Faculdade de Computação

Programação Procedimental

1° Desafio de Programação

Prof. Cláudio C. Rodrigues

Instruções:

1. Apresentar as soluções usando a ***linguagem C***;
2. O trabalho deve ser desenvolvido em grupo composto de 1 até 3 (um até três) estudantes e qualquer identificação de plágio sofrerá penalização;
3. O trabalho deve possuir uma capa, com identificação da disciplina, identificação da atividade, nomes dos estudantes, números de matrícula e e-mail;
4. Entregar o relatório com soluções, impreterivelmente, no dia **24/07/2013**;
5. O relatório impresso deverá ser entregue ao aluno monitor da disciplina, que registrará a entrega;
6. Enviar via e-mail, ao professor da disciplina, um arquivo zipado com os códigos fontes das soluções propostas – Assunto: **PP - Primeiro Desafio de Programação 2013-1**
7. Cada relatório deverá apresentar uma capa com a formatação apresentada abaixo:

Formatação da Capa do Relatório:

Programação Procedimental

1° Desafio de Programação

Prof. Cláudio C. Rodrigues

Autores:

Fulano de Tal	2013XXX145	fulano@ufu.br
Beltrano da Silva	2013XXX157	beltrano@ufu.br

Uberlândia, 24 de julho de 2013.

Cofrinhos da Vó Vitória

Arquivo fonte: cofre.c, cofre.cc, cofre.cpp ou cofre.pas

Vó Vitória mantém, desde o nascimento dos netos Joãozinho e Zezinho, um ritual que faz a alegria dos meninos. Ela guarda todas as moedas recebidas como troco em dois pequenos cofrinhos, um para cada neto. Quando um dos cofrinhos fica cheio, ela chama os dois netos para um alegre almoço, ao final do qual entrega aos garotos as moedas guardadas nos cofrinhos de cada um.

Ela sempre foi muito zelosa quanto à distribuição igualitária do troco arrecadado. Quando, por força do valor das moedas, ela não consegue depositar a mesma quantia nos dois cofrinhos, ela memoriza a diferença de forma a compensá-la no próximo depósito.

1. Tarefa

Vó Vitória está ficando velha e tem medo que deslizes de memória a façam cometer injustiças com os netos, deixando de compensar as diferenças entre os cofrinhos. Sua tarefa é ajudar Vó Vitória, escrevendo um programa de computador que indique as diferenças entre os depósitos, de forma que ela não tenha que preocupar-se em memorizá-las.

2. Entrada

A entrada é composta de vários conjuntos de teste. A primeira linha de um conjunto de teste contém um número inteiro N, que indica o número de depósitos nos cofrinhos. As N linhas seguintes descrevem cada uma um depósito nos cofrinhos; o depósito é indicado por dois valores inteiros J e Z, separados por um espaço em branco, representando respectivamente os valores, em centavos, depositados nos cofres de Joãozinho e Zezinho. O final da entrada é indicado por $N = 0$.

Exemplo de Entrada

```
3
20 25
10 5
10 10
4
0 5
12 0
0 20
17 1
0
```

3. Saída

Para cada conjunto de teste da entrada seu programa deve produzir um conjunto de linhas na saída. A primeira linha deve conter um identificador do conjunto de teste, no formato “Teste n”, onde n é numerado seqüencialmente a partir de 1. A seguir seu programa deve escrever uma linha para cada depósito do conjunto de testes. Cada linha deve conter um inteiro que representa a diferença (em centavos) entre o valor depositado nos cofrinhos do Joãozinho e do Zezinho. Deixe uma linha em branco ao final de cada conjunto de teste. A grafia mostrada no Exemplo de Saída, abaixo, deve ser seguida rigorosamente.

