][from_index[1]]->getColor()) <<
"'s " << cps[from_index[0]][from_index[1]]->getType() << " cannot move to " << to << "!"
<< endl;
593:         return -1;
594:     }
595:
596:     if(cps[to_index[0]][to_index[1]]!=nullptr){
597:         taking = " taking "
+colorToString(cps[to_index[0]][to_index[1]]->getColor()) + "'s "
+cps[to_index[0]][to_index[1]]->getType();
598:     }
599:
600:     // has no chesspiece, check the move logic
601:     // has chesspiece, both check the move logic and if the chesspiece can be
eaten
602:     if
(cps[from_index[0]][from_index[1]]->move(from_index,to_index,cps,status)==-1){
603:         cout << colorToString(cps[from_index[0]][from_index[1]]->getColor()) <<
"'s " << cps[from_index[0]][from_index[1]]->getType() << " cannot move to " << to << "!"
<< endl;
604:         return -1;
605:     }
606:
607:     // if success
608:     cps[to_index[0]][to_index[1]]=cps[from_index[0]][from_index[1]];
609:     cps[from_index[0]][from_index[1]]=nullptr;
610:
611:     cout << colorToString(cps[to_index[0]][to_index[1]]->getColor()) << "'s " <<
cps[to_index[0]][to_index[1]]->getType() << " moves from " << from << " to " << to <<
taking << endl;
612:     turn = (turn=='w')?'b':'w'; // switch turn
613:     updateStatus();
614:     isStatusCheck();
615:
616: }
617:
618: // load the state of the chess board
619: void ChessBoard::loadState(string FEN){
620:     // sample: "rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq"
621:     string FEN1 = FEN.substr(0,FEN.find(' '));
622:     string FEN2 = FEN.substr(FEN.find(' ')+1, FEN.length());
```

There is no check for whether the characters in from/to are valid (i.e. within the board) -3

```
623:     vector<string> tokens = splitString(FEN1, '/');
624:
625:     //load the chessboard
626:     for(int i=0;i<8;i++){
627:         for(int j=0;j<tokens[i].length();j++){
628:             // if 8, go to the next line
629:             if(tokens[i][j]=='8'){
630:                 break;
631:             }
632:             else if(tokens[i][j]==' ' || tokens[i][j]=='.'){
633:                 // cout << "leave this space" << endl;
634:             }
635:             else{
636:                 Color color = (islower(tokens[i][j])?Black:White);
637:                 switch((char)tolower(tokens[i][j])){
638:                     case 'p':
639:                         cps[i][j] = new Pawn(color);
640:                         cps[i][j]->setSymbol(tokens[i][j]);
641:                         break;
642:                     case 'n':
643:                         cps[i][j] = new Knight(color);
644:                         cps[i][j]->setSymbol(tokens[i][j]);
645:                         break;
646:                     case 'b':
647:                         cps[i][j] = new Bishop(color);
648:                         cps[i][j]->setSymbol(tokens[i][j]);
649:                         break;
650:                     case 'r':
651:                         cps[i][j] = new Rook(color);
652:                         cps[i][j]->setSymbol(tokens[i][j]);
653:                         break;
654:                     case 'q':
655:                         cps[i][j] = new Queen(color);
656:                         cps[i][j]->setSymbol(tokens[i][j]);
657:                         break;
658:                     case 'k':
659:                         cps[i][j] = new King(color);
660:                         cps[i][j]->setSymbol(tokens[i][j]);
661:                         break;
662:                     default:
663:                         break;
664:                 }   // end switch
665:             }   // end if-else
666:         }   // end for loop of j
667:     }   // end for loop of i
668:     cout << "A new board state is loaded!\n";
669:
670:     //load the turn and castling info
671:     turn = FEN2[0];
672:     castling=FEN2.substr(FEN2.find(' ')+1,FEN2.length());
673:
674:     updateStatus();
675: }
676:
677: // print each piece of the chess board
678: void ChessBoard::printBoard(){
679:     for(int i=0;i<8;i++){
680:         for(int j=0;j<8;j++){
681:             if(cps[i][j]!=nullptr)
682:                 cout << *cps[i][j];
683:             else{
684:                 cout << " ";
685:             }
686:         }
687:         cout  << endl;
```



