Program 3: Chess Positions

Due date: Wednsday, November 26, 11:59pm. (late submission rules apply)

Purpose

To write a program requiring use multidimensional arrays.

Assignment

This is an **team** assignment. I recommend that you stay in the same teams as in **Lab 6.** If, due to course attrition, your team numbers less than three people, contact the instructor to receive new team assignment. Each team is expected to collaborate on the program. All other forms of collaboration, such as code sharing between members of different groups or soliciting/receiving help from individuals *not enrolled in the course* are strictly prohibited. All non-collaboration, non-plagiarism policies of the course are in full effect and will be enforced.

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.

Program Description

In this assignment you will continue working on the chess board rendering function library `chessboard.h` that you have started in **Lab 6**.

The goal of this assignment is to build the chess board library and a matching C program that takes as input a description of a chess position, and produces as output a PPM file showing this position.

To achieve this task, you will have to make certain changes and additions to the `chessboard.h` library. At the same time, you will be able to take full advantage of the functions **already in** the `chessboard.h` library. Therefore, **it is important that each group starts its work by completing the assignment of Lab 6**.

General Program Specification

Deliverables. You will continue developing the `chessboard.h` function library. In addition, you will develop a C program `position.c`, which will accept as input, the description of a chess position, and will write to standard output the PPM file displaying the position.

Input specification. Your input will consist of a sequence of lines. Each line of input will contain three inputs in the following order:

1. `char piece`. A `char` value representing both the chess piece and its color.
2. `char vertical`. A `char` value representing the vertical (`a` — `h`) of the chess piece location.
3. `int horizontal`. An `int` value representing the vertical (1 — 8) of the chess piece location.

The chess piece and its color are represented as shown in the table below:

Chess piece	White	Black
pawn	'P'	'p'
rook	'R'	'r'
knight	'N'	'n'
bishop	'B'	'b'
queen	'Q'	'q'
king	'K'	'k'

The **last line** of the input will always be

```
z z 0
```

Examples. The following input:

```
P a 2
P b 2
N b 1
p f 7
p g 7
r h 8
z z 0
```

describes a chess position which contains two white pawns on $a2$ and $b2$, a white knight on b, two black pawns on $f7$ and $g7$ and a black rook on $h8$.

Use of arrays. The new parts of the `chessboard.h` library will follow in their design the `image.h` library each of you built in **Lab 7**. In particular, to facilitate rendering of more than one chess piece without complications, you will represent the chessboard image as a binary array

```
char board[800][800];
```

(your declaration of all arrays shall follow the traditional style rules). We use `char` to represent single pixels because our chessboard is black-and-white. `board[i][j]` will store the information about the color of the pixel `(i,j)` of the PPM image rendering the chessboard, but rather than storing the entire RGB set of values, for this programming assignment, it is sufficient to store 0 for **black** and 1 for **white**. Your functions will be repsonsible for converting this information into specific colors when necessary.

New functions will take the `board[][800]` array as their first parameter and will modify its contents.

New Functions

The new functionality of the `chessboard.h` library shall be facilitated by the following new functions declared and defined in it.

```
void drawImage(char image[][WIDTH]);
int getStartX(int v);
int getStartY(int h) ;
void initChessboard(char image[][WIDTH]);
void putPawn(char image[][WIDTH], int horizontal, int vertical, int color);
void putRook(char image[][WIDTH], int horizontal, int vertical, int color);
void putKnight(char image[][WIDTH], int horizontal, int vertical, int color);
void putBishop(char image[][WIDTH], int horizontal, int vertical, int color);
void putQueen(char image[][WIDTH], int horizontal, int vertical, int color);
void putKing(char image[][WIDTH], int horizontal, int vertical, int color);
```

(remember, `WIDTH` must be correctly `#define`d in `chessboard.h`)

`void drawImage(char image[][WIDTH])`. This function takes as input the array representing current state of the chessboard, and outputs it as a PPM file. This function shall operate precisely in the same manner as the `drawImage()` function from your `image.h` library from **Lab 7**. The PPM file is sent, byte-by-byte to standard output. This function is responsible for ensuring tha the PPM file header gets printed.

