9ª MARATONA DE PROGRAMAÇÃO DA FIPP

DIA 15/05 - 14H

CADERNO DE PROBLEMAS

15/05/2013 – 14:00 às 18:00

Leia atentamente estas instruções antes de iniciar a resolução dos problemas. Este caderno é composto de 6 problemas, sendo que 2 deles estão descritos em inglês.

1. É permitido o uso de material impresso, livros, apostilas e dicionários bem como lápis, caneta ou lapiseira, borracha, régua e papel para rascunho. O acesso à internet é bloqueado durante a realização da prova. A ajuda do ambiente de desenvolvimento como C, C++ ou Java pode ser utilizada.
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 os nomes indicados nos problemas para nomear os seus arquivos-fonte, 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 exibi-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 C: `scanf`, `getchar`, `printf`, `putchar`;
- em C++: as mesmas de C ou os objetos `cin` e `cout`;
- 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);
```

Em C ou C++: use `sprintf(str,"%d",num);` em vez de `itoa(num,str,10);`

Faculdade de Informática de Presidente Prudente

http://www.unoeste.br/fipp/maratona

Loop Musical (1 – vermelha)

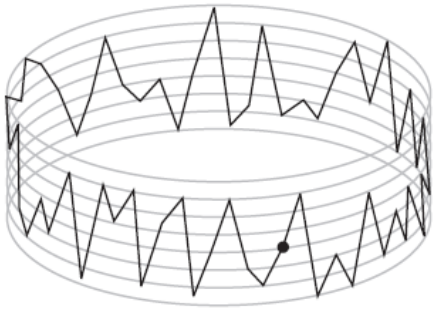
Arquivo fonte: *loop.c*, *loop.cpp* ou *loop.java*

Um loop musical é um trecho de música que foi composto para repetir continuamente (ou seja, o trecho inicia novamente toda vez que chega ao final), sem que se note descontinuidade. Loops são muito usados na sonorização de jogos, especialmente jogos casuais pela internet.

Loops podem ser digitalizados, por exemplo, utilizando PCM. PCM, do inglês *Pulse Code Modulation*, é uma técnica para representação de sinais analógicos, muito utilizada em áudio digital. Nessa técnica, a magnitude do sinal é amostrada em intervalos regulares de tempo, e os valores amostrados são armazenados em sequência. Para reproduzir a forma de onda amostrada, o processo é invertido (demodulação).

Fernandinha trabalha para uma empresa que desenvolve jogos e compôs um bonito loop musical, codificando-o em PCM. Analisando a forma de onda do seu loop em um software de edição de áudio, Fernandinha ficou curiosa ao notar a quantidade de "picos" existentes. Um pico em uma forma de onda é um valor de uma amostra que representa um máximo ou mínimo local, ou seja, um ponto de inflexão da forma de onda. A figura abaixo ilustra: (a) um exemplo de forma de onda; e (b) o loop formado com essa forma de onda, contendo 48 picos.

(a) Uma forma de onda



(b) Mesma forma de onda em loop

Fernandinha é uma amiga muito querida e pediu sua ajuda para determinar quantos picos existem no seu loop musical.

Entrada

A entrada contém vários casos de teste. A primeira linha de um caso de teste contém um inteiro N, representando o número de amostras no loop musical de Fernandinha ($2 \leq N \leq 10^4$). A segunda linha contém N inteiros H_i, separados por espaços,

representando a sequência de magnitudes das amostras ($-10^4 \leq H_i \leq 10^4$ para $1 \leq i \leq N$, $H_1 \neq H_N$ e $H_i \neq H_{i+1}$ para $1 \leq i < N$). Note que H_1 segue H_N quando o loop é reproduzido.

O final da entrada é indicado por uma linha que contém apenas o número zero.

A entrada deve ser lida da entrada padrão.

Saída

Para cada caso de teste da entrada seu programa deve imprimir uma única linha, contendo apenas um inteiro, o número de picos existentes no loop musical de Fernandinha.

O resultado de seu programa deve ser escrito na saída padrão.

