COMP07027 Introduction to Programming

Coursework – 2014/15

Programming Project

This project is worth 80% of the overall module assessment.

The development work for this coursework may be completed in pairs (recommended because it is good to have someone to discuss the problem and solution with, but you may NOT work in groups larger than two) or individually.

If you work as a pair this means working on all parts of the program together (the report is an individual report, but the development and testing of the code should be done jointly) – it is not intended that you delegate different parts of the implementation work to partners individually, though it is acceptable for you to identify who will do the lead work for each function/part (as in, who types the code in to the computer once the algorithm/logic has been agreed and who sits and watches over their shoulder!), and you may wish to decide that where one person takes the lead on implementing some function that the other person takes the lead responsibility for testing it.

Your project work is to be submitted to the Programming Project assignment on Moodle no later than **Friday May 1st at 4:00 pm**. As you are being allowed to work in pairs, there will be time in the laboratory sessions for you to work on the project, but this will not be sufficient to finish the project, you will need to arrange to meet outwith scheduled lab time. You should start the project NOW – it will take you longer than you think to implement this program and every year there are students who do not succeed in producing a working program who write in their critical appraisal that they would have done a much better job if only they had started the work earlier. If you submit and fail this project, you will have to do another one, on your own, over the summer. Note that under regulations that came into effect last session, if you fail to submit this project and do not have acceptable mitigating circumstances you will not be given a project to do over the summer but will need to take the whole module again next year. In that scenario you may not be allowed to proceed into the next year of your degree, so please do make sure that you submit something for this project.

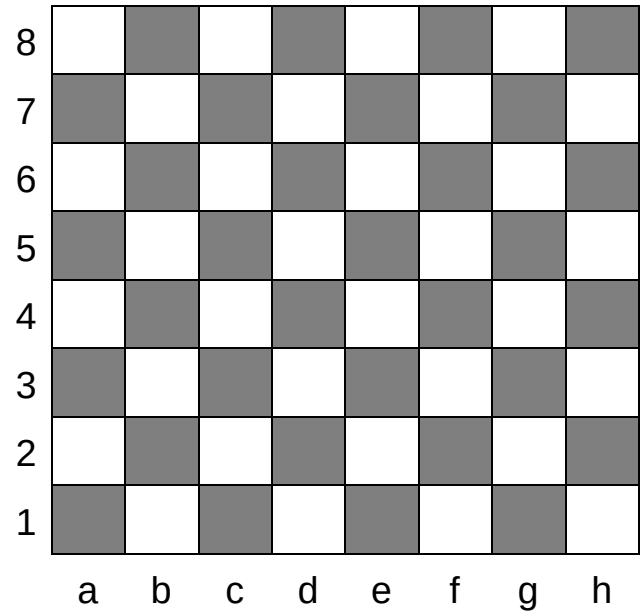
See the submission notes at the end of this handout for more details of what you are required to submit. Late submissions will be penalized at the standard

university rate: a reduction of 10% for submissions up to 1 week late (so that a submission that would have received 50% would receive 40%), with zero marks awarded after that (i.e. there is no point in submitting a project that is 1 week plus 1 day late). Extensions for project work will only be allowed on the presentation of a valid medical certificate or evidence of some similar valid cause.

The Open Knight's Tour Problem

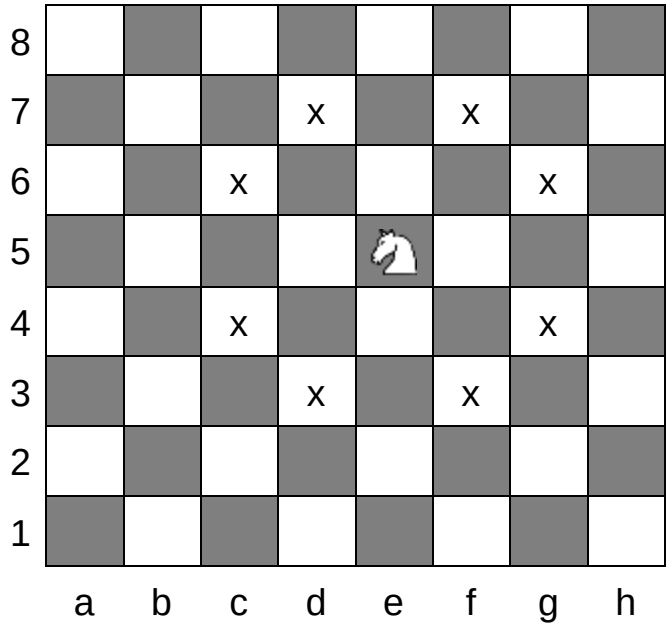
The knight's tour is a classic chess problem, the first systematic analysis of which that is known was done by the mathematician, Euler, in 1759 (by then it was already an ancient problem).