```
688:     }
689: }
690:
691: int ChessBoard::castling_move(ChessPiece* piece, string from, string to, ChessPiece* cb[8][8], int* status[8][8]){
692:     if(castling.find("piece")==-1){
693:         // invalid castling move
694:         return -1;
695:     }
696:     // if the king/queen or the rook has been moved
697:     if(!piece->getFirstMove()){
698:         return -1;
699:     }
700:
701:     // the king/queen is in check
702:     if(from.length()!=2 || to.length()!=2){
703:         cout << "invalid move input" << endl;
704:         return -1;
705:     }
706:     int from_index[2]={'8'-from[1],from[0]-'A'};
707:     int to_index[2]={'8'-to[1],to[0]-'A'};
708:     int attacked[2]={0,0};
709:     attacked[0]=(piece->getColor()==Black)?2:-1;
710:     attacked[1]=1;
711:     if(*status[from_index[0]][from_index[1]]==attacked[0] || *status[from_index[0]][from_index[1]]==attacked[1]){
712:         return -1;
713:     }
714:
715:     // if after castling, the king will be attacked
716:     if(*status[to_index[0]][to_index[1]]==attacked[0] || *status[to_index[0]][to_index[1]]==attacked[1]){
717:         return -1;
718:     }
719:
720:     // if there is a piece between the king/queen and the rook
721:     if(*from_index == *to_index){
722:         int step= (*(from_index+1)< *(to_index+1))?1:-1;
723:         int y = *(from_index+1) +step;
724:         while(y<8 && y>=0 && y!=*(to_index+1)){
725:             if(cb[*from_index][y]!=nullptr){
726:                 // if there is piece, cannot move
727:                 return -1;
728:             }
729:             y+=step;
730:         }
731:     }
732:
733:     // castling
734:     return 0;
735: }
736:
737: // defined string split function
738: vector<string> splitString(const string& input, char delimiter) {
739:     vector<string> result;
740:     size_t start = 0;
741:     size_t end = input.find(delimiter);
742:
743:     while (end != string::npos) {
744:         result.push_back(input.substr(start, end - start));
745:         start = end + 1;
746:         end = input.find(delimiter, start);
747:     }
748:
749:     // Add the last substring after the last delimiter
750:     result.push_back(input.substr(start));
```

```
751:
752:     return result;
753: }
754:
755: string colorToString(Color c){
756:     switch (c) {
757:         case Black:
758:             return "Black";
759:         case White:
760:             return "White";
761:         default:
762:             return "UNKNOWN";
763:     }
764: }
765:
```

Would have been better to overload the << operator -2


```
1: #include <iostream>
2: #include "ChessPiece.h"
3: #include <string>
4: #include "Color.h"
5:
6: /*
7:  * @Author: shihan
8:  * @Date: 2023-12-02 18:30:46
9:  * @version: 1.0
10:  * @description: implementation of chess piece
11:  */
12:
13: using namespace std;
14:
15: // ChessPiece class
16: ChessPiece::ChessPiece(){
17:     symbol = ' ';
18:     first_move = true;
19: }
20:
21: ChessPiece::~ChessPiece(){}
22:
23: ChessPiece::ChessPiece(Color _color):color(_color){
24:     first_move = true;
25: }
26:
27: int ChessPiece::move(int *from_index, int *to_index, ChessPiece* cb[8][8], int* ↙
status[8][8]){
28:     cout << "general chess piece move\n";
29:     return 0;
30: }
31:
32: void ChessPiece::setSymbol(char _symbol){
33:     symbol = _symbol;
34: }
35:
36: void ChessPiece::setColor(Color _color){
37:     this->color=_color;
38: }
39:
40: void ChessPiece::setFirstMove(bool _firstMove){
41:     this->first_move = _firstMove;
42: }
43:
44: Color ChessPiece::getColor(){
45:     return color;
46: }
47:
48: char ChessPiece::getSymbol(){
49:     return symbol;
50: }
51:
52: bool ChessPiece::getFirstMove(){
53:     return first_move;
54: }
55:
56:
57: // King class
58: int King::move(int *from_index, int *to_index, ChessPiece* cb[8][8], int* ↙
status[8][8]){
59:     int diff = (*from_index)*10 + (*(from_index+1)) - (*to_index)*10 - ↙
(*(to_index+1));
60:     if(abs(diff)!=1 && abs(diff)!=10){
61:         // it is not a square
62:         return -1;
63:     }
```

It is not good to pass the chessboard raw pointers to the chesspiece. Access should be done via getters that return necessary information - passing raw pointers around is dangerous because it means the chess pieces can edit the board, which we don't want! -3