Exemplo de entrada	**Saída para o exemplo de entrada**
2 1 -3 6 40 0 -41 0 41 42 4 300 450 449 450 0	2 2 4

Curso Universitário (2 – rosa)

Arquivo fonte: *curso.c*, *curso.cpp* ou *curso.java*

Há alguns anos a Universidade de Pinguinhos introduziu um novo sistema flexível de créditos para os alunos ingressantes de cursos de graduação. No novo sistema os alunos podem escolher as disciplinas que desejam cursar em um semestre, com a única restrição de não poderem cursar uma dada disciplina *A* sem antes terem cursado todas as disciplinas que tiverem sido estabelecidas como pré-requisitos de *A*. Após alguns semestres o reitor notou que muitos estudantes estavam sendo reprovados em muitas disciplinas, simplesmente porque os estudantes estavam cursando muitas disciplinas por semestre – alguns estudantes chegavam a se matricular em até quinze disciplinas em um semestre. Sendo muito sábio, esse ano o reitor introduziu uma regra adicional limitando o número de disciplinas que cada estudante pode cursar por semestre a certo valor *N*. Essa regra adicional, no entanto, fez com que os alunos ficassem muito confusos na hora de escolher as disciplinas a serem cursadas em cada semestre.

É aí que você entra na estória. O reitor resolveu disponibilizar um programa de computador para ajudar os alunos a fazerem suas escolhas de disciplinas, e solicitou sua ajuda. Mais precisamente, o reitor quer que o programa sugira as disciplinas a serem cursadas durante o curso da seguinte forma. A cada disciplina é atribuída uma *prioridade*. Se mais do que *N* disciplinas podem ser cursadas em um determinado semestre (obedecendo ao sistema de pré-requisitos), o programa deve sugerir que o aluno se matricule nas *N* disciplinas de maior prioridade. Se *N* ou menos disciplinas podem ser cursadas em um determinado semestre, o programa deve sugerir que o aluno matricule-se em todas as disciplinas disponíveis.

Portanto, dadas a descrição de pré-requisitos para cada disciplina, a prioridade de cada disciplina, e o número máximo de disciplinas por semestre, seu programa deve calcular o número necessário de semestres para concluir o curso, segundo a sugestão do reitor, e a lista de disciplinas que o aluno deve matricular-se em cada semestre.

Entrada

A entrada contém vários casos de teste. Se uma disciplina não tem qualquer pré-requisito ela é denominada *básica*; caso contrário ela é denominada *avançada*. A primeira linha de um caso de teste contém dois números inteiros, $1 \le N \le 100$ e $1 \le M \le 10$, indicando, respectivamente, o número de disciplinas avançadas do curso e o número máximo de disciplinas que podem ser cursadas por semestre. Cada uma das *N* linhas seguintes tem o formato

```
STR0 K STR1 STR2 ... STRK
```

sendo STR0 o nome de uma disciplina avançada, $1 \le K \le 15$ o número de disciplinas que são pré-requisitos de `STR0`, e `STR1`, `STR2`, . . . `STRK` os nomes das disciplinas que são pré-requisitos de `STR0`. O nome de uma disciplina é uma cadeia com no mínimo um e no máximo sete caracteres alfanuméricos maiúsculos (‘`A`’-‘`Z`’ e ‘`0`’-‘`9`’). Note que as disciplinas básicas são aquelas que aparecem apenas como pré-requisito de alguma disciplina avançada. Para concluir o curso, um aluno deve cursar (e passar!) em todas as disciplinas básicas e avançadas. A prioridade das disciplinas é determinada pela ordem em que elas aparecem pela primeira vez na entrada: a que aparece primeiro tem maior prioridade, e a que aparece por último tem a menor prioridade. Não há circularidade nos

pré-requisitos (ou seja, se a disciplina *B* tem como pré-requisito a disciplina *A* então *A* não tem *B* como pré-requisito, direta ou indiretamente). O número total de disciplinas é no máximo igual a 200. O final da entrada é indicado por $N = M = 0$.

A entrada deve ser lida da entrada padrão.

Saída

Para cada caso de teste da entrada seu programa deve produzir a saída na seguinte forma. A primeira linha deve conter a frase '`Formatura em S semestres`', sendo S o número de semestres necessários para concluir o curso segundo a sugestão do reitor. As `S` linhas seguintes devem conter a descrição das disciplinas a serem cursadas em cada semestre, um semestre por linha, no formato mostrado no exemplo de saída abaixo. Para cada semestre, a lista das disciplinas deve ser dada em ordem lexicográfica.

