Lab session 4: Programming Fundamentals

Problem 1: Assembly code

In this exercise you are expected to write a program that simulates a small (hypothetical) processor that has only eight assembly instructions and two registers `R1` and `R2`. The registers can only contain integer values. In the following description of the machine instructions, `Rx` and `Ry` represent the registers (where `x` can be 1 or 2, `y` can be 1 or 2, and possibly `x==y`):

- `OUT Rx` This instruction prints the value of the corresponding register on the output (followed by a newline).
- `LCS Rx n` This instruction Loads a ConStant value `n` in the specified register.
- `INC Rx` This instruction INCrements the value of the specified register.
- `DEC Rx` This instruction DECrements the value of the specified register.
- `ADD Rx Ry` This instruction implements `Ry = Rx + Ry`.
- `SUB Rx Ry` This instruction implements `Ry = Rx - Ry`.
- `MUL Rx Ry` This instruction implements `Ry = Rx * Ry`.
- `END` Each machine program must end with this instruction.

Write a program that simulates the execution of a machine code program that consists of (some of) the above instructions.
[Note: this problem has 20 test cases in Themis.]

Example 1:

input:

```
LCS R1 1
LCS R2 5
ADD R1 R2
LCS R1 2
MUL R2 R2
INC R1
DEC R2
OUT R1
OUT R2
END
```

output:

```
3
35
```

Example 2:

input:

```
LCS R1 1
LCS R2 0
INC R1
ADD R1 R1
DEC R1
LCS R2 2
MUL R1 R2
LCS R1 7
MUL R2 R1
OUT R1
END
```

output:

```
42
```

Problem 2: Pangram

A *Pangram* is a sentence using every letter of the alphabet at least once. A famous English pangram is "THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG".

Write a program that reads from the input a sentence, and outputs whether the sence is a *pangram*, or *no pangram*. The input consists of a sentence containing spaces and uppercase letters from the conventional 26 letter alphabet ('A', 'B', .., 'Z'). The sentence ends with a dot ('.'). Make sure that the output of your program has the same format as in the following examples. Note that there are no spaces in the output, and that the output ends with a newline (`\n`):

Example 1:

input:

```
THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG.
```

output:

```
PANGRAM
```

Example 2:

input:

```
AMAZINGLY FEW DISCOTHEQUES PROVIDE JUKEBOXES.
```

output:

```
PANGRAM
```

Example 3:

input:

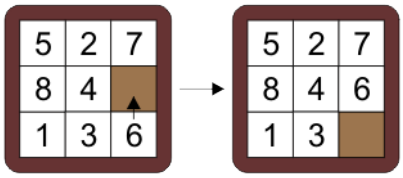
```
THIS SENTENCE IS CLEARLY NOT A PANGRAM.
```

output:

```
NO PANGRAM
```

Problem 3: Sliding Puzzle

A *sliding n-puzzle* is a number puzzle, that is played on an $n \times n$ grid containing $n^2 - 1$ tiles, and an empty space. The tiles are numbered $1, 2, .., n^2 - 1$. Tiles can be moved using the moves `UP`, `DOWN`, `LEFT`, `RIGHT`. For example, in the following figure, the move made is `UP 6`, meaning that the tile with the number 6 was moved up (i.e, the empty space went down).



The object of the puzzle is to place the tiles in ascending order by making a series of sliding moves. A possible solution is given in the following figure. Note that the location of the empty space may be anywhere in a solution, so the empty space need not be in the bottom right position (as in the figure).

Write a program that reads from the input an integer `n`, where $2 \leq$ `n` ≤ 8. The next `n` lines contain the rows of an `n` $\times$ `n` grid of numbers. The grid is the initial configuration of an `n`-puzzle. The empty space is encoded as the number 0. The rest of the input consists of a series of moves, terminated using the word END.

