Chaves

Nome do arquivo: `chaves.c`, `chaves.cpp`, `chaves.pas`, `chaves.java`, `chaves.js`, `chaves.py2` *ou* `chaves.py3`

Seu amigo Juca está enfrentando problemas com programação. Na linguagem C, algumas partes do código devem ser colocadas entre chaves `"{ }"` e ele frequentemente esquece de colocá-las ou as coloca de forma errada. Porém, como Juca tem dificuldade para entender os erros de compilação, ele nunca sabe exatamente o que procurar. Por isso ele te pediu para fazer um programa que determine se um código está com as chaves balanceadas, ou seja, se é válido. Um código está com as chaves balanceadas se:

- Não há chaves (como por exemplo "`Bom`" ou "`Correto`");
- O código é composto por uma sequência de códigos válidos (como por exemplo "`Bom Correto`" ou "`{}{}`" ou "`{}Correto`"); ou
- O código é formado por um código válido entre chaves (como por exemplo "`{{}}`" ou "`{Bom}`").

O código de Juca é composto por N linhas de até 100 caracteres cada. Pode haver linhas vazias e espaços consecutivos.

Entrada

A primeira linha contém um inteiro N, representando o número de linhas no código. As N linhas seguintes contém até 100 caracteres.

Saída

Seu programa deve produzir uma única linha, contendo uma única letra, `"S"` se o código está com as chaves balanceadas e `"N"`, caso contrário.

Restrições

- $1 \leq N \leq 10^3$.

Informações sobre a pontuação

- Em um conjunto de casos de teste equivalente a 50 pontos, $N = 1$ e todos os carateres são `"{"` ou `"}"` (como no terceiro exemplo).

Exemplos

Entrada	**Saída**

6
#include <stdio.h>
 	
int main(void)
{
  printf("Hello World\n");
}
	S

Entrada	**Saída**

5
{I{N{
}F{])
 	
}O}R{
}M}A{T}I{C@!!{onze}!!}
	S

Entrada	**Saída**
1 {{}}{{}}}	N

Entrada	**Saída**
1 {{{3}}}{{{2}}a{{1}}{0}	N

Expressões

Nome do arquivo fonte: `expressoes.c`, `expressoes.cpp`, `expressoes.pas`, `expressoes.java`, *ou* `expressoes.py`

Pedrinho e Zezinho estão precisando estudar resolução de expressões matemáticas para uma prova que irão fazer. Para isso, eles querem resolver muitos exercícios antes da prova. Como sabem programar, então decidiram fazer um gerador de expressões matemáticas.

O gerador de expressões que eles criaram funciona em duas fases. Na primeira fase é gerada uma cadeia de caracteres que contém apenas os caracteres '{', '[', '(', '}', ']' e ')'. Na segunda fase, o gerador adiciona os números e operadores na estrutura criada na primeira fase. Uma cadeia de caracteres é dita *bem definida* (ou válida) se atende as seguintes propriedades:

1. Ela é uma cadeia de caracteres vazia (não contém nenhum caractere).
2. Ela é formada por uma cadeia *bem definida* envolvida por parênteses, colchetes ou chaves. Portanto, se a cadeia S é *bem definida*, então as cadeias (S), $[S]$ e $\{S\}$ também são *bem definidas*.
3. Ela é formada pela concatenação de duas cadeias *bem definidas*. Logo, se as cadeias X e Y são *bem definidas*, a cadeia XY é *bem definida*.

Depois que Pedrinho e Zezinho geraram algumas expressões matemáticas, eles perceberam que havia algum erro na primeira fase do gerador. Algumas cadeias não eram *bem definidas*. Eles querem começar a resolver as expressões o mais rápido possível, e sabendo que você é um ótimo programador (e participa da OBI) resolveram pedir que escreva um programa que dadas várias cadeias geradas na primeira fase, determine quais delas são *bem definidas* e quais não são.

Entrada

A entrada é composta por diversas instâncias. A primeira linha da entrada contém um inteiro T indicando o número de instâncias. Em seguida temos T linhas, cada uma com uma cadeia A.

Saída

Para cada instância imprima uma linha contendo a letra `S` se a cadeia é *bem definida*, ou a letra `N` caso contrário.

Restrições

- $1 \leq T \leq 20$.
- a cadeia de caracteres A tem entre 1 e 100000 caracteres.
- a cadeia de caracteres A contém apenas caracteres '{', '[', '(', '}', ']' e ')'.

