Miguel Arnáiz Montes G2391
Sofia Xiaofan Fernández

The eight queens.

Describe the eight queens problem. Add a reference.

The eight queens problem consists on placing eight queens on a chessboard. The queens can not be placed attacking each other. This can also be extracted to an nxn board, the called n queens problems.

Explain the algorithm that solves the problem (not just a description but a reasoning of the steps).

The algorithm consists on 3 steps:

- Creating a list with the range told (from A to B)
- Permutate the values inside the list
- Check for each permutation if the queens do not collide in a diagonal

Describe the Prolog predicates used in the solution. Add screenshots of the code execution.

Here we see the solution found.

```
?- queens_1(8, X).
X = [1, 5, 8, 6, 3, 7, 2, 4] ,

?-
```

Now we will show the functionality of the predicates used through the trace execution of the code:

Miguel Arnáiz Montes G2391
Sofia Xiaofan Fernández

range(A,B,L)

```
?- trace, queens_1(8, X).
   Call: (11) queens_1(8, _10944) ? creep
   Call: (12) range(1, 8, _11474) ? creep
   Call: (13) 1<8 ? creep
   Exit: (13) 1<8 ? creep
   Call: (13) _11614 is 1+1 ? creep
   Exit: (13) 2 is 1+1 ? creep
   Call: (13) range(2, 8, _11464) ? creep
   Call: (14) 2<8 ? creep
   Exit: (14) 2<8 ? creep
   Call: (14) _11846 is 2+1 ? creep
   Exit: (14) 3 is 2+1 ? creep
   Call: (14) range(3, 8, _11696) ? creep
   Call: (15) 3<8 ? creep
   Exit: (15) 3<8 ? creep
   Call: (15) _12078 is 3+1 ? creep
   Exit: (15) 4 is 3+1 ? creep
   Call: (15) range(4, 8, _11928) ? creep
   Call: (16) 4<8 ? creep
   Exit: (16) 4<8 ? creep
   Call: (16) _12310 is 4+1 ? creep
   Exit: (16) 5 is 4+1 ? creep
   Call: (16) range(5, 8, _12160) ? creep
   Call: (17) 5<8 ? creep
   Exit: (17) 5<8 ? creep
   Call: (17) _12542 is 5+1 ? creep
   Exit: (17) 6 is 5+1 ? creep
   Call: (17) range(6, 8, _12392) ? creep
   Call: (18) 6<8 ? creep
   Exit: (18) 6<8 ? creep
   Call: (18) _12774 is 6+1 ? creep
   Exit: (18) 7 is 6+1 ? creep
   Call: (18) range(7, 8, _12624) ? creep
   Call: (19) 7<8 ? creep
   Exit: (19) 7<8 ? creep
   Call: (19) _13006 is 7+1 ? creep
   Exit: (19) 8 is 7+1 ? creep
   Call: (19) range(8, 8, _12856) ? creep
   Exit: (19) range(8, 8, [8]) ? creep
   Exit: (18) range(7, 8, [7, 8]) ? creep
   Exit: (17) range(6, 8, [6, 7, 8]) ? creep
   Exit: (16) range(5, 8, [5, 6, 7, 8]) ? creep
   Exit: (15) range(4, 8, [4, 5, 6, 7, 8]) ? creep
   Exit: (14) range(3, 8, [3, 4, 5, 6, 7, 8]) ? creep
   Exit: (13) range(2, 8, [2, 3, 4, 5, 6, 7, 8]) ? creep
   Exit: (12) range(1, 8, [1, 2, 3, 4, 5, 6, 7, 8]) ? creep
```

This predicate creates a list from A to B and stores it in L.