Exemplo de Saída

```
Teste 1
-5
0
0

Teste 2
-5
7
-13
3
```

(esta saída corresponde ao exemplo de entrada acima)

4. Restrições

$0 \leq N \leq 100$ ($N = 0$ apenas para indicar o fim da entrada)
$0 \leq J \leq 100$ (valor de cada depósito no cofre de Joãozinho)
$0 \leq Z \leq 100$ (valor de cada depósito no cofre de Zezinho)

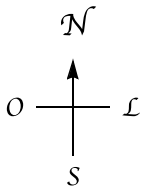
Supermercado

Arquivo fonte: super.c, super.cc, super.cpp ou super.pas

A rede de supermercados BemBom, da cidade de Planalto, decidiu reformular o armazenamento de seus estoques. No sistema atual, cada uma das lojas da rede possui espaço para armazenar um pequeno estoque, sendo freqüentemente necessário transportar mercadorias de uma loja para outra. Para racionalizar o transporte e aumentar a capacidade de estoque, a direção da rede BemBom decidiu instalar um depósito central. De forma a diminuir os custos com transporte, ficou definido que o novo depósito deve ser localizado em um quarteirão que minimize a soma das distâncias dele até todas as lojas da rede.

Por ser uma cidade planejada, Planalto possui uma característica muito peculiar. Todas as suas ruas são orientadas na direção leste-oeste ou norte-sul, e todos os quarteirões são do mesmo tamanho. Veja uma parte do mapa de Planalto na figura abaixo. Os quarteirões em Planalto são identificados pelo número de quadras, em cada direção, que os separam da localização da prefeitura (0,0). Localizações a leste e a norte da prefeitura são identificadas por coordenadas positivas, e localizações a oeste e a sul por coordenadas negativas.

-2, 2	-1, 2	0, 2	1, 2	2, 2
-2, 1	-1, 1	0, 1	1, 1	2, 1
-2, 0	-1, 0	0, 0	1, 0	2, 0
-2, -1	-1, -1	0, -1	1, -1	2, -1
-2, -2	-1, -2	0, -2	1, -2	2, -2



Parte do mapa de Planalto

1. Tarefa

A sua tarefa é, dadas as coordenadas dos quarteirões onde estão localizados todos os supermercados da rede, determinar o quarteirão onde deve ser instalado o novo depósito. A localização deste depósito deve ser tal que a soma das distâncias entre o depósito e as lojas, em número de quarteirões em ambas as direções, seja a menor possível. A distância entre dois quarteirões é dada pela distância entre eles na direção leste-oeste mais a distância na direção norte-sul. Por exemplo, a distância entre os quarteirões (2,-1) e (4, 3) é 2 + 4 = 6.

2. Entrada

A entrada é composta de vários conjuntos de teste. A primeira linha de cada conjunto de teste contém um número inteiro *S* que é o número de supermercados da rede. A seguir, são dadas *S* linhas, cada uma contendo dois números inteiros *X* e *Y*, representando as coordenadas do quarteirão onde se situa um dos supermercados. *X* representa a coordenada na direção leste-oeste e *Y* represetna a coordenada na direção norte-sul. O final da entrada é dado por um conjunto de teste com *S* = 0.

Exemplo de Entrada

```
4
1 2
-3 -3
4 -1
-5 0
5
1 3
7 13
25 13
15 14
25 1
0
```

3. Saída

Para cada conjunto de teste, o seu programa deve escrever três linhas na saída. A primeira linha deve conter um identificador do conjunto de teste, no formato "Teste *n*", onde *n* é numerado seqüencialmente a partir de 1. A segunda linha deve conter as coordenadas *X* e *Y* do quarteirão onde deve ser instalado o novo depósito, separadas por um espaço em branco. Se mais de um quarteirão puder ser escolhido como localização do depósito, seu programa pode imprimir qualquer um deles. A terceira linha deve ser deixada em branco. O formato do exemplo de saída abaixo deve ser seguido rigorosamente.

