Lab 6: Conditional Statements

Due date: Monday, January 30, 11:59pm.

Lab Assignment

Assignment Preparation

This is a pair programming assignment. For this assignment, you pick your own partners. The only condition is that you must pick a different partner than the one you worked with on **Lab 5**.

Collaboration. Students work in pairs, and it is considered cheating, if members of the team (pair) do not work together. Communication between pairs during lab time is allowed, but no direct sharing of code is allowed.

Purpose. The lab allows you to practice the use of conditional statements. It also facilitates learning the use of boolean expressions as conditions in `if` and `switch` statements.

Programming Style. All submitted C programs must adhere to the programming style described in detail at

http://users.csc.calpoly.edu/∼cstaley/General/CStyle.htm

When graded, the programs will be checked for style. Any stylistic violations are subject to a 10% penalty. Significant stylistic violations, epsecially those that make grading harder, may yield stricter penalties. Also note the the Lab 2 requirement for the content of the header comment in each file you submit applies **to each assignment** (lab, programming assignment, homework) in this course.

Testing and Submissions. Any submission that does not compile using the

```
gcc -ansi -Wall -Werror -lm
```

compiler settings will receive an automatic score of 0.

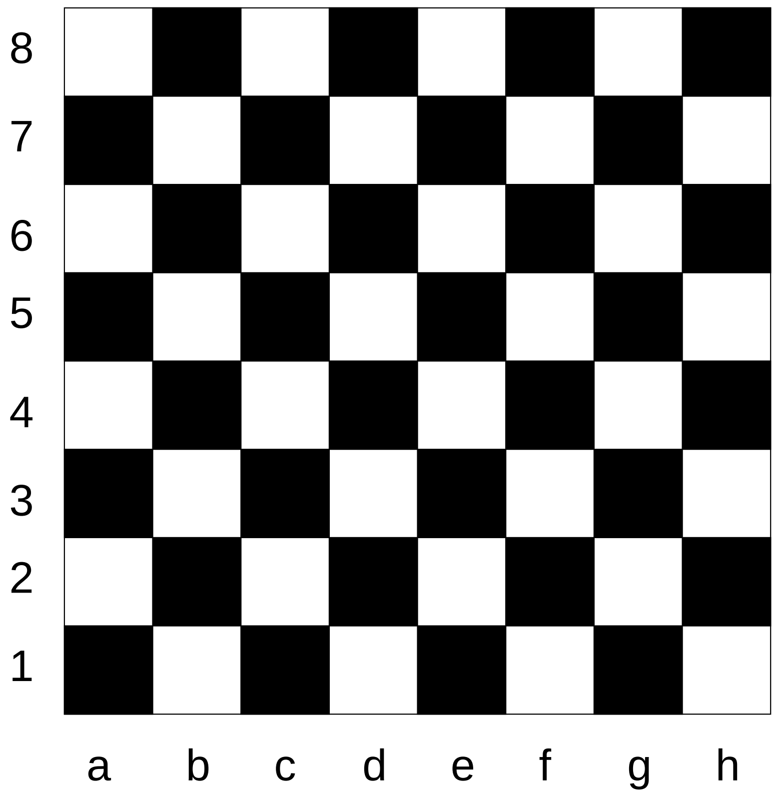

Figure 1: The chessboard.

The Task

Part 1: Chess piece moves.

The game of chess is played on an 8×8 chessboard (Figure ??). The columns, referred to as *files* are identified by letters `a` through `h` (left to right). The rows, referred to as *ranks* are numbered 1 through 8 (bottom to top).

There are six different types of game pieces in chess: pawn, knight, bishop, rook, queen and king. Each chess piece moves according to a special set of rules.

The goal of this assignment is to create a collection of functions that test if a specific move by a given chess piece is legal or not.

CR-1. Organization. You shall create six functions named

```
int isLegalMovePawn(...)
int isLegalMoveKnight(...)
int isLegalMoveBishop(...)
int isLegalMoveRook(...)
int isLegalMoveQueen(...)
int isLegalMoveKing(...)
```

The specifications for each functions are given below. All functions shall be declared in a file called `chess.h` (which should also be used to `#define` any constants) and defined in a file called `chess.c`.

Additionally, you shall create a C file `moveTest.c` which shall contain an `int main()` function that uses `checkit.h` macros to unit-test each individual function. There shall be at least 10 tests for each function, and you shall make sure to test all execution paths in each of the functions[1]. The tests of each function shall be separated by five printed lines as follows:

```

----------------------------------
Testing <ChessPiece> Moves
----------------------------------

```

where `<ChessPiece>` is one of `Pawn`, `Knight`, `Bishop`, `Rook`, `Queen`, `King`. The first and the last lines are empty.

A template `moveTest.c` is provided to you.

To compile your program, use the following command:

```
> gcc -ansi -Wall -Werror -lm -o chessTest chessTest.c chess.c
```

CR-2. Chess piece moves. In chess, pieces can be either black or white. For knights, bishops, rooks, queens and kings, the legality of a move does not depend on their color. For pawns the legality of the move depends on what color they are.

Your functions will be determing the legality of a move under the assumption that the chess piece is the only piece on the board. Thus, there will be no obstructions to any moves, and there will be no moves of one piece taking another.

Pawn moves. A **white pawn** can be located on ranks $2 - - - 8$ on any of the files of the board. A legal move advances a pawn from rank i to rank $i + 1$. Additionally, a white pawn located on rank 2 can be legally advanced to rank 4.

A **black pawn** can be located on ranks $1 - - - 7$ of the board. A legal move advances a pawn from rank i to rank $i - 1$. Additionally, a black pawn located on rank 7 can be legally advanced to rank 5.

