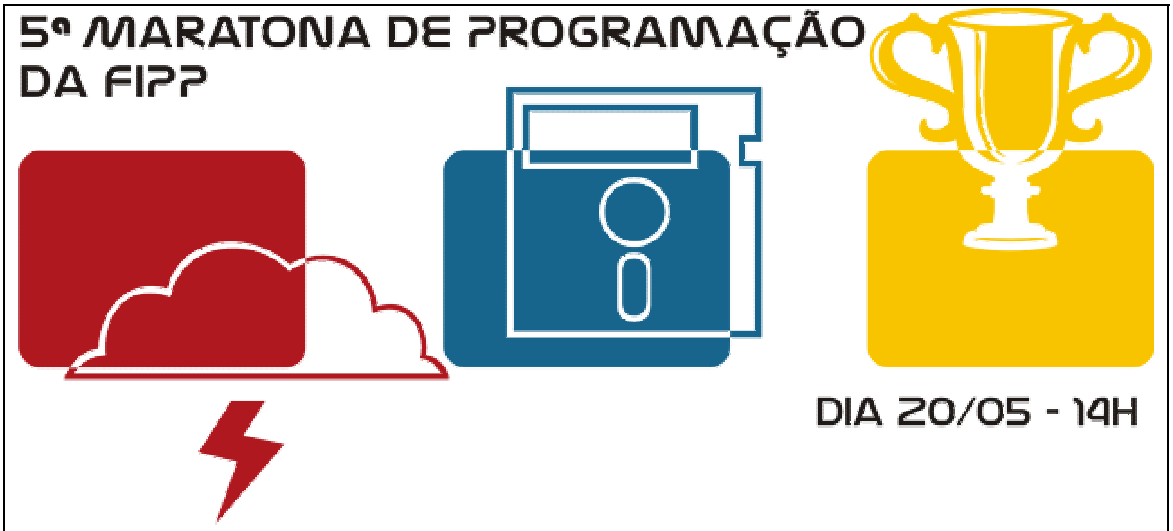

CADERNO DE PROBLEMAS

20/05/2008 – 14:00 às 18:00

Leia atentamente estas instruções antes de iniciar a resolução dos problemas. Este caderno é composto de 5 problemas, 2 deles estão descritos em inglês.

1. É proibido utilizar livros, apostilas impressas, manuais ou qualquer outro material que não seja fornecido pela comissão organizadora durante a prova, com exceção de dicionários de inglês. É permitida apenas a consulta do *help* do ambiente de programação se este estiver disponível.
2. A correção das resoluções dos problemas será automatizada por meio do sistema de submissão eletrônica BOCA, baseada nos resultados obtidos com uma série de execuções dos algoritmos de cada equipe.
3. Siga atentamente as exigências da tarefa quanto ao formato da entrada e saída do seu algoritmo. Não implemente nenhum recurso gráfico nas suas soluções (janelas, menus, etc), nem utilize qualquer rotina para limpar a tela ou posicionar o cursor.
4. Os problemas não estão ordenados, neste caderno, por ordem de dificuldade. Procure resolver primeiro os problemas mais fáceis.
5. Utilize para nomear os seus arquivos-fonte os nomes indicados nos problemas de acordo com a linguagem de programação utilizada.
6. Os problemas devem ser resolvidos utilizando o raciocínio entrada-processamento-saída, ou seja, não é necessário armazenar toda a saída para exibí-la.

Observações:

Não utilize arquivos para entrada ou saída. Todos os dados devem ser lidos da entrada padrão (teclado) e escritos na saída padrão (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:
```
Scanner sc = new Scanner(); int i = sc.nextInt();
String s = sc.next(); float f = sc.nextFloat();
System.out.println(); System.out.printf("%d",i);
```

Quando a linguagem escolhida para a resolução dos problemas for o Pascal, recomendamos que não seja utilizada a instrução: `uses newdelay, crt;`

Gerente de Espaço (1 – vermelho)

Arquivo fonte: *espaco.c*, *espaco.cpp*, *espaco.pas* ou *espaco.java*

É bem verdade que a maioria das pessoas não se importa muito com o que ocorre dentro de um computador, desde que ele execute as tarefas que devem ser desempenhadas. Existem, no entanto, alguns poucos *nerds* que sentem prazer em acompanhar o movimento de *bits* e *bytes* dentro da memória do computador.

É para esse público, constituído principalmente de adolescentes, que a multinacional de software ACM (*Abstractions of Concrete Machines*) deseja desenvolver um sistema que acompanhe e produza um relatório das operações efetuadas em um disco rígido. Um disco rígido é composto de uma sequência de células atômicas de armazenamento, cada uma de tamanho 1Kb.

