Lab session 5: Programming Fundamentals

Problem 1: Histogram

A teacher has made a list of the grades that his students scored on their midterm. However, such a list gives little insight in the distribution of the grades. Therefore, he would like to have a program that plots a simple histogram of the grades. In this exercise you will make this program for him.

The first line of the input contains the number of grades. This is followed by the grades themselves. Grades are integer values from the interval $[1, 10]$. For testing purposes, there are a few input files available in Themis. For example, you can offer the file `1.in` as an input to your program as follows:

```
./a.out < 1.in
```

In Themis, you will also find the file `histogram.c` which contains an incomplete program. You need to perform the following steps to complete the program.

1. Add a function `readGrades` that reads the grades from the input and returns a dynamically allocated array in which they are stored.
2. Add a function `computeFrequencies` which, given an array with grades, returns a small array with the frequency distribution of the grades (a so-called *histogram*).
3. Add a function `printHistogram` which, given the histogram, prints a simple frequency plot on the screen.

Make sure that your program produces output exactly the same as in the following example. Note that the columns of the plot are separated by a (single) space-character.

input:
```
29
6 3 8 6 7 4 8 9 2 10 4 9 5 7 4 8 6 7 2 10 4 1 8 3 6 3 6 9 4
```
output:
```
. . . * . * . . . .
. . . * . * . * . .
. . * * . * * * * .
. * * * . * * * * *
* * * * * * * * * *
1 2 3 4 5 6 7 8 9 10
```

Problem 2: Resolution rule

In propositional logic, a *clause* is an expression which is the *disjunction* of a finite collection of literals (atoms or their negations).
In this exercise, we impose two extra rules on clauses:

- For the atoms, we only allow the upper case letters A, B, .., Z. Hence, there can be at most 26 atoms in a clause.
- A clause never contains an atom and its negation. This makes sense, because $A \vee \neg A \vee \alpha$ reduces to **true** for any expression α.

An example of a valid clause is:

$$A \vee B \vee \neg P \vee \neg C \vee \neg Q \vee E$$

We represent this clause with the following ASCII notation (where ˜ is used to denote negation): `[A,B,~P,~C,~Q,E]`.

The *resolution rule* is an inference rule that produces a new clause implied by two clauses containing *complementary* literals. Two literals are said to be complements if one is the negation of the other. The resolution rule is given by the following formula ((where α and β are propositional expressions):

$$\frac{A \vee \alpha, \quad \neg A \vee \beta}{\alpha \vee \beta}$$

The concluding expression $\alpha \vee \beta$ is called the *resolvent*.

As a concrete example, from the clauses $A \vee \neg B \vee C$ and $A \vee P \vee B \vee \neg D$ we produce $A \vee C \vee A \vee P \vee \neg D$ which reduces to (eliminating the duplicate occurence of A) the resolvent $A \vee C \vee P \vee \neg D$.

Write a program that reads from the input the ASCII representation of two clauses. The output of the program should be the ASCII representation of the resolvent of the two clauses, or `NO RESOLVENT` if the two clauses do not contain complementary literals. If the resolvent exists, then the ASCII representation of the resolvent must be sorted in alphabetical order. Note that you may assume that the two input clauses are chosen such that they produce at most one resolvent (in other words, there is at most one pair of complementary literals).

Example 1:
input:
```
[A,~B,C][A,P,B,~D]
```
output:
```
[A,C,~D,P]
```

Example 2:
input:
```
[A][~A]
```
output:
```
[]
```

Example 3:
input:
```
[A,B][C,~D]
```
output:
```
NO RESOLVENT
```

Problem 3: Zeckendorf representation

Recall that the Fibonacci numbers $F(i)$ are defined as:

$$F(0) = 0,\ F(1) = 1,\ F(n) = F(n-1) + F(n-2)\ \ (\text{for } n > 1)$$

Zeckendorf's theorem states that every positive integer n can be represented uniquely as the sum of one or more distinct Fibonacci numbers $F(i)$ (where $i \geq 2$) in such a way that the sum does not include any two consecutive Fibonacci numbers. We call such a sum the *Zeckendorf representation* of n.
For example, the Zeckendorf representation of 64 is $64 = F(10) + F(6) + F(2) = 55 + 8 + 1$. Also $64 = F(10) + F(5) + F(4) + F(2) = 55 + 5 + 3 + 1$, but this is not a Zeckendorf representation because the consecutive Fibonacci numbers $F(4)$ and $F(5)$ are used.

The input of this problem is a positive integer n, where $1 \leq n \leq 10^9$. The output should be the Zeckendorf representation of n in the format as given in the below examples.

