NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER ENGINEERING
COMPUTER AND COMPUTER TECHNOLOGY SECTOR

Spring 2021

PROGRAMMING LANGUAGES I.

Exercise 2

Closing date and time of electronic submission: 30/5/2021, 23:59:59

ICU under constant pressure, again (0.25 points)

This problem is known from the first series of exercises this year. The
The purpose of this exercise is to write its solution in Prolog. Because the systems
Prolog do not run native code, the time limit for exercise will be significant
increased. Your plan should contain a **longest / 2** predicate which will
has the input filename as the first argument and will return to its second argument
the answer. For example the pronunciation of the first series, your predicate will
should behave as shown below. [1]

```
? - longest ('f.txt', Answer), writeln (Answer), fail.
5
false .
```

To read the input, see the model given in the second exercise.

Loops in video games, again (0.25 points)

And this problem is known from the first series of exercises this year. The
The purpose of this exercise is to write its solution in Python. Because its implementations
Python do not run native code, the time limit for this exercise will be again
increased. Your program should behave in the same way as the programs in
C / C ++ that you delivered for the first set of exercises. For examples of pronunciation
of the first row, the output of your program must be as follows:

```
$ python3 loop_rooms.py maze1.txt
4

$ python3 loop_rooms.py maze2.txt
2
```

Sort by tail and stack (0.25 + 0.25 = 0.5 points)

Give N nonnegative at most three digit integers (where $1 \le N \le 42$) [2] , which
placed in a queue. A stack is also given, which is initially empty. The purpose of it
of the exercise is to sort the numbers in the queue in ascending order,

using a sequence consisting of the following movements:

- **Q** : Remove the first element of the tail and place it on top of the stack.
- **S** : Remove the item at the top of the stack and place it at the end of the tail.

[1] In all the examples of this series of exercises, depending on the Prolog system you will use, on cases where there is a solution, the line with **false .** can say **fail .** or **no .** Also note that using **writeln** and **fail** , as shown in the examples, you can also understand if its execution Your program's main category is deterministic or not.

[2] Do not take the upper limit (N ≤ 42) in cash. The test cases will allow the exhaustive search.

Page 2

The movements of the sequence are applied one after the other in sequence and after the execution of the latter the stack should be empty and the numbers in the queue should be sorted in ascending order, with the smallest number at the beginning of the queue. If they exist more sequences of movements that lead to the desired result, we are interested in find the one that has the shortest length (ie the smallest number of movements). If they exist more sequences of equal length, we are interested in finding the lexically shorter one.

For example, suppose that N = 4 and that the numbers 7, 17, 3, 42 are initially in the queue, with this series. The sequence of 10 movements "QQSQSSQQSS" achieves the desired result, as shown in the table below (the beginning of the queue is on the left and its end right, while the top of the stack is on the right):

Step	Movement	Condition	Step	Movement	Condition
0	initially	tail: 7, 17, 3, 42 stack: (empty)	6	running S queue:	42, 17, 3, 7 stack: (empty)
1	running Q queue:	17, 3, 42 stack: 7	7	running Q queue:	17, 3, 7 stack: 42
2	running Q queue:	3, 42 stack: 7, 17	8	running Q queue:	3, 7 stack: 42, 17
3	running S queue:	3, 42, 17 stack: 7	9	running S queue:	3, 7, 17 stack: 42
4	running Q queue:	42, 17 stack: 7, 3	10	running S queue:	3, 7, 17, 42 stack: (empty)
5	running S queue:	42, 17, 3 stack: 7			

The exercise asks you to write two programs (one in Prolog and one in Python) which will answer the above question. The numbers that are initially placed in the queue will are known and the response of your programs may be based on appropriate processing of these numbers. Because Prolog and Python implementations are not running native code, the time limit for your programs will be increased.

The entry of your programs is read from a text file that will contain two lines. The first line will contain the number *N* of the numbers, while the second will contain the *N* numbers, divided by two with a space. The output of the programs you must be the requested sequence of movements. If it is empty, the output should is the word "empty". Examples are given below, in Prolog and Python.

In Prolog

? - qssort ('qs1.txt', Answer), writeln (Answer), fail.

QQSQSSQQSS
false .

? - qssort ('qs2.txt', Answer), writeln (Answer), fail.
QQQQSQSSSQSSQS
false .

? - qssort ('qs3.txt', Answer), writeln (Answer), fail.
QQSQQSQSSS
false .

? - qssort ('qs4.txt', Answer), writeln (Answer), fail.
QQQQSSSS
false .

? - qssort ('qs5.txt', Answer), writeln (Answer), fail.
empty
false .

Page 3

In Python

$ python3 qssort.py qs1.txt
QQSQSSQQSS

$ python3 qssort.py qs2.txt
QQQQSQSSSQSSQS

$ python3 qssort.py qs3.txt
QQSQQSQSSS

$ python3 qssort.py qs4.txt
QQQQSSSS

$ python3 qssort.py qs5.txt
empty

$ cat qs1.txt
4
7 17 3 42

$ cat qs2.txt
6
17 7 3 42 1 8

$ cat qs3.txt
6
1 0 1 0 1 0

$ cat qs4.txt
5
5 4 3 2 1

$ cat qs5.txt
4
1 2 3 4

The first example is the one described above. Notice that and the sequence QSQSQSQQSQSS movement achieves the desired result, but is longer (12 movements instead of 10). In the third example, the sequence of movements "QQSQQSSSQS" as well achieves the desired result, but is of equal length (10 movements) and the "QQSQQSQSSS" is lexically smaller. In the fifth example, the numbers are already classified.

Further instructions for the exercises

- You can work in groups of up to two people. You can if you want to form different group compared to the previous series of exercises - the groups in the system submission is anyway new to each series of exercises.

- You are not allowed to share your programs with fellow students outside your group or put them in a place where others can find them (eg on a web page, on talk sites,…). In case "strange" similarities are observed in programs, the grade of the students involved in *all the series of exercises* is done automatically zero regardless of which team ... was "inspired" by the other.

- You can use "helper" code (eg some code that it manages some data structure) that you found on the internet in your programs, with provided that your program contains in the comments the assumption about the origin of this code and a link to it.

- Programs in Python must be in a file and work in Python 3.7.3. (Note that Python 2 is a different dialect of Python!)

- Programs in Prolog must be in a file and work on one of them following systems SWI Prolog (8.0.2), GNU Prolog (1.3.0) or YAP (6.2.2).

- The submission of programs will be done electronically through moodle, as in previous exercise, and to be able to submit them, your team members (and both) should already have a moodle account. There will be a relevant announcement as soon as the submission system becomes active. Your programs should read it input as stated and must not have any other type of output other than requested because they will not be accepted by the submission system.