Especificamente, a ACM deseja acompanhar três tipos de operações:

- `insere NOME T`
  insere no disco o arquivo `NOME`, de tamanho T. Você pode supor que um arquivo com esse nome não existe ainda no disco. O tamanho T de um arquivo é dado na forma *X*Kb, *X*Mb, ou *X*Gb, onde *X* é um inteiro ($0 < X \leq 1023$). `NOME` é uma cadeia de caracteres com comprimento máximo 10.
- `remove NOME`
  remove o arquivo `NOME` do disco. Se um arquivo com esse nome não existe, não faz nada;
- `otimiza`
  compacta o disco, deslocando os arquivos existentes na direção do início do disco, eliminando espaços livres entre dois arquivos subsequentes, e preservando a ordem em que os arquivos aparecem no disco, de modo a deixar um espaço de memória livre no final do disco.

A capacidade de um disco é sempre um número múltiplo de 8Kb. No início, o disco está vazio, ou seja, contém um bloco livre do tamanho da capacidade do disco. Um arquivo é sempre armazenado em um bloco de células de armazenamento contíguas. O arquivo a ser inserido deve ser sempre colocado no início do menor bloco livre cujo tamanho é maior ou igual ao tamanho do arquivo. Se mais de um bloco livre é igualmente adequado, escolha o mais próximo do começo do disco. Caso não seja possível inserir o arquivo por falta de um bloco livre suficientemente grande, deve-se executar automaticamente o comando otimiza. Se após a otimização ainda não for possível inserir o arquivo, uma mensagem de erro deve ser produzida.

No caso de todas as operações serem executadas sem erro, seu programa deve produzir uma estimativa aproximada do estado final do disco, conforme descrito abaixo.

Lembre que 1Mb corresponde a 1024Kb, enquanto 1Gb corresponde a 1024Mb.

Entrada

A entrada é constituída de vários casos de teste. A primeira linha de um caso de teste contém um único inteiro *N* indicando o número de operações no disco ($0 < N \leq 10000$). A segunda linha de um caso de teste contém a descrição do tamanho do disco, composta por um inteiro *D* ($0 < D \leq 1023$), seguido de um especificador de unidade; o

especificador de unidade é uma cadeia de dois caracteres no formato Kb, Mb ou Gb. Cada uma das *N* linhas seguintes contém a descrição de uma operação no disco (insere, remove ou otimiza, conforme descrito acima). O final da entrada é indicado por $N = 0$.

A entrada deve ser lida da entrada padrão.

Exemplo de entrada

```
3
8Kb
insere arq0001 7Kb
insere arq0002 3Mb
remove arq0001
6
8Mb
insere arq0001 4Mb
insere arq0002 1Mb
insere arq0003 512Kb
remove arq0001
remove arq0001
insere arq0001 5Mb
0
```

Saída

Para cada caso de teste seu programa deve produzir uma linha na saída. Se todas as operações de inserção forem executadas sem erro, seu programa deve produzir uma linha contendo uma estimativa aproximada do estado do disco, apresentada como se segue. Divida o número de *bytes* do disco em oito blocos contíguos de mesmo tamanho. Para cada um dos oito blocos seu programa deve verificar a porcentagem *P* de *bytes* livres daquele bloco, e apresentar a estimativa do estado final no formato

[C][C][C][C][C][C][C][C]

onde C é ‘ ’, ‘-’ ou ‘#’, dependendo se $75 < P \leq 100$, $25 < P \leq 75$ ou $0 \leq P \leq 25$, respectivamente. Caso um arquivo não possa ser inserido por falta de espaço, seu programa deve produzir uma linha contendo a expressão `ERRO`: disco cheio; nesse caso, operações subsequentes do caso de teste devem ser ignoradas.

A saída deve ser escrita na saída padrão.

Exemplo de Saída

ERRO: disco cheio
[#][#][#][#][#][#][-][ ]

Head or Tail (2 – branco)

Source file: *headtail.c*, *headtail.cpp*, *headtail.pas* ou *headtail.java*

John and Mary have been friends since nursery school. Since then, they have shared a playful routine: every time they meet, they play Head or Tail with a coin, and whoever wins has the priviledge of deciding what they are going to play during the day. Mary always choose Head, and John always choose Tail.

Nowadays they are in college, but continue being truly good friends. Whenever they meet, they still play Head and Tail, and the winner decides which film to watch, or which restaurant to have dinner together, and so on.