Knight moves. A knight located on chessboard cell xi can legally move to a cell that is located:

- two ranks away from it along the file (vertically) and one file away from it along the rank (horizontally) in *any* direction, or
- two files away from it along the rank (horizontally) and one rank away from it along the file (vertically) in *any* direction.

A knight located in the central part of the board has a total of eight legal moves. E.g., a knight in position d4 can move legally to b3, b5, c2, c6, e2, e6, f3, f5.

[1]Which may mean more than 10 tests in some cases.

Bishop moves. A bishop located on any chessboard cell can legally move to any other cell that lies on the same diagonal as the cell is occupies.

For example, a bishop located on cell d4 can legally move to the following cells: a7,a1, b6, a2, c3, c5, e3, e5, f2, f6, g1, g7 and h8.

Rook moves. A rook located on file x and rank i can legally move to any other position on the same file or to any other position on the same rank.

For example, a rook located on cell d4 can legally move to the following locations: a4, b4, c4, d1, d2, d3, d5, d6, d7, d8, e4, f4, g4, h4.

Queen moves. A queen can legally move from a location is it on to any location that is a legal move for a bishop, or a legal move to a rook.

King moves. A king can move to any location that is adjacent to its current location horizontally, vertically or diagonally.

CR-3. `isLegalMovePawn()`. The `int isLegalMovePawn()` function shall take as input five parameters:

Parameter	Type	Purpose
fromFile	`char`	Initial file position of the pawn
fromRank	`int`	Initial rank position of the pawn
color	`char`	Color of the pawn
toFile	`char`	File position of the location the pawn is being moved to
toRank	`int`	Rank position of the location the pawn is being moved to

The function shall do the following:

1. Check that the value of every parameter passed to it is correct. File values (fromFile and toFile) should be between `'a'` and `'h'`; rank values (fromRank and toRank) should be between 1 and 8. The color value is either `'w'` for *white pawn* or `'b'` for *black pawn*.

   If the value of any of the parameters is incorrect, your function shall return the -1.

2. Check that the starting position of the pawn is legal (see comments above about the legality of pawn locations on the board). If the starting location of the pawn is not legal, your function shall return -2.

3. Check if the move of the pawn of the specified color from the starting location (fromFile,fromRank) to the end location (toFile,toRank) is legal. Use the specifications from requirement CR-2 to check move legality.

   If the move is legal, your function shall return 1. If the move is not legal, your function shall return 0.

CR-4. Other functions. The remaining functions,

```
int isLegalMoveKnight(...)
int isLegalMoveBishop(...)
int isLegalMoveRook(...)
int isLegalMoveQueen(...)
int isLegalMoveKing(...)
```

shall check the legality of the move of the chess piece specified in the name of the function. Each function shall take as input the following parameters:

Parameter	Type	Purpose
fromFile	`char`	Initial file position of the pawn
fromRank	`int`	Initial rank position of the pawn
toFile	`char`	File position of the location the pawn is being moved to
toRank	`int`	Rank position of the location the pawn is being moved to

Each function shall do the following:

1. Check that the value of every parameter passed to it is correct. File values (fromFile and toFile) should be between `'a'` and `'h'`; rank values (fromRank and toRank) should be between 1 and 8.

   If the value of any of the parameters is incorrect, your function shall return the -1.

2. Check if the move of the chess piece from the starting location (fromFile,fromRank) to the end location (toFile,toRank) is legal. Use the specifications from requirement CR-2 to check move legality.

   If the move is legal, your function shall return 1. If the move is not legal, your function shall return 0.

Concluding notes. Some additional notes for you. No chess piece can legally move to the location it is currently on, i.e., a move is only legal if the chess piece leaves its current location. Only moves that involve a single chess piece are considered. This **excludes from consideration in this assignment** the moves of capture of other chess pieces by pawns (in a traditional way, or *en passant*) and *castling* of the king.

Part 2: Grade determination

Write a function `char toLetterGrade()` which takes as input two integer numbers, grade and maxGrade, that represent the grade a student received for an assignment, and the maximal possible grade for this assignment.

The function shall perform the following tasks:

1. check that the input parameters are valid. maxGrade must be positive. grade must be non-negative and shall not exceed maxGrade. If either of the parameters is invalid, the function shall return `'0'` (the character).

2. determine the letter grade the assignment shall receive. The letter grade depends on the percent of the maximal grade the student is received and is determined as follows:

From	To	Letter Grade
90%	100%	`A`
80%	90%	`B`
70%	80%	`C`
60%	70%	`D`
0%	60%	`F`

   The lower bound of each range is always inclusive. The upper bound of 100% is inclusive, all other upper bounds are *exclusive.*

3. return the computed letter grade.

Additional requirement. You are allowed to use `if` statements in the body of the function to test whether the input values are valid. However, **the determination of the letter grade shall be done using the** `switch` statement or statements.

Put the declaration and the definition of the `toLetterGrade()` function into a file `grades.h`. Additionally, create a collection of `checkit.h`-style unit tests for this function which (a) contains at least 10 tests and (b) covers every execution path in your function. Put these tests into an `int main()` function in a file `checkGrades.c`. You will be submitting both files.

Submission.

Files to submit. You shall submit five files: `chess.h`, `testChess.c`, `grades.h`, `testGrades.c` and `team.txt`.

Submission procedure. Only one submission per pair is required, but please make sure that the same person submits **all** files. Use the `handin` command.

Section 01 students:

```
> handin dekhtyar lab06 <files>
```

Please, DO NOT submit binary files.