Your program should output `SOLVED` if the series of moves (starting from the initial configuration) solves the puzzle. If the puzzle is not solved, your program should output `UNSOLVED`. The program should output `INVALID` in case an invalid move occurs in the move series. Note that there are no spaces in the output, and that the output ends with a newline (`\n`):

Example 1:

input:

```
3
1 3 0
4 2 5
6 7 8
RIGHT 3
UP 2
END
```

output:

```
SOLVED
```

Example 2:

input:

```
3
1 0 3
4 2 5
6 7 8
LEFT 3
RIGHT 3
END
```

output:

```
UNSOLVED
```

Example 3:

input:

```
2
3 1
0 2
LEFT 2
DOWN 1
RIGHT 3
RIGHT 2
END
```

output:

```
INVALID
```

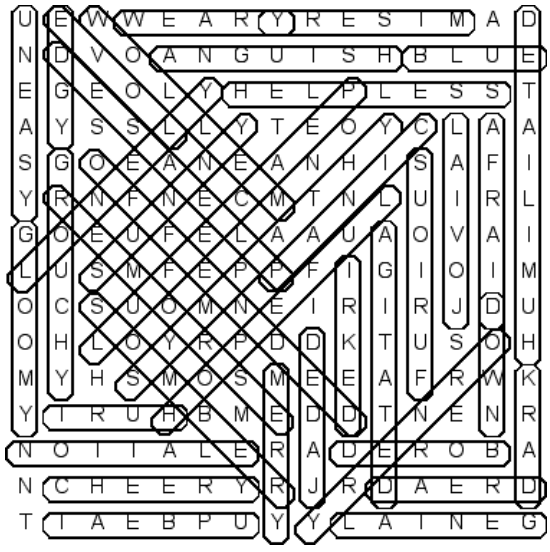
Problem 4: Word search puzzle

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 word 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 $4 \leq \texttt{n} \leq 20$. 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 dot ('.'). 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`

Problem 5: Supernacci (ADT)

In the lecture, you have learned that the following code fragment computes the `n`th Fibonacci number (where `n` is read from the input):

```
int n, a=0, b=1;
scanf("%d", &n);
printf("fib(%d)=", n);
while (n > 0) {
  b = a + b;
  a = b - a;
  n--;
}
printf("%d\n", a);
```

However, for inputs `n` that are greater than 46 this program will fail, because of integer overflow. Replacing the type `int` by `unsigned long long` helps a bit, but not much because that version will overflow for $n > 93$.

On Themis you can download the file `supernacci.c`, in which you can find a abstract data stucture (ADT) that can represent natural numbers that have at most 1000 decimal digits.

```
#define MAXDIG 1000

typedef struct {
  int numDigits;       // number of digits
  int digits[MAXDIG];  // decimal digits of big integer
} unsignedBigInt;
```

You are required to implement the following ADT operations:

```
unsignedBigInt makeUnsignedBigIntFromInt(unsigned int n);
  // returns the unsignedBigInt representation of n.

void printBigInt(unsignedBigInt n);
  // print the number n on the screen (not followed by a newline)

void printlnBigInt(unsignedBigInt big);
  // print the number n on the screen (followed by a newline)

unsignedBigInt add(unsignedBigInt a, unsignedBigInt b);
  // returns the unsignedBigInt representation of a+b.

unsignedBigInt subtract(unsignedBigInt a, unsignedBigInt b);
  // returns the unsignedBigInt representation of a-b.
  // You may assume that 0<=b<=a, so the result is non-negative.
```

Using this data type, the following program fragment is able to compute `fib(n)` als long as `fib(n)` has at most 1000 digits.

```
int main(int argc, char *argv[]) {
  int n;
  scanf("%d", &n);
  printf("fib(%d)=", n);

  unsignedBigInt a = makeUnsignedBigIntFromInt(0);
  unsignedBigInt b = makeUnsignedBigIntFromInt(1);

  while (n > 0) {
    b = add(a,b);
    a = subtract(b,a);
    n--;
  }

  printlnBigInt(a);
  return 0;
}
```

Example 1:

input:

```
5
```

output:

```
fib(5)=5
```

Example 2:

input:

```
50
```

output:

```
fib(50)=12586269025
```

Example 3:

input:

```
142
```

output:

```
fib(142)=212207101440105399533740733471
```