EECS 22L: Project 1

Prepared by: Mina Moghadam, Mihnea Chirila, Hamid Nejatollahi, Bo Tsai
Prof. Quoc-Viet Dang and Prof. Rainer Dömer

January 8, 2018

1 The Game of Chess

This is the first team programming project of EECS 22L, "Software Engineering Project in C language".

The goal of this programming exercise is to design and develop a chess program in which a user can play interactive chess against the computer. This is a big task, but it is not as difficult as it may sound.

This project is designed to be an interesting exercise where you can practice all elements of software engineering and work in teams. In particular, you will practice specifying and documenting the software program, designing data structures and algorithms, designing software modules, writing original source code, testing and debugging the software program, and collaborating and communicating effectively in a team.

1.1 The rules of chess

As a first step, you should make yourself familiar with the terminology and rules of the chess game. The rules are available on the **TA Infos** tab of the web page for this course.

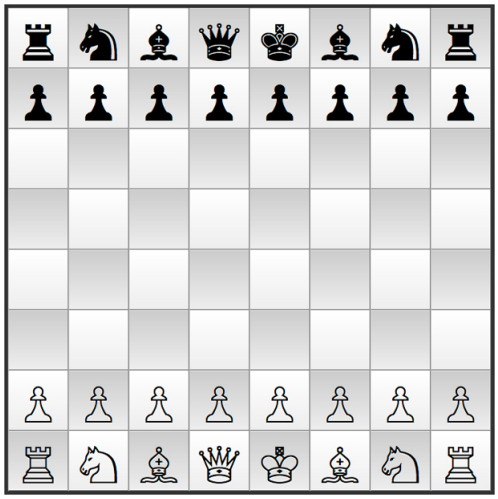

Figure 1: A chess board with initial setup

Using the information from this document, learn the rules of chess. Make yourself familiar with the goal of the game, the game play and the chess board. Specifically, learn about the six different pieces and how they can move on the board.

1.2 Chess pieces:

- The **queen** can move horizontally, vertically, and diagonally across the board.
- A **rook** can move horizontally and vertically across the board.
- A **bishop** can move diagonally across the board.
- A **knight** can jump to eight different squares which are two steps forward plus one step sideways from its current position.
- The **king** can move in any direction, but only one step at a time. Also, the king must never move into check. There is also a special "castling" move for the king.
- A **pawn** can move only forward towards the end of the board, but captures sideways. From its initial position, a pawn may make two steps, otherwise only a single step at a time. If the pawn reaches the end of the board, it is automatically promoted to another piece (usually a queen). There is also a special "en passant" move for the pawn.

1.3 Special moves

- "Castling" is a special move in the game of chess involving the king and either of the original rooks of the same color. It is the only move in chess (except promotion) in which a player moves two pieces at the same time. Castling consists of moving the king two squares towards a rook on the player's first rank (row), then moving the rook onto the square over which the king crossed. Castling can only be done if the king has never moved, the rook involved has never moved, the squares between the king and the rook involved are not occupied, the king is not in check, and the king does not cross over or end on a square in which it would be in check.
- "En passant" is a special pawn capture which can occur immediately after a player moves a pawn two squares forward from its starting position, and an enemy pawn could have captured it had the same pawn moved only one square forward. The opponent captures the just-moved pawn as if taking it "as it passes" through the first square. The resulting position is the same as if the pawn had moved only one square forward and the enemy pawn had captured normally.

  The en passant capture must be done on the very next turn, or the right to do so is lost. Such a move is the only occasion in chess in which a piece captures but does not move to the square of the captured piece. If an en passant capture is the only legal move available, it must be made.

The chess game ends as soon as one king is trapped in "checkmate". That is, there is no move for the player possible which would get his king out of check. Then the player loses.

2 Program Specification

In this project, we want a software program to be built, where a human player can interactively play chess against the computer. The human user chooses a color (white or black) and the computer plays the opponent.

