**10050 – Hartals** **02/08/2010**

**Topic Data Structures : Lists**

## Problem D: Hartals

A social research organization has determined a simple set of parameters to simulate the behavior of the political parties of our country. One of the parameters is a positive integer *h* (called the *hartal parameter*) that denotes the average number of days between two successive *hartals* (strikes) called by the corresponding party. Though the parameter is far too simple to be flawless, it can still be used to forecast the damages caused by *hartals*. The following example will give you a clear idea:

Consider three political parties. Assume $h_1 = 3$, $h_2 = 4$ and $h_3 = 8$ where $h_i$ is the *hartal parameter* for party $i$ ( $i = 1, 2, 3$). Now, we will simulate the behavior of these three parties for $N = 14$ days. One must always start the simulation on a Sunday and assume that there will be no *hartals* on weekly holidays (on Fridays and Saturdays).

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Days | | | | | | | | | | | | | | |
| | Su | Mo | Tu | We | Th | Fr | Sa | Su | Mo | Tu | We | Th | Fr | Sa |
| Party 1 | | | x | | | x | | | x | | | x | | |
| Party 2 | | | | x | | | | x | | | | x | | |
| Party 3 | | | | | | | | x | | | | | | |
| Hartals | | | 1 | 2 | | | | 3 | 4 | | | 5 | | |

The simulation above shows that there will be exactly 5 *hartals* (on days 3, 4, 8, 9 and 12) in 14 days. There will be no *hartal* on day 6 since it is a Friday. Hence we lose 5 working days in 2 weeks.

In this problem, given the hartal parameters for several political parties and the value of *N*, your job is to determine the number of working days we lose in those *N* days.

### Input

The first line of the input consists of a single integer *T* giving the number of test cases to follow.

The first line of each test case contains an integer *N* ( 7<=N<=3650 ) giving the number of days over which the simulation must be run. The next line contains another integer *P* (1<=P<=100 ) representing the number of political parties in this case. The *i* - *t*h of the next *P* lines contains a positive integer $h_i$ (which will never be a multiple of 7) giving the *hartal parameter* for party *i* (1<=i<=P ).

### Output

For each test case in the input output the number of working days we lose. Each output must be on a separate line.

### Sample Input

```
2
14
3
3
4
8
100
4
12
15
25
40
```

### Sample Output

```
5
15
```

## 442 - Matrix Chain Multiplication

# Matrix Chain Multiplication

Suppose you have to evaluate an expression like A*B*C*D*E where A,B,C,D and E are matrices. Since matrix multiplication is associative, the order in which multiplications are performed is arbitrary. However, the number of elementary multiplications needed strongly depends on the evaluation order you choose.

For example, let A be a 50*10 matrix, B a 10*20 matrix and C a 20*5 matrix. There are two different strategies to compute A*B*C, namely (A*B)*C and A*(B*C).

The first one takes 15000 elementary multiplications, but the second one only 3500.

Your job is to write a program that determines the number of elementary multiplications needed for a given evaluation strategy.

### Input Specification

Input consists of two parts: a list of matrices and a list of expressions.

The first line of the input file contains one integer *n* ( 1<=n<=26 ), representing the number of matrices in the first part. The next *n* lines each contain one capital letter, specifying the name of the matrix, and two integers, specifying the number of rows and columns of the matrix.

The second part of the input file strictly adheres to the following syntax (given in EBNF):

```
SecondPart = Line { Line } <EOF>
Line       = Expression <CR>
Expression = Matrix | "(" Expression Expression ")"
Matrix     = "A" | "B" | "C" | ... | "X" | "Y" | "Z"
```

### Output Specification

For each expression found in the second part of the input file, print one line containing the word "error" if evaluation of the expression leads to an error due to non-matching matrices. Otherwise print one line containing the number of elementary multiplications needed to evaluate the expression in the way specified by the parentheses.

