HACETTEPE UNIVERSITY DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

BIL137 PROGRAMMING LABORATORY, EXPERIMENT 2

Subject:	Functions and Arrays
Submission Date:	26.11.2010
Due Date:	14.12.2010
Advisors:	Sevil ŞEN, Erkut ERDEM, Safa SOFUOĞLU

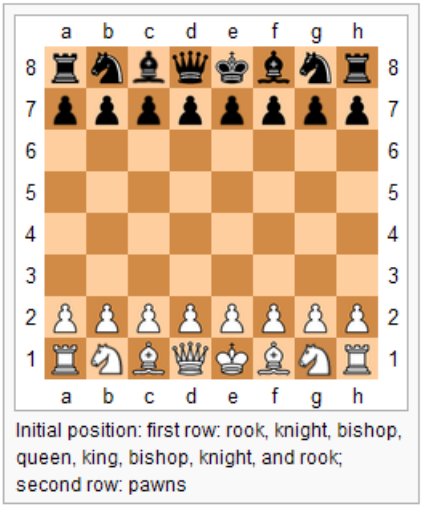
1. INTRODUCTION

The aim of this experiment is to help you practice functions and arrays in C. For this purpose, you are going to write an interactive chess simulator which will be able to take specific commands from the user, execute the commands and return the results (if any) to the user.

2. BACKGROUND INFORMATION

2.1. CHESS

Chess is a two-player board game played on a chessboard, a square-checkered board with 64 squares arranged in an eight-by-eight grid. Each player begins the game with sixteen pieces: one king, one queen, two rooks, two knights, two bishops, and eight pawns. The object of the game is to checkmate the opponent's king, whereby the king is under immediate attack (in "check") and there is no way to remove or defend it from attack on the next move.



Initial position: first row: rook, knight, bishop, queen, king, bishop, knight, and rook; second row: pawns

SETUP

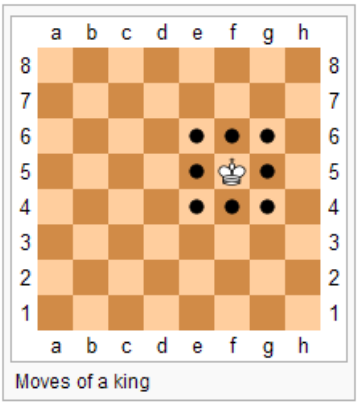
Chess is played on a square board of eight rows (called ranks and denoted with numbers 1 to 8) and eight columns (called files and denoted with letters *a* to *h*) of squares. The colors of the sixty-four squares alternate and are referred to as "light squares" and "dark squares". The chessboard is placed with a light square at the right hand end of the rank nearest to each player, and the pieces are set out as shown in the diagram, with each queen on its own color. The pieces are divided, by convention, into white and black sets. The players are referred to as "White" and "Black" and each begins the game with sixteen pieces of the specified color. These consist of one king, one queen, two rooks, two bishops, two knights, and eight pawns.

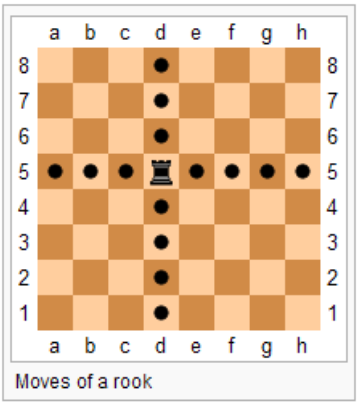
MOVEMENT

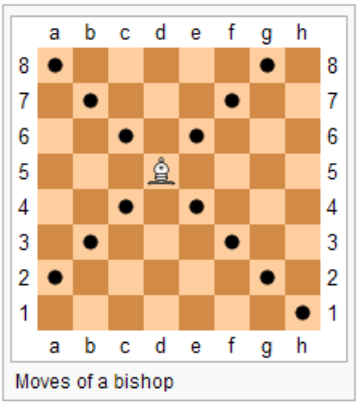
White always moves first. After the initial move, the players alternately move one piece at a time (with the exception of castling, when two pieces are moved). Pieces are moved to either an unoccupied square or one occupied by an opponent's piece, capturing it and removing it from play. With the sole exception of en passant, all pieces capture opponent's pieces by moving to the square that the opponent's piece occupies. A player may not make any move that would put or leave his king under attack. If the player to move has no legal moves, the game is over; it is either a checkmate—if the king is under attack—or a stalemate—if the king is not.

