

**CPAS'2009@CIC**

**FINAL**

---

**29 de Abril 2009**  
**Colégio Internato dos Carvalhos**

### Exercício 1

Escreva um programa que leia um número inteiro **X** ( $X \geq 1$ ) e que calcule, partindo do número fornecido, uma sequência de números, de acordo com as regras:

- **caso seja par**, dividir por 2;
- **caso seja ímpar**, multiplicar por 3 e somar 1;
- **quando o resultado atingir o valor 1**, terminar a sequência.

O programa indica o tamanho da sequência gerada da forma: **#tamanho**.

#### Exemplo 1

**Entrada:**

9

**Saída:**

```
28
14
7
22
11
34
17
52
26
13
40
20
10
5
16
8
4
2
1
#19
```

## QUESTÕES SOBRE O FUNCIONAMENTO DO SISTEMA

### INPUT/OUTPUT (ENTRADA/SAÍDA)

- ✓ Em todos os problemas o input é feito pelo standard input e o output é feito pelo standard output. Portanto, os vossos programas podem e devem usar as funções "normais" de escrita leitura, como sejam o scanf e printf (em C) ou o read/readln e write/writeln (em Pascal).
- ✓ Os inputs/outputs devem ser exactos, isto é, os dados de entrada e saída devem ser EXACTAMENTE IGUAIS (no conteúdo, na formatação, na sequência, no respeito pelas maiúsculas e minúsculas, no espaçamento, nas mudanças de linha, etc.) aos apresentados no enunciado.
- ✓ O standard input pode ser pensado como uma stream equivalente a qualquer ficheiro. Portanto, quando acaba, tem um end-of-file.  
Em C, quando o final de input é atingido, um fgets retorna NULL, um scanf retorna zero argumentos lidos e um feof(stdin) retorna true. Em FP, quando o final de input é atingido, EOF retorna true. Para simular um end-of-file se estiver a escrever via teclado, pressione Ctr+D em Linux e Ctr+Z em Windows.
- ✓ A última linha do output deve incluir mudança de linha (writeln em Pascal ou \n em C).

### Importante:

- Em C/C++:
- ✓ Não utilizar fflush(stdin), pois a sua utilização torna todas as submissões erradas.
  - ✓ Deve ter o cuidado de colocar a instrução return 0 na função int main () (o que é interpretado pelo Sistema de Submissão como fim do programa).
  - ✓ São permitidas apenas as bibliotecas: **stdio**, **strings**, **math** e **stdlib**.
- Em FP:
- ✓ Não é permitida a inclusão de qualquer biblioteca externa.

**SITUAÇÕES OMISSAS OU NÃO REGULAMENTADAS SERÃO DECIDIDAS PELO JÚRI DO CONCURSO. EM TODOS OS CASOS, AS EQUIPAS COMPROMETEM-SE A RESPEITAR A DECISÃO SOBERANA DO JÚRI, NÃO HAVENDO DIREITO A RECURSO.**

### Exercício 2

Implemente um programa que leia uma lista de números inteiros (entre 1 e 10000), contabilizando:

- > **+1**, se dois valores consecutivos estão em ordem crescente;
- > **-1**, se estão em ordem decrescente;
- > **0 (zero)**, se são iguais.

A lista de valores termina quando é fornecido o valor 0 (zero) e é apresentado o resultado da soma. (O zero final não é considerado como pertencendo à lista.)

#### Exemplo 1

##### Entrada:

```
12
14
15
11
11
12
0
```

##### Saída:

```
2
```

### Exercício 3

Implemente um programa que, dada uma lista de palavras denominadas palavras-chave, determine o número de ocorrências de cada palavra-chave no texto.

O programa lê o número **N** ( $1 \leq N \leq 20$ ) de palavras-chave, a lista de palavras-chave (uma por linha), o número **M** ( $1 \leq M \leq 50$ ) de linhas do texto e cada uma das linhas do texto (*string* com comprimento máx. 100 caracteres).

Assuma o seguinte:

- o texto e as palavras-chave são escritas utilizando apenas caracteres da língua inglesa (A..Z; a..z) e hífenes;
- ignoram-se as diferenças entre maiúsculas e minúsculas;
- não existem palavras-chave repetidas;
- a lista de ocorrências é apresentada em minúsculas.

