

## CONCURSO PA

### Preliminares

Este é um conjunto de dez novos exercícios sobre arrays. Todos os problemas dizem respeito a arrays de int. Em cada problema é preciso programar uma função.

No enunciado dos exercícios, quando se escreve “dado um array de int”, isso significa que a função terá um argumento de tipo `int *`, para o array e um argumento de tipo `int` para o tamanho do array, como habitualmente. Implicitamente, e também tal como habitualmente, as funções que construírem arrays de saída, devolverão o tamanho do array construído.

Cada uma das funções deve vir acompanhada pela sua função de teste unitário, com pelo menos quatro casos de teste.

Todas as funções devem tratar devidamente arrays vazios, isto é, arrays cujo tamanho é zero.

As funções de teste devem escrever os arrays usando a função **`ints_println_special`**. Esta função é análoga à função **`ints_println_basic`**, diferindo dela apenas no caso dos arrays vazios: no caso de arrays vazios, a função **`ints_println_basic`** escreve uma linha em branco; a função **`ints_println_special`** escreve uma linha com um asterisco.

Nas funções de teste, todos os arrays devem ser declarados com capacidade 1000, salvo menção em contrário. Os arrays devem ser lidos até à ocorrência do terminador -1, que já não pertencerá ao array, usando a função **`ints_get_until`**.

Deverá submeter no **Concurso PI\_2223\_PA no Mooshak**.

### Compilação separada

Para estes exercícios, os professores fornecem um ficheiro chamado **`our_ints.c`** com um conjunto de funções sobre arrays de int, de uso geral: **`ints_get`**, **`ints_println_basic`**, **`ints_max`**, **`ints_sum`**, etc. Junto a este ficheiro, vem outro, chamado header file, de nome **`our_ints.h`**, que contém os cabeçalhos das funções existentes no primeiro, **`our_ints.c`**.

Estes dois ficheiros devem ser colocados na sua diretoria sources, mas fica vedado modificá-los (a não ser que o professor dê instruções explícitas nesse sentido).

A partir de agora, para usar aquelas funções no seu programa, não será preciso copiá-las. Em vez disso, adotamos a seguinte técnica:

1. Inserir no programa a diretiva **#include "our\_ints.h"**, logo à cabeça, a seguir aos outros **#include**.
2. Compilar separadamente o programa e o ficheiro **our\_ints.c** e linkar em conjunto o resultado das duas compilações, para produzir o programa executável.

Concretizando, suponhamos que o seu programa reside num ficheiro de nome **my\_arrays.c**. Então as primeiras linhas deste ficheiro serão assim, em esquema:

```
#include <stdio.h>
#include <assert.h>
#include "our_ints.h"
// ...
```

Note que enquanto os **#include** de ficheiros de biblioteca vêm entre < e >, os **#include** de ficheiros nossos vêm entre aspas. Depois, para compilar, damos o seguinte comando:

**gcc -Wall my\_arrays.c our\_ints.c**

Não havendo erros, em resultado desta compilação é criado o ficheiro **a.out** ou **a.exe**, consoante o sistema operativo.

Na verdade, não há uma compilação, mas sim duas: a do ficheiro **my\_arrays.c** e a do ficheiro **our\_ints.c**. São duas compilações separadas, isto é, não interferem uma na outra.

Se, por exemplo, o programa **my\_arrays.c** usar a função **ints\_max**, o compilador conhecerá o cabeçalho da função **ints\_max**, pois encontrou-o quando processou a diretiva **#include "our\_ints.h"**, mas não conhecerá a definição dessa função, pois essa definição está noutro ficheiro, que não no ficheiro **my\_arrays.c**. Tudo bem, é mesmo assim.

Na verdade, aquilo a que habitualmente chamamos “compilação” é um processo em dois passos:

1. Compilação propriamente dita.
2. Linkagem.

Durante a compilação propriamente dita, cada ficheiro contendo um programa C é analisado e, se não tiver erros, é traduzido para outra linguagem dita linguagem-objeto, produzindo um ficheiro, chamado ficheiro-objeto, o qual contém o programa-objeto. Por oposição, o programa escrito em C, é o programa-fonte, e reside no ficheiro-fonte. E a linguagem de programação C é a linguagem-fonte.