permu(Xs,Zs)

```
   Call: (12) permu([1, 2, 3, 4, 5, 6, 7, 8], _10944) ? creep
   Call: (13) del(_13488, [1, 2, 3, 4, 5, 6, 7, 8], _13550) ? creep
   Exit: (13) del(1, [1, 2, 3, 4, 5, 6, 7, 8], [2, 3, 4, 5, 6, 7, 8]) ? creep
   Call: (13) permu([2, 3, 4, 5, 6, 7, 8], _13490) ? creep
   Call: (14) del(_13626, [2, 3, 4, 5, 6, 7, 8], _13688) ? creep
   Exit: (14) del(2, [2, 3, 4, 5, 6, 7, 8], [3, 4, 5, 6, 7, 8]) ? creep
   Call: (14) permu([3, 4, 5, 6, 7, 8], _13628) ? creep
   Call: (15) del(_13764, [3, 4, 5, 6, 7, 8], _13826) ? creep
   Exit: (15) del(3, [3, 4, 5, 6, 7, 8], [4, 5, 6, 7, 8]) ? creep
   Call: (15) permu([4, 5, 6, 7, 8], _13766) ? creep
   Call: (16) del(_13902, [4, 5, 6, 7, 8], _13964) ? creep
   Exit: (16) del(4, [4, 5, 6, 7, 8], [5, 6, 7, 8]) ? creep
   Call: (16) permu([5, 6, 7, 8], _13904) ? creep
   Call: (17) del(_14040, [5, 6, 7, 8], _14102) ? creep
   Exit: (17) del(5, [5, 6, 7, 8], [6, 7, 8]) ? creep
   Call: (17) permu([6, 7, 8], _14042) ? creep
   Call: (18) del(_14178, [6, 7, 8], _14240) ? creep
   Exit: (18) del(6, [6, 7, 8], [7, 8]) ? creep
   Call: (18) permu([7, 8], _14180) ? creep
   Call: (19) del(_14316, [7, 8], _14378) ? creep
   Exit: (19) del(7, [7, 8], [8]) ? creep
   Call: (19) permu([8], _14318) ? creep
   Call: (20) del(_14454, [8], _14516) ? creep
   Exit: (20) del(8, [8], []) ? creep
   Call: (20) permu([], _14456) ? creep
   Exit: (20) permu([], []) ? creep
   Exit: (19) permu([8], [8]) ? creep
   Exit: (18) permu([7, 8], [7, 8]) ? creep
   Exit: (17) permu([6, 7, 8], [6, 7, 8]) ? creep
   Exit: (16) permu([5, 6, 7, 8], [5, 6, 7, 8]) ? creep
   Exit: (15) permu([4, 5, 6, 7, 8], [4, 5, 6, 7, 8]) ? creep
   Exit: (14) permu([3, 4, 5, 6, 7, 8], [3, 4, 5, 6, 7, 8]) ? creep
   Exit: (13) permu([2, 3, 4, 5, 6, 7, 8], [2, 3, 4, 5, 6, 7, 8]) ? creep
   Exit: (12) permu([1, 2, 3, 4, 5, 6, 7, 8], [1, 2, 3, 4, 5, 6, 7, 8]) ? creep
```

Permu predicate transforms the list Xs into a permutation in Zs. It uses an auxiliary predicate: **del(X, L1, L2).** This predicate deletes the element X from the list L1 and returns the remaining list in L2.