Exemplos

Entrada	**Saída**
12 () [] {} (] }{ ([{}]) {}()[] ()] {[] ((([{}{}()[]])(){}) ((((((((((){([])}])))))))))	S S S N N S S N N N S N

Telefone

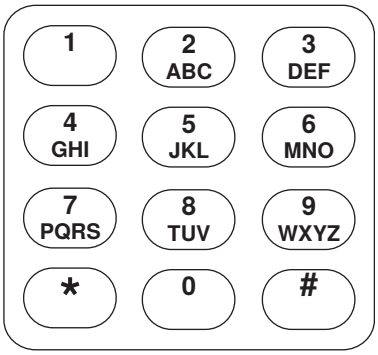
Nome do arquivo fonte: `telefone.c`, `telefone.cpp` *ou* `telefone.pas`

As primeiras redes públicas de telefonia foram construídas pela AT&T no começo do século XX. Elas permitiam que seus assinantes conversassem com a ajuda de uma *telefonista*, que conectava as linhas dos assinantes com um cabo especial.

Essas redes evoluíram muito desde então, com a ajuda de vários avanços tecnológicos. Hoje em dia, essas redes atendem centenas de milhões de assinantes; ao invés de falar diretamente com uma telefonista, você pode simplesmente discar o número da pessoa desejada no telefone.

Cada assinante recebe um número de telefone — por exemplo, 55–98–234–5678. Qualquer pessoa que discar esse número consegue então falar com a pessoa do outro lado da linha. Os hífens no número de telefone são só para facilitar a leitura, e não são discados no telefone.

Para que fique mais fácil de se lembrar de um número de telefone, muitas companhias divulgam números que contém letras no lugar de dígitos. Para convertê-los de volta para dígitos, a maioria dos telefones tem letras nas suas teclas:



Ao invés de discar uma letra, disca-se a tecla que contém aquela letra. Por exemplo, se você quiser discar o número 0800–FALE–SBC, você na realidade discaria 0800–3253–722.

A sua avó tem reclamado de problemas de vista — em particular, ela não consegue mais enxergar as letrinhas nas teclas do telefone, e por isso queria que você fizesse um programa que convertesse as letras em um número de telefone para dígitos.

Entrada

A entrada contém um único conjunto de testes, que deve ser lido do *dispositivo de entrada padrão* (normalmente o teclado). A entrada é composta de apenas uma linha, contendo o número de telefone que deve ser traduzido. O número de telefone contém entre 1 e 15 caracteres, que podem ser dígitos e '`0`' a '`9`', letras de '`A`' a '`Y`' e hífens ('`-`').

Saída

Seu programa deve imprimir, na *saída padrão*, uma única linha, contendo o número de telefone com as letras convertidas para dígitos. Hífens no número telefone devem ser mantidos no número de telefone de saída.

Exemplo de entrada	**Exemplo de saída**
`55-98-234-5678`	`55-98-234-5678`

Exemplo de entrada	**Exemplo de saída**
`0800-FALE-SBC`	`0800-3253-722`

Botas Trocadas

Nome do arquivo: `botas.c`, `botas.cpp`, `botas.pas`, `botas.java`, `botas.js` *ou* `botas.py`

A divisão de Suprimentos de Botas e Calçados do Exército comprou um grande número de pares de botas de vários tamanhos para seus soldados. No entanto, por uma falha de empacotamento da fábrica contratada, nem todas as caixas entregues continham um par de botas correto, com duas botas do mesmo tamanho, uma para cada pé. O sargento mandou que os recrutas retirassem todas as botas de todas as caixas para reembalá-las, desta vez corretamente.

Quando o sargento descobriu que você sabia programar, ele solicitou com a gentileza habitual que você escrevesse um programa que, dada a lista contendo a descrição de cada bota entregue, determina quantos pares corretos de botas poderão ser formados no total.

Entrada

A primeira linha da entrada contém um inteiro N indicando o número de botas individuais entregues. Cada uma das N linhas seguintes descreve uma bota, contendo um número inteiro M e uma letra L, separados por um espaço em branco. M indica o número do tamanho da bota e L indica o pé da bota: $L =$ '`D`' indica que a bota é para o pé direito, $L =$ '`E`' indica que a bota é para o pé esquerdo.

Saída

Seu programa deve imprimir uma única linha contendo um único número inteiro indicando o número total de pares corretos de botas que podem ser formados.

Restrições

- $2 \leq N \leq 10^4$
- N é par
- $30 \leq M \leq 60$
- L é o caractere '`D`' ou o caractere '`E`'

Exemplos

Entrada	**Saída**
4 40 D 41 E 41 D 40 E	2

Entrada	**Saída**
6 38 E 39 E 40 D 38 D 40 D 37 E	1