Lab session 5: Imperative Programming

The central theme of this lab is recursion. This means that each problem should be solved using recursion. Themis only checks whether your program produces the right output, and does not check whether you used recursion or not. Therefore, non-recursive solution that is accepted by Themis will still be rejected by manual inspection by the teaching assistants.

Problem 1: Palindromic sentence

A palindrome is a word which reads the same forward as backward (for example `madam`). In this problem we consider palindromic sentences. A sentence is called a *palindromic sentence* if the sentence reads the same forward as backward, ignoring any character which is not a letter. Moreover, the case (lower/upper) of letters is irrelevant. For example, the following sentence is palindromic:

Sir, I'm Iris!

Write a program that reads a sentence from the input (terminated by a newline character), and determines whether the sentence is palindromic or not. Your program must make use of a *recursive* function `isPalindromic`.

Example 1:

input:

```
Sir, I'm Iris!
```

output:

```
The sentence is a palindrome.
```

Example 2:

input:

```
Some men interpret nine memos.
```

output:

```
The sentence is a palindrome.
```

Example 3:

input:

```
I'm not palindromic.
```

output:

```
The sentence is not a palindrome.
```

Problem 2: Up-Down series

In this problem we consider, for a given n, the series $1, 2, 3, \ldots, n$. We want to compute all permutations (reorderings) such that the permuted series alternates from increasing to decreasing. For example, for $n = 4$ the series 1, 2, 3, 4 can be reordered in the following 5 up-down-up series:

```
1 3 2 4
1 4 2 3
2 3 1 4
2 4 1 3
3 4 1 2
```

Write a program that reads from the input an integer `n` (where $2 \leq$ `n` ≤ 11), and outputs the number of possible up-down permutations constructed from the series $1, 2, \ldots,$ `n`. Of course, you program should make use of recursion.

Example 1:
input:
`2`
output:
`1`

Example 2:
input:
`3`
output:
`2`

Example 3:
input:
`4`
output:
`5`

Problems 3: Compressor

Pictures taken with modern digital cameras have many pixels, and hence produce large files. Most cameras therefore use image compression to save storage space. In this problem we will implement one of the techniques used for image compression.

In this exercise we make the simplifying assumption that the input is an $n \times n$ matrix with only two characters (representing foreground and background pixels). Moreover, n is a power of two (i.e. $n = 2, 4, 8, 16, 32, ..$). This property allows us to partition the two-dimensional $n \times n$ grid by recursively subdividing it into four sub-grids of size $\frac{n}{2} \times \frac{n}{2}$.

The recursive partitioning works as follows. There are two base cases. The first considers a 1×1 region (i.e. a single location). In this case, we simply output the character at that location. The other base case is the situation in which all locations of an $n \times n$ region (where $n > 1$) are filled with the same value `X`, then we output `1X` and stop processing this region. The recursive case is the situation in which the region is non-constant. In this case we output `0` (a zero), and we recursively subdivide the region into four $\frac{n}{2} \times \frac{n}{2}$ regions. These regions are recursively processed in clock-wise order, staring with the upper-left region.

The input of the program consists of a positive integer `n` (a power of two), which is the size of the $n \times n$ grid and `n` lines (terminated by newlines) containing `n` characters. Your output should be the number `n` (on a separate line), followed by the compression of the input.

Example 1:

input:

```
8
****....
****....
****....
****....
........
........
........
........
```

output:

```
8
01*1.1.1.
```

Example 2: inpt:

```
8
..**....
..**....
**......
**......
........
........
......*.
......*.
```

output:

```
8
001.1*1.1*1.01.1.0*..*1.1.
```

Example 3:

input:

```
8
++++++++
++++++++
+#+++#++
###+####+
+######++
++###+++
+++#++++
++++++++
```

output:

```
8
001+1+0+++#0+###01+1+0+++#0+###00###+#1+1+1+00+#++1#0+#++1+
```

Problem 4: Decompressor

Write a decompressor for the output of the compressor that you wrote in the previous problem. Of course, your solution must make use of recursion.

Example 1:

input:

```
8
01*1.1.1.
```

output:

```
8
****....
****....
****....
****....
........
........
........
........
```

Example 2:

input:

```
8
001.1*1.1*1.01.1.0*..*1.1.
```

output:

```
8
..**....
..**....
**......
**......
........
........
......*.
......*.
```

Example 3:

input:

```
8
001+1+0+++#0+###01+1+0+++#0+###00###+#1+1+1+00+#++1#0+#++1+
```

output:

```
8
++++++++
++++++++
+#+++#++
###+####+
+######++
++####+++
+++#++++
++++++++
```