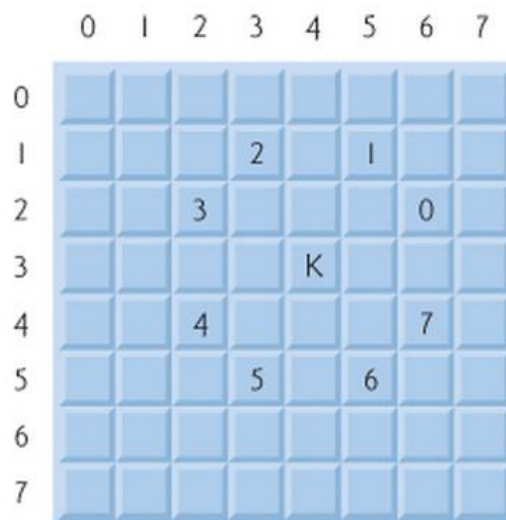


This is Problem 7.22 & 7.23 in your book

### ***(Knight's Tour)***

One of the more interesting puzzles for chess buffs is the Knight's Tour problem. The question is this: Can the chess piece called the knight move around an empty chess-board and touch each of the 64 squares **once and only once**? We study this intriguing problem in depth in this exercise.

The knight makes L-shaped moves (over two in one direction then over one in a perpendicular direction). Thus, from a square in the middle of an empty chessboard, the knight can make eight different moves (numbered 0 through 7) as shown in the figure.



The eight possible moves of the knight.

a) Draw an 8-by-8 chessboard on a sheet of paper and attempt a Knight's Tour by hand. Put a 1 in the first square you move to, a 2 in the second square, a 3 in the third, etc. Before starting the tour, estimate how far you think you'll get, remembering that a full tour consists of 64 moves. How far did you get? Was this close to your estimate?

b) Now let's develop a program that will move the knight around a chessboard. The board is represented by an 8-by-8 two-dimensional array board. Each of the squares is initialized to zero. We describe each of the eight possible moves in terms of both their horizontal and vertical components. For example, a move of type 0, consists of moving two squares horizontally to the right and one square vertically upward. Move 2 consists of moving one square horizontally to the left and two squares vertically upward. Horizontal moves to the left and vertical moves upward are indicated with negative

numbers. The eight moves may be described by two one-dimensional arrays, horizontal and vertical, as follows:

```
horizontal[ 0 ] = 2   vertical[ 0 ] = -1
horizontal[ 1 ] = 1   vertical[ 1 ] = -2
horizontal[ 2 ] = -1  vertical[ 2 ] = -2
horizontal[ 3 ] = -2  vertical[ 3 ] = -1
horizontal[ 4 ] = -2  vertical[ 4 ] = 1
horizontal[ 5 ] = -1  vertical[ 5 ] = 2
horizontal[ 6 ] = 1   vertical[ 6 ] = 2
horizontal[ 7 ] = 2   vertical[ 7 ] = 1
```

Let the variables `currentRow` and `currentColumn` indicate the row and column of the knight's current position. To make a move of type `moveNumber`, where `moveNumber` is between 0 and 7, your program uses the statements

```
currentRow += vertical[ moveNumber ];
currentColumn += horizontal[ moveNumber ];
```

Keep a counter that varies from 1 to 64. Record the latest count in each square the knight moves to. Remember to test each potential move to see if the knight has already visited that square, and, of course, test every potential move to make sure that the knight does not land off the chessboard. Now write a program to move the knight around the chessboard. Run the program. How many moves did the knight make?

c) After attempting to write and run a Knight's Tour program, you've probably developed some valuable insights. We'll use these to develop a heuristic (or strategy) for moving the knight. Heuristics do not guarantee success, but a carefully developed heuristic greatly improves the chance of success. You may have observed that the outer squares are more troublesome than the squares nearer the center of the board. In fact, the most troublesome, or inaccessible, squares are the four corners.

Intuition may suggest that you should attempt to move the knight to the most troublesome squares first and leave open those that are easiest to get to, so when the board gets congested near the end of the tour, there will be a greater chance of success.

We may develop an "accessibility heuristic" by classifying each square according to how accessible it's then always moving the knight to the square (within the knight's L-shaped moves, of course) that is most inaccessible. We label a two-dimensional array `accessibility` with numbers indicating from how many squares each particular square is accessible. On a blank chessboard, each center square is rated as 8, each corner square is rated as 2 and the other squares have accessibility numbers of 3, 4 or 6 as follows:

```
2 3 4 4 4 4 3 2
3 4 6 6 6 6 4 3
4 6 8 8 8 8 6 4
4 6 8 8 8 8 6 4
4 6 8 8 8 8 6 4
3 4 6 6 6 6 4 3
2 3 4 4 4 4 3 2
```

Now write a version of the Knight's Tour program using the accessibility heuristic. At any time, the knight should move to the square with the lowest accessibility number. In case of a tie, the knight may move to any of the tied squares. Therefore, the tour may begin in any of the four corners. [Note: As the knight moves around the chess-board, your program should reduce the accessibility numbers as more and more squares become occupied. In this way, at any given time during the tour, each available square's accessibility number will remain equal to precisely the number of squares from which that square may be reached.] Run this version of your program. Did you get a full tour? Now modify the program to run 64 tours, one starting from each square of the chessboard. How many full tours did you get?

d) Write a version of the Knight's Tour program which, when encountering a tie between two or more squares, decides what square to choose by looking ahead to those squares reachable from the "tied" squares. Your program should move to the square for which the next move would arrive at a square with the lowest accessibility number.

#### (Knight's Tour: Brute Force Approaches)

In the previous exercise, we developed a solution to the Knight's Tour problem. The approach used, called the "accessibility heuristic," generates many solutions and executes efficiently.

As computers continue increasing in power, we'll be able to solve more problems with sheer computer power and relatively unsophisticated algorithms. This is the "brute force" approach to problem solving.

a) Use **random number generation** to enable the knight to walk around the chessboard (in its legitimate L-shaped moves, of course) at random. Your program should run one tour and print the final chessboard. How far did the knight get?

b) Most likely, the preceding program produced a relatively short tour. Now modify your program to attempt 1000 tours. Use a one-dimensional array to keep track of the number of tours of each length. When your program finishes attempting the 1000 tours, it should print this information in neat tabular format. What was the best result?

c) Most likely, the preceding program gave you some "respectable" tours, but no full tours. Now "pull all the stops out" and simply let your program run until it produces a full tour. [Caution: This version of the program could run for hours on a powerful computer.] Once again, keep a table of the number of tours of each length, and print this table when the first full tour is found. How many tours did your program attempt before producing a full tour? How much time did it take?

d) Compare the brute force version of the Knight's Tour with the accessibility heuristic version. Which required a more careful study of the problem? Which algorithm was more difficult to develop? Which required more computer power? Could we be certain (in advance) of obtaining a full tour with the accessibility heuristic approach? Could we be certain (in advance) of obtaining a full tour with the brute force approach? Argue the pros and cons of brute force problem solving in general.