Competidor(a): _____

Número de inscrição: _____–_____ *(opcional)*

Este Caderno de Tarefas não pode ser levado para casa após a prova. Após a prova entregue este Caderno de Tarefas para seu professor guardar. Os professores poderão devolver os Cadernos de Tarefas aos competidores após o término do período de aplicação das provas (23 de agosto de 2022).

Olimpíada Brasileira de Informática

OBI2022

Caderno de Tarefas

Modalidade Programação • Nível 1 • Fase 2

23 de agosto de 2022

A PROVA TEM DURAÇÃO DE 2 HORAS

Promoção:

Sociedade Brasileira de Computação

Apoio:

FUNDAÇÃO BEHRING

Coordenação:

INSTITUTO DE COMPUTAÇÃO UNICAMP

Instruções

LEIA ATENTAMENTE ESTAS INSTRUÇÕES ANTES DE INICIAR A PROVA

- Este caderno de tarefas é composto por 7 páginas (não contando a folha de rosto), numeradas de 1 a 7. Verifique se o caderno está completo.
- A prova deve ser feita individualmente.
- É proibido consultar a Internet, livros, anotações ou qualquer outro material durante a prova. É permitida a consulta ao *help* do ambiente de programação se este estiver disponível.
- As tarefas têm o mesmo valor na correção.
- A correção é automatizada, portanto siga atentamente as exigências da tarefa quanto ao formato da entrada e saída de seu programa; em particular, seu programa não deve escrever frases como "Digite o dado de entrada:" ou similares.
- Não implemente nenhum recurso gráfico nas suas soluções (janelas, menus, etc.), nem utilize qualquer rotina para limpar a tela ou posicionar o cursor.
- As tarefas **não** estão necessariamente ordenadas, neste caderno, por ordem de dificuldade; procure resolver primeiro as questões mais fáceis.
- Preste muita atenção no nome dos arquivos fonte indicados nas tarefas. Soluções na linguagem C devem ser arquivos com sufixo *.c*; soluções na linguagem C++ devem ser arquivos com sufixo *.cc* ou *.cpp*; soluções na linguagem Pascal devem ser arquivos com sufixo *.pas*; soluções na linguagem Java devem ser arquivos com sufixo *.java* e a classe principal deve ter o mesmo nome do arquivo fonte; soluções na linguagem Python 3 devem ser arquivos com sufixo *.py*; e soluções na linguagem Javascript devem ter arquivos com sufixo *.js*.
- Na linguagem Java, **não** use o comando *package*, e note que o nome de sua classe principal deve usar somente letras minúsculas (o mesmo nome do arquivo indicado nas tarefas).
- Para tarefas diferentes você pode escolher trabalhar com linguagens diferentes, mas apenas uma solução, em uma única linguagem, deve ser submetida para cada tarefa.
- Ao final da prova, para cada solução que você queira submeter para correção, copie o arquivo fonte para o seu diretório de trabalho ou pen-drive, conforme especificado pelo seu professor.
- Não utilize arquivos para entrada ou saída. Todos os dados devem ser lidos da entrada padrão (normalmente é o teclado) e escritos na saída padrão (normalmente é a tela). Utilize as funções padrão para entrada e saída de dados:
  - em Pascal: *readln, read, writeln, write*;
  - em C: *scanf, getchar, printf, putchar*;
  - em C++: as mesmas de C ou os objetos *cout* e *cin*.
  - em Java: qualquer classe ou função padrão, como por exemplo *Scanner, BufferedReader, BufferedWriter* e *System.out.println*
  - em Python: *read,readline,readlines,input,print,write*
  - em Javascript: *scanf, printf*
- Procure resolver a tarefa de maneira eficiente. Na correção, eficiência também será levada em conta. As soluções serão testadas com outras entradas além das apresentadas como exemplo nas tarefas.

Tanque de combustível

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

Cássio alugou um carro para a viagem de férias. O carro tem consumo de combustível constante (em kilometros rodados por litro de combustível), independente da velocidade com que trafega. Ao fim da viagem, Cássio deve devolver o carro no aeroporto.

Cássio está terminando sua viagem de férias e está no momento na rodovia que leva ao aeroporto, em direção ao aeroporto para devolver o carro. Mais precisamente Cássio está no último posto de combustível existente na rodovia em que ele pode abastecer o carro antes de devolvê-lo.

Para economizar o máximo possível em combustível, Cássio quer devolver o carro com o menor número de litros possível no tanque – idealmente, com o tanque *zerado*, ou seja, sem combustível.

Dados o consumo do carro, a distância em que se encontra do aeroporto e a quantidade de combustível presente no tanque antes do abastecimento, determine qual deve ser a menor quantidade de combustível que Cássio deve comprar.