#### Exemplo 1

##### Entrada:

```
6
hoje
ONTEM
amanha
sol
chuva
nevoeiro
```

3

Hoje esta um dia de sol

Ontem esteve um dia de chuva

Amanha preve-se chuva

##### Saída:

```
hoje-1
ontem-1
amanha-1
sol-1
chuva-2
nevoeiro-0
```

## REGULAMENTO

### INÍCIO DE SESSÃO E SOFTWARE

- ✓ Deve efectuar o início de sessão com o utilizador **cpas** e password **cpas123**
- ✓ Todo o software necessário ao desenvolvimento da prova foi previamente instalado, estando os respectivos atalhos disponíveis no ambiente de trabalho.

### ADVERTÊNCIAS

- ✓ Não podem consultar quaisquer materiais de apoio, para além dos que forem fornecidos pela organização.
- ✓ Não é permitida a utilização de dispositivos de memória secundária (PenDisks, etc.).
- ✓ Os elementos da equipa podem comunicar apenas entre si, com os cuidados devidos de forma a não perturbar as restantes equipas.
- ✓ Devem desligar o telemóvel ou outros aparelhos de comunicação.
- ✓ Não podem usar a Internet/Intranet para comunicar, consultar informação ou partilhar ficheiros.
- ✓ Só podem abandonar a sala com a autorização do membro da organização presente na sala.
- ✓ Não podem instalar, eliminar ou alterar o software e a configuração dos postos.

**No caso do não cumprimento das disposições anteriores a equipa é automaticamente desclassificada.**

### RECOMENDAÇÕES

- ✓ Por prevenção, devem guardar as resoluções com alguma regularidade no disco local, numa pasta criada para o efeito, com o nome cpas1 para o exercício 1, cpas2 para o exercício 2, etc..

### DÚVIDAS

- ✓ Em caso de dúvida, devem dirigir-se ao membro da organização presente na sala.

### ATRIBUIÇÃO DA CLASSIFICAÇÃO

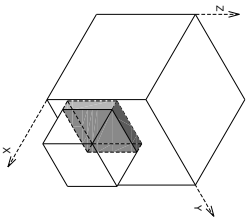
- ✓ 100 pontos por cada problema resolvido com sucesso (pressupõe que o programa ultrapassou com sucesso a bateria de testes efectuada automaticamente pelo Sistema de Submissão).
- ✓ Cada problema pode ser submetido 5 vezes, no máximo (esgotadas as 5 tentativas a submissão desse problema fica bloqueada).
- ✓ A classificação final é determinada pela soma dos pontos obtidos nas submissões efectuadas. Em caso de empate, aplicam-se os seguintes critérios:
  1. Tempo gasto pela equipa (em minutos) desde a hora de início do concurso até ao instante em que o último problema é submetido com sucesso.
  2. Após a aplicação do critério anterior, se o empate persistir, o júri decide a classificação em função da qualidade global das soluções apresentadas.

### Exercício 10

Implemente um programa que calcule o volume da intersecção de **N** ( $2 \leq N \leq 100$ ) cubos.

A entrada consiste numa série de cubos para a qual o volume da intersecção deve ser calculado. A primeira linha contém o número **N** de cubos e, seguidamente, são indicados os **N** cubos (um por linha). Cada linha descreve um cubo através de quatro números inteiros (entre 1 e 100). Os três primeiros números são as coordenadas **x**, **y**, e **z**, do canto do cubo e o quarto inteiro (positivo) é a dimensão das arestas do cubo (nas 3 direcções paralelas aos eixos **x**, **y**, e **z**).

#### Exemplo 1



**Entrada:**

```
2
0 0 0 10
9 1 1 5
```

**Saída:**

```
25
```

#### Exemplo 2

**Entrada:**

```
3
0 0 0 10
9 1 1 5
8 2 2 3
```

**Saída:**

```
9
```

### Exercício 4

Implemente um programa que simule a «roda da sorte».

Esta roda está dividida em 20 fatias, cada fatia corresponde a um prémio de 10, 20 ou 40 euros. Em cada quadrante (a roda é constituída por 4 quadrantes) há 5 fatias com prémios pela ordem seguinte: 10, 20, 40, 20 e 10 euros.

