Desenferrujando...

2020/2021

Depois deste sobressalto com época de exames remotos, todos estaremos um pouco enferrujados, em relação à nossa programação com C.

Assim, o objetivo deste guião é ajudar a desenferrujar.

Vamos usar alguns dos problemas que saíram em exames no ano passado e este ano. Alguns alunos já conhecem os problemas, outros não. Alguns até já os resolveram no exame, mas "à pressão". Agora temos oportunidade de pensar neles com mais calma e de usar melhor as técnicas que temos aprendido.

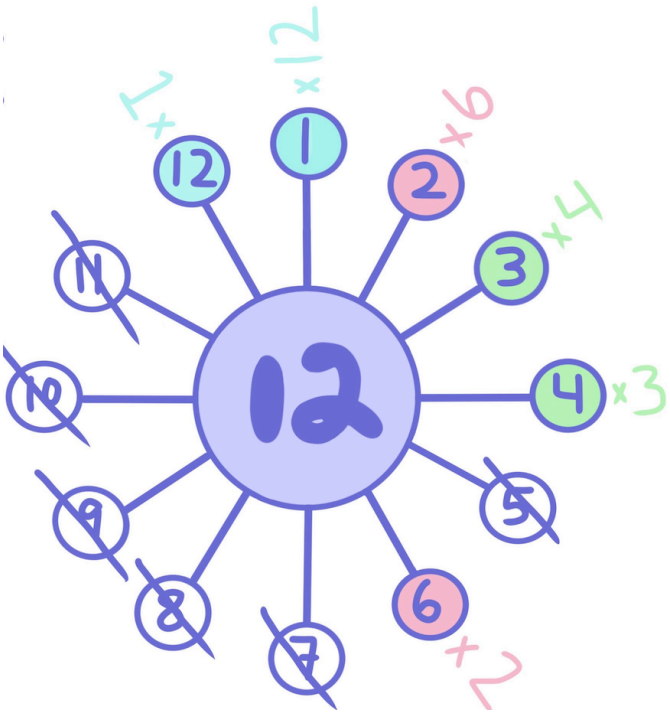
Antes de começar, prepare a sua diretoria de trabalho, para a nova cadeira, Laboratório de Programação: pasta LP_2021, e lá dentro as diretorias `sources`, `docs` e `work`, pelo menos. Copie para a diretoria `sources` as novas versões das nossas bibliotecas `our_ints.c` (e `our_ints.h`) e `our_doubles.c` (e `our_double.h`), a partir dos ficheiros na tutoria. São estes ficheiros que estão instalados no Mooshak, a partir de agora.

Estas novas versões são bastante mais extensas do que as anteriores. Dê uma vista de olhos. Há lá coisas que ainda não aprendemos, mas os mais atrevidos vão gostar de usar mesmo antes de "a matéria ter sido dada".

Na pasta `work`, crie uma diretoria para os ficheiros de teste deste guião, de nome `desenferrujando`, por exemplo. A cada um dos exemplos usados na descrição dos problemas deve corresponder um ficheiro de teste. Nomeie os ficheiros de maneira sistemática, para ser possível distingui-los rapidamente. Depois use nos seus testes, sistematicamente, compilando o programa na diretoria `sources` e correndo na diretoria `desenferrujando`, por redireção do *standard input*. Acrescente também os seus próprios casos de teste.

Note bem: esta arrumação é "obrigatória". Os professores verificarão se está a ser respeitada. Se não estiver, os programas não validam.

Maiores divisores ímpares



A tarefa é programar uma função que, dado um array de números `int`, todos positivos, calcule outro vetor em que cada elemento é o maior divisor ímpar do correspondente elemento do array dado. O maior divisor ímpar de um dado número é isso mesmo: o maior de entre os divisores desse número que são ímpares. Por exemplo, os divisores de 60 são 1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30 e 60. Destes, o maior dos ímpares é 15. Na verdade, se o número for ímpar, o maior divi-

sor ímpar é próprio número; se o número for par, é o maior divisor ímpar de metade do número.

A função de teste lerá um array a partir da consola, chama a função e escreve na consola uma linha com o array resultado.

Indicações

Trata-se de um problema em que, dado um array, vamos construir um novo array em que cada elemento resulta da transformação do correspondente elemento do array dado. É um caso muito frequente, que devemos conhecer bem. Faz-se sempre com um ciclo for que percorre o array dado e a única coisa que muda é a função de transformação.

