Lab session 10: Programming Fundamentals

Lab session 10 consists of five exercises. Each exercise is worth two grade points. All exercises must be solved using recursion in C.

Problem 1: Matching ABC Strings

In this exercise, a *wildcard pattern* is defined as a string consisting of the letters `'A'`, `'B'`, `'C'`, and the wildcard character `'?'`. Such a pattern is used to specify the set of all strings that can be formed by replacing the wildcard characters by a `'A'`, `'B'`, or `'C'`.

For example, for the wilcard pattern `AB?AC?` the possible combinations in lexicographic (i.e. alphabetical) order are:

```
ABAACA
ABAACB
ABAACC
ABBACA
ABBACB
ABBACC
ABCACA
ABCACB
ABCACC
```

Write a program that reads from the input a wilcard pattern and outputs in lexicographic order all possible combinations that match this pattern. You may assume that the pattern has at most 32 characters.

Example 1:

input:
```
AB?AC?
```
output:
```
ABAACA
ABAACB
ABAACC
ABBACA
ABBACB
ABBACC
ABCACA
ABCACB
ABCACC
```

Example 2:

input:
```
??
```
output:
```
AA
AB
AC
BA
BB
BC
CA
CB
CC
```

Example 3:

input:
```
?A?
```
output:
```
AAA
AAB
AAC
BAA
BAB
BAC
CAA
CAB
CAC
```

Problem 2: Well formed Strings of Parentheses

A *well formed string of parentheses* is defined by the following (recursive) rules:

- The empty string is well formed.
- If s is a well formed string, then `(`s`)` is a well formed string.
- If s and t are well formed strings, then their concatenation st is a well formed string.

For example, `"((()))"` and `"()()()"` are well formed strings, while `"(()"`, `")(()"` and `"("` are not well formed strings.
There are exactly 5 well formed strings that have length 6: `"((()))"`, `"(()())"`, `"(())()"`, `"()(())"`, and `"()()()"`.
The input of this problem consists of a non-negative integer `n` (where $1 \leq n \leq 30$). Your program should output the number of well formed strings of parentheses that have length `n`.

Example 1:
input:
```
6
```
output:
```
5
```

Example 2:
input:
```
3
```
output:
```
0
```

Example 3:
input:
```
10
```
output:
```
42
```

Problem 3: Reachable?

Given an array of non-negative integers, you are initially positioned at the first index of the array (i.e. index 0). Each element in the array represents your maximum jump length at that position. Your program should output `YES` if you are able to reach the last index, otherwise it should output `NO`.

For example, if the input arrays is $[3, 0, 3, 1, 0, 1, 0]$, then the output should be `YES` which is shown in the following steps:

1. From index 0, we can make a maximal 'jump' of length 3, so we can reach the indexes 1, 2, and 3. We decide to jump to index 2.
2. From index 2, we can jump to indexes 3, 4, or 5. We choose to jump to index 5.
3. From index 5, we can only move to index 6 which is the index we want to reach.

An input for which the output must clearly be `NO` is $[3, 2, 1, 0, 4]$.

The input for this problem consists of a line containing an integer n ($1 \leq n < 100$), the length of the array, followed by a line with the contents of the array, i.e. n non-negative integers.

Example 1:
input:
```
7
3 0 3 1 0 1 0
```
output:
```
YES
```

Example 2:
input:
```
5
3 2 1 0 4
```
output:
```
NO
```

Example 3:
input:
```
10
4 7 6 5 4 3 2 1 0 0
```
output:
```
NO
```

Problem 4: Arithmetic Expressions

Given the ordered series of digits 1,2,..,9, you must construct arithmetic expressions by placing in between neighbouring digits a + (plus), a − (minus), or a $\star$ (multiplication). Given a positive integer $n > 0$ on the input, you are asked to compute the number of arithmetic expressions using these three operations such that they evaluate to n. Note that multiplication binds stronger than (takes precedence over) addition and subtraction. For example, for $n = 362881$ there is only one possible expressions: $362881 = 1 + 2 * 3 * 4 * 5 * 6 * 7 * 8 * 9$
You do not have to worry about overflow, since the largest value that you can construct is the above example (i.e. $1 + 9!$) which easily fits in a 32 bit integer.
[Note: Your program should not contain hard coded values. The TAs will check this!]

Example 1:
input:
```
5
```
output:
```
28
```

Example 2:
input:
```
6
```
output:
```
23
```

Example 3:
input:
```
362881
```
output:
```
1
```

Problem 5: 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 $2 \leq \texttt{n} \leq 11$), and outputs the number of possible up-down permutations constructed from the series $1, 2, \ldots, \texttt{n}$. 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
```