Exemplo de Saída

```
Teste 1
-2 0

Teste 2
15 13
```

(esta saída corresponde ao exemplo de entrada acima)

4. Restrições

$0 \leq S \leq 1000$ ($S = 0$ apenas para indicar o final da entrada)
$-1000 \leq X \leq 1000$
$-1000 \leq Y \leq 1000$

Macaco-prego

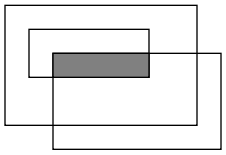
Arquivo fonte: macaco.c, macaco.cc, macaco.cpp ou macaco.pas

O macaco-prego é um animal irrequieto e barulhento, merecedor também dos adjetivos desordeiro e despudorado. A sua cabeça, encimada por uma densa pelagem negra ou marrom-escura, semelhante a um gorro, torna seu aspecto inconfundível. Apesar de ser o macaco mais comum nas matas do país, uma de suas sub-espécies encontra-se seriamente ameaçada de extinção: o macaco-prego-do-peito-amarelo, que se distingue das demais pela coloração amarelada do peito e da parte anterior dos braços.

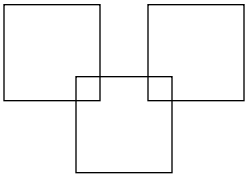
Um grande esforço foi feito pelos primatologistas para aumentar a população dos macacos-prego-do-peito-amarelo. Sabe-se que eles se alimentam de plantas, das quais consomem preferencialmente frutos e brotos. Alimentam-se também de muitos animais, preferencialmente lesmas, lagartas e rãs, e preferem as florestas mais densas. Para determinar o melhor local do país para criar uma nova reserva ambiental para os macacos-prego-do-peito-amarelo, o governo fez um levantamento das regiões no país onde as condições preferidas desses animais ocorrem: regiões de floresta densa, regiões com frutos, regiões com muitos brotos, etc. Ajude a salvar os macacos-prego-do-peito-amarelo.

1. Tarefa

As regiões propícias para o macaco-prego-do-peito-amarelo foram determinadas como retângulos cujos lados são todos verticais ou horizontais. Sua tarefa é encontrar o local ideal para a reserva ambiental, definida como a interseção de todas as regiões dadas.



Conjunto de três regiões com interseção preenchida em cinza



Conjunto de três regiões com interseção vazia

As regiões foram divididas de tal forma que uma região não tangencia qualquer outra região. Assim, a interseção entre quaisquer duas regiões ou é um retângulo ou é vazia.

2. Entrada

Seu programa deve ler vários conjuntos de teste. A primeira linha de um conjunto de teste contém um inteiro não negativo, *N*, que indica o número de regiões (o valor $N = 0$ indica o final da entrada). Seguem-se *N* linhas, cada uma contendo quatro números inteiros *X*, *Y*, *U* e *V* que descrevem uma região: o par *X*, *Y* representa a coordenada do canto superior esquerdo e o par *U*, *V* representa a coordenada do canto inferior direito de um retângulo.

Exemplo de Entrada

```
3
0 6 8 1
1 5 6 3
2 4 9 0
3
0 4 4 0
3 1 7 -3
6 4 10 0
0
```

3. Saída

Para cada conjunto de teste da entrada seu programa deve produzir três linhas na saída. A primeira linha deve conter um identificador do conjunto de teste, no formato "Teste *n*", onde *n* é numerado a partir de 1. A segunda linha deve conter as coordenadas do retângulo de interseção encontrado pelo seu programa, no mesmo formato utilizado na entrada. Caso a interseção seja vazia, a segunda linha deve conter a expressão "nenhum". A terceira linha deve ser deixada em branco. A grafia mostrada no Exemplo de Saída, abaixo, deve ser seguida rigorosamente.