O programa deverá ler a força com que a roda é girada, isto é, o número **F** ( $F > 0$ ) correspondente ao número de fatias pelas quais o ponteiro passa, o quadrante e a fatia inicial (os dois últimos valores lidos na mesma linha).

A saída do programa consiste na sequência de fatias e no prémio obtido.

#### Exemplo 1:

Força 7 (o ponteiro passa por 7 fatias), quadrante inicial 1 e fatia inicial 2.

**Entrada:**

```
7
1 2
```

**Saída:**

```
1 20
1 40
1 20
1 10
2 10
2 20
2 40
PREMIO=40
```

### Exercício 5

Uma estratégia utilizada para remover elementos de uma lista é a denominada «remoção peguquosa».

Neste tido de remoção, quando queremos eliminar um ou mais elementos com um dado valor de uma lista, apenas o marcamos como «del».

Quando o número de elementos eliminados desta forma é igual ou superior ao número de elementos «efectivos», procedemos à remoção «de facto» dos elementos da lista.

Quando pretendemos inserir um novo elemento e temos elementos «del», inserimos na primeira posição desse tipo, caso contrário, inserimos no final da lista.

Implemente um programa que:

- leia uma lista de **N** ( $1 \leq N \leq 100$ ) números inteiros entre 1 e 10000;
- aceite o conjunto de operações sobre a lista: - para remover da lista todos os elementos iguais ao indicado, + para inserir o elemento indicado e \* para terminar a introdução;
- apresente a lista resultante, indicando os elementos no estado «del».

#### Exemplo 1

Entrada:

```
7
4 3 8 5 8 7 8
8
+
20
*
```

Saída:

```
4 3 20 5 del 7 del
```

#### Exemplo 2

Entrada:

```
4
12 3 6 8
-
3
-
6
+
1
1
*
```

Saída:

```
12 8 1
```

### Exercício 9

Os militares americanos perceberam que os talibãs estão a escutar as suas transmissões. Sabem, também, que o equipamento usado é desactualizado e que não consegue detectar a transmissão se esta for curta. Torna-se, por isso, importante "comprimir" o máximo de informações na transmissão.

Para resolver este problema, implemente um programa que comprima todas as palavras de uma mensagem numa única *string*, concatenando e sobrepondo palavras com um ou mais caracteres em comum, para formar uma «palavra-comboio» que é menor do que a soma de todas as palavras da mensagem.

Por exemplo, a expressão "norman" e "mankind" pode ser associada de uma forma mais curta na *string* "normankind", que contém ambas as palavras como *substrings*.

Por conseguinte, o objectivo do programa é o de construir a *string* mais curta que contém todas as palavras da mensagem. A ordem das palavras no «comboio» não é importante, uma vez que uma outra mensagem é transmitida indicando a ordem das palavras da mensagem no «comboio».

O programa lê um número inteiro **N** ( $1 \leq N \leq 8$ ) linhas, contendo uma única palavra (em minúsculas). O comprimento máximo de cada palavra é dez caracteres. Se existir mais do que uma *string* optimizada, o programa poderá gerar qualquer uma dessas *strings*.

#### Exemplo 1

Entrada:

```
3
kin
mankind
norman
```

Saída:

```
normankind
```

#### Exemplo 2

Entrada:

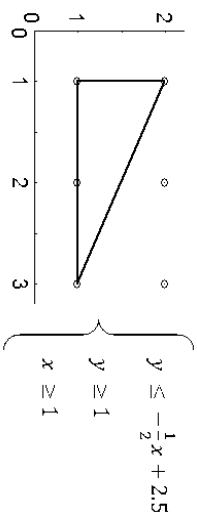
```
7
domain
reed
aint
ring
freedom
integer
gerring
```

Saída:

```
Freedomaintegering
```

### Exercício 8

Considere-se **R** restrições no plano, cuja intersecção define um polígono convexo. Por exemplo, o triângulo da figura seguinte é definido por 3 restrições:



Implemente o método de **Monte Carlo** para calcular aproximadamente a área deste polígono. Este método consiste em tomar um retângulo que contém o polígono e escolher pontos deste retângulo aleatoriamente (neste caso são dados no input), contabilizando os pontos que pertencem ao polígono (sucessos). Assim, a área do polígono é aproximadamente a área do retângulo multiplicada pela fracção do número de sucessos pelo número total de pontos considerados (ver exemplo).