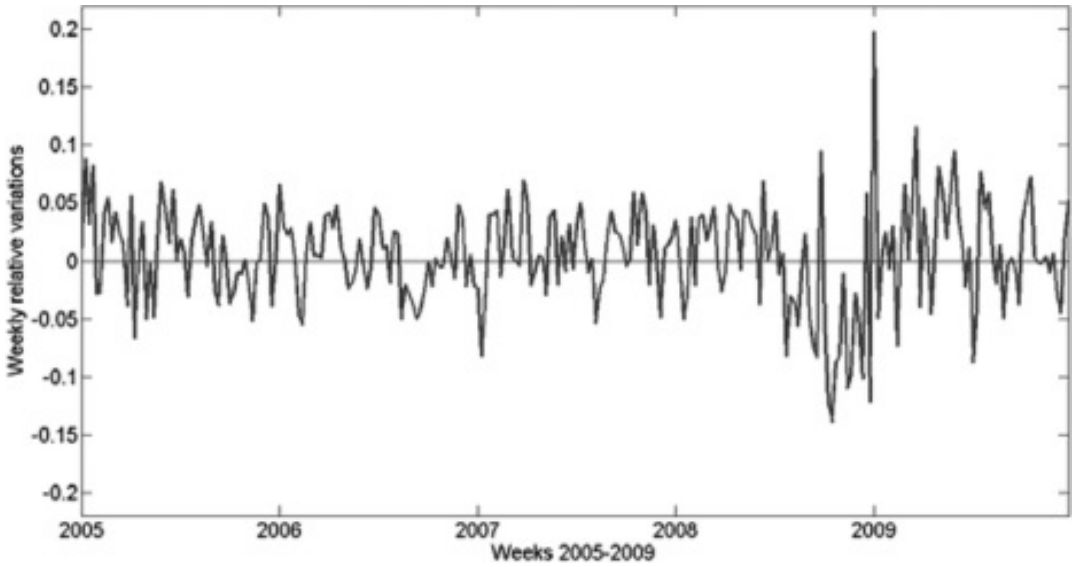
Portanto, devemos programar logo a função de transformação: neste caso a função que, dado um número inteiro positivo, calcula o maior divisor ímpar desse número.

Em geral, é boa ideia autonomizar a função de transformação, exceto nos casos em que basta usar diretamente uma expressão aritmética simples.

Programe a função que calcula o maior divisor ímpar de duas maneiras: recursiva e iterativa. Use testes unitários "puxados", para garantir que as suas funções estão bem, tanto a função que calcula o maior divisor ímpar como a que calcula o array que resulta da transformação.

Submeta no problema A.

Variações relativas



A tarefa é programar uma função que, dado um array de números `double`, todos positivos, calcule a maior variação relativa de um número em relação ao anterior. A variação relativa de *x* em relação a *y* é o quociente *(x-y)/y*.

A função de teste deve ler um array a partir da consola, chamar a função e escrever na consola uma linha com o resultado, usando a cadeia de formato "%f".

Indicações

Queremos a maior variação relativa. Por outras palavras, queremos a maior das variações relativas. Se já tivéssemos o array das variações relativas, bastaria usar uma das funções da nossa biblioteca `our_double.c`. Não temos, mas podemos construir.

Para construir o array das variações relativas devemos usar a mesma técnica da transformação de arrays. A diferença aqui é que cada elemento do array novo resulta de uma "combinação" de dois elementos adjacentes no array dado. Portanto, a função de transformação tem dois argumentos, e, claro, é a função que está expressa no enunciado por "A variação relativa de *x* em relação a *y é o quociente* (x-*y*)/*y*".

Inclua funções de teste unitário para a função da variação relativa e para a função de transformação de arrays.

Submeta no problema B.

Pingão

Os supermercados Pingão têm uma campanha permanente, que consiste em oferecer um bilhete de cinema por cada 20 euros, em compras de mais de 20 euros. Ora, eu faço quase sempre compras pequenas, que não chegam a 20 euros, portanto, poucas vezes ganho bilhetes, o que é uma pena.

Por isso, estou a considerar mudar a minha estratégia, agrupando as compras de maneira que o valor seja sempre maior ou igual a 20 euros.

Para estimar o meu lucro em bilhetes, vou analisar o histórico das minhas compras no Pingão no ano passado e calcular quantos bilhetes teria conseguido, se tivesse agrupado as compras, a partir da primeira, de maneira a perfazer, em cada caso, um preço maior ou igual a 20 euros. Por exemplo, se as minhas compras fossem <7, 11, 16, 15, 18, 17>, eu agruparia 7+11+16, 15+18, e sobrava última compra. Ganharia assim 2 bilhetes. Note que se eu tivesse agrupado 7+11+16+15 e 18+17, conseguiria ganhar 3 bilhetes, e se tivesse 7+11+16+15+18+17 ganharia 4 bilhetes, mas é claro que não posso esperar até ao fim do ano para fazer as compras todas (o que seria sempre vantajoso em número de bilhetes).