Definição: considere as cadeias de caracteres $S_a = a_1a_2...a_m$ e $S_b = b_1b_2 \ldots b_n$. S_a precede S_b em ordem lexicográfica se e apenas se S_b é não-vazia e uma das seguintes condições é verdadeira:

- S_a é uma cadeia vazia;
- $a_1 < b_1$ na ordem '0' < '1' < '2' < . . . < '9' < 'A' < 'B' < . . . < 'Z';
- $a_1 = b_1$ e a cadeia $a_2a_3 \ldots a_m$ precede a cadeia $b_2b_3 \ldots b_n$.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	**Saída para o exemplo de entrada**

`2 2`
`B02 3 A01 A02 A03`
`C01 2 B02 B01`
`3 2`
`ARTE2 1 ARTE1`
`PROG3 1 PROG2`
`PROG2 2 MAT1 PROG1`
`0 0`
	`Formatura em 4 semestres`
	`Semestre 1 : A01 A02`
	`Semestre 2 : A03 B01`
	`Semestre 3 : B02`
	`Semestre 4 : C01`
	`Formatura em 4 semestres`
	`Semestre 1 : ARTE1 MAT1`
	`Semestre 2 : ARTE2 PROG1`
	`Semestre 3 : PROG2`
	`Semestre 4 : PROG3`

Symmetric Order (3 – azul)

Source file: *symmetric.c*, *symmetric.cpp* or *symmetric.java*

In your job at Albatross Circus Management (yes, it's run by a bunch of clowns), you have just finished writing a program whose output is a list of names in nondescending order by length (so that each name is at least as long as the one preceding it). However, your boss does not like the way the output looks, and instead wants the output to appear more symmetric, with the shorter strings at the top and bottom and the longer strings in the middle. His rule is that each pair of names belongs on opposite ends of the list, and the first name in the pair is always in the top part of the list. In the first example set below, Bo and Pat are the first pair, Jean and Kevin the second pair, etc.

Input

The input consists of one or more sets of strings, followed by a final line containing only the value 0. Each set starts with a line containing an integer, *n*, which is the number of strings in the set, followed by *n* strings, one per line, sorted in nondescending order by length. None of the strings contain spaces. There is at least one and no more than 30 strings per set. Each string is at most 25 characters long.

The input must be read from standard input.

Output

For each input set print "`SET n`" on a line, where *n* starts at 1, followed by the output set as shown in the sample output.

The output must be written to standard output.

Sample Input	Output for the sample Input
7 Bo Pat Jean Kevin Claude William Marybeth 6 Jim Ben Zoe Joey Frederick Annabelle 5 John Bill Fran Stan Cece 0	SET 1 Bo Jean Claude Marybeth William Kevin Pat SET 2 Jim Zoe Frederick Annabelle Joey Ben SET 3 John Fran Cece Stan Bill

Factorial Again! (4 – verde)

Source file: *factorial.c*, *factorial.cpp* or *factorial.java*

Mathew, an engineering freshman, is developing an original positional notation for representing integer numbers. He called it "A Curious Method" (ACM for short). The ACM notation uses the same digits as the decimal notation, i.e., 0 through 9.

To convert a number A from ACM to decimal notation you must add k terms, where k is the number of digits of A (in the ACM notation). The value of the i-th term, corresponding the i-th digit a_i, counting from right to left, is $a_i \times i!$. For instance 719_{ACM} is equivalent to 53_{10}, since $7 \times 3! + 1 \times 2! + 9 \times 1! = 53$.

Mathew has just begun studying number theory, and probably does not know which properties a numbering system should have, but at the moment he is only interested in converting a number from ACM to decimal. Could you help him?

Input

Each test case is given in a single line that contains a non-empty string of at most 5 digits, representing a number in ACM notation. The string does not have leading zeros.

The last test case is followed by a line containing one zero.

The input must be read from standard input.

Output

For each test case output a single line containing the decimal representation of the corresponding ACM number.

The output must be written to standard output.

Sample input	Output for the sample input
719 1 15 110 102 0	53 1 7 8 8

Alarme Despertador (5 – branca)

Arquivo fonte: *alarme.c*, *alarme.cpp* ou *alarme.java*

Daniela é enfermeira em um grande hospital, e tem os horários de trabalho muito variáveis. Para piorar, ela tem sono pesado e uma grande dificuldade para acordar com relógios despertadores.

Recentemente ela ganhou de presente um relógio digital, com alarme com vários tons, e tem esperança que isso resolva o seu problema. No entanto, ela anda muito cansada e quer aproveitar cada momento de descanso. Por isso, carrega seu relógio digital despertador para todos os lugares, e sempre que tem um tempo de descanso procura dormir, programando o alarme despertador para a hora em que tem que acordar. No entanto, com tanta ansiedade para dormir, acaba tendo dificuldades para adormecer e aproveitar o descanso.