A standard chess board contains 64 squares arranged in 8 rows (called *ranks* and in the standard algebraic chess notation are numbered 1 to 8) and 8 columns (called *files* and labelled a to h):



In a game of chess there are two players, White, whose pieces occupy ranks 1 and 2 at the beginning of the game, and Black, whose pieces occupy ranks 7 and 8 initially. By convention, a chess board is displayed with the White position at the bottom. This project does not involve playing chess, however, but concerns one of the pieces in a chess set.

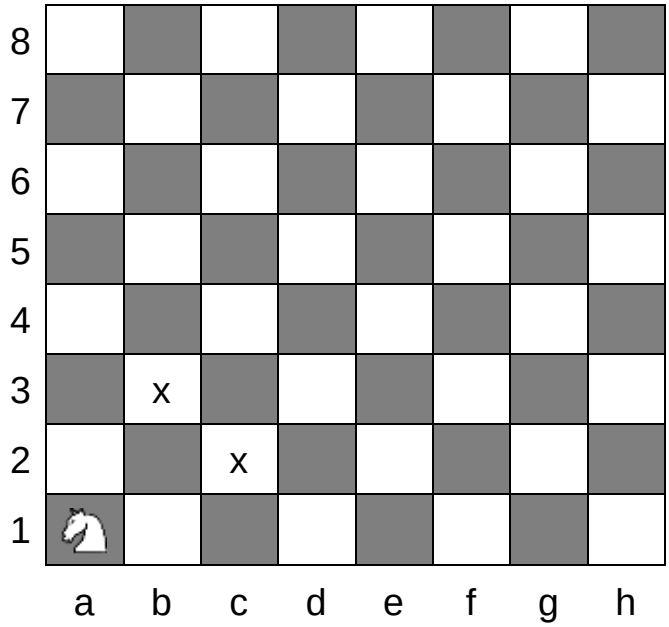
The piece in question is the Knight, which in a standard (Staunton pattern) chess set is shaped like the head of a horse. The squares on a chess board are coloured black and white and a Knight's move is such that each move takes it to a square of the opposite colour to the one it is on.



As illustrated above, a Knight on e5 can move to c4, c6, d3, d7, f3, f7, g4, or g6. If you think of the board in terms of rows and columns, with a Knight at position (column, row) it can move to each of the eight squares at:

```
(column+1, row-2)       (column+1, row+2)
(column+2, row-1)       (column+2, row+1)
(column-1, row-2)       (column-1, row+2)
(column-2, row-1)       (column-2, row+1)
```

Of course, a Knight cannot move off the board so a Knight on a1, for example, can only move to two squares: b3 or c2.



The open Knight's tour (or Knight's path) problem asks the question, **from some chosen starting square on the board, is it possible to construct a sequence of Knight moves such that it visits every square on the board exactly once?** This problem is called the "open" knight's tour as the knight is not required to finish on a square where it can return to its starting point in 1 move (solving that problem is called the closed Knight's tour). In the 1990s it was proved that an open tour is possible for any NxN board for which N is larger than or equal to 5.

Even for a 5x5 board it can be difficult to find a path, and it may be several moves after a mistake (a move that does not lead to a complete path) is made before the problem solver realises that they have hit a dead end. So to make the testing of your program simpler your problem is to find a tour for a much smaller board, one that is 3x4 (three ranks, four files).

For a 3x4 board a path around the board exists if the Knight starts in one of the outside files (*a* or *d*). For your program you can just start the Knight in a1. The solution is much easier to find for this case not just because there are fewer moves to make (only 12 squares, after all) but also because form any starting square on a board this size for which there is a path the Knight never has more than two available moves (to unvisited squares) from any square that it occupies.

As with the practice Towers of Hanoi program, you are NOT being asked to write a program that will solve the problem – this can be done fairly straightforwardly and you can find programs online that do this, but your program is to allow the user to choose each move in the sequence that the Knight is to make.

