Maratona de Programação
1998
Caderno de Problemas
Categoria - Júnior

Problema 1: Pirâmide

```
Arquivo:  piramide.c ou piramide.pas
Entrada:  piramide.in
Saída:  piramide.out
```

Os segredos das pirâmides vêm instigando cientistas e místicos há séculos, dando asas à imaginação das pessoas. Uma dessas curiosidades são os chamados *números piramidais*. Dizemos que um número inteiro é piramidal se for igual à soma de três números primos consecutivos. Assim, por exemplo, o número 15 é piramidal, pois $15 = 3 + 5 + 7$. O número 10 também é piramidal, já que $10 = 2 + 3 + 5$. O número 23 é piramidal, pois $23 = 5 + 7 + 11$.
Neste programa você receberá uma seqüência de números inteiros maiores que 0, terminada por um zero, e deverá verificar se cada um dos números da seqüência é ou não piramidal.

Entrada

A entrada é composta por uma seqüência de números inteiros positivos, cada um maior que 0 e menor que 10000, cada um deles em uma linha. Na última linha haverá um zero.

Saída

Para cada linha da entrada você deverá imprimir uma linha na saída, colocando a mensagem `O numero ... e piramidal` ou `O numero ... nao e piramidal`. A linha com o número 0 não deve causar nenhuma impressão.

Exemplo

Entrada

```
15
7
6
97
0
```

Saída

```
O numero 15 e piramidal
O numero 7 nao e piramidal
O numero 6 nao e piramidal
O numero 97 e piramidal
```

Problema 2: Presentes

`Arquivo: presente.c ou presente.pas`
`Entrada: presente.in`
`Saída: presente.out`

Este problema envolve determinar, de um grupo de amigos que estão se presenteando, quanto a mais cada pessoa dá do que recebe (ou vice-versa para aqueles que vêem presentear com cinismo).
Neste problema cada pessoa coloca de lado algum dinheiro para presentear alguns de seus amigos no grupo. Este dinheiro é dividido de forma igual entre todos os presentes que a pessoa dará.
Entretanto, em qualquer grupo de amigos, algumas pessoas são mais generosas que outras e algumas pessoas têm mais dinheiro que outras.
Dado um grupo de amigos, o dinheiro que cada um do grupo gastou com presentes, e uma (sub)lista de amigos que cada um presenteará, você deve escrever um programa que determina quanto a mais (ou a menos) cada pessoa do grupo deu do que recebeu. Note que o valor de todos os presentes dados são inteiros. Cada pessoa dá a mesma quantidade inteira de dinheiro a cada amigo que presenteia, e dá o máximo possível.

A entrada

A entrada é uma seqüência de grupos de pessoas. Cada grupo consiste de muitas linhas:

- o número de pessoas no grupo,
- a lista de nomes das pessoas do grupo,
- uma linha para cada pessoa do grupo, com o nome desta pessoa, a quantidade de dinheiro que ela tem para presentear, o número de pessoas que ela presenteou e o nome dos amigos que ela presenteou.

Todo nome é formado por uma única palavra, só usa letras minúsculas e não possui mais que 12 letras. Dinheiro é um inteiro positivo menor que 2000. Todo grupo tem pelo menos uma pessoa. A entrada consiste de um ou mais grupos e é terminada por um 0.

A saída

Para cada grupo de amigos, deve ser impressa uma linha para cada pessoa do grupo. Cada linha deve conter o nome desta pessoa, seguido do ganho (ou perda) líquida da pessoa, ou seja, a diferença entre o que ela recebeu e o que deu. Os nomes em um grupo devem ser impressos na mesma ordem em que apareceram pela primeira vez na entrada.
A saída para cada grupo deverá ser separada do próximo grupo por uma linha em branco.

Exemplo

Entrada

```
5
dave laura owen vick amr
dave 200 3 laura owen vick
owen 500 1 dave
amr 150 2 vick owen
laura 0 2 amr vick
vick 0 0
3
liz steve dave
liz 30 1 steve
steve 55 2 liz dave
dave 0 2 steve liz
0
```

Saída

```
dave 302
laura 66
owen -359
vick 141
amr -150

liz -3
steve -24
dave 27
```

Problema 3: Robô

`Arquivo: robo.c ou robo.pas`
`Entrada: robo.in`
`Saída: robo.out`

A idéia deste problema é seguir os passos de vários robôs que se movem em um mundo formado por uma grade retangular. Uma *posição do robô* consiste de uma coordenada da grade (um par de inteiros: coordenada x seguida por coordenada y) e uma direção (N, S, E, W para norte, sul, leste e oeste). Uma *instrução* é uma string das letras "L", "R" e "F" que representam respectivamente as instruções:

- L = *Esquerda:* o robô vira 90 graus para a esquerda e fica parado;
- R = *Direita:* o robô vira 90 graus para a direita e fica parado;
- F = *Para frente:* o robô se move uma posição na direção atual e mantém a mesma direção.