Exemplo de Saída

```
Teste 1
2 4 6 3

Teste 2
nenhum
```

(esta saída corresponde ao exemplo de entrada acima)

4. Restrições

$0 \leq N \leq 10000$ ($N = 0$ apenas para indicar o fim da entrada)
$-10000 \leq X \leq 10000$
$-10000 \leq Y \leq 10000$
$-10000 \leq U \leq 10000$
$-10000 \leq V \leq 10000$

Meteoros

Arquivo fonte: meteoro.c, meteoro.cc, meteoro.cpp ou meteoro.pas

Em noites sem nuvens pode-se muitas vezes observar pontos brilhantes no céu que se deslocam com grande velocidade, e em poucos segundos desaparecem de vista: são as chamadas estrelas cadentes, ou *meteoros.* Meteoros são na verdade partículas de poeira de pequenas dimensões que, ao penetrar na atmosfera terrestre, queimam-se rapidamente (normalmente a uma altura entre 60 e 120 quilômetros). Se os meteoros são suficientemente grandes, podem não queimar-se completamente na atmosfera e dessa forma atingem a superfície terrestre: nesse caso são chamados de *meteoritos*.

Zé Felício é um fazendeiro que adora astronomia e descobriu um portal na Internet que fornece uma lista das posições onde caíram meteoritos. Com base nessa lista, e conhecendo a localização de sua fazenda, Zé Felício deseja saber quantos meteoritos caíram dentro de sua propriedade. Ele precisa de sua ajuda para escrever um programa de computador que faça essa verificação automaticamente.

1. Tarefa

São dados:

- uma lista de pontos no plano cartesiano, onde cada ponto corresponde à posição onde caiu um meteorito;
- as coordenadas de um retângulo que delimita uma fazenda.

As linhas que delimitam a fazenda são paralelas aos eixos cartesianos. Sua tarefa é escrever um programa que determine quantos meteoritos caíram dentro da fazenda (incluindo meteoritos que caíram exatamente sobre as linhas que delimitam a fazenda).

2. Entrada

Seu programa deve ler vários conjuntos de testes. A primeira linha de um conjunto de testes quatro números inteiros X_1, Y_1, X_2 e Y_2, onde (X_1, Y_1) é a coordenada do canto superior esquerdo e (X_2, Y_2) é a coordenada do canto inferior direito do retângulo que delimita a fazenda. A segunda linha contém um inteiro, *N*, que indica o número de meteoritos. Seguem-se *N* linhas, cada uma contendo dois números inteiros *X e Y*, correspondendo às coordenadas de cada meteorito. O final da entrada é indicado por $X_1 = Y_1 = X_2 = Y_2 = 0$.

Exemplo de Entrada

```
2 4 5 1
2
1 2
3 3
2 4 3 2
3
1 1
2 2
3 3
0 0 0 0
```

3. Saída

Para cada conjunto de teste da entrada seu programa deve produzir três linhas na saída. A primeira linha deve conter um identificador do conjunto de teste, no formato "Teste *n*", onde *n* é numerado a partir de 1. A segunda linha deve conter o número de meteoritos que caíram dentro da fazenda. A terceira linha deve ser deixada em branco. A grafia mostrada no Exemplo de Saída, abaixo, deve ser seguida rigorosamente.

Exemplo de Saída

```
Teste 1
1

Teste 2
2
```

(esta saída corresponde ao exemplo de entrada acima)

4. Restrições

$0 \leq N \leq 10.000$
$0 \leq X \leq 10.000$
$0 \leq Y \leq 10.000$
$0 \leq X_1 < X_2 \leq 10.000$
$0 \leq Y_2 < Y_1 \leq 10.000$

Calculando

Arquivo fonte: calcula.c, calcula.cc, calcula.cpp ou calcula.pas

A disseminação dos computadores se deve principalmente à capacidade de eles se comportarem como outras máquinas, vindo a substituir muitas destas. Esta flexibilidade é possível porque podemos alterar a funcionalidade de um computador, de modo que ele opere da forma que desejarmos: essa é a base do que chamamos programação.

1. Tarefa

Sua tarefa é escrever um programa que faça com que o computador opere como uma calculadora simples. O seu programa deve ler expressões aritméticas e produzir como saída o valor dessas expressões, como uma calculadora faria. O programa deve implementar apenas um subconjunto reduzido das operações disponíveis em uma calculadora: somas e subtrações.