You do not need to simulate or represent the colours of the squares or produce a graphical application. Instead, you should just display 3 rows and 4 columns of text to represent the board. The value 1 is placed in the starting square for the Knight, and on each subsequent move the square being moved to should show the number of the move.

To simplify the code in displaying the board your program may display the board with the first rank at the top, rather than at the bottom as in the above diagrams (as noted above, the convention in chess diagrams is to show White playing up the board and Black down the board, here we are showing a mirror image of the board with White's side at the top).

So a run of the program (with user input in bold) might go something like this:

```
Welcome to the open Knight’s tour problem! This program allows you to move
a Knight around a 3 by 4 chessboard. Your task it to have it visit each of
the 12 squares on the board exactly once (i.e. without visiting any square
twice).

An empty board would be displayed in the following format, with “White” at
the top:

     a   b   c   d
1:   -   -   -   -
2:   -   -   -   -
3:   -   -   -   -

As the knight moves, the sequence of moves is shown by numbers in those
squares it was or is occupying – the highest number indicating where the
knight is currently located. For example, in the board below the knight is
on b1, having started at a1, moved to c2, then to a3, and then to b1:

     a   b   c   d
1:   1   4   -   -
2:   -   -   2   -
3:   3   -   -   -

When referring to a square to move to, give the coordinates in the
standard chess algebraic format “a1” or “c3”. If you run into a dead end
and can make no further progress, type “q” to quit.

The problem starts NOW. This is the initial position:

     a   b   c   d
1:   1   -   -   -
2:   -   -   -   -
3:   -   -   -   -

The Knight is on a1, where do you want to move the Knight to (enter your
move in the format <file><rank>, for example: a1, or q to quit)? b3

     a   b   c   d
1:   1   -   -   -
2:   -   -   -   -
3:   -   2   -   -

The Knight is on b3, where do you want to move the Knight to (enter your
move in the format <file><rank>, for example: a1, or q to quit)? d2

     a   b   c   d
1:   1   -   -   -
2:   -   -   -   3
3:   -   2   -   -
```

```
The Knight is on d2, where do you want to move the Knight to (enter your
move in the format <file><rank>, for example: a1, or q to quit)? b1

      a   b   c   d
1:    1   4   -   -
2:    -   -   -   3
3:    -   2   -   -

The Knight is on b1, where do you want to move the Knight to (enter your
move in the format <file><rank>, for example: a1, or q to quit)? a3

      a   b   c   d
1:    1   4   -   -
2:    -   -   -   3
3:    5   2   -   -

The Knight is on a3, where do you want to move the Knight to (enter your
move in the format <file><rank>, for example: a1, or q to quit)? c2

      a   b   c   d
1:    1   4   -   -
2:    -   -   6   3
3:    5   2   -   -

The Knight is on c2, where do you want to move the Knight to (enter your
move in the format <file><rank>, for example: a1, or q to quit)? q

OK. You have elected to quit – you have not found a tour of the board.
Good luck next time! Goodbye.
```

Comments on the Program's Behaviour

This program can terminate in one of two ways: either the user finds a path for the Knight that visits all 12 squares, or they run out of moves (as in the sequence above) when the Knight arrives on a square from which it cannot move to a square that it has not already been. To pass this project, your program need not detect the case that the user has run into a dead end, users should recognise this for themselves and elect to quit. But the program should not allow a Knight to move to a square that it has already been, so if instead of typing "q" at the end of the sequence above the user were to type "a1" or "a3" the program should report that the Knight has already visited that square and not allow the move.

Your program should, of course, detect when the user successfully navigates the Knight through a tour of the board and display a congratulatory message.

In case you have trouble finding a solution to check that your program does detect one, here is a sequence of moves that successfully completes an open tour of a 3x4 board starting at *a1*:

c2, a3, b1, d2, b3, c1, d3, b2, d1, c3, a2

The final board displayed after entering this sequence should be:

```
     a   b   c   d
1:   1   4   7  10
2:  12   9   2   5
3:   3   6  11   8
```

Comments on Representing the Program's Data

Given the format of the program's output, the simplest way to represent the board is to use a two-dimensional array of **`int`**. The array will have three rows (the ranks) and four columns (the files).

It should be clear that the value of each element in the array should either be zero (the square has not been visited by the Knight) or the move number that it was visited on, with 1 representing the square where the Knight "lands" on the board, 2 where it moves next, *etc*.