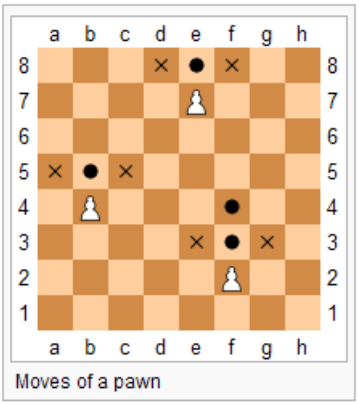
Each chess piece has its own style of moving. In the diagrams, the dots mark the squares where the piece can move if no other pieces (including one's own piece) are on the squares between the piece's initial position and its destination.

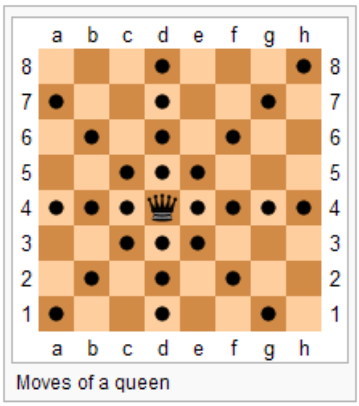
- The king moves one square in any direction.
- The rook can move any number of squares along any rank or file, but may not leap over other pieces.
- The bishop can move any number of squares diagonally, but may not leap over other pieces.
- The queen combines the power of the rook and bishop and can move any number of squares along rank, file, or diagonal, but it may not leap over other pieces.
- The knight moves to any of the closest squares that are not on the same rank, file, or diagonal, thus the move forms an "L"-shape two squares long and one square wide. The knight is the only piece that can leap over other pieces.
- The pawn may move forward to the unoccupied square immediately in front of it on the same file, or on its first move it may advance two squares along the same file provided both squares are unoccupied, or it may move to a square occupied by an opponent's piece, which is diagonally in front of it on an adjacent file, capturing that piece.[1]

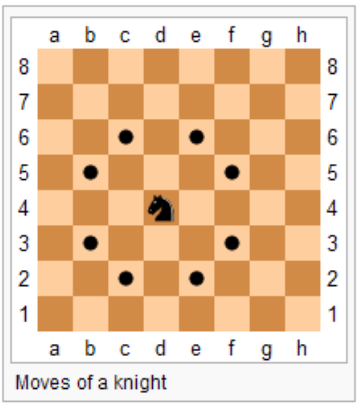

Moves of a king


Moves of a rook


Moves of a bishop


Moves of a pawn


Moves of a queen


Moves of a knight

2.2. READING AND COMPARING STRINGS

In this experiment, you are going to write code that reads strings from standard input and compares them to a predefined string value. Below is a sample code that reads two words (strings) from console and prints "`correct!`" if the input is "`hello world`".

[1] From http://en.wikipedia.org/wiki/Chess

```c
#include <stdio.h>
#include <string.h>

int main(void)
{
	char first[32], second[32];
	scanf("%s", first);
	if(strcmp("hello", first) == 0)
	{
		scanf("%s", second);
		if(strcmp("world", second) == 0)
		{
			printf("correct!");
		}
		else
		{
			printf("second word is not correct");
		}
	}
	else
	{
		printf("first word is not correct");
	}
}
```

2.3. ONLINE RESOURCES

- http://en.wikipedia.org/wiki/Chess
- http://en.wikibooks.org/wiki/C_Programming/Arrays
- http://en.wikibooks.org/wiki/C_Programming/Procedures_and_functions

3. PROBLEM

In this experiment, you are going to develop a chess application that implements a subset of rules of the chess game. Your program will be able to place and move pieces on the chess board, and also print the board's layout and clear the board.

When the program is requested to move pieces on the board, it should firstly check whether the piece is allowed to move to the destination square or not.

When the program is requested to place or move pieces on the board, it should check whether the destination square already contains another piece. If there is an opponent piece on the destination square and the program determines that the move is valid, moving piece will capture the piece at the destination square, removing it from play. If there is a friendly piece on the destination square, the move should be considered invalid. The pieces on the board will be represented with letters, so that:

K: Black King	k: White King
Q: Black Queen	q: White Queen
R: Black Rook	r: White Rook
B: Black Bishop	b: White Bishop
N: Black Knight	n: White Knight
P: Black Pawn	p: White Pawn

For this experiment, you will be handling pawns different than the actual chess game. White pawns will only be able to move upwards (from 1 to 8) and black pawns will only be able to move downwards. The two-square starting move of pawns will not be available.

Your program should be able to process and respond to 6 different commands, which are defined below. Optional arguments to the command are given in brackets. Upon execution, your program should be waiting for user commands and when it completes processing a command, it should be waiting for the next command.

When the program starts, the chess board should be empty, i.e. not containing any pieces.