A direção *norte* corresponde à direção que vai do ponto (x, y) para o ponto $(x, y + 1)$ da grade. A direção *leste* corresponde à direção que vai do ponto (x, y) para o ponto $(x + 1, y)$ da grade.
Como a grade é retangular e limitada, um robô que se move para fora da fronteira é perdido para sempre. Entretanto, robôs perdidos deixam um perfume que proíbe outros robôs de caírem fora do mundo pelo mesmo ponto da grade. O perfume é deixado na última posição da grade que o robô ocupou antes de desaparecer. Uma instrução de mover para fora do mundo de um ponto da grade de onde algum robô anteriormente foi perdido é simplesmente ignorada pelo robô atual, independente da orientação que tinha o robô que se perdeu anteriormente.
Dadas as dimensões de uma grade retangular, uma seqüência de posições de robôs e instruções, você deve escrever um programa que determina a posição final de cada robô.

Entrada

A primeira linha da entrada são as coordenadas superior direita do mundo retangular, a coordenada inferior esquerda é (0,0).
O resto da entrada consiste de uma seqüência de posições de robôs e instruções (duas linhas por robô). Uma posição consiste de dois inteiros especificando as coordenadas iniciais de um robô e uma direção (N,S,E,W), todos separados por espaços em uma linha. Uma instrução para o robô é uma string de letras "L", "F" e "R" em uma linha.
Cada robô é processado seqüencialmente, ou seja, a execução das instruções de um robô termina antes de iniciar as instruções do próximo robô.
A entrada termina com um "end-of-file".

Você pode assumir que todas as posições iniciais dos robôs estão dentro dos limites da grade especificada. O valor máximo para qualquer coordenada é 50.

Saída

Para cada posição/instrução de robô da entrada, a saída deverá indicar a posição final da grade e direção do robô. Se um robô se perde, a palavra `LOST` deverá ser impressa depois da última posição e direção.

Exemplos

Entrada

```
5 3
1 1 E
RFRFRFRF
3 2 N
FRRFLLFFRRFLL
0 3 W
LLFFFLFLFL
```

Saída

```
1 1 E
3 3 N LOST
2 3 S
```

Entrada

```
0 0
0 0 E
RF
0 0 N
FFFFFF
```

Saída

```
0 0 S LOST
0 0 N
```

Problema 4: Panquecas

```
Arquivo: pank.c ou pank.pas
Entrada: pank.in
Saída: pank.out
```

Dada uma pilha de panquecas, você deve escrever um programa que indica como a pilha pode ser ordenada de forma que a maior panqueca está no fundo e a menor panqueca está no topo. O tamanho de uma panqueca é dado pelo seu diâmetro. Todas as panquecas de uma pilha têm diâmetros diferentes.
A ordenação da pilha é feita através de uma seqüência de "giros" (*flips*) de panquecas. Um giro consiste em inserir a espátula entre duas panquecas em uma pilha e girar (reverter) todas as panquecas que estão em cima da espátula (revertendo a subpilha). Um giro é especificado dando a posição da panqueca da subpilha que será revertida (em relação à pilha completa). A panqueca do fundo da pilha inteira tem posição 1 e a panqueca do topo de uma pilha com n panquecas tem posição n.
Uma pilha é especificada dando o diâmetro de cada panqueca da pilha na ordem em que as panquecas aparecem.
Por exemplo, considere as três pilhas de panquecas abaixo (nas quais a panqueca 8 está no topo da pilha mais à esquerda):

8	7	2
4	6	5
6	4	8
7	8	4
5	5	6
2	2	7

A pilha mais à esquerda pode ser transformada na pilha do meio fazendo a operação *flip(3)*. A pilha do meio pode ser transformada na pilha à direita através do comando *flip(1)*.

A entrada

A entrada consiste de uma seqüência de pilhas de panquecas. Cada pilha consistirá de 1 a 30 panquecas e cada panqueca terá um diâmetro inteiro entre 1 e 100. A entrada é terminada por "end-of-file". Cada pilha é dada através de duas linhas. Na primeira linha se encontra o número de panquecas da pilha e, na segunda linha, as panquecas, com a panqueca do topo da pilha aparecendo primeiro nesta linha, a panqueca do fundo aparecendo por último, e todas as panquecas separadas por um espaço.

A saída

Para cada pilha de panquecas, a saída deverá ecoar a pilha original em uma linha, seguida por uma seqüência de "flips" que resultam numa pilha de panquecas ordenadas de tal forma que a panqueca de maior diâmetro está no fundo e a de menor diâmetro no topo. Para cada pilha, a seqüência de “flips” deverá ser terminada por um zero (indicando que não são mais necessários “flips”). Uma vez que a pilha esteja ordenada, nenhum “flip” deve ser feito.

