Guião da Ficha Trabalho 3

Introdução aos *Arrays*

Arrays encontram-se entre as estruturas de dados mais antigas e importantes em programação.

Em C, arrays são uma fonte muito comum de erros, falhas de segurança e de transtorno emocional. Isto deve-se ao facto de estarem intimamente relacionados com o conceito de apontador de memória, que tende a gerar alguma confusão. Um array corresponde a um conjunto de elementos, que se encontram armazenados num espaço continuo em memória.

Por exemplo, o array:

int arr[5] = {10, 2, 13, 64, 7};

Encontra-se representado em memória da seguinte forma:

Address	Value	Index
0x3bf4fbfd28	10	0
0x3bf4fbfd2c	2	1
0x3bf4fbfd30	13	2
0x3bf4fbfd34	64	3
0x3bf4fbfd38	7	4

Como tal, é possível aceder a um elemento de um array de tamanho N pelo seu indice, ou seja, a sua posição no *array*. Este indice começa em 0 e termina em N-1.

Qualquer variável declarada como *array* é na verdade **um apontador para o primeiro elemento do array**. Ou seja, a seguinte expressão é verdadeira (onde & é o operador referência, ou seja "endereço de":

```
arr == &arr[0]
```

Isto também permite o uso de **aritmética de apontadores**, como por exemplo (onde * é o operador de desreferência):

```
arr[2] == *(arr+2);
```

Isto mostra-se também no uso idiomático de **char* para representar strings**, que não são