There are several features that we expect the chess program to have. We distinguish between the minimum requirements for this project, and more advanced options that are goals and optional features (bonus points).

2.1 Basic functions that are required:

1. The game follows the official rules of chess.
2. The program shows a game interface where the player can see the game board and make moves.
3. The program supports an interactive player (human user) and an automatic player (computer).

4. The human user chooses the side to play (white or black).
5. The program keeps a human readable log of all the moves (in a text file).
6. The computer player makes its moves in reasonable time (less than 1 minute per move).

2.2 Advanced options that are desirable (but optional): (Bonus)

1. The human user can choose to play against a second human user or let the computer play against itself
2. The human player can withdraw previous moves
3. The program may support different levels of the computer player, such as beginner, intermediate, and expert
4. The computer player may provide hints on possible good moves to the human player
5. The program may provide a graphical user interface (GUI) (as introduced in Lecture 4)
6. The program may provide clocks (timers) for both players
7. The program can take a given board setup and start the game from there
8. The program supports the official algebraic notation of chess moves
9. Any other options that make the game more fun to play...

2.3 Tournament support

At the end of this project, we will host a competitive chess tournament among all teams with the chance to win extra bonus points. The teams' computer programs will play chess against each other, using separate computer terminals in the lab which will be controlled by a team member. Each win of a computer player will earn extra points for the team.

More details will be announced later, but a few early hints are in order so that you can choose your program features wisely.

- Hint 1: The basic functions (Section 2.1 above) are sufficient to participate in the tournament.
- Hint 2: An illegal move immediately ends the game. The opponent will be declared the winner. It is therefore most important to get the chess rules implemented correctly.
- Hint 3: A feature rich program with nice graphics is desirable, but if the computer player makes only dumb moves, it will not earn much credit.

3 Software Engineering Approach

In the design and implementation of this project, we will follow basic principles of software engineering in a close-to-real-world setting. You will practice the major tasks in software engineering to build your own software product. We will not provide detailed instructions on how to design the program as we did for the assignments in EECS22. Instead, your team needs to come up with your own choices and practice designing the software architecture of a medium-size program and document it.

3.1 Team work

The software design programming in this course will be performed by student teams. Teams of 5-6 students will be formed at the beginning of this project. Team work is an essential aspect of this class and every student needs to contribute to the team effort. While tasks may be assigned in a team to individual members, all members eventually share the responsibility for the project deliverables.

The overall tasks of software design, implementation and documentation should be partitioned among the team members, for example, to be performed by individuals or pairs (pair programming). A possible separation into tasks or program modules may include:

- main program
- user interface (textual and/or graphics)
- chess objects (data structures for pieces, boards, moves)
- lists (or trees) of moves
- chess rules (possible moves, legal moves)
- log file module
- strategy (artificial intelligence, AI) module
- documentation
- testing

When planning the team partitioning, keep in mind that certain tasks depend on others and that some tasks are best handled by everyone together.

A team account will be provided on the servers (`bondi`, `laguna`, `crystalcove` and `zuma`) for each team to share source codes, data and document files among the team members. Since teams will compete in the projects, sharing of files across teams is not permitted.

Every student is expected to show up and participate in team meetings. Attendance of the weekly discussion and lab sections is mandatory for the sake of successful team work.

3.2 Major project tasks

We list several steps here to approach the medium-sized programming project.

- *Design the software application specification*: work as a team to decide the functionalities of the program, the *input* and *output* of the program, and other things that describe the *features* of the program for the users.
- *Design the software architecture specification*: work as a team to design the data structure, program modules, application programming interface (API) functions between modules, and basic algorithms that will be used to solve the problem.
- *Build the software package*: write the source code and implement the program. Each team member may be in charge of their own modules and ideally work in parallel on implementation and module testing. Use Makefile for rule-based compilation to integrate the modules from different owners.
- *Version control and collaboration*: use a version control application, i.e. CVS (introduced in Lecture 3) to maintain the team project documentation and source code files. Team members can synchronize their own work with the others through the team repository located in the team account.
- *Test and debug the software*: work as a team to decide the testing strategies, write automated test programs or scripts, and debug the program when some of the test cases fail.