Yesterday Mary confided to John that she has being keeping a record of the results of every play since they started, in nursery school. It came as a surprise to John! But since John is studying Computer Science, he decided it was a good opportunity to show Mary his skills in programming, by writing a program to determine the number of times each of them won the game over the years.

Input

The input contains several test cases. The first line of a test case contains a single integer N indicating the number of games played ($1 \le N \le 10000$). The following line contains N integers R_i, separated by space, describing the list of results. If $R_i = 0$ it means Mary won the ith game, if $R_i = 1$ it means John won the ith game ($1 \le i \le N$). The end of input is indicated by $N = 0$.

The input must be read from standard input.

Output

For each test case in the input your program should output a line containing the sentence "`Mary won X times and John won Y times`", where $X \ge 0$ and $Y \ge 0$.

The output must be written to standard output.

Sample Input	Output for the Sample Input
5 0 0 1 0 1 6 0 0 0 0 0 1 0	Mary won 3 times and John won 2 times Mary won 5 times and John won 1 times

Fake Tickets (3 – laranja)

Source file: *fake.c*, *fake.cpp*, *fake.pas* ou *fake.java*

Your school organized a big party to celebrate your team brilliant win in the prestigious, world famous ICPC (International Collegiate Poetry Contest). Everyone in your school was invited for an evening which included cocktail, dinner and a session where your team work was read to the audience. The evening was a success – many more people than you expected showed interested in your poetry – although some critics of yours said it was food rather than words that attracted such an audience.

Whatever the reason, the next day you found out why the school hall had seemed so full: the school director confided he had discovered that several of the tickets used by the guests were fake. The real tickets were numbered sequentially from 1 to N ($N \leq 10000$). The director suspects some people had used the school scanner and printer from the Computer Room to produce copies of the real tickets. The director gave you a pack with all tickets collected from the guests at the party's entrance, and asked you to determine how many tickets in the pack had 'clones', that is, another ticket with the same sequence number.

Input

The input contains data for several test cases. Each test case has two lines. The first line contains two integers N and M which indicate respectively the number of original tickets and the number of persons attending the party ($1 \leq N \leq 10000$ and $1 \leq M \leq 20000$). The second line of a test case contains i integers Ti representing the ticket numbers in the pack the director gave you ($1 \leq Ti \leq N$). The end of input is indicated by $N = M = 0$.

Output

For each test case your program should print one line, containing the number of tickets in the pack that had another ticket with the same sequence number.

Sample input	Output for the sample input
5 5 3 3 1 2 4 6 10 6 1 3 6 6 4 2 3 1 2 0 0	1 4

Histórico de Comandos (4 – amarelo)

Arquivo fonte: *hist.c*, *hist.cpp*, *hist.pas* ou *hist.java*

Uma *interface por linha de comando* (ILC) é um dos tipos de interface humano-computador mais antigos que existem. Uma ILC permite a interação com o software através de um *interpretador de comandos*, sendo normalmente acessível em um terminal (ou janela) de texto. A vantagem de um interpretador de comandos é que ele permite que o usuário opere o sistema usando apenas o teclado. Ainda hoje em dia, em que estamos acostumados com interfaces gráficas sofisticadas, muitos aplicativos e sistemas operacionais incluem algum tipo de interface por linha de comando, e muitos usuários ainda preferem usá-la para grande parte das tarefas.

Um dos recursos mais úteis de um interpretador de comandos é o *histórico* de comandos. Quando um comando é digitado e executado, ele é colocado no histórico de comandos do terminal. O comando pode ser exibido novamente no terminal apertando a tecla '↑'; a tecla *Enter* executa o comando novamente quando o comando está sendo exibido no terminal. Todos os comandos executados são guardados no histórico: pressionar a tecla '↑' duas vezes exibe o penúltimo comando executado, pressioná-la três vezes exibe o antepenúltimo comando, e assim sucessivamente.

Por exemplo, se o histórico inicial é (*A,B,C,D*), para repetir o comando *C* basta pressionar duas vezes a tecla '↑'. O histórico será então atualizado para (*A,B,C,D,C*). Nesse ponto, para repetir o comando *A* será necessário pressionar cinco vezes a tecla '↑'; o histórico será atualizado para (*A,B,C,D,C,A*). Nesse ponto, para repetir mais uma vez o comando *A* basta pressionar uma vez a tecla '↑'; o histórico será atualizado para (*A,B,C,D,C,A,A*).