```
64:     else{
65:         // if the move will make the king be attacked
66:         int attacked = (symbol=='k')?-1:2;
67:         if
(*status[*to_index][*(to_index+1)]==1||*status[*to_index][*(to_index+1)]==attacked){
68:             return -1;
69:         }
70:         return 0;
71:     }
72:
73: }
74:
75: King::King(Color _color) : ChessPiece(_color) {}
76:
77: string King::getType() const{
78:     return "King";
79: }
80:
81:
82: // Rook class
83: int Rook::move(int *from_index, int *to_index,ChessPiece* cb[8][8], int*
status[8][8]){
84:     // 1. check if the move obeys the rule
85:     if(*from_index!=*to_index && *(from_index+1)!=*(to_index+1)){
86:         return -1;
87:     }
88:     // 2. check if there is pieces between the from and the to
89:     if(*from_index == *to_index){
90:         int step= (*(from_index+1)< *(to_index+1))?1:-1;
91:         int y = *(from_index+1) +step;
92:         while(y<8 && y>=0 && y!=*(to_index+1)){
93:             if(cb[*from_index][y]!=nullptr){
94:                 // if there is piece, cannot move
95:                 return -1;
96:             }
97:             y+=step;
98:         }
99:     }
100:    if(*(from_index+1) == *(to_index+1)){
101:        int step= (*(from_index)< *(to_index))?1:-1;
102:        int x = *(from_index) +step;
103:        while(x<8 && x>=0 && x!=*(to_index)){
104:            if(cb[x][*(from_index+1)]!=nullptr){
105:                return -1;
106:            }
107:            x+=step;
108:        }
109:    }
110:
111:    //valid move
112:    return 0;
113: }
114:
115: Rook::Rook(Color _color):ChessPiece(_color){}
116:
117: string Rook::getType() const{
118:     return "Rook";
119: }
120:
121:
122: // Bishop class
123: int Bishop::move(int *from_index, int *to_index, ChessPiece* cb[8][8], int*
status[8][8]){
124:     // 1. check if the input obeys the rule
125:     if(abs(*from_index - *to_index)!=abs(*(from_index+1) - *(to_index+1))){
126:         return -1;
```

Checking for the presence of pieces between the source and destination is a common process shared by all of the pieces. This logic should be abstracted into the parent piece class, with the individual piece providing information on which squares it will pass during its move. The parent piece class should query the chess board to find out if the squares are unoccupied rather than accessing the raw pointers -5


```
127:     }
128:     // 2. check if there is pieces between the from and the to
129:     int steps = abs(*from_index - *to_index);
130:     int each_x = (*to_index - *from_index)/steps;
131:     int each_y = (*(to_index+1) - *(from_index+1))/steps;
132:     int index_x = *from_index;
133:     int index_y = *(from_index+1);
134:     for(int i=0;i<steps-1;i++){
135:         index_x+=each_x;
136:         index_y+=each_y;
137:         if(cb[index_x][index_y]!=nullptr){
138:             return -1;
139:         }
140:     }
141:     // valid move
142:     return 0;
143:
144: }
145:
146: Bishop::Bishop(Color _color):ChessPiece(_color){}
147:
148: string Bishop::getType() const {
149:     return "Bishop";
150: }
151:
152:
153: // Queen class
154: Queen::Queen(Color _color):ChessPiece(_color){}
155:
156: string Queen::getType() const{
157:     return "Queen";
158: }
159:
160: int Queen::move(int *from_index, int *to_index, ChessPiece* cb[8][8], int*
status[8][8]){
161:     // 1. check if it is a rook's rule or a bishop's rule
162:     // for a rook:
163:     if(*from_index == *to_index){
164:         int step= (*(from_index+1)< *(to_index+1))?1:-1;
165:         int y = *(from_index+1) +step;
166:         while(y<8 && y>=0 && y!=*(to_index+1)){
167:             if(cb[*from_index][y]!=nullptr){
168:                 return -1;
169:             }
170:             y+=step;
171:         }
172:         return 0;
173:     }
174:     else if(*(from_index+1) == *(to_index+1)){
175:         int step= (*(from_index)< *(to_index))?1:-1;
176:         int x = *(from_index) +step;
177:         while(x<8 && x>=0 && x!=*(to_index)){
178:             if(cb[x][*(from_index+1)]!=nullptr){
179:                 return -1;
180:             }
181:             x+=step;
182:         }
183:         return 0;
184:     }
185:     // for a bishop
186:     else if(abs(*from_index - *to_index)==abs(*(from_index+1) - *(to_index+1))){
187:         int steps = abs(*from_index - *to_index);
188:         int each_x = (*to_index - *from_index)/steps;
189:         int each_y = (*(to_index+1) - *(from_index+1))/steps;
190:         int index_x = *from_index;
191:         int index_y = *(from_index+1);
```

This is clear duplication with the rook and bishop class - should have been abstracted into a function -3