2. Entrada

A entrada é composta de vários conjuntos de testes. A primeira linha de um conjunto de testes contém um número inteiro m ($1 \leq m \leq 100$), indicando o número de operandos da expressão a ser avaliada. A segunda linha de um conjunto de testes contém a expressão aritmética a ser avaliada, no seguinte formato:

$X_1\ s_1\ X_2\ s_2\ ...\ X_{m-1}\ s_{m-1}\ X_m$

onde

- X_i , $1 \leq i \leq m$, é um *operando* ($0 \leq X_i \leq 100$);
- s_j, $1 \leq j < m$, é um *operador*, representado pelos símbolos ‘+’ ou ‘–’;
- não há espaços em branco entre operandos e operadores.

O final da entrada é indicado pelo valor m=0.

Exemplo de Entrada

```
3
3+7-22
3
5-10-77
10
1+2+3+4+5+6+7+8+9+10
0
```

3. Saída

Para cada conjunto de testes da entrada seu programa deve produzir três linhas. A primeira linha deve conter um identificador da expressão, no formato “Teste *n*”, onde *n* é numerado a partir de 1. Na segunda linha deve aparecer o resultado encontrado pelo seu programa. A terceira linha deve ser deixada em branco. A grafia mostrada no Exemplo de Saída, abaixo, deve ser seguida rigoro-

samente.

Exemplo de Saída

```
Teste 1
-12

Teste 2
-82

Teste 3
55
```

(esta saída corresponde ao exemplo de entrada acima)

4. Restrições

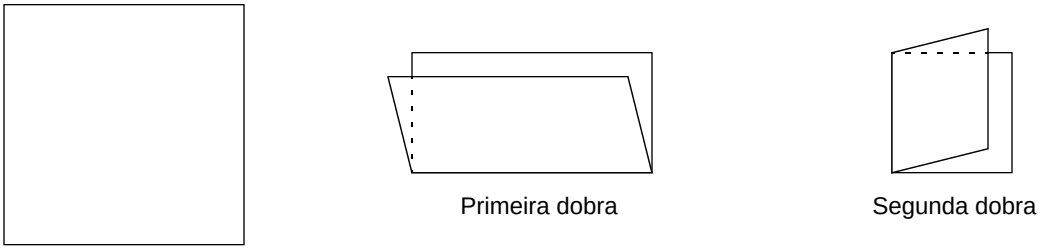
$1 \leq m \leq 100$
$0 \leq X_i \leq 100$ para todo $1 \leq i \leq m$

Dobradura

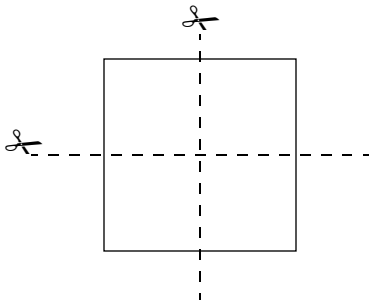
arquivo fonte: dobra.pas, dobra.c, dobra.cc ou dobra.cpp

1. Tarefa

Zezinho tem aulas de Iniciação Artística em sua escola, e recentemente aprendeu a fazer dobraduras em papel. Ele ficou fascinado com as inúmeras possibilidades de se dobrar uma simples folha de papel. Como Zezinho gosta muito de matemática, resolveu inventar um quebra-cabeça envolvendo dobraduras. Zezinho definiu uma operação de dobradura D que consiste em dobrar duas vezes uma folha de papel quadrada de forma a conseguir um quadrado com 1/4 do tamanho original, conforme ilustrado na figura.



Depois de repetir N vezes esta operação de dobradura D sobre o papel, Zezinho cortou o quadrado resultante com um corte vertical e um corte horizontal, conforme a figura abaixo.



Zezinho lançou então um desafio aos seus colegas: quem adivinha quantos pedaços de papel foram produzidos?