Leandro é administrador de sistemas e usa frequentemente o interpretador de comandos para gerenciar remotamente os servidores que administra. Em geral, ele precisa apenas repetir comandos que já havia digitado antes. Enquanto estava trabalhando em um servidor, ele teve uma curiosidade: quantas vezes ele precisa pressionar a tecla '↑' para executar uma determinada sequência de comandos? Ele sabe quais são as posições no histórico dos comandos que ele necessita executar, mas não sabe resolver esse problema. Por isso, pediu que você fizesse um programa que respondesse à pergunta dele.

Entrada

A entrada é composta de vários casos de teste. A primeira linha de cada caso de teste contém um número inteiro N, indicando o número de comandos que Leandro deseja executar ($1 \leq N \leq 1.000$). A segunda linha de um caso de teste contém N inteiros $P_1, P_2, \ldots, P_N$, que indicam as posições dos comandos no histórico ($1 \leq P_i \leq 1.000.000$) no momento inicial, na ordem em que os comandos devem ser executados. Ou seja, o primeiro comando que deve ser executado está inicialmente na posição P_1 do histórico; depois deve ser executado o comando que está inicialmente na posição P_2 no histórico, e assim por diante, até P_N, que é a posição inicial do último comando que deve ser executado. Note que pode haver $P_i = P_j$.

As posições são dadas em função do número de vezes que a tecla '↑' deve ser pressionada: um comando na posição 5 necessita que a tecla '↑' seja pressionada cinco

vezes antes de aparecer no terminal (note que à medida que comandos vão sendo executados, a posição de um dado comando no histórico pode mudar).

O final da entrada é indicado por $N = 0$.

A entrada deve ser lida da entrada padrão.

Saída

Para cada caso de teste, seu programa deve imprimir apenas uma linha, contendo o número de vezes que Leandro precisa pressionar a tecla '↑' para executar todos os comandos.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	**Saída para o exemplo de entrada**
3 2 5 3 4 2 1 4 3 5 1 2 3 4 5 4 1 3 1 3 0	13 16 25 9

Esquerda, Volver! (5 – verde)

Arquivo fonte: *esquerda.c*, *esquerda.cpp*, *esquerda.pas* ou *esquerda.java*

Este ano o sargento está tendo mais trabalho do que de costume para treinar os recrutas. Um deles é muito atrapalhado, e de vez em quando faz tudo errado – por exemplo, ao invés de virar à direita quando comandado, vira à esquerda, causando grande confusão no batalhão.

O sargento tem fama de durão e não vai deixar o recruta em paz enquanto este não aprender a executar corretamente os comandos. No sábado à tarde, enquanto todos os outros recrutas estão de folga, ele obrigou o recruta a fazer um treinamento extra. Com o recruta marchando parado no mesmo lugar, o sargento emitiu uma série de comandos "esquerda volver!" e "direita volver!". A cada comando, o recruta deve girar sobre o mesmo ponto e dar um quarto de volta na direção correspondente ao comando. Por exemplo, se o recruta está inicialmente com o rosto voltado para a direção norte, após um comando de "esquerda volver!" ele deve ficar com o rosto voltado para a direção oeste. Se o recruta está inicialmente com o rosto voltado para o leste, após um comando "direita, volver!" ele deve ter o rosto voltado para o sul.

No entanto, durante o treinamento, em que o recruta tinha inicialmente o rosto voltado para o norte, o sargento emitiu uma série tão extensa de comandos, e tão rapidamente, que até ele ficou confuso, e não sabe mais para qual direção o recruta deve ter seu rosto voltado após executar todos os comandos. Você pode ajudar o sargento?

Entrada

A entrada contém vários casos de teste. A primeira linha de um caso de teste contém um inteiro N que indica o número de comandos emitidos pelo sargento ($1 \leq N \leq 1000$). A segunda linha contém N caracteres, descrevendo a série de comandos emitidos pelo sargento. Cada comando é representado por uma letra: 'E' (para "esquerda, volver!") e 'D' (para "direita, volver!").

O final da entrada é indicado por $N = 0$.

A entrada deve ser lida da entrada padrão.

Saída

Para cada caso de teste da entrada seu programa deve produzir uma única linha da saída, indicando a direção para a qual o recruta deve ter sua face voltada após executar a série de comandos, considerando que no início o recruta tem a face voltada para o norte. A linha deve conter uma letra entre 'N', 'L', 'S' e 'O', representando respectivamente as direções norte, leste, sul e oeste.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	**Saída para o exemplo de entrada**
3 DDE 2 EE 0	L S