- *Software release*: release the software package with the executable program and documentation, e.g. the README file, user manual, etc. Release also the source code as a package for the future developers or maintainers.

3.3 Deliverables

Each team needs to work together and submit one set of deliverables each week. Here is the checklist of the files the team needs to submit and the due dates (hard deadlines).

Table 1: The Chess Project Deliverables

Week	File Name	File Description	Due Date
1	`Chess_UserManual.pdf`	The application specification	01/15/18 at 12:00pm (noon)
2	`Chess_SoftwareSpec.pdf`	The software architecture specification	01/22/18 at 12:00pm (noon)
3	`Chess_Alpha.tar.gz` `Chess_Alpha_src.tar.gz`	The alpha version of the chess program, including the program source code and documentation	01/29/18 at 12:00pm (noon)
4	`Chess_V1.0.tar.gz` `Chess_V1.0_src.tar.gz`	The released software package for the chess game and the program source code and documentation	02/05/18 at 12:00pm (noon)

Note that we do require these exact file names. If you use different file names, we will not see your files for grading.

We will separately provide detailed templates (document skeleton, table of expected contents) for the textual documents and a detailed list of contents (directory structure and expected files) for the file archives. These grading criteria will be provided at the Projects tab of the course webpage.

3.4 Submission for grading

To submit your team's work, you have to be logged in the server `zuma` or `crystalcove` by using your **team's account**. Also, you need to create a directory named `chess` in your team account, and put all the deliverables in that directory. Next, change the current directory to the directory containing the `chess` directory. Then type the command:

```
% ~eecs22/bin/turnin.sh
```

which will guide you through the submission process.

For each deliverable, You will be asked if you want to submit the script file. Type yes or no. If you type "n" or "y" or just plain return, they will be ignored and be taken as a no. You can use the same command to update your submitted files until the submission deadline.

Below is an example of how you would submit your team work:

```
zuma% ls # This step is just to make sure that you are in the correct directory that contains chess/
chess/
zuma% ~eecs22/bin/turnin.sh
============================================================
EECSL 22L Winter 2018:
Project "chess" submission for team1
Due date:  Mon Jan 15 12:00:00PM 2018
* Looking for files:
* Chess_UserManual.pdf
```

```
============================================================
Please confirm the following:  *
"I have read the Section on Academic Honesty in the *
UCI Catalogue of Classes (available online at *
http://www.editor.uci.edu/catalogue/appx/appx.2.htm#gen0) *
and submit original work accordingly." *
Please type YES to confirm.  y
============================================================
Submit Chess_UserManual.pdf [yes, no]?  y
File Chess_UserManual.pdf has been submitted
============================================================
Summary:
============================================================
Submitted on Mon Jan 8 00:05:31 2018
You just submitted file(s):
Chess_UserManual.pdf
zuma% _
```

For a binary package, we expect the user to read the documentation and run the executable program as follows:

```
% gtar xvzf BinaryArchive.tar.gz
% evince chess/doc/chess.pdf
% chess/bin/chess
```

For a source code package, we expect the developer to read the documentation and build the software as follows:

```
% gtar xvzf SourceArchive.tar.gz
% evince chess/doc/chess_software.pdf
% cd chess
% make
% make test
% make clean
```

Again, please ensure that these commands execute cleanly on your submitted packages.

4 Design and Implementation Hints

For this first project in this course, we would like to provide a few suggestions towards effective software design and implementation.

4.1 A simple user interface

In this section, we will show an example for a simple ASCII text-based chess board. You may use it as the initial interface of the chess game.