```
192:         for(int i=0;i<steps-1;i++){
193:             index_x+=each_x;
194:             index_y+=each_y;
195:             if(cb[index_x][index_y]!=nullptr){
196:                 return -1;
197:             }
198:         }
199:         return 0;
200:     }
201:
202:     // else, the logic is wrong and no valid move
203:     return -1;
204: }
205:
206:
207: // Knight class
208: Knight::Knight(Color _color):ChessPiece(_color){}
209:
210: string Knight::getType() const {
211:     return "Knight";
212: }
213:
214: int Knight::move(int *from_index, int *to_index, ChessPiece* cb[8][8], int* ⬈
status[8][8]){
215:     int diff_x = abs(*from_index - *to_index);
216:     int diff_y = abs(*(from_index+1)- *(to_index+1));
217:     // check whether the move follows the rule
218:     if(diff_x + diff_y !=3){
219:         return -1;
220:     }
221:     if(diff_x ==2 && diff_y ==1){
222:         return 0;
223:     }
224:     if(diff_x ==1 && diff_y ==2){
225:         return 0;
226:     }
227:     return -1;
228: }
229:
230:
231: // Pawn class
232: Pawn::Pawn(Color _color):ChessPiece(_color){}
233:
234: string Pawn::getType() const{
235:     return "Pawn";
236: }
237:
238: int Pawn::move(int *from_index, int *to_index, ChessPiece* cb[8][8], int* ⬈
status[8][8]){
239:     // for Black's Pawn
240:     if(symbol == 'p'){
241:         // one step is okay
242:         if(*(from_index+1)==*(to_index+1) && (*from_index +1 == *to_index) && ⬈
cb[*to_index][*(to_index+1)]==nullptr ){
243:             return 0;
244:         }
245:         // two steps need to check if overleap, if it is the first move
246:         if(first_move && *(from_index+1)==*(to_index+1) && (*from_index +2 == ⬈
*to_index) && cb[*from_index +1][*(from_index+1)]==nullptr && ⬈
cb[*to_index][*(to_index+1)]==nullptr ){
247:             first_move = false;
248:             return 0;
249:         }
250:         // diagonally in front of it and have piece
251:         if((*from_index +1 == *to_index)){
252:             if(abs(*(from_index+1) - *(to_index+1))==1 && ⬈
```



```
cb[*(to_index)][*(to_index+1)]!=nullptr && cb[*(to_index)][*(to_index+1)]->getColor()!= ⬈
cb[*(from_index)][*(from_index+1)]->getColor()){
253:                 return 0;
254:             }
255:         }
256:
257:     }
258:
259:     // for White's Pawn
260:     else if(symbol == 'P'){
261:         // one step is okay
262:         if(*(from_index+1)==*(to_index+1) && (*from_index -1 == *to_index) && ⬈
cb[*to_index][*(to_index+1)]==nullptr){
263:             return 0;
264:         }
265:         // two steps need to check if overleap, if it is the first move
266:         if(first_move && *(from_index+1)==*(to_index+1) && (*from_index -2 == ⬈
*to_index) && cb[*from_index -1][*(from_index+1)]==nullptr && ⬈
cb[*to_index][*(to_index+1)]==nullptr){
267:             first_move = false;
268:             return 0;
269:         }
270:         // diagonally in front of it and have piece
271:         if((*from_index -1 == *to_index)){
272:             if(abs(*(from_index+1) - *(to_index+1))==1 && ⬈
cb[*(to_index)][*(to_index+1)]!=nullptr && cb[*(to_index)][*(to_index+1)]->getColor()!= ⬈
cb[*(from_index)][*(from_index+1)]->getColor()){
273:                 return 0;
274:             }
275:         }
276:
277:     }
278:
279:     // else, the move is invalid
280:     return -1;
281: }
282:
```