2. Entrada

A entrada é composta de vários conjuntos de teste. Cada conjunto de teste é composto de uma única linha, contendo um número inteiro *N* que indica o número de vezes que a operação de dobradura D foi aplicada. O final da entrada é indicado por *N* = -1.

Exemplo de Entrada

```
1
0
-1
```

3. Saída

Para cada conjunto de teste da entrada seu programa deve produzir três linhas na saída. A primeira linha deve conter um identificador do conjunto de teste, no formato “Teste *n*”, onde *n* é numerado

a partir de 1. A segunda linha deve conter o número de pedaços de papel obtidos depois de cortar a dobradura, calculado pelo seu programa. A terceira linha deve ser deixada em branco. A grafia mostrada no Exemplo de Saída, abaixo, deve ser seguida rigorosamente.

Exemplo de Saída

```
Teste 1
9

Teste 2
4
```

(esta saída corresponde ao exemplo de entrada acima)

4. Restrições

$-1 \leq N \leq 15$ ($N = -1$ apenas para indicar o fim da entrada)

Chocolate

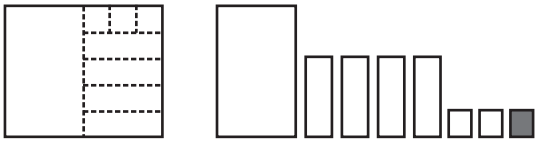
Nome do arquivo fonte: `choc.c`, `choc.cpp`, *ou* `choc.pas`

Juliana é uma famosa doceira reconhecida internacionalmente pelos seus bombons, exportados para todo o mundo. Embora não revele a ninguém as suas receitas, ela já deu entrevistas contando alguns de seus segredos. Sua fábrica de bombons utiliza somente chocolates comprados de um único produtor suíço, que envia barras gigantescas que são cortadas por grandes máquinas.

Dada uma barra grande de chocolate, Juliana realiza divisões sucessivas da barra até obter uma barra que contém a quantidade exata de chocolate para aquela receita. Após cada divisão, ela seleciona um dos pedaços resultantes e armazena os demais para uso futuro. As divisões são determinadas por critérios técnicos relacionados ao tamanho das barras e aos equipamentos disponiveis em um dado momento.

Por exemplo, se ela deseja obter uma barra de 100g de chocolate a partir de uma barra de 3Kg, primeiro ela divide a barra ao meio. Em seguida, um dos pedaços é dividido em cinco partes iguais e por fim, um desses pedaços de 300g é dividido em 3 pedaços, resultando no pedaço de 100g necessário para a receita. Nesse processo, 1 pedaço é utilizado para a receita e 7 pedaços de diferentes tamanhos serão guardados para uso futuro. A figura abaixo ilustra esse cenário.

Barra original



Divisões na barra original e pedaços da barra dividida
O pedaço marcado será utilizado na receita

Tarefa

Dada uma sequência de divisões realizadas por Juliana em uma barra de chocolate, determinar quantos pedaços serão armazenados em estoque para uso futuro.

Entrada

A entrada contém um único conjunto de testes, que deve ser lido do *dispositivo de entrada padrão* (normalmente o teclado). A primeira linha da entrada contém um inteiro N que indica o número de divisões feitas na barra de chocolate original ($1 \leq N \leq 1.000$). A linha seguinte contém N inteiros I ($2 \leq I \leq 10$) representando o número de pedaços em que o pedaço atual foi dividido. Sempre que é feita uma divisão, um pedaço é utilizado para a próxima divisão e os demais são separados para serem armazenados em estoque.

Saída

Seu programa deve imprimir, na *saída padrão*, uma única linha, contendo o número de pedaços de chocolate que serão armazenados em estoque.

Entrada	**Entrada**	**Entrada**
3 2 3 5	5 2 2 2 3 3	7 2 3 4 5 6 7 8
Saída	**Saída**	**Saída**
7	7	28