Lab session 11: Programming Fundamentals

Problem 1: Tri-sums

The input of this problem consists of four lines. The first line contains a positive integer s. Each remaining line contains an array. Each line starts with a positive integer n followed by a colon (':'). The value of n is the length of the array of integers that follows.
The output of your program must be the number of unique triplets (a, b, c) such that $a+b+c = s$ and a is drawn from the first array, b is drawn from the second array, and c is drawn from the third array. The arrays may contain duplicates, but you should return the number of *unique* triplets.

Example 1:

input:
```
42
5: 3 5 6 9 5
7: 19 15 13 8 5 11 3
8: 11 4 13 12 5 5 20 16
```
output:
```
2
```

Example 2:

input:
```
150
10: 5 17 9 16 80 20 43 64 70 70
10: 64 37 52 84 5 58 96 50 21 52
10: 27 0 11 67 12 60 11 2 77 21
```
output:
```
4
```

Example 3:

input:
```
30
10: 16 18 13 0 14 17 17 7 17 15
10: 19 13 9 18 1 5 3 11 12 16
10: 0 18 12 1 14 10 9 0 11 15
```
output:
```
29
```

Problem 2: Cryptarithmetic multiplication

Cryptarithmetic puzzles are mathematical puzzles where letters are used to represent digits, and the goal is to find a valid digit assignment to the letters such that a given arithmetic equation holds true. In the lecture, we discussed the puzzle `SEND+MORE=MONEY`, which has the solution `9567 + 1085 = 10652 (S=9, E=5, N=6, D=7, M=1, O=0, R=8, Y=2).`

In cryptarithmetic problems, we must adhere to several constraints:

- The arithmetic equation must hold true.
- Distinct Digits: Each letter must represent a distinct digit from 0 to 9. In other words, no two letters can have the same assigned digit.
- No Leading Zeros: In the example `SEND+MORE=MONEY` the letters `S` and `M` cannot be assigned the digit 0.

In this problem we consider cryptarithmetic multiplication puzzles, i.e. puzzles where the multiplication of two strings yields the resulting string. An example of such a puzzle is `SIX*TWO=TWELVE`. It has three solutions (see example 1).

Write a program that reads from the input a cryptarithmetic multiplication puzzle. All letters are in uppercase. The input contains no white space (apart from a terminating newline character). The output of your program must be all solutions of the puzzle, or nothing at all if no solutions exist. The solutions must be printed as in the given examples: variables in alphabetic order, and for each variable the assigned digit in ascending order.

[*Note: for this problem, all test cases are visible in Themis. Do not abuse this! Hardcoding solutions automatically means that you will get zero points for this exercise.*]

Example 1:

input:
```
SIX*TWO=TWELVE
```
output:
```
E=0,I=6,L=3,O=8,S=9,T=2,V=7,W=1,X=5
E=0,I=7,L=3,O=5,S=9,T=1,V=8,W=6,X=2
E=0,I=8,L=1,O=5,S=9,T=3,V=7,W=4,X=6
```

Example 2:

input:
```
LA*MUL=TIANI
```
output:
```
A=3,I=1,L=7,M=5,N=9,T=4,U=6
A=5,I=0,L=6,M=3,N=4,T=2,U=1
A=7,I=6,L=8,M=3,N=9,T=2,U=0
A=8,I=4,L=3,M=6,N=1,T=2,U=5
A=9,I=4,L=6,M=5,N=1,T=3,U=0
A=9,I=8,L=2,M=6,N=0,T=1,U=5
```

Example 3:

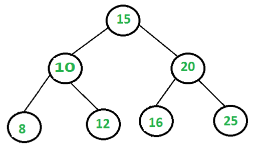
input:
```
IT*WAS=NICE
```
output:
```
A=0,C=2,E=6,I=1,N=7,S=9,T=4,W=5
A=0,C=2,E=6,I=1,N=9,S=7,T=8,W=5
A=0,C=5,E=2,I=1,N=9,S=4,T=3,W=7
A=0,C=6,E=8,I=1,N=5,S=4,T=7,W=3
A=2,C=4,E=3,I=6,N=8,S=9,T=7,W=1
A=3,C=0,E=6,I=5,N=7,S=9,T=4,W=1
A=3,C=4,E=6,I=1,N=9,S=8,T=7,W=5
A=3,C=6,E=8,I=7,N=9,S=2,T=4,W=1
A=4,C=1,E=0,I=3,N=9,S=5,T=8,W=2
A=4,C=2,E=0,I=6,N=9,S=8,T=5,W=1
A=4,C=9,E=6,I=3,N=5,S=2,T=8,W=1
A=6,C=8,E=0,I=1,N=9,S=5,T=2,W=7
A=7,C=0,E=6,I=1,N=8,S=9,T=4,W=5
A=8,C=7,E=0,I=3,N=6,S=2,T=5,W=1
A=9,C=0,E=4,I=3,N=6,S=7,T=2,W=1
A=9,C=3,E=8,I=2,N=5,S=4,T=7,W=1
```

