# Mais alguns exercícios com arrays

2018/2019


## Preliminares

Este é um conjunto de mais dez 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`", quer-se significar 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`.

# Compilação separada

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

Estes dois ficheiros devem ser colocados na sua diretoria `sources`, mas, salvo ordem em contrário, fica vedado modificá-los.

A partir de agora, para usar aquelas funções no seu programa, não vai ser 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 ao 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:

```
// Santo Isidoro, ajuda-me com estes arrays malucos.
#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`.

Não 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 seu 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 uma outra linguagem dita *linguagem-objeto*, produzindo um ficheiro, chamado *ficheiro-objeto*.

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 C 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, depois de criar o ficheiro executável.

No entanto, é possível instruir o compilador para produzir apenas o ficheiro-objeto e não fazer mais nada. 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, de cada vez que damos o comando

```
gcc -Wall my_arrays.c our_ints.o
```

É mais sensato substituir o ficheiro `our_ints.c` por `our_ints.o` neste comando, pois assim o compilador apenas traduz o nosso ficheiro `my_array.c` e depois linka logo os ficheiros `my_array.o` e `our_ints.o` e as bibliotecas, sem ter compilado de novo o ficheiro `our_ints.c`.

Por outro lado, neste fase, é boa ideia dar um nome diferente a cada 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 página os ficheiros `our_ints.c`, `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.

# Novo estilo da função main

Programe todas as funções destes exercícios no mesmo ficheiro. Para poder manter todas as funções de teste operacionais, use o esquema que o professor tem usado nas aulas teóricas, assim:

```
int main(int argc, char **argv)
{
  unit_tests();
```

```
  int x = 'A';
  if (argc > 1)
    x = *argv[1];
  if (x == 'A')
    test_ints_greater_than();
  else if (x == 'B')
    test_ints_less_than();
  else if (x == 'C')
    test_decimal_weights();
  // ...
  else if (x == 'U')
    printf("All unit tests PASSED.\n");
  return 0;
}
```

# Submissão ao Mooshak

Para efeitos de 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 ser diferente do funcionamento observado no seu computador.

# 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 resultantes com a função `ints_println_special`.

Submeta no problema A.

# LessThan

Programe a função `ints_less_than`, análoga à anterior, mas para os valores menores que o número dado.

Submeta no problema B.

# 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 problema C.

# 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 feitas com a função `ints_println_special`.

Submeta no problema D.

# 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 valor de **X**. Para cada um destes valores, chama a função `ints_take`, sempre para o mesmo array, escrevendo o array resultante com a com a função `ints_println_special`.

Submeta no problema E.

# 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.

# 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.

# 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.

# 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.

# 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.