Discrete Optimization Specialization: Assignment 9
Birthday Parade

1 Problem Statement

The Great Turtle is grateful to Nüwa for making him the Prime Minister of the Undersea Kingdom. He is treating her to a military parade by the finest Shrimp Soldiers and Crab Generals (all termed as "soldiers" hereafter) to honour Nüwa's 10000 year-old birthday. In the initial formation, the soldiers are lined up in columns with the captain in front. And then soldiers will march out from the formation and parade in front of Nüwa one after the other, starting with the captain. Each soldier can only parade in front of her when all soldiers ahead of that soldier in their column have already paraded.

Nüwa considers a good parade to have all soldiers of the same height. So she remains pleased during the parade until the height difference between two consecutive soldiers is discernible to her. The aim is to determine an ordering of the soldiers to march in front of Nüwa in the parade so as to keep her pleased for as long as possible.

2 Data Format Specification

The input for the Birthday Parade is contained in a file named `data/parade`p`.dzn`, where

- p is the problem number,
- n is the number of soldiers in the parade,
- *height* is an array of heights for each soldier,
- *cols* is the number of columns they are arrayed in, and
- *delta* is the height difference which is indiscernible with Nüwa's aging (and thus failing) eyesight.

The aim is to generate assignment *pos* of soldiers to the positions.

The captain is always soldier number 1, soldier 2 is the first soldier in the first column, soldier $cols + 1$ is the first soldier in the last column, and soldier $cols + 2$ is the second soldier in the first column, *etc*.

The data declarations and main decisions are hence:

```
int: n;                          % number of soldiers
set of int: SOLDIER = 1..n;
set of int: POS = 1..n;
array[SOLDIER] of int: height;   % height of each soldier
int: cols;                       % number of columns of soldiers
int: delta;                      % height delta allowed

array[SOLDIER] of var POS: pos; % order position of each soldier
```

The output required for every stage of this assignment is of the form:

```
pos = [order position of each soldier in the parade];
obj = how long Nüwa is happy;
```

Note Nüwa is happy until the first position in the parade order where the soldier is noticeably different in height (difference is greater than *delta*) than the soldier after them. If this never occurs she is happy for the entire parade ($obj = n$).

The aim of this assignment is to model the problem, and give a basic search strategy (just one `int_search` annotation) that will keep Nüwa happy as long as possible. Notice that this is a permutation problem and it is possible to build models using different viewpoints.

For example given the data file

```
n = 12;
height = [4, 3, 3, 5, 2, 5, 3, 4, 2, 1, 2, 4];
cols = 4;
delta = 1;
```

the soldiers are arranged initially in 4 columns behind the captain, as

	1		
2	3	4	5
6	7	8	9
10	11	12	

An optimal solution is given by

```
obj = 11;
pos = [1, 9, 4, 2, 5, 11, 6, 3, 7, 12, 8, 10];
```

Note for example that soldier 11 is paraded 8th after both the soldiers in front of him in his column. That is after soldier 3 (paraded 4th) and soldier 7 (paraded 6th) are already paraded. Also note that the heights of the soldiers in the order paraded are $4, 5, 4, 3, 2, 3, 2, 2, 3, 4, 5, 1$ and Nüwa is happy until the second last soldier parades.

3 Instructions

Edit `parade.mzn` to solve the optimization problem described above. Your `parade.mzn` implementation can be tested on the data files provided. In the MiniZinc IDE, use the *Run* icon to test your model locally. At the command line use,

```
mzn-gecode ./parade.mzn ./data/<inputFileName>
```

to test locally. In both cases, your model is compiled with MiniZinc and then solved with the GECODE solver.

Resources You will find several problem instances in the `data` directory provided with the handout.

Handin This assignment contains 3 solution submissions and 5 model submissions. For solution submissions, we will retrieve the best/last solution the solver has found using your model and check its correctness and quality. For model submissions, we will retrieve your model file (.mzn) and run it on some hidden data to perform further tests.

From the MiniZinc IDE, the *Submit to Coursera* icon can be used to submit assignment for grading. From the command line, `submit.py` is used for submission. In both cases, follow the instructions to apply your MiniZinc model(s) on the various assignment parts. You can submit multiple times and your grade will be the best of all submissions.[1] It may take several minutes before your assignment is graded; please be patient. You can track the status of your submission on the *programming assignments* section of the course website.

4 Technical Requirements

For completing the assignment you will need MiniZinc 2.1.x and the GECODE 5.0.x solver. Both of these are included in the bundled version of the MiniZinc IDE 2.1.x (`http://www.minizinc.org`). To submit the assignment from the command line, you will need to have Python 3.5.x installed.

[1] Solution submissions can be graded an unlimited number of times. However, there is a limit on grading of **model submissions**.