`int getStartX(int v)`. This function may come handy at some points. It takes as input the vertical (as an integer number) and outputs the starting column for the chessboard squares on this vertical. For example, `getStartX(1)` shall return 0, `getStartX(2)` shall return 100, and so on.

`int getStartY(int h)`. This function is a companion function to `getStartX()`. It takes as input the horizontal on the chessboard and outputs the starting row for the chessboard squares on this vertical. For example, `getStartY(1)` shall return 700, `getStartX(2)` shall return 600, and so on. (remember that rows are numbered bottom-to-top on the chessboard, but pixels rows are numbered top to bottom).

`void initChessboard(char image[][WIDTH])`. This function takes as input the array representing the current state of the chessboard, and modifies the array to contain an image of an empty chess board. This function is somewhat similar to the `blankImage()` function from `Lab 7.`, except instead of a blank image, it rendered an empty chess board.

`void putPawn(char image[][WIDTH], int horizontal, int vertical, int color)`. This is the first of the six functions that render different chess pieces on the board. It takes as input an array representing the current state of the board, the location of the new pawn on the chessboard (horizontal, vertical) and the pawn color. The function modifies the array to put an appropriate rendering of the pawn in the chessboard square determined by the `horizontal` and `vertical` parameters.

Note, that if a different chess piece occupied that chess board square prior to the `putPawn()` call, its rendering shall completely disappear from the new image (i.e., `putPawn()` shall completely overwrite the contents of the designated chessboard square).

All other chess board squares shall remain intact!

`void putRook(char image[][WIDTH], int horizontal, int vertical, int color)`.

`void putKnight(char image[][WIDTH], int horizontal, int vertical, int color)`.

`void putBishop(char image[][WIDTH], int horizontal, int vertical, int color)`.

`void putQueen(char image[][WIDTH], int horizontal, int vertical, int color)`.

`void putKing(char image[][WIDTH], int horizontal, int vertical, int color)`. The remaining five functions render the respective chess pieces in the same manner as `putPawn()` does. The input to all these functions is an array representing the current state of the board, the location of the new pawn on the chessboard (horizontal, vertical) and the pawn color. The functions modify the array to put an appropriate rendering of the chess piese (rook, knight, bishop, queen, king) in the chessboard square determined by the `horizontal` and `vertical` parameters. Just as `putPawn()` does, all these functions overwrite the contents of the `(vertical,horizontal)` square on the chessboard, while leaving the remaining chess board squares intact.

Hint. The functions that you have created for `chessboard.h` library during **Lab 6.** may come very useful in implementation of the new `put` functions. Some of the old functions *may require some tweaking* — this will depend on your implementation. Alternatively, you can write `putPawn()`,...,`putKing()` from scratch, but this may signficantly increase your workload.

Extra Credit

For extra credit, come up with better, more realsitic renderings of the chess pieces. If you choose to do so, you shall prepare a new function library, `chessboardExtra.h`. This library shall have exactly the same structure as the `chessboard.h`. The difference will be in how you render individual chess pieces. You may choose to use the old function hierarchy from **Lab 7**, or you may choose to alter it.

Examples of improvements include: introducing new graphics primitives (an oval, for example) and using them in rendering of individual chess pieces; using current graphics primitives to render nicer looking chess pieces; abandoning the graphics primitives, and using other means (e.g., the analogs of `putPixel()` function from `Lab 7.`) to draw nice chess piece icons.

The low-hanging fruit are the improvements in the shape of a knight and a queen.

Submission

What to submit. You must submit `team.txt` file describing your team. **Please DO it right away!** For the regular part of the assignment you shall submit two files: `chessboard.h` and `position.c`. For the extra credit part, you need to submit the `chessboardExtra.h` and `positionExtra.h` — the extra credit analogs of the `chessboard.h` and `position.c` files. In addition. if submitting extra credit, include a `README` file, describing briefly the improvements made (e.g., which chess piece renderings were improved, and/or how the improvements were achieved).

Handin. You will submit to `dekhtyar`. As usual, ssh to vogon. The `handin` commands are:

```
> handin dekhtyar program03-09 <files>
```

and

```
> handin dekhtyar program03-11 <files>
```

for Sections 09 and 11 of the course respectively.

Appendix A: Specifications from Lab 6

Game of Chess

The game of chess is played on an 8×8 square board, with alternating black and white squares. The game is played by two players, with two identical sets of chess pieces: white and black.