A questão é: se eu tivesse agrupado as compras da maneira elementar, isto é, até o valor ser maior ou igual a 20, quanto bilhetes teria conseguido?

Tarefa

Escreva um programa que, dado o valor de referência, isto é, o valor que dá direito a um bilhete de cinema (que seria 20, no caso da explicação anterior), e da-

dos os valores das minhas compras, por ordem cronológica, calcule o número de bilhetes que eu teria conseguido se usasse a estratégia indicada.

Input

O input é um ficheiro de texto, que será lido por redireção da consola. A primeira linha tem o valor de referência; as restantes linhas, em número indeterminado, contêm os valores das compras por ordem cronológica. Todos os valores são números inteiros positivos. O número compras é maior ou igual a zero.

Output

O output terá uma linha, com o número de bilhetes.

Exemplo 1

Input

```
20
7
11
16
15
18
17
```

Output

```
2
```

Explicação

Este é o exemplo usado na descrição do problema.

Exemplo 2

Input

```
10
7
5
12
```

```
8
25
7
1
5
8
12
5
```

Output

```
8
```

Explicação

Neste caso, o valor de referência é 10. Os agrupamentos são 7+5, 12, 8+25, 7+1+5, 8+12 e sobra 5. Isto dá 1+1+3+1+2 = 8 bilhetes.

Indicações

Neste problema temos de agrupar sucessivamente números, até que a soma atinja ou ultrapasse um valor dado.

Em casos destes, a técnica é calcular o *primeiro* grupo, e depois, usando os mesmos cálculos, calcular o primeiro grupo do *resto* do array (isto é, o primeiro grupo do subarray que começa logo após o primeiro grupo calculado anteriormente) e assim sucessivamente.

Portanto, deve haver uma função para calcular o primeiro grupo. Programe-a, por favor. Calcular o grupo é calcular o número de elementos do grupo.

A seguir, usando essa função, calcule o array dos grupos. Depois, transforme o array dos grupos no array das somas dos grupos. A seguir, calcule o array dos bilhetes, por transformação do valor do array das somas. Finalmente, calcule o número de bilhetes, somando o array dos bilhetes.

Cada uma destas operações deve ser representada por uma função. Algumas dessas funções até poderão já existir na nossa biblioteca `our_ints.c`.

Alguns alunos, mais económicos, dirão que o problema se pode resolver sem arrays nenhuns, para além dos array dados, só com uma passagem. É verdade, mas não vale a pena, porque fica muito mais complicado, com várias operações

a serem feitas em cada passo. Torna-se mais difícil perceber o que se passa, e se nos enganarmos, fica mais difícil encontrar o erro e corrigi-lo.

Ainda assim, poderíamos mudar ligeiramente a função que calcula os grupos, fazendo-a calcular o array das somas por grupo, em vez de o número de números por grupo. Com isso pouparíamos trabalho, porque para calcular os grupos, temos de somar. E, se já somámos, devia ser escusado somar de novo, na função seguinte.

Por isso, o melhor mesmo seria que a função que calcula o array dos grupos calculasse em simultâneo o array das somas. Isso sim! Em todo o caso, este é um melhoramento que poderíamos deixar para uma segunda fase do desenvolvimento.

Inclua funções de teste unitário pelo menos para os dois casos de teste indicados no texto.

Submeta no problema C.

Oceano

Os supermercados Oceano têm uma campanha permanente, que consiste em oferecer um bilhete de cinema por cada 20 euros, em compras de mais de 20 euros. Ora, eu faço quase sempre compras pequenas, que não chegam a 20 euros, e poucas vezes ganho sequer um bilhete, o que é uma pena.

Por isso, estou a considerar mudar a minha estratégia, de maneira a ter sempre compras cujo valor seja maior que 20 euros.

Para não comprar coisas de que não necessito, decidi que se a minha compra não chegar a 20 euros, vou comprar mais uma unidade, e uma unidade apenas, de um dos produtos que estão na minha lista de compras, de maneira a atingir o limite, ultrapassando-o o menos possível.