Depois, durante a linkagem, os vários ficheiros-objeto são linkados entre si e com as bibliotecas da linguagem, para produzir o programa executável.

Até agora só tínhamos um ficheiro-fonte de cada vez. Nos exercícios de hoje temos dois, e, em geral, de futuro teremos vários.

Podemos admitir que os ficheiros-objeto são ficheiros temporários, que não é preciso guardar. Por isso, o compilador apaga-os sistematicamente, após criar o ficheiro executável.

No entanto, é possível instruir o compilador para produzir apenas o ficheiro-objeto e não fazer mais nada, deixando-o na diretoria source, ao lado dos ficheiros- fonte. Observe:

**gcc -Wall -c our\_ints.c**

Se der este comando, notará que surge na sua pasta sources um ficheiro de nome **our\_ints.o**, em MacOS/Linux ou **out\_ints.obj**, em Windows. Esse é o ficheiro-objeto.

Aquela opção -c significa “compile só, não linke”.

Aliás, podemos observar que se o ficheiro **our\_ints.o** é o resultado da tradução do ficheiro **our\_ints.c** e este não muda, então, guardando o ficheiro **our\_ints.o**, escusamos de “cansar” o compilador a traduzir repetidamente a mesma coisa. Em vez do comando anterior, que compilava os dois ficheiros, podemos dar o seguinte comando:

**gcc -Wall my\_arrays.c our\_ints.o**

Assim, o compilador apenas traduz o nosso ficheiro **my\_arrays.c** e depois linka logo os ficheiros **my\_arrays.o** e **our\_ints.o** e as bibliotecas, sem ter compilado de novo o ficheiro **our\_ints.c**. No fim, apaga o ficheiro **my\_arrays.c** e deixa ficar o ficheiro **our\_ints.o**.

Adicionalmente, nesta fase, é boa ideia dar um nome diferente a cada programa executável, em vez de nos ficarmos sempre por **a.out**. Fazemos isso, recorrendo à opção de compilação -o.

Observe: **gcc -Wall my\_arrays.c our\_ints.o -o my\_arrays.out**

### Ficheiros fornecidos

Estão disponíveis na tutoria os ficheiros **our\_ints.h** e **our\_ints.c**. Estude-os. Use como referência o ficheiro **our\_ints\_unit\_tests.c**. Este ficheiro apenas invoca todas as funções de teste unitário presentes no ficheiro **our\_ints.c**. Experimente.

### Submissão ao Mooshak

Na submissão ao Mooshak, apenas interessa o ficheiro **my\_arrays.c**. O Mooshak tem lá os ficheiros **our\_int.c** e **our\_int.h** e compila o programa submetido da maneira que explicámos.

Isto também significa que se, por falta de atenção, você modificasse o ficheiro **our\_ints.c**, o funcionamento do seu programa no Mooshak poderia diferir do funcionamento observado no seu computador.

## Programa A – GreaterThan

Programe a função **ints\_greater\_than**, que, dado um array de **int** e um número **int**, copia para um array de saída os elementos do array dado cujo valor é maior que esse número.

A função de teste lerá primeiro o array até ao terminador e depois uma sequência de valores de comparação, até ao fim dos dados, chamando a função para o array e para cada um dos valores de comparação. Após cada chamada da função, escreve o array resultante com a função **ints\_println\_special**.

Submeta no **problema A**.

## Programa B – LessThan

Programe a função **ints\_less\_than**, análoga à anterior, mas para os valores menores que o número dado.

Submeta no **Programa B**.

## Programa C – Digits Sums

Programe a função **digits\_sums** que, dado um array de **int** onde todos os valores são não negativos, constrói outro array em que cada elemento contém a soma dos algarismos decimais do correspondente elemento do array de entrada. Por exemplo, se o array dado for <32, 9, 10000, 717, 2099>, o array de saída será <5, 9, 1, 15, 20>.