### Sample Input

```
9
A 50 10
B 10 20
C 20 5
D 30 35
E 35 15
F 15 5
G 5 10
H 10 20
I 20 25
A
B
C
(AA)
(AB)
(AC)
(A(BC))
((AB)C)
(((((DE)F)G)H)I)
(D(E(F(G(HI)))))
((D(EF))((GH)I))
```

**Sample Output**

```
0
0
0
error
10000
error
3500
15000
40500
47500
15125
```

---

### 11234 - Expressions

## Problem E: Expressions

Arithmetic expressions are usually written with the operators in between the two operands (which is called infix notation). For example, *(x+y)*(z-w)* is an arithmetic expression in infix notation. However, it is easier to write a program to evaluate an expression if the expression is written in postfix notation (also known as reverse polish notation). In postfix notation, an operator is written behind its two operands, which may be expressions themselves. For example, *x y + z w - **  is a postfix notation of the arithmetic expression given above. Note that in this case parentheses are not required.

To evaluate an expression written in postfix notation, an algorithm operating on a stack can be used. A stack is a data structure which supports two operations:

1. **push**: a number is inserted at the top of the stack.
2. **pop**: the number from the top of the stack is taken out.

During the evaluation, we process the expression from left to right. If we encounter a number, we push it onto the stack. If we encounter an operator, we pop the first two numbers from the stack, apply the operator on them, and push the result back onto the stack. More specifically, the following pseudocode shows how to handle the case when we encounter an operator O:

```
a := pop();
b := pop();
push(b O a);
```

The result of the expression will be left as the only number on the stack.

Now imagine that we use a queue instead of the stack. A queue also has a push and pop operation, but their meaning is different:

1. **push**: a number is inserted at the end of the queue.
2. **pop**: the number from the front of the queue is taken out of the queue.

Can you rewrite the given expression such that the result of the algorithm using the queue is the same as the result of the original expression evaluated using the algorithm with the stack?

#### Input Specification

The first line of the input contains a number **T** ($T \leq 200$). The following **T** lines each contain one expression in postfix notation. Arithmetic operators are represented by uppercase letters, numbers are represented by lowercase letters. You may assume that the length of each expression is less than *10000* characters.

#### Output Specification

For each given expression, print the expression with the equivalent result when using the algorithm with the queue instead of the stack. To make the solution unique, you are not allowed to assume that the operators are associative or commutative.

**Sample Input**

```
2
xyPzwIM
abcABdefgCDEF
```

**Sample Output**

```
wzyxIPM
gfCecbDdAaEBF
```

---

**133 - The Dole Queue**

## The Dole Queue

In a serious attempt to downsize (reduce) the dole queue, The New National Green Labour Rhinoceros Party has decided on the following strategy. Every day all dole applicants will be placed in a large circle, facing inwards. Someone is arbitrarily chosen as number 1, and the rest are numbered counter-clockwise up to N (who will be standing on 1's left). Starting from 1 and moving counter-clockwise, one labour official counts off k applicants, while another official starts from N and moves clockwise, counting m applicants. The two who are chosen are then sent off for retraining; if both officials pick the same person she (he) is sent off to become a politician. Each official then starts counting again at the next available person and the process continues until no-one is left. Note that the two victims (sorry, trainees) leave the ring simultaneously, so it is possible for one official to count a person already selected by the other official.

### Input

Write a program that will successively read in (in that order) the three numbers (N, k and m; k, m > 0, 0 < N < 20) and determine the order in which the applicants are sent off for retraining. Each set of three numbers will be on a separate line and the end of data will be signalled by three zeroes (0 0 0).

### Output

For each triplet, output a single line of numbers specifying the order in which people are chosen. Each number should be in a field of 3 characters. For pairs of numbers list the person chosen by the counter-clockwise official first. Separate successive pairs (or singletons) by commas (but there should not be a trailing comma).

### Sample input

```
10 4 3
0 0 0
```

### Sample output

```
ΔΔ 4 ΔΔ 8, ΔΔ 9 ΔΔ 5, ΔΔ 3 ΔΔ 1, ΔΔ 2 ΔΔ 6, Δ 10, ΔΔ 7
```

where Δ represents a space.