```cpp
/*
 * @Author: shihan
 * @Date: 2023-11-22 21:38:19
 * @version: 1.0
 * @description:
 */
#include"ChessBoard.h"

#include<iostream>

using std::cout;

void location();
void testQueen();
void testKnight();
void testPawn();
void testCancelMove();
void testtransferFEN();

int main() {

  // test
  // 1. test location transformation
  // location();
  // 2. test piece move
  // testQueen();
  // testKnight();
  // testPawn();
  // testCancelMove();
  // 3. test helper function
  // testtransferFEN();



  // cout << "========================\n";
  // cout << "Testing the Chess Engine\n";
  // cout << "========================\n\n";

  ChessBoard cb;
  // cb.loadState("rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq");
  // cb.loadState("qqqqqqqq/8/8/8/8/8/8/RNBQKBNR w KQkq");
  cout << '\n';
  // A new board state is loaded!

  // // It is not Blackâ\200\231s turn to move!
  // cb.submitMove("D7", "D6");
  // cout << '\n';

  // // There is no piece at position D4!
  // cb.submitMove("D4", "H6");
  // cout << '\n';

  // // Whiteâ\200\231s Pawn moves from D2 to D4
  // cb.submitMove("D2", "D4");
  // cout << '\n';

  // // cb.submitMove("A8", "B4");
  // // Black's rook cannot move to B4


  // // Blackâ\200\231s Bishop cannot move to B4!
  // cb.submitMove("F8", "B4");
  // cout << '\n';

  // cout << "========================\n";
  // cout << "Alekhine vs. Vasic (1931)\n";
```



```cpp
  // cout << "========================\n\n";

  // ↙
cb.loadState("rnbqk.nr/ppp..ppp/....p.../...p..../...PP.../..bB..../PPP..PPP/R.BQK.NR w ↙
KQkq");
  // cb.loadState("....b.../8/8/8/8/8/8/8 w KQkq");
  //  cb.printBoard();
  // cb.printStatus();


  // // A new board state is loaded!
  cb.loadState("rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq");
  cout << '\n';

  cb.submitMove("E2", "E4");
  cb.submitMove("E7", "E6");
  // cb.printBoard();
  cout << '\n';

  cb.submitMove("D2", "D4");
  cb.submitMove("D7", "D5");
  // cb.printBoard();
  cout << '\n';

  cb.submitMove("B1", "C3");
  cb.submitMove("F8", "B4");
  // cb.printBoard();
  cout << '\n';

  cb.submitMove("F1", "D3");
  cb.submitMove("B4", "C3");
  // // Blackâ\200\231s Bishop moves from B4 to C3 taking Whiteâ\200\231s Knight
  // // White is in check

  cout << '\n';



  // Whiteâ\200\231s Pawn moves from B2 to C3 taking Blackâ\200\231s Bishop
  cb.submitMove("B2", "C3");
  cb.submitMove("H7", "H6");
  // cb.printBoard();
  cout << '\n';

  cb.submitMove("C1", "A3");
  cb.submitMove("B8", "D7");
  // cb.printBoard();
  cout << '\n';

  cb.submitMove("D1", "E2");
  // Blackâ\200\231s Pawn moves from D5 to E4 taking Whiteâ\200\231s Pawn
  cb.submitMove("D5", "E4");
  // cb.printBoard();
  cout << '\n';

  // Whiteâ\200\231s Bishop moves from D3 to E4 taking Blackâ\200\231s Pawn
  cb.submitMove("D3", "E4");
  cb.submitMove("G8", "F6");
  // cb.printBoard();
  cout << '\n';

  cb.submitMove("E4", "D3");
  cb.submitMove("B7", "B6");
  cout << '\n';