A função de teste lê um array até o terminador -1, calcula e escreve o array resultante com a função **ints\_println\_special**.

Submeta no **Programa C**.

## Programa D – Append

Programe a função **ints\_append**, que, dados dois arrays de **int**, copia para um terceiro array, primeiro os elementos do primeiro array, e depois, a seguir, os elementos do segundo.

O cabeçalho da função deve ser assim:

```
int ints_append(const int *a, int n, const int *b, int m, int *c)
```

A função de teste lerá dois arrays, primeiro um array A, depois um array B, cada um até ao respetivo terminador.

A seguir calcula “**append(A, B)**” e escreve o array resultante e depois o calcula “**append(B, A)**”, e escreve o array resultante. Ambas as escritas são realizadas com a função **ints\_println\_special**.

**Submeta no Programa D.**

## Programa E – Take

Programe a função **ints\_take**, que, dado um array de **int** e um número X de tipo **int**, copia para um array de saída os X primeiros elementos do array dado ou todos, se X for maior que o tamanho do array dado. Se X for negativo, o array de saída será vazio.

A função de teste lê primeiro o array até ao terminador -1 e depois uma sequência de valores de X. Para cada um destes valores, chama a função **ints\_take**, sempre para o mesmo array, escrevendo o array resultante com a função **ints\_println\_special**.

**Submeta no Programa E.**

## Programa F – Drop

Programe também a função **ints\_drop**, análoga à anterior, que copia todos os elementos exceto os X primeiros ou não copia nenhuns se X for maior que o tamanho. Neste caso, se X for negativo, o array de saída será igual ao array dado.

**Submeta no problema F.**

## Programa G – Ascending

Programe a função **ints\_ascending**, que, dado um array de **int**, copia para um array de saída os elementos do array dado cujo valor é maior ou igual que o valor de cada um dos elementos à sua esquerda. Por exemplo, se o array de entrada for <3,1,8,2,4,5,7,9,5,9,4,15,13,11>, então o array de saída será <3,8,9,9,15>.

A função de teste lê os valores para o array até ao terminador -1, calcula e escreve o array resultante com a função **ints\_println\_special**.

**Submeta no problema G.**

## Programa H – Accumulate

Programe a função **ints\_accumulate**, que, dado um array de **int**, regista num array de saída as somas parciais do array de entrada. Mais exatamente, no array de saída o valor do elemento na posição K é soma dos K primeiros elementos do array de entrada. Por exemplo, se o array de entrada for <5,1,4,1,3,4>, então o array de saída será <0,5,6,10,11,14,18>.

Note bem: o tamanho do array de saída é igual ao tamanho do array de entrada mais 1.

A função de teste lê os valores para o array até terminador -1, calcula e escreve o array resultante com a função **ints\_println\_special**.

**Submeta no problema H.**

## Programa I – Unaccumulate

Programe a função **ints\_unaccumulate**, que, dado um array de **int** não vazio, realiza a operação inversa de **ints\_accumulate**.

Note bem: o tamanho do array de saída é igual ao tamanho do array de entrada menos 1.

A função de teste lê os valores para o array até ao terminador -1, calcula e escreve o array resultante com a função **ints\_println\_special**.

**Submeta no problema I.**

## Programa J – Find Triple

Programe a função **ints\_find\_triple**, que, dado um array de **int** devolva a posição do primeiro elemento do primeiro triplo de elementos consecutivos iguais no array ou -1 se não existirem no array triplos de elementos consecutivos iguais. Por exemplo, se o array for <6,2,8,5,9,9,9,1,9,2,2,2> a função devolverá 4; se for <1,6,3,8,8,3,8,2> a função devolverá -1.

Sugestão: use um ciclo for da forma: for (int i = 0; i < n-2; i++)

A função de teste lê os valores para o array até ao fim dos dados, calcula e escreve uma linha com o resultado.

**Submeta no problema J.**