Example 1:

input:
```
64
```
output:
```
F(10)+F(6)+F(2)=55+8+1
```

Example 2:

input:
```
42
```
output:
```
F(9)+F(6)=34+8
```

Example 3:

input:
```
997
```
output:
```
F(16)+F(6)+F(3)=987+8+2
```

Problem 4: Searching sums

In this exercise, you are given a positive integer n and a square matrix of size $m \times m$, where $1 \leq m \leq 100$. The objective is to find how many times you can get the value n, by summing consecutive values from the matrix horizontally, vertically, or diagonally. For example, in the following matrix the sum $n = 42$ can be constructed in four different ways: $42 = 10+20+12 = 9+13+20 = 9+11+22 = 20+0+22$.

0	3	7	9	**10**
9	**13**	**20**	5	**20**
12	**11**	33	**0**	**12**
17	39	**22**	3	18
12	15	32	1	17

Write a program that accepts as its input two positive integers `n` and `m`: `n` is the target sum, while `m` defines the size of the `m`×`m` matrix. The remaining input consist of `m` lines, containing `m` non-negative integers (the rows of the matrix). You program should print the number of ways that the sum `n` can be found in the matrix. Note that a sum must contain at least two terms!

Example 1:

input:
```
42 5
0 3 7 9 10
9 13 20 5 20
12 11 33 0 12
17 39 22 3 18
12 15 31 1 17
```
output:
```
4
```

Example 2:

input:
```
42 5
0 3 7 9 10
9 13 20 5 20
12 11 33 1 12
17 39 22 3 18
42 15 31 1 17
```
output:
```
4
```

Example 3:

input:
```
42 5
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
16 17 18 19 20
21 22 23 24 25
```
output:
```
7
```

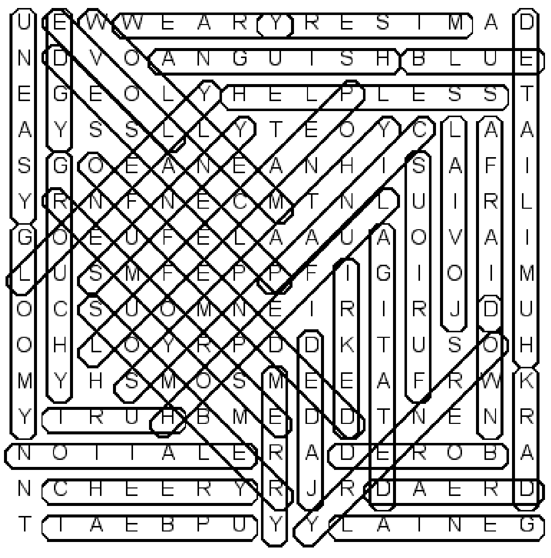
Problem 5: Word search puzzle

A *word search puzzle* is a word game that consists of a square grid filled with letters and a list of words. The objective of the puzzle is to find and mark the words, which are hidden in the grid. The words may be placed horizontally, vertically, or diagonally. Moreover, the spelling may be in reverse (e.g. `ELZZUP` instead of `PUZZLE`). Note that some letters in the grid may be part of several words. Once the puzzle is completed (i.e. all words are found), the remaining unmarked letters form the solution of the puzzle.

In the following figure, you see an example of such a puzzle and its solution. The remaining letters form the word `ASTONISHMENT`, which is the solution of the puzzle.

```
AFRAID ANGUISH BLUE BORED CHEERY DARK DOWN HURT FURIOUS
DREAD EDGY ELATION GENIAL GLOOMY GROUCHY PANIC AGITATED
HELPLESS HOPEFUL HUMILIATED IRKED JADED JOVIAL UPBEAT
LONELY LOVE MELLOW MERRY MISERY OFFENDED ORNERY WEARY
PEACEFUL PLEASED REMORSE SOMBER SUNNY SYMPATHY UNEASY
```

```
U E W W E A R Y R E S I M A D
N D V O A N G U I S H B L U E
E G E O L Y H E L P L E S S T
A Y S S L L Y T E O Y C L A A
S G O E A N E A N H I S A F I
Y R N F N E C M T N L U I R L
G O E U F E L A A U A O V A I
L U S M F E P P F I G I O I M
O C S U O M N E I R I R J D U
O H L O Y R P D D K T U S O H
M Y H S M O S M E E A F R W K
Y T R U H B M E D D T N E N R
N O I T A L E R A D E R O B A
N C H E E R Y R J R D A E R D
T T A E B P U Y Y L A I N E G
```



Write a program that reads from the input a word search puzzle, and outputs the solution of the puzzle. The input is given by a line containing a positive integer `n`, which denotes the size of the `n × n` grid. You may assume that $4 \le \mathbf{n} \le 20$. The next `n` lines contain the rows of the grid, i.e. strings with `n` uppercase letters (followed by newlines). The remaining input consists of a list of hidden words (each having a length ≤ 20). Each word is given on a separate line, and is written using uppercase letters. The list is terminated with a line containing only a period (`'.'`). In Themis, you can find three example input files (`1.in`, `2.in`, and `3.in`) and their corresponding output files (solutions). These can be used for *input redirection*, which saves a lot of typing in the testing of your program: `./a.out < 1.in`