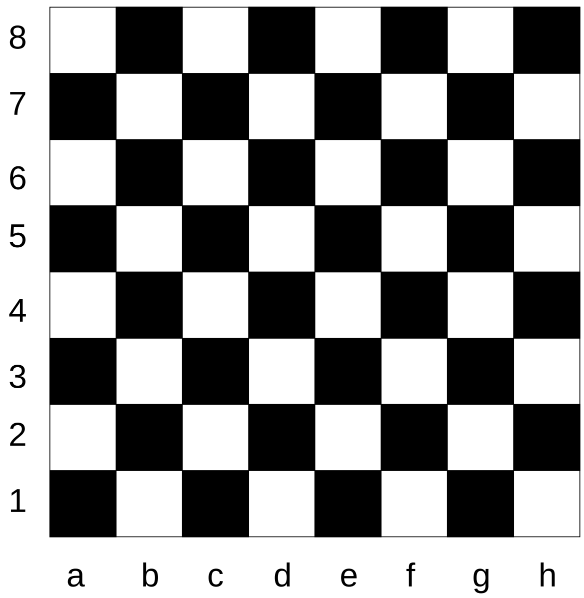


Figure 1: Chessboard.

The full set of chess pieces includes:

8 pawns
2 rooks
2 knights
2 bishops
1 queen
1 king

The rows of the chessboard are called horizontals and are numbered (bottom-to-top) 1 through 8. The columns of the chessboard are called verticals and are referred to (left-to-right) 'a' – 'h'. Thus, the bottom left corner of the chessboard is referred to as a1, the top right corner — h8 and the square in the third row and third column is c3. (see Figure 1).

Specifications

Chess board image. Your program/function library will work with PPM files of the size 800×800. Each square of the chessboard is represented by a 100×100 area of the image. Square a1 (bottom left) is black.

Chess Pieces. Essentially, chess pieces are 100×100 '"icons". Chess pieces come in six varieties (pawn, rook, knight, bishop, queen, king) and two colors (white, black).

Graphical Elements. Each chess piece is constructed out of simple graphical elements. In this lab, there are only two types of graphical elements that are used to construct chess pieces: circle and rectangle.

Figure 2 shows how each chess piece is constructed out of circles and rectangles. Each image is a $100\times$ 100 icon. The bottom solid line on each image is the bottom of the chess square. Each chess piece is elevated 10 pixels above the

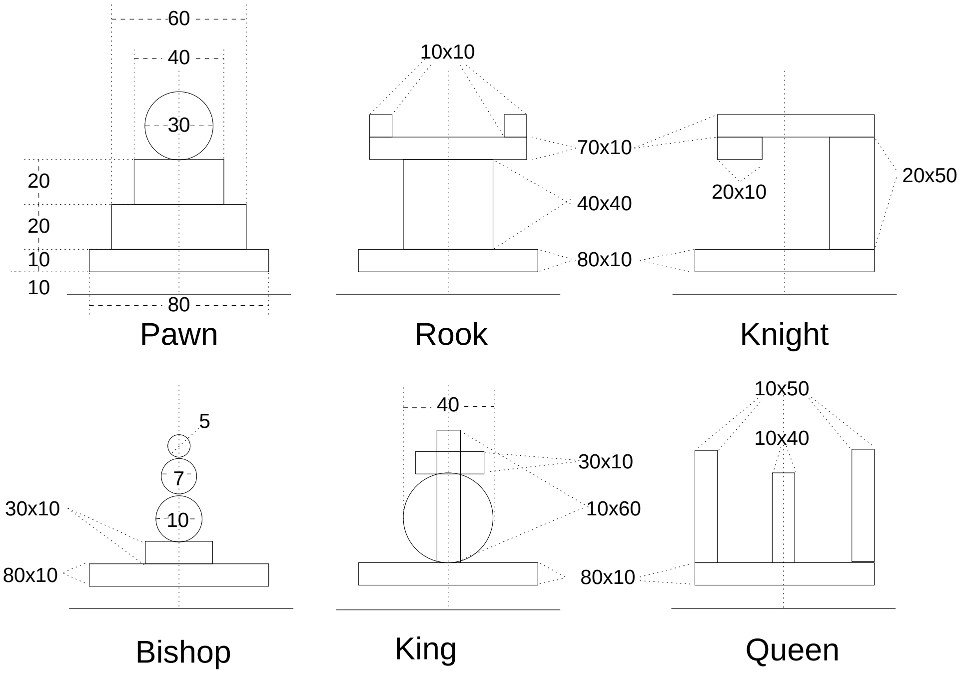
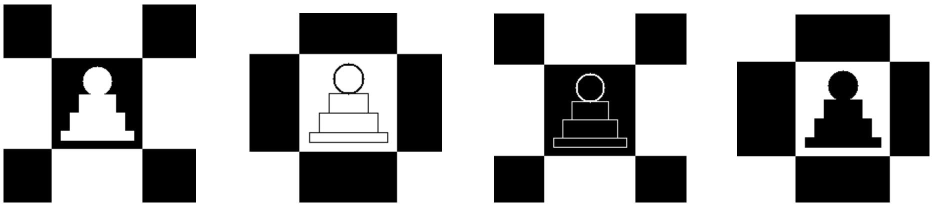