Example 4:

input:
```
CHAOS*US=TRUMP
```
output:
```
UNSOLVABLE
```

Problem 3: Fully Balanced Search Tree

A *search tree* is a tree structure used for storing data in such a way that data items can be searched for efficiently. A tree is called a search tree if the value for each node is greater than any value in subtrees on the left of it, and less than any keys in subtrees on the right of it. An example of such a tree is given in the following figure:



Apart from being a search tree, the tree in the figure is also a *fully balanced tree*. A tree is called fully balanced if the length of all shortest paths from the top node (called the root node) to terminal nodes (called the leaf nodes) is the same. In the example figure, node 15 is the root node, and the nodes 8, 12, 16, and 25 are the leaf nodes. The length of each shortest path from the root node to any leaf is 2. So, the tree is fully balanced.
It is easy to see that the number of nodes in a fully balanced tree with n layers is equal to $2^n - 1$.
We can print a tree on the output in textual format as follows:

- For a tree containing only one node with value `n`, we print `Leaf n`. For example, the tree containing only 42 is printed as `Leaf 42`.
- For a tree with root node n and subtrees L and R, we print `(Tree (L) n (R))`, where `L` and `R` are the recursively printed subtrees L and R. For example, the fully balanced search tree containing the values 1, 2, and 3 is printed as `Tree (Leaf 1) 2 (Leaf 3)`.

The input for this problem is an integer n (where $1 \leq n < 20$), followed by a line containing $2^n - 1$ integers (without duplicates). The output must be the fully balanced binary search tree containing these input values printed according to the described output format.

Example 1:

input:
```
1
42
```
output:
```
Leaf 42
```

Example 2:

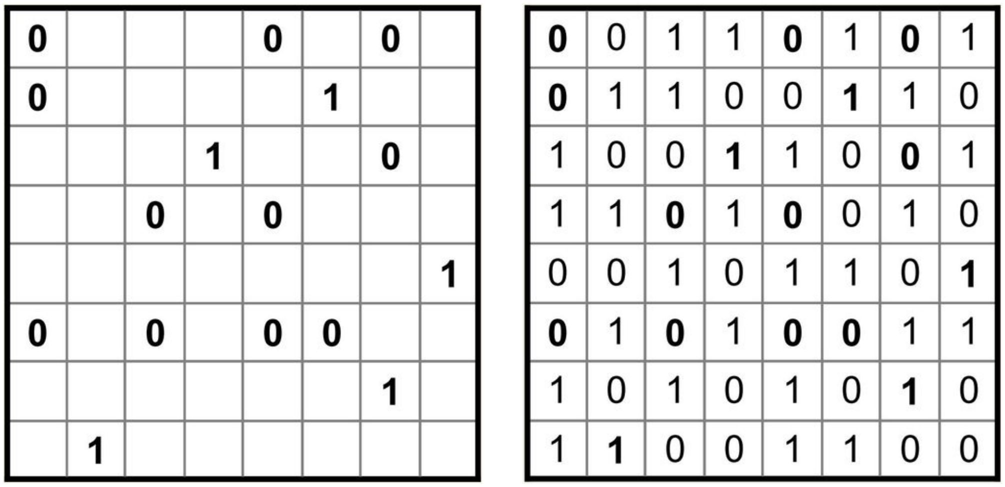
input:
```
2
1 2 3
```
output:
```
Tree (Leaf 1) 2 (Leaf 3)
```

Example 3:

input:
```
3
8 10 12 15 16 20 25
```
output:
```
Tree (Tree (Leaf 8) 10 (Leaf 12)) 15 (Tree (Leaf 16) 20 (Leaf 25))
```

Problem 4: Takuzu Solver

A *takuzu* is a logic number placement puzzle on an $n \times n$ grid, where n is even. In this exercise you may assume that $2 \leq n \leq 14$. The objective is to fill the grid with 1s and 0s, where there is an equal number of 1s and 0s in each row and column (hence four 0s and four 1s) and no more than two of either number adjacent to each other. Moreover, there can be no identical rows, nor can there be identical columns. An example of a Takuzu puzzle and its solutions are given in the following figure.



The first line of the input contains the number n. The remaining n lines consist of the grid. An empty cell is denoted by a full stop ('`.`'). The objective is to fill the empty cells of the grid with 1s and 0s, according to the given rules.

The output of your program must be the solved takuzu. The test sets in Themis are such that each test case has exactly one unique solution.