Um problema que a tem atormentado na hora de dormir é saber quantos minutos ela teria de sono se adormecesse imediatamente e acordasse somente quando o despertador tocasse. Mas ela realmente não é muito boa com números, e pediu sua ajuda para escrever um programa que, dada a hora corrente e a hora do alarme, determine o número de minutos que ela poderia dormir.

Entrada

A entrada contém vários casos de teste. Cada caso de teste é descrito em uma linha, contendo quatro números inteiros H_1, M_1, H_2 e M_2, com H_1:M_1 representando a hora e minuto atuais, e H_2:M_2 representando a hora e minuto para os quais o alarme despertador foi programado ($0 \leq H_1 \leq 23$, $0 \leq M_1 \leq 59$, $0 \leq H_2 \leq 23$, $0 \leq M_2 \leq 59$).

O final da entrada é indicado por uma linha que contém apenas quatro zeros, separados por espaços em branco.

A entrada deve ser lida da entrada padrão.

Saída

Para cada caso de teste da entrada seu programa deve imprimir uma linha, cada uma contendo um número inteiro, indicando o número de minutos que Daniela tem para dormir.

A saída deve ser escrita na saída padrão.

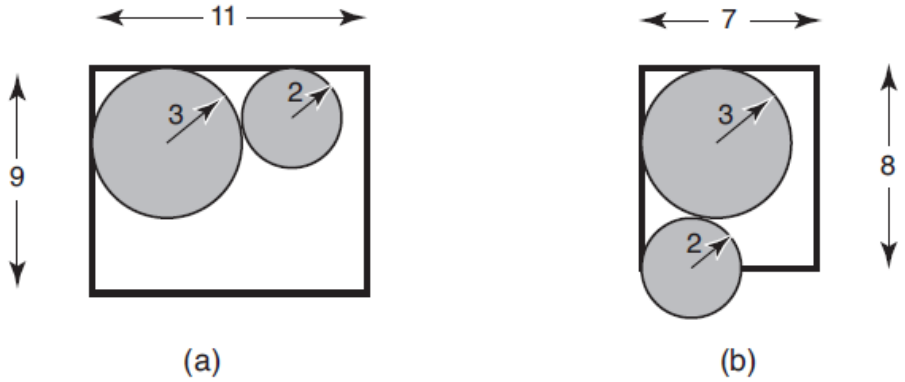
Exemplo de entrada	Saída para o exemplo de entrada
1 5 3 5 23 59 0 34 21 33 21 10 0 0 0 0	120 35 1417

Elevador (6 – amarela)

Arquivo fonte: *elevador.c*, *elevador.cpp* ou *elevador.java*

A FCC (Fábrica de Cilindros de Carbono) fabrica vários tipos de cilindros de carbono. A FCC está instalada no décimo andar de um prédio e utiliza os vários elevadores do prédio para transportar seus cilindros. Por questão de segurança, os cilindros devem ser transportados na posição vertical; como são pesados, no máximo, dois cilindros podem ser transportados em uma única viagem de elevador. Os elevadores têm formato de paralelepípedo e sempre têm altura maior que a altura dos cilindros.

Para minimizar o número de viagens de elevador para transportar os cilindros, a FCC quer, sempre que possível, colocar dois cilindros no elevador. A figura abaixo ilustra, esquematicamente (vista superior), um caso em que isto é possível (a), e um caso em que isto não é possível (b):



Como existe uma quantidade muito grande de elevadores e de tipos de cilindros, a FCC quer que você escreva um programa que, dadas as dimensões do elevador e dos dois cilindros, determine se é possível colocar os dois cilindros no elevador.

Entrada

A entrada contém vários casos de teste. A primeira e única linha de cada caso de teste contém quatro números inteiros L, C, R_1 e R_2, separados por espaços em branco, indicando, respectivamente, a largura do elevador ($1 \leq L \leq 100$), o comprimento do elevador ($1 \leq C \leq 100$), e os raios dos cilindros ($1 \leq R_1, R_2 \leq 100$).

O último caso de teste é seguido por uma linha que contém quatro zeros separados por espaços em branco.

A entrada deve ser lida da entrada padrão.

Saída

Para cada caso de teste, o seu programa deve imprimir uma única linha com um único caractere: 'S' se for possível colocar os dois cilindros no elevador e 'N' caso contrário.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	**Saída para o exemplo de entrada**
11 9 2 3 7 8 3 2 10 15 3 7 8 9 3 2 0 0 0 0	S N N S