  cb.submitMove("E2", "E6");
```

```
131:   // Whiteâ\200\231s Queen moves from E2 to E6 taking Blackâ\200\231s Pawn
132:   // Black is in check
133:   // cb.printBoard();
134:   // cb.printStatus();
135:
136:   //Blackâ\200\231s Pawn moves from F7 to E6 taking Whiteâ\200\231s Queen
137:   cb.submitMove("F7", "E6");
138:   cout << '\n';
139:
140:   // Whiteâ\200\231s Bishop moves from D3 to G6
141:   // Black is in checkmate
142:   cb.submitMove("D3", "G6");
143:
144:   // cb.submitMove("E8","E7");
145:   // cb.printBoard();
146:   // cb.printStatus();
147:   cout << '\n';
148:
149:   return 0;
150: }
151:
152: void location(){
153:     int x=3,y=3;
154:     string str;
155:     cout << (char)(y+'A') << endl;
156:     cout << (char)('8'-x) << endl;
157:     str[0]=(char)(y+'A');
158:     str[1]=(char)('8'-x);
159:     cout << str << endl;
160:     // string str = (char)(y+'A')+""+(char)('8'-x);
161:     // cout << str << endl;
162: }
163:
164: void testQueen(){
165:     ChessBoard cb;
166:
167:   // cb.loadState("8/8/8/8/...Q..../8/8/8 w KQkq");
168:     // cb.submitMove("D4","D3"); // ok
169:     // cb.submitMove("D4","D5"); // ok
170:     // cb.submitMove("D4","B4"); // ok
171:     // cb.submitMove("D4","H4"); // ok
172:     // cb.submitMove("D4","B6"); // ok
173:     // cb.submitMove("D4","H8"); // ok
174:     // cb.submitMove("D4","A1"); // ok
175:     // cb.submitMove("D4","G1"); // ok
176:
177:
178:     cb.loadState("8/8/8/.....r../...Q..../8/8/8 w KQkq");
179:     // cb.submitMove("D4","H4"); // not ok
180:     // cb.submitMove("D4","E6"); // not ok
181:     cb.submitMove("D4","A3"); // not ok
182:
183:
184:     // cb.printStatus();
185: }
186:
187: void testKnight(){
188:     ChessBoard cb;
189:     cb.loadState("8/8/8/.....r../...N.r../8/8/8 w KQkq");
190:     // cb.submitMove("D4","F5"); // ok
191:     // cb.submitMove("D4","E6"); // ok
192:     // cb.submitMove("D4","B3"); // ok
193:     // cb.submitMove("D4","F4"); // not ok
194:     cb.submitMove("D4","F5"); // ok
195:
196: }
```



```
197:
198: void testPawn(){
199:     ChessBoard cb;
200:     // cb.loadState("8/..p...../8/8/8/8/8/8 b KQkq");
201:     // cb.submitMove("C7","C6"); // ok
202:     // cb.submitMove("C7","C5"); // ok
203:     cb.submitMove("C7","C8"); // not ok
204:
205:     // cb.loadState("8/..p...../..R...../8/8/8/8/8 b KQkq");
206:     // cb.submitMove("C7","C5"); // not ok
207:     // cb.submitMove("C7","C6"); // ok
208:
209:     cb.loadState("8/..p...../.R...../8/8/8/8/8 b KQkq");
210:     cb.submitMove("C7","D6"); // not ok
211:     // cb.submitMove("C7","B6"); // ok
212:
213: }
214:
215: void testCancelMove(){
216:     ChessBoard cb;
217:     cb.loadState("8/..p...../.R...../8/8/8/8/8 b KQkq");
218:     cb.submitMove("C7","B6"); // ok
219:     cb.printBoard();
220: }
221:
222: void testtransferFEN(){
223:     ChessBoard cb;
224:     cb.loadState(
"......../..p...../.R...../......../......../......../......../....... b KQkq ");
225:     cout << cb.transferFEN() << endl;
226: }
227:
228:
229:
```

```cpp
/*
 * @Author: shihan
 * @Date: 2023-12-03 12:31:51
 * @version: 1.0
 * @description:
 */
#include <iostream>
#include <string>
#include <cstdlib> // For abs function (integer)
#include "ChessBoard.h"
using namespace std;

void move(string from, string to){
    cout << from[0]-'A'<< endl;
    int from_index[2]={'8'-from[1],from[0]-'A'};
    cout<< from_index[0]<< from_index[1]<< endl;

}

void int_array_to_int(int *from, int *to){
    int diff = (*from)*10 + (*(from+1)) - (*to)*10 - (*(to+1));
    cout << abs(diff) << endl;
    if(abs(diff)!=1 && abs(diff)!=10){
        cout << "break the king's rule"<< endl;
    }
    else{
        cout<<"okay for king"<< endl;
    }
    // cout << *to << endl;
    // cout << *(to+1) << endl;
    // cout << (*to)*10 + (*to+1)<< endl;
}

void check_line(int *from, int *to){
    if(abs(*from - *to)==abs(*(from+1) - *(to+1))){
        cout << "ok" << endl;
    }
    else{
        cout << "not ok" << endl;
    }

    int steps = abs(*from - *to);
    int each_x = (*to - *from)/steps;
    int each_y = (*(to+1) - *(from+1))/steps;
    cout << each_x;
    cout << " " << each_y << endl;

    int index_x = *from;
    int index_y = *(from+1);
    for(int i=0;i<steps;i++){
        index_x+=each_x;
        index_y+=each_y;
        cout << index_x << index_y << endl;
    }