#### Entrada e Saída

A primeira linha da entrada contém 4 reais correspondendo ao canto inferior esquerdo e superior direito do retângulo a considerar: **x1 y1 x2 y2**. De seguida vem o número **R** ( $3 \leq R \leq 20$ ) de restrições seguido por **R** linhas com a definição de cada uma num dos 3 formatos dados no exemplo (os únicos operadores possíveis são **<=** ou **>=** podendo estes aparecer em qualquer um dos formatos). Por fim vem o número **P** ( $1 \leq P \leq 1000$ ) de pontos e suas coordenadas reais **x,y**.

#### Exemplo 1

Entrada:

```
1.0 1.0 3.0 2.0
3
```

```
y <= -0.5x + 2.5
```

```
y >= 1
```

```
x >= 1
```

```
5
```

```
1.5 1.5
```

```
2.5 1.7
```

```
2.0 2.0
```

```
1.0 2.0
```

```
1.8 2.0
```

Saída:

```
0.80
```

Explicação:

Apenas o 1º e o 4º ponto pertencem ao triângulo formado pelas restrições. Assim, o resultado = [área\_rectângulo=2] x [sucessos/número\_pontos=2/5].

Autor: Miguel Oliveira, vencedor do CPAS 2005

### Exercício 6

Implemente um programa que leia um número inteiro **X** ( $0 \leq X \leq 999$ ) e o escreva por extenso, em minúsculas.

**Exemplo 1**

Entrada:

```
25
```

Saída:

```
vinte e cinco
```

**Exemplo 2**

Entrada:

```
999
```

Saída:

```
novecientos e noventa e nove
```

### Exercício 7

Implemente o seguinte programa baseado no jogo das minas. Suponha um campo minado de 3x5 células e 3 minas (as minas são indicadas por \* e as células vazias por .):

```
* *
. . .
. . . .
. *
. . . .
```

Calculando o número de minas adjacentes a cada célula (numa das 8 células vizinhas), teríamos os seguintes valores associados a cada uma das células do campo minado:

```
* * 100
33200
1 * 100
```

Quando «cliquemos» numa determinada célula, temos uma de 3 situações:

- **foi clicada uma MINA**, o jogador morre (o programa emite a mensagem **MORREU**);
- a célula **TEM minas adjacentes**, é destapada (é mostrado o número associado a essa célula);
- a célula **NÃO TEM minas adjacentes**, são então destapadas todas as células vizinhas, ou vizinhas das vizinhas, até se encontrarem células com minas adjacentes (número maior do que 0).

Supondo que nos encontramos no estado inicial, ao clicarmos na posição 1,5 (linha 1, coluna 5), serão destapadas todas as células vizinhas, e vizinhas das vizinhas, até se encontrarem células com minas adjacentes (número maior do que 0):

```
--100
--200
--100
```

**Nota:** As células tapadas são indicadas por -

O programa lê o número **N** ( $1 \leq N \leq 100$ ) de linhas e **M** ( $1 \leq M \leq 100$ ) de colunas do campo minado e as minas (indicadas por \*) ou o número de minas adjacentes à célula respectiva (de 0 a 8).

De seguida, lê as coordenadas **X** ( $1 \leq X \leq N$ ), **Y** ( $1 \leq Y \leq M$ ) da célula a destapar.

Por fim, apresenta o campo minado resultante (com a célula ou células destapadas) ou a mensagem **MORREU**.

#### Exemplo 1

**Entrada:**

```
8 8
*2100000
2*210000
12*21000
012*2100
0012*210
00012*21
000012*2
0000012*
6 2
```

**Saida:**

```
-----
12-----
012-----
0012-----
00012---
000012---
000001--
```

#### Exemplo 2

**Entrada:**

```
6 10
*****1***3*
5**32356*3
**543****4
45***5*7**
**5*43*6*5
*33*212***
2 2
```

**Saida:**

MORREU

#### Exemplo 3

**Entrada:**

```
4 4
*100
1100
0000
0000
1 2
```

**Saida:**

```
-1--
-----
-----
```