Por exemplo, se na minha lista eu tiver um produto de 2 euros, outro de 5 euros e outro de 9 euros, o valor da compra será 16 euros, o que não dá bilhete. Mas, se eu comprar mais um daqueles produtos que custam 5 euros, pagarei 21 euros, e ganharei um bilhete! Claro que se em vez desse tivesse comprado outro dos de 9 euros também ganharia um bilhete, mas pagava mais, e não quero isso.

Tarefa

Escreva um programa que, dado o valor de referência, isto é, o valor que dá direito a um bilhete de cinema (que seria 20, no caso da explicação anterior), e dados os valores dos produtos na minha lista de compras, calcule o valor do produto de que eu devo comprar mais uma unidade, para que o valor da compra atinja o valor de referência ou o ultrapasse o menos que for possível. Se não houver nenhum produto nas condições indicadas o resultado deve ser 0 (zero).

Input

O input é um ficheiro de texto, que será lido por redireção da consola. A primeira linha tem o valor de referência; as restantes linhas, em número indeterminado, contêm os valores dos produtos na minha lista de compras. Todos os valores são números inteiros positivos. O número de produtos é maior ou igual a zero. Em todos os casos de teste, a soma dos valores dos produtos é menor que o valor de referência.

Output

O output terá uma linha, com o valor do produto do qual devo comprar mais uma unidade ou zero ser não for possível.

Exemplo 1

Input

```
20
2
5
9
```

Output

```
5
```

Explicação

Este é o exemplo usado na descrição do problema.

Exemplo 2

Input

```
100
23
12
30
```

Output

```
0
```

Explicação

Neste caso, o valor de referência é 100. Mesmo que comprasse outra unidade do produto mais caro, não atingiria o valor de referência.

Exemplo 3

Input

```
60
10
8
12
22
```

Output

```
8
```

Explicação

Neste caso, a soma dos valores é 52. Basta comprar outra unidade do produto de valor 8 para atingir 60.

Indicações

Pense em termos de arrays, e não em termos de if's.

Inclua funções de teste unitário pelo menos para os três casos de teste indicados no texto.

Submeta no problema D.

Bolsa

Fiz a minha fortuna jogando na bolsa. Como me sinto generoso, decidi revelar metade da estratégia secreta com a qual enriqueci.

A arte está em comprar ações quando estão a descer e vendê-las quanto estão a subir, na altura exata.

Ora, para vender o que eu faço é o seguinte. Em cada dia, estudo a variação da cotação das ações de cada uma das companhias em que invisto, desde que tenho registos até "hoje". Se estamos "hoje" numa série de cotações ascendentes tão comprida como a maior série de cotações ascendentes alguma vez observada an-

teriormente ou mais comprida do que essa, então vendo todas as ações dessa companhia.

Não vou revelar ainda como faço para decidir comprar…

Tarefa

Escrever um programa para analisar uma sequência de números representando as cotações das ações de uma certa companhia, ao longo do tempo até "hoje" e que indique se devo vender ou não. (O último valor da sequência é o valor da cotação "hoje".)

Veja também o exemplo abaixo e a explicação.

Input

O input contém números `double` em número indeterminado, representando a sequência de cotações.

Output

O output contém uma linha, com a menção `YES` ou `NO`, consoante seja de vender ou não.

Restrições

O número de cotações é maior ou igual a 2 e menor ou igual a 1000.

Exemplo

Input

```
3.1 4.2 5.5 2.0 2.0
4.0 3.4 5.5 2.9 3.0
5.4 6.5 7.1 5.0 4.5
5.2 5.1 4.1 3.4 5.8
5.1 6.1 8.1 9.6 12.5
```

Output

```
YES
```

Explicação

Os valores aumentam nos três primeiros dias, <3.1 4.2 5.5>; depois, aumentam nos cinco dias a partir do nono dia <2.9 3.0 5.4 6.5 7.1>; no final, há outra subida de comprimento 5 terminando "hoje" <5.1 6.1 8.1 9.6 12.5>. Pelo meio há outras subidas mais pequenas. Logo, vou vender, "hoje".

"Ontem" não teria vendido.

Note que quando se diz, por exemplo, "aumentam nos cinco dias, etc." isso significa que houve quatro aumentos sucessivos: do primeiro dia para o segundo, do segundo para o terceiro, do terceiro para o quarto e do quarto para quinto.

Indicações

Pense em termos de operações sobre arrays. Algumas já estarão programadas, outras são variantes de operações que também já encontrámos.

Inclua funções de teste unitário pelo menos para os casos de teste indicados no texto e mais um ou dois.

Submeta no problema E.