[*Note: the grading of this exercise differs from the grading of the other exercises. There is no subtraction for (too) many submissions, and you will receive 0.2 grade points for each passed test case. Moreover, all test cases are visible in Themis. Do not abuse this! Hardcoding solutions automatically means that you will get zero points for this exercise.*]

Example 1:

input:
```
2
10
01
```
output:
```
10
01
```

Example 2:

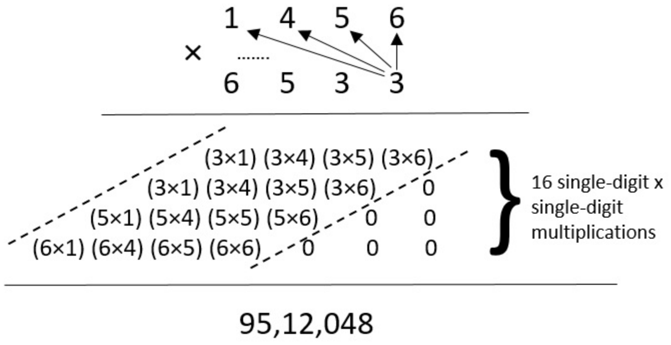
input:
```
2
1.
..
```
output:
```
10
01
```

Example 3:

input:
```
4
.1.0
..0.
.0..
11.0
```
output:
```
0110
1001
0011
1100
```

Problem 5: Karatsuba Multiplication

In the following figure, the multiplication 1456×6533 is performed using the *long multiplication* algorithm (as taught on primary school). For two n-digit numbers, this method performs n^2 single-digit products.



In 1962, Anatoly Karatsuba published a faster multiplication algorithm, which is nowadays known as *Karatsuba multiplication*. It is a divide-and-conquer algorithm that reduces the multiplication of two n-digit numbers to three multiplications of $n/2$-digit numbers. By repeating this reduction recursively, at most $n^{\log_2 3} \approx n^{1.58}$ single-digit multiplications are needed, which is asymptotically more efficient than the long multiplication algorithm.

The algorithm works as follows. Let x and y be $2n$-digit numbers (so, in the above example, $n = 2$, $x = 1456$, $y = 6533$), and we want to compute $z = x \times y$.
Clearly, can write $x = a \times 10^n + b$ and $y = c \times 10^n + d$ (in the example, $a = 14$, $b = 56$, $c = 65$, and $d = 33$). Using simple arithmetic, we find

$$x \times y = (a \times 10^n + b) \times (c \times 10^n + d) = (a \times c)10^{2n} + (a \times d + b \times c)10^n + b \times d$$

At first sight, this calculation did not help much. But there is a trick.

$$(a \times d + b \times c) = (a + b) \times (c + d) - a \times c - b \times d$$

In other words, if we compute $a \times c$ and $b \times d$ first (using two n-digit multiplications), then we can reuse these results to compute $(a \times d + b \times c)$ using only one n-digit multiplication (at the expensive of some extra additions and subtractions).

For the above example, we first compute $a \times c = 14 \times 65 = 910$, $b \times d = 56 \times 33 = 1848$, and next we compute $(14 + 56) \times (65 + 33) - 910 - 1848 = 70 \times 98 - 910 - 1848 = 4102$. The final answer is then $x \times y = 910 \cdot 10^4 + 4102 \cdot 10^2 + 1848 = 9512048$.

Clearly, the numbers in this example are that small that there is no need to use an algorithm like this. However, there are many applications in which we need to deal with huge numbers (for example in number theory, and in particular in cryptography. In case you are interested, look up the RSA algorithm). For such big numbers, the algorithm recursively computes the products $a \times c$, $b \times d$ and $(a + b) \times (c + d)$.

On Themis, you can find the file `karatsuba.zip`. It contains 6 files: `main.c`, `natnum.c`, `natnum.h`, `karatsuba.h`, `karatsuba.c`, and a `Makefile`. After unzipping, you can compile the program by simply typing `make`. It will produce an `a.out` that expects two natural numbers x and y on the input, and it outputs the product $x \times y$.

The program is fully functional, but it implements natural numbers as `unsigned longs`. In other words, the computed product will only be correct for values less than 2^{64}. For example, the program indeed returns 9512048 for inputs 1456 and 6533. But it returns an incorrect answer for inputs 1234567890 and 98765432100, because $1234567890 \times 98765432100 = 121932631112635269000 > 2^{64}$.

Your task is to replace the data type for `natNumber` in the file `natnum.h` by some data structure that is able to store natural numbers with an unbounded number of digits. Moreover, you will need to re-implement all operations in the file `natnum.c` accordingly. You are not allowed to make changes in any of the other files.

If you do this correctly, then the program should be able to compute the above example product without overflow. You need to submit to Themis all 6 files (so including the Makefile, not in a zip file!).

Example 1:

input:
```
1456 6533
```
output:
```
9512048
```

Example 2:

input:
```
6533 1456
```
output:
```
9512048
```

Example 3:

input:
```
1234567890 98765432100
```
output:
```
121932631112635269000
```