With this representation, and assuming that you name the array `board`, the corner square, a1, is `board[0][0]` and the opposite corner square d3 is `board[2][3]` (note, in the array indexing the row number comes first, in the chess notation for the square the column comes first). This means that file *a* is column 0 and rank *1* is row 0 in the array, so you need to convert the user input into this form. If the variable, `file`, is a `char` that contains the name of the file (a, b, c, or d) then you can calculate the column using the formula:

```
int column = file – 'a';
```

If the variable, `rank`, is an `int` that holds the rank number (1, 2, or 3) then you can calculate the row (I'm sure you will already have worked this out!) using the formula:

```
int row = rank – 1;
```

You will also find it useful to define a class to represent the position of a square on the board. This should store the row and column number of the square. Using this class as the method's return-type, you can write a method to get the user's move and have it return the coordinates of the square to which the Knight is to move. It also means that you can use a variable of this

type to track the current location of the Knight, without having to search the board looking for the highest number to determine where the Knight is.

Comments on Representing the Program's Output

To get the numbers on the screen to line up remember that you can specify a field width in TextIO's put() and putln() methods. For example TextIO.put(x, 4); will display the value of x right justified in a field of 4 characters (so if the value were, say, 2, then there would be three leading spaces before the 2 in the output, if the value were 10 then there would two leading spaces). You could alternatively use a format String and TextIO.putf() to control the number of characters a value occupies on the screen (see Lab07 and section 2.4.1 of the textbook).

Marking Scheme

Your program should address the following points:

Point No.	**Description (the facilities you need to implement)**	**Possible Marks**
1	The program follows the above specification in providing a description of the problem, the user instructions, and the initial state of the board (the Knight is initially on a1 in a 3x4 board).	10
2	The program defines a suitable set of classes and variables to represent the program state. This includes representing the board, where the Knight has visited and where the Knight and is currently located.	10
3	For each move, the program gets the user input and checks that, if the input is a move, that the move is a legal Knight's move and that it does not revisit a square that has already been visited. If the input is a request to quit the program displays a suitable message and exits.	10
4	For each valid move, the program updates the program state (i.e. moves the Knight) and displays the new program state.	10
5	After each move, the program checks if the Knight has completed the tour and if so it stops the sequence and displays a congratulatory message.	10

6	Additional functionality. Marks will be awarded for providing additional features relevant to the problem. Examples could include: a. Allow the user to choose the starting square b. Detect when a Knight's path will lead or has led to a dead end and alert the user to the situation c. Allow the user to retract their last move and try an alternative d. Allow the user to retract a series of moves and try an alternative to the earliest move in the retracted sequence, OR e. At the end of the game replay each move of the game's move sequence for the user to a point where the user would like to deviate from the previous sequence and try an alternative f. Extend the program to deal with 5x5 or larger boards (in which case, the user could choose the starting square) g. Provide an option to save the Knight's path (the sequence of moves) to a text file for later analysis	25
7	Program structure. The functionality of the program is distributed across a number of methods with appropriate parameters and return types, and/or across a number of classes each of which encapsulate some aspect of the problem or solution.	15
8	Report. The submission includes an individual report on the work done, how the program was developed and tested, and any areas where the program could be improved or further developed given more time and effort. **(Note: this report MUST be included or the submission will not be accepted – and a report must be produced and submitted by each of you individually even if you did the rest of the work as part of a pair).**	10
	Total Marks	100

What you should submit

Each of you should submit your project, whether or not you worked individually or as a pair. To submit your project, upload it using the link in the Assignments subpage of the Moodle site for the module.

The submission for the project should include the following:

1. A ZIP file containing all of the source code files (these are the files whose name ends in .java) that make up your project, and your individual report. Your report should be in Word format, rich text format, or a searchable PDF. As a rough guide to preparing your project for submission:
    a. Make sure ALL of the project's source files are in the project's folder, and that your report is also there. If any are not in that folder, move them there or get help to do so.
    b. Go up to the containing folder and create a ZIP file of your whole project. Assuming you're using Windows, just right-click on the folder and select Send To..Compressed (zipped) folder). For other operating systems, you may need to look up the correct method for creating a ZIP file – email for help if you need it.

2. Submit the ZIP file using the **Programming Project** link in the Assignments area of the module site on Moodle no later than **Friday May 1st 2015 at 4:00 pm**.