Entrada

A primeira linha contém um inteiro, C, o consumo do carro em kilômetros rodados por litro de combustível. A segunda linha contém um inteiro D, a distância do aeroporto, em kilometros. A terceira linha contém um inteiro T, o número de litros de combustível presente no tanque antes do abastecimento. Você pode assumir que o tanque tem capacidade suficiente para armazenar todo o combustível que Cássio comprar.

Saída

Seu programa deve produzir uma única linha, contendo um único valor, com um dígito de precisão, indicando a quantidade de combustível que Cássio deve comprar, para chegar ao aeroporto com o tanque contendo a menor quantidade de combustível possível.

Restrições

- $1 \leq C \leq 50$
- $1 \leq D \leq 1000$
- $0 \leq T \leq 100$

Exemplos

Exemplo de entrada 1	**Exemplo de saída 1**
2 10 0	5.0

Explicação do exemplo 1: O consumo é 2 km/l, Cássio está a 10 km do aeroporto e o tanque não tem combustível. Para chegar ao aeroporto o carro vai gastar 5.0 litros de combustível. Como o tanque não tem combustível, Cássio precisa comprar 5.0 litros de combustível.

Exemplo de entrada 2	**Exemplo de saída 2**
30 100 2	1.3

Explicação do exemplo 2: O consumo é 30 km/l, Cássio está a 100 km do aeroporto e o tanque tem 2 litros combustível. Para chegar ao aeroporto o carro vai gastar 3.33 litros de combustível. Como o tanque já tem 2 litros de combustível, Cássio precisa comprar 1.3 litros de combustível (note o arredondamento).

Exemplo de entrada 3	**Exemplo de saída 3**
50 120 3	0.0

Explicação do exemplo 3: O consumo é 50 km/l, Cássio está a 120 km do aeroporto e o tanque tem 3 litros combustível. Para chegar ao aeroporto o carro vai gastar 2.4 litros de combustível. Como o tanque já tem 3 litros de combustível, Cássio não precisa comprar combustível.

Exemplo de entrada 4	**Exemplo de saída 4**
50 73 0	1.5

Explicação do exemplo 4: O consumo é 50 km/l, Cássio está a 73 km do aeroporto e o tanque não tem combustível. Para chegar ao aeroporto o carro vai gastar 1.46 litros de combustível. Como o tanque não tem combustível, Cássio precisa comprar 1.5 litros de combustível (note o arredondamento).

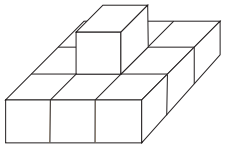
Pirâmide

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

O rei da Nlogônia decidiu construir uma pirâmide no jardim do Palácio Real, usando cubos de pedra de mesmo tamanho. A *dimensão* de uma pirâmide é o número de cubos de pedra num dos lados da base (primeira camada) da pirâmide. A base da pirâmide é quadrada, ou seja, cada lado tem o mesmo número de cubos de pedra.

Na pirâmide, a partir da segunda camada, cada cubo de pedra deve ser empilhado exatamente em cima de outro cubo de pedra que não esteja na borda da camada abaixo. Além disso, o número de camadas deve ser o maior possível para uma dada dimensão, e em cada camada deve ser usado o maior número de cubos de pedra possível.

A figura abaixo à esquerda mostra uma pirâmide de dimensão 3; a figura à direita mostra o *plano de contrução* para essa pirâmide, indicando quantos cubos de pedra devem ser empilhados em cada posição.



1	1	1
1	2	1
1	1	1

O rei ainda não decidiu qual a dimensão da pirâmide que vai construir, mas como é muito detalhista já avisou os Arquitetos Reais que antes de iniciar a contrução eles devem produzir um plano de contrução para a dimensão escolhida.

Ajude os Arquitetos Reais, escrevendo um programa que, dada a dimensão da pirâmide, produza o seu plano de construção.

Entrada

A primeira e única linha da entrada contém um número inteiro N, a dimensão da pirâmide.

Saída

Seu programa deve produzir o plano de contrução da pirâmide, constituído por N linhas, cada linha contendo N números inteiros.

Restrições

- $1 \leq N \leq 100$

Informações sobre a pontuação

- Para um conjunto de casos de testes valendo 10 pontos, $1 \leq N \leq 3$.

Exemplos

Exemplo de entrada 1	**Exemplo de saída 1**

`3`	`1 1 1`
	`1 2 1`
	`1 1 1`

Explicação do exemplo 1: Para uma pirâmide de dimensão 3, o maior número de camadas possível é 2.