}
int main(){
    ChessBoard cb;
  cb.loadState("8/8/......Q./8/8/8/8/8 w KQkq");
    cb.printStatus();

    return 0;
}
```



```cpp
/*
 * @Author: shihan
 * @Date: 2023-12-02 18:46:21
 * @version: 1.0
 * @description:
 */
#include <iostream>
#include <vector>
#include <string>
#include"ChessBoard.h"

using namespace std;


// vector<string> splitString(const string& input, char delimiter) {
//     vector<string> result;
//     size_t start = 0;
//     size_t end = input.find(delimiter);

//     while (end != string::npos) {
//         result.push_back(input.substr(start, end - start));
//         start = end + 1;
//         end = input.find(delimiter, start);
//     }

//     // Add the last substring after the last delimiter
//     result.push_back(input.substr(start));

//     return result;
// }

class Test{
    public:
        Test(){cout << "test\n";}
};

void testChessBoard();
void testCheck();
int main() {

    // testChessBoard();
    testCheck();
    // cout << islower('8')<< endl;


    // for (const auto& token : tokens) {
    //     cout << token << endl;
    // }
    // delete cb;
    return 0;
}

void testSubstr(){
    string s="hello world";
    char delimiter = ' ';
    string token = s.substr(s.find(delimiter)+1, s.length()); // token is "scott"
    cout << token;
}

void testCheck(){
    ChessBoard cb;
    cb.loadState("rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq");
  cout << '\n';

  cb.submitMove("E2", "E4");
  cb.submitMove("E7", "E6");
```

```
67:   cout << '\n';
68:
69:   cb.submitMove("D2", "D4");
70:   cb.submitMove("D7", "D5");
71:   cout << '\n';
72:
73:   cb.submitMove("B1", "C3");
74:   cb.submitMove("F8", "B4");
75:   cout << '\n';
76:
77:   cb.submitMove("F1", "D3");
78:   cb.submitMove("B4", "C3");
79:     cb.printBoard();
80: }
81: void testChessBoard(){
82:     ChessBoard cb;
83:   cb.loadState("rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq");
84:   // cb.submitMove("D7", "D6");
85:   // cb.submitMove("D4", "H6");
86:   cb.submitMove("D2", "D4");
87:     cb.printBoard();
88:
89:
90: }
91: void testTypeid(){
92:     Test t;
93:     cout << typeid(t).name()<< endl;;
94: }
95: void testSplitString(){
96:     string input = "Hello,World,This,Is,C++";
97:     vector<string> tokens = splitString(input, ',');
98:
99:     cout <<  tokens[0][0] << endl;
100:    for(int i=0;i<2;i++){
101:        cout << tokens[i].length() << endl;
102:        for(int j=0;j<tokens[i].length();j++){
103:            cout<< tokens[i][j] << endl;
104:        }
105:    }
106: }
```



```
1: // File: colors.cpp
2:
3: #include "Color.h"
4:
5: #include <string>
6: using namespace std;
7: // Implementation of colorToString function
8:
9: string colorToString(Color c){
10:     switch (c) {
11:         case Black:
12:             return "Black";
13:         case White:
14:             return "White";
15:         default:
16:             return "UNKNOWN";
17:     }
18: }
```

```
chess: ChessBoard.o ChessPiece.o Color.o ChessMain.o
  g++ -Wall -g ChessBoard.o ChessPiece.o ChessMain.o -o chess

Color.o : Color.cpp Color.h
  g++ -Wall -g -c Color.cpp

ChessPiece.o: ChessPiece.cpp ChessPiece.h
  g++ -Wall -g -c ChessPiece.cpp

ChessBoard.o: ChessBoard.cpp ChessBoard.h
  g++ -Wall -g -c ChessBoard.cpp

ChessMain.o: ChessMain.cpp ChessBoard.h ChessPiece.h Color.h
  g++ -Wall -g -c ChessMain.cpp

clean:
  rm -rf *.o ChessMain

```