Miguel Arnáiz Montes G2391
Sofia Xiaofan Fernández

test(Qs)

```
Call: (12) test([1, 2, 3, 4, 5, 6, 7, 8]) ? creep
Call: (13) test([1, 2, 3, 4, 5, 6, 7, 8], 1, [], []) ? creep
Call: (14) _4566 is 1-1 ? creep
Exit: (14) 0 is 1-1 ? creep
Call: (14) memberchk(0, []) ? creep
Fail: (14) memberchk(0, []) ? creep
Redo: (13) test([1, 2, 3, 4, 5, 6, 7, 8], 1, [], []) ? creep
Call: (14) _4792 is 1+1 ? creep
Exit: (14) 2 is 1+1 ? creep
Call: (14) memberchk(2, []) ? creep
Fail: (14) memberchk(2, []) ? creep
Redo: (13) test([1, 2, 3, 4, 5, 6, 7, 8], 1, [], []) ? creep
Call: (14) _5018 is 1+1 ? creep
Exit: (14) 2 is 1+1 ? creep
Call: (14) test([2, 3, 4, 5, 6, 7, 8], 2, [0], [2]) ? creep
Call: (15) _5168 is 2-2 ? creep
Exit: (15) 0 is 2-2 ? creep
Call: (15) memberchk(0, [0]) ? creep
Exit: (15) memberchk(0, [0]) ? creep
Fail: (14) test([2, 3, 4, 5, 6, 7, 8], 2, [0], [2]) ? creep
Fail: (13) test([1, 2, 3, 4, 5, 6, 7, 8], 1, [], []) ? creep
Fail: (12) test([1, 2, 3, 4, 5, 6, 7, 8]) ? creep
Redo: (20) permu([], _3888) ? creep
Call: (21) del(_5468, [], _5530) ? creep
Fail: (21) del(_5468, [], _5574) ? creep
Fail: (20) permu([], _3888) ? creep
Redo: (20) del(_3886, [8], _5662) ? creep
Call: (21) del(_3886, [], _5652) ? creep
Fail: (21) del(_3886, [], _5652) ? creep
Fail: (20) del(_3886, [8], _5800) ? creep
Fail: (19) permu([8], _3750) ? creep
Redo: (19) del(_3748, [7, 8], _5888) ? creep
Call: (20) del(_3748, [8], _5878) ? creep
Exit: (20) del(8, [8], []) ? creep
Exit: (19) del(8, [7, 8], [7]) ? creep
Call: (19) permu([7], _3750) ? creep
Call: (20) del(_6058, [7], _6120) ? creep
Exit: (20) del(7, [7], []) ? creep
Call: (20) permu([], _6060) ? creep
Exit: (20) permu([], []) ? creep
Exit: (19) permu([7], [7]) ? creep
Exit: (18) permu([7, 8], [8, 7]) ? creep
Exit: (17) permu([6, 7, 8], [6, 8, 7]) ? creep
Exit: (16) permu([5, 6, 7, 8], [5, 6, 8, 7]) ? creep
Exit: (15) permu([4, 5, 6, 7, 8], [4, 5, 6, 8, 7]) ? creep
Exit: (14) permu([3, 4, 5, 6, 7, 8], [3, 4, 5, 6, 8, 7]) ? creep
Exit: (13) permu([2, 3, 4, 5, 6, 7, 8], [2, 3, 4, 5, 6, 8, 7]) ? creep
Exit: (12) permu([1, 2, 3, 4, 5, 6, 7, 8], [1, 2, 3, 4, 5, 6, 8, 7]) ? creep
Call: (12) test([1, 2, 3, 4, 5, 6, 8, 7]) ? creep
Call: (13) test([1, 2, 3, 4, 5, 6, 8, 7], 1, [], []) ? creep
Call: (14) _6738 is 1-1 ? creep
Exit: (14) 0 is 1-1 ? creep
Call: (14) memberchk(0, []) ? creep
Fail: (14) memberchk(0, []) ? creep
Redo: (13) test([1, 2, 3, 4, 5, 6, 8, 7], 1, [], []) ? creep
Call: (14) _6964 is 1+1 ? creep
Exit: (14) 2 is 1+1 ? creep
Call: (14) memberchk(2, []) ? creep
Fail: (14) memberchk(2, []) ? creep
Redo: (13) test([1, 2, 3, 4, 5, 6, 8, 7], 1, [], []) ? creep
Call: (14) _7190 is 1+1 ? creep
Exit: (14) 2 is 1+1 ? creep
Call: (14) test([2, 3, 4, 5, 6, 8, 7], 2, [0], [2]) ? creep
Call: (15) _7340 is 2-2 ? creep
Exit: (15) 0 is 2-2 ? creep
Call: (15) memberchk(0, [0]) ? creep
Exit: (15) memberchk(0, [0]) ? creep
Fail: (14) test([2, 3, 4, 5, 6, 8, 7], 2, [0], [2]) ? creep
Fail: (13) test([1, 2, 3, 4, 5, 6, 8, 7], 1, [], []) ? creep
Fail: (12) test([1, 2, 3, 4, 5, 6, 8, 7]) ? creep
Redo: (20) permu([], _6060) ? creep
Call: (21) del(_7640, [], _7702) ? creep
Fail: (21) del(_7640, [], _7746) ? creep
Fail: (20) permu([], _6060) ? creep
Redo: (20) del(_6058, [7], _7834) ? creep
Call: (21) del(_6058, [], _7824) ? creep
```

This predicate gives a solution in Qs to the problem using another auxiliary method with same name but different arguments: **test([Y|Ys],X,Cs,Ds)**.

What do you think about the solution? Have you come up with another way of solving it?

It is a pretty good solution. However it may be a bit inefficient as it has to make every permutation in the list and check it until it reaches a valid solution.

We didn't come up with any other solution.