The following text shows the chess board with grids, pieces, and letter/numerical notations. It can be drawn by displaying the symbols, such as *'+'*, *'-'*, *'|'*, as well as numbers and letters on the screen. Different chess pieces can be represented in letters which indicate their color ('b' for black, 'w' for white), and names (*'R'* for Rook, *'N'* for Knight, *'B'* for Bishop, *'Q'* for Queue, *'K'* for King, and *'P'* for Pawn).

This is the initial setup of the pieces on the board.

```
    +----+----+----+----+----+----+----+----+
  8 | bR | bN | bB | bQ | bK | bB | bN | bR |
    +----+----+----+----+----+----+----+----+
  7 | bP | bP | bP | bP | bP | bP | bP | bP |
    +----+----+----+----+----+----+----+----+
```

```
6 |    |    |    |    |    |    |    |    |
  +----+----+----+----+----+----+----+----+
5 |    |    |    |    |    |    |    |    |
  +----+----+----+----+----+----+----+----+
4 |    |    |    |    |    |    |    |    |
  +----+----+----+----+----+----+----+----+
3 |    |    |    |    |    |    |    |    |
  +----+----+----+----+----+----+----+----+
2 | wP | wP | wP | wP | wP | wP | wP | wP |
  +----+----+----+----+----+----+----+----+
1 | wR | wN | wB | wQ | wK | wB | wN | wR |
  +----+----+----+----+----+----+----+----+
    a    b    c    d    e    f    g    h
```

The following text shows the chess board with one white pawn being moved from grid *e2* to *e4*.

```
  +----+----+----+----+----+----+----+----+
8 | bR | bN | bB | bQ | bK | bB | bN | bR |
  +----+----+----+----+----+----+----+----+
7 | bP | bP | bP | bP | bP | bP | bP | bP |
  +----+----+----+----+----+----+----+----+
6 |    |    |    |    |    |    |    |    |
  +----+----+----+----+----+----+----+----+
5 |    |    |    |    |    |    |    |    |
  +----+----+----+----+----+----+----+----+
4 |    |    |    |    | wP |    |    |    |
  +----+----+----+----+----+----+----+----+
3 |    |    |    |    |    |    |    |    |
  +----+----+----+----+----+----+----+----+
2 | wP | wP | wP | wP |    | wP | wP | wP |
  +----+----+----+----+----+----+----+----+
1 | wR | wN | wB | wQ | wK | wB | wN | wR |
  +----+----+----+----+----+----+----+----+
    a    b    c    d    e    f    g    h
```

Alternatively, you may also choose to provide a graphical user interface (GUI), which we will introduce in Lecture 4.

4.2 An object-oriented data structure

While the ANSI-C programming language is not explicitly an object-oriented language (that would be C++), it is nevertheless prudent to think of the objects at hand when designing a data structure for a project.

For the chess program, the following are some of the basic objects that require representation in the program's data structures:

- a player (who is either black or white)
- a piece type (either king, queen, bishop, knight, rook, or pawn)
- a piece (a combination of player/color and piece type)
- a board (8x8 matrix of squares, each with or without a piece on them)
- a position of a piece (known to the user as "e2", for example)
- a move (a combination of a start and end position, e.g. "e2 e4")
- a log (a list of moves)

Together with such object data, methods or functions that manipulate the objects are also needed. For the chess game, suitable basic functions include the following:

- on a board, lookup a piece at a given position
- on a board, put a given piece onto a given position
- on a board, move a piece from a position to another
- etc.

Additionally, specific higher-level functions are needed for various purposes, including:

- for a piece on the board, compute all reachable positions
- for a piece on the board, compute all legal moves
- on a board, check whether or not a player's king is in check
- on a board, check whether or not a player's king is in checkmate
- for a player and a given board, compute all legal moves
- from a list of moves, select the best one
- etc.

4.3 Overall chess program control flow

The basic overall control flow of a chess game is the following:

- setup
- display the board
- repeat
  - white player makes a move
  - display the board
  - if black is in checkmate, white wins!
  - black player makes a move
  - display the board
  - if white is in checkmate, black wins!

Let the games begin!