Exemplo

Entrada

```
3
4 6 8
5
5 4 3 2 1
5
5 1 2 3 4
```

Saída

```
4 6 8
0
5 4 3 2 1
1 0
5 1 2 3 4
1 2 0
```

Problema 5: Míssil

`Arquivo: missil.c ou missil.pas`
`Entrada: missil.in`
`Saída: missil.out`

O Departamento de Defesa americano está terminando os testes preliminares de um novo míssil defensivo chamado "the catcher" que é capaz de interceptar diversos mísseis vindos ao mesmo tempo. Os militares reputam que o "catcher" é um excelente míssil defensivo. Ele pode se mover para frente, para o lado e para trás muito rapidamente, e pode interceptar um míssil inimigo sem ser destruído. Mas o míssil tem um grande problema. Embora ele possa ser lançado para atingir qualquer elevação inicial, ele não tem energia para ir mais alto do que o último míssil que ele interceptou.
Os testes feitos pelo Depto. de Defesa foram simulações por computador de condições de batalha e ataque hostil. Como eram testes preliminares, as simulações testavam somente a capacidade do catcher se mover verticalmente. Em cada simulação atiravam contra o catcher uma seqüência de mísseis hostis que vinham em intervalos fixos de tempo. A única informação que o catcher tinha era a seqüência de mísseis e a altura de cada um deles.
O resultado de cada teste é o número total desses mísseis que foram interceptados pelo catcher naquele teste.
O escritório geral quer ter certeza de que os testes simulados foram suficientes, dadas as restrições do catcher. Você deve escrever um programa que, dados diversos testes com seqüências de mísseis, imprime o número máximo de mísseis que o catcher pode interceptar. Para cada míssil hostil, o catcher pode interceptá-lo se e somente se uma destas condições estiver satisfeita:

- o míssil hostil é o primeiro a ser interceptado neste teste; **ou**
- o míssil foi disparado depois que o último míssil foi interceptado e ele não está mais alto do que o último míssil que foi interceptado.

Entrada e Saída

A entrada de qualquer teste é uma seqüência de um ou mais inteiros não negativos, todos eles menores ou iguais a 32767, representando cada um a altura de um míssil hostil. O último número em cada seqüência é um -1, que significa o fim dos dados daquele teste em particular e não é altura de um míssil. O fim dos dados é dado pelo número -1 no início de uma seqüência, e não deve ser considerado como um teste em separado.
A saída para cada teste consiste do número do teste (`Teste #1`, `Teste #2`, etc) e o número máximo de mísseis hostis que o catcher poderia possivelmente interceptar neste teste. O número máximo aparece depois de uma mensagem. Deve existir pelo menos uma linha em branco na saída entre dois testes sucessivos.

Nota: Esperaremos 2 minutos até interromper a execução de um programa. Seu programa deverá executar cada teste neste tempo. O número de mísseis de um dado teste pode ser grande (até 50).

Exemplo

Entrada

```
389
207
155
300
299
170
350
158
65
-1
20
23
34
21
-1
30
21
30
25
30
10
-1
-1
```

Saída

```
Teste #1:
  numero maximo de interceptacoes: 6

Teste #2:
  numero maximo de interceptacoes: 2

Teste #3:
  numero maximo de interceptacoes: 4
```

Problema 6: Fatorial

`Arquivo: fatorial.c ou fatorial.pas`
`Entrada: fatorial.in`
`Saída: fatorial.out`

A expressão $N!$, que se lê "N fatorial", denota o produto dos primeiros N números inteiros maiores que 0, (suponha $N > 0$). Assim, por exemplo, $1! = 1$, $2! = 2$, $3! = 6$, $4! = 24$, $5! = 120$, e assim por diante.
Você deve escrever um programa que calcula o último dígito diferente de zero de qualquer fatorial, para $0 < N < 10000$. Por exemplo, se o seu programa receber como entrada o número 5, deve devolver o último dígito não nulo de $5!$, que é 2.

Entrada

A entrada é uma seqüência de números inteiros maiores que 0 e menores que 10000, cada um em uma linha. Na última linha da entrada deverá estar um zero.

Saída

Para cada número da entrada, seu programa deve imprimir exatamente uma linha com o valor da entrada ajustado à direita (ocupando portanto as colunas de 1 a 5). As colunas 6 a 8 devem conter "`-> `". A coluna 9 deve conter o último dígito não nulo de $N!$.

Exemplo

Entrada

```
1
2
26
125
3125
9999
0
```

Saída

```
    1-> 1
    2-> 2
   26-> 4
  125-> 8
 3125-> 2
 9999-> 8
```