Command: `place`

Arguments: `position piece`

Description: Places the given new pieces onto the given positions on the board. If pieces overlap during the placement, the operation should fail.

Output on successful operation: `OK`

Output on failure: `FAILED`

Command: `showmoves`

Arguments: `position`

Description: Lists the possible target positions where the piece at the given position can move. The order of the positions will be from a to h and from 1 to 8 increasing, i.e. a3 will be printed before a5 and a7 will be printed before b2.

Output: `position1 position2 position3 …`

Output on failure: `FAILED`

Command: `move`

Arguments: `position1 position2`

Description: Moves the piece at position1 to position2. The operation should fail if there is no piece at position 1 or if the move is not valid.

Output on successful operation: `OK`

Output on failure: `FAILED`

Command: `print`

This command does not take any arguments.

Description: Prints the status of the board to the console.

Output: The output will be 8 lines of 8 characters, where the rows will be 8 to 1 from top to bottom and the columns will be a to h from left to right. Pieces will be represented with their corresponding letters and empty squares will be represented as a single whitespace.

Command: `clear`

This command does not take any arguments.

Description: Removes every piece from the board to result in an empty board.

This command does not output any information to the console.

Command: `exit`

This command does not take any arguments.

Description: Instructs the program to exit.

This command does not output any information to the console.

4. SPECIFICATIONS

4.1. EXECUTION AND TESTING

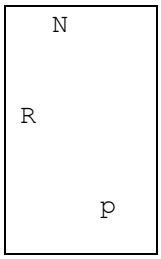
Your program should be interactively accepting input from console and writing output to the console. A command and all of its arguments will be entered as a single line and there will be at least one whitespace between the command and arguments. Your program must not output anything other than what the command is expected to output. Outputs that are different than what's specified will cause you a grade loss. Below is an example session with inputs and outputs:

```
> place a8 R
OK
> place c8 R
OK
> place d8 R
OK
> place a5 b
OK
> place f2 p
OK
> print
```

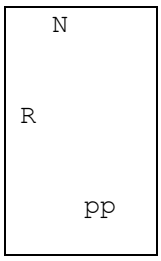
```
R RR


b


     p
```

```
> showmoves a8
a5 a6 a7 b8
```

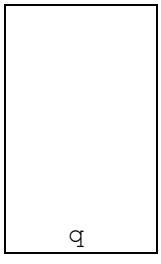
```
> move a8 a4
FAILED
> move a8 a5
OK
> print
```




```
> place e2 p
OK
> place f2 p
FAILED
> print
```




```
> clear
> place d1 q
OK
> print
```




```
> exit
```

4.2. DESIGN EXPECTATIONS

- Use understandable variable and function names.
- Add comments to your code where necessary.
- Conform to modular programming practices.

4.3. VALID PLATFORMS AND DEVELOPMENT ENVIRONMENT

You are going to use the Code::Blocks IDE to develop your project. Your code will be compiled against gcc version 4.1.2.

5. EVALUATION

5.1. REQUIRED FILES

You should create and submit a ZIP archive in the following structure for evaluation. An invalid structured archive will cause you partial or full score loss.

Directory	Files	Description
source	*.c, *.h (optional), *.cbp	Program source/header files and CodeBlocks project file
report	*.pdf	Your report (Only pdf format is accepted)

5.2. ACCEPTANCE PROCEDURE

Your final works will be accepted on-line from December 8 to December 10, 23:59, sharp. Any late arrival will not be accepted.

Any archive with significant errors (not a ZIP file, Microsoft Word or other invalid format report, missing files/directories, etc) will not be accepted. This is not a valid excuse for late submission.

5.3. REPORTS

Your reports must be PDF documents and adhere to the Hacettepe University Computer Science Department Report Writing Guidelines. Submissions with poorly written code can't be expected to receive a high score for the report. Here are some additional guidelines that will help you write your report for this experiment:

- Briefly explain what you understand from the problem.
- Provide a detailed description of your solution.
- Explain the execution flow between subprograms. Use flowcharts where necessary.
- Do not copy-paste from the experiment sheet.

6. NOTES

- SAVE all your work until the experiment is graded.
- The assignment must be original, INDIVIDUAL work. Shared source codes will be considered as cheating. Also the students who share their works will be punished in the same way.
- It is your responsibility to protect your work from unauthorized access.
- You can ask your questions via course's news group (*dersler.137* on news.cs.hacettepe.edu.tr). Discussion of the solution on the newsgroups will be considered "group work", which means "cheating".
- You are expected to follow the course's news group and you will be held responsible for the announcements made there.
- Cheaters will not be tolerated, and anyone caught will be reported to university authorities.