Figure 2: How to draw chess pieces on the chess board.



(a) white-on-black (b) white-on-white (c) black-on-black (d) black-on-white

Figure 3: Four ways of drawing a pawn.

bottom of the square, and sits on an 80×10 base rectangle. The dimensions of all rectangles, and the diameters of all circles are included in the Figure.

Coloring chess pieces. Each chess piece can come in one of two colors and may need to be drawn on either a black or a white square. Thus, there are four possible drawings you have to be able to make for each chess piece:

Color of chess piece	Color of square	Drawing style
white	black	solid white shape (shape boundaries are white)
white	white	black outline of each shape boundary
black	black	white outline of each shape boundary
black	white	solide black shape (shape boundaries are black)

Figure 3 shows an example of a pawn drawn in all four styles. The renderings of the other five types of chess pieces are shown in Figures 4 (rooks), 5 (bishops), 6 (knights), **??** (queens) and **??** (kings).

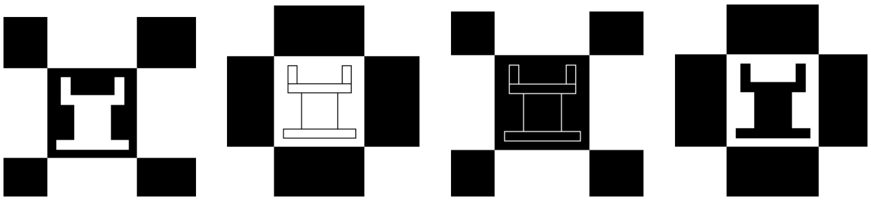


Figure 4: Four ways of drawing a rook.

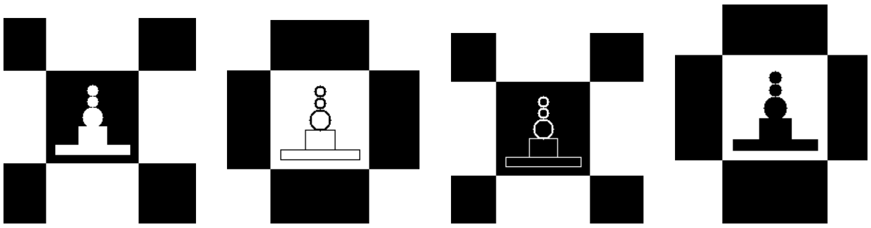


Figure 5: Four ways of drawing a bishop.

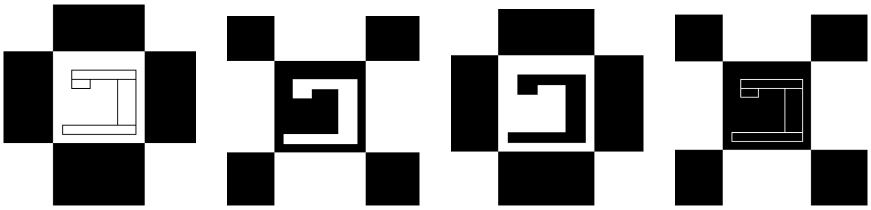


Figure 6: Four ways of drawing a knight.