Exemplo de entrada 2	**Exemplo de saída 2**
8	1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 1 1 2 3 3 3 3 2 1 1 2 3 4 4 3 2 1 1 2 3 4 4 3 2 1 1 2 3 3 3 3 2 1 1 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1

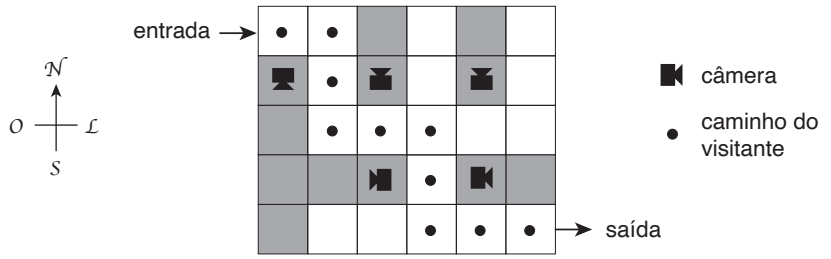
Explicação do exemplo 2: Para uma pirâmide de dimensão 8, o maior número de camadas possível é 4.

Câmeras

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

Uma exposição vai ser montada num espaço retangular, dividido em $N \times M$ células dispostas em N colunas por M linhas. Uma *célula* é o espaço delimitado pela interseção de uma coluna com uma linha. As colunas estão na direção Norte-Sul e as linhas na direção Oeste-Leste. Para segurança das obras foram instaladas K câmeras, em células selecionadas. Cada câmera pode estar apontada para uma de quatro direções: Norte, Sul, Leste ou Oeste. Uma câmera observa todas as células da coluna ou linha na direção em que está apontada, a partir da célula em que está instalada (incluindo a célula em que está instalada).

A porta de entrada da exposição está na célula mais ao norte e mais à oeste, a porta de saída está na célula mais ao sul e mais ao leste. A figura abaixo ilustra um espaço de exposição com 6 colunas, 5 linhas e 5 câmeras instaladas.



Preocupado com a segurança, o organizador da exposição deseja saber se é possível que um visitante entre pela porta de entrada e saia pela porta de saída, movendo-se somente nas quatro direções (Norte, Sul, Leste ou Oeste) sem que seja observado por qualquer das câmeras instaladas.

Entrada

A primeira linha contém três inteiros N, M e K indicando respectivamente o número de colunas, o número de linhas e o número de câmeras instaladas. As colunas estão numeradas de 1 a N e as linhas estão numeradas de 1 a M. A coluna 1 é a coluna mais à Oeste e a linha 1 é a linha mais ao Norte. Cada uma das K linhas seguintes descreve uma câmera e contém dois inteiros C_i, L_i e um caractere D_i, indicando respectivamente a coluna, a linha e a direção em que a câmera está instalada. O caractere D_i pode ser `N`, `S`, `L` ou `O`, indicando respectivamente que a câmera está instalada direcionada para o Norte, Sul, Leste ou Oeste.

Saída

Seu programa deve produzir uma única linha, contendo um único caractere, que deve ser `S` se é possível que um visitante entre pela porta de entrada e saia pela porta de saída sem que seja observado por qualquer das câmeras instaladas, ou `N` caso contrário.

Restrições

- $2 \leq N \leq 30$; $2 \leq M \leq 30$; $1 \leq K \leq 30$
- $1 \leq C_i \leq N$, para $1 \leq i \leq K$
- $1 \leq L_i \leq M$, para $1 \leq i \leq K$
- D_i pode ser `N`, `S`, `L` ou `O`.

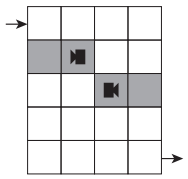
Informações sobre a pontuação

- Para um conjunto de casos de testes valendo 10 pontos, $M = 2$ e $K = 1$.
- Para um conjunto de casos de testes valendo outros 10 pontos, $N = 3$, $M = 3$ e $K = 2$.
- Para um conjunto de casos de testes valendo outros 80 pontos, nenhuma restrição adicional.

Exemplos

Exemplo de entrada 1	**Exemplo de saída 1**
4 5 2 2 2 O 3 3 L	N

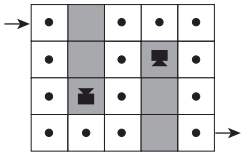
Explicação do exemplo 1:



Neste caso a resposta é Não.

Exemplo de entrada 2	**Exemplo de saída 2**
5 4 2 2 3 N 4 2 S	S

Explicação do exemplo 2:



Neste caso a resposta é Sim.

Exemplo de entrada 3	**Exemplo de saída 3**
6 5 5 1 2 S 3 2 N 5 2 N 3 4 O 5 4 L	S

Explicação do exemplo 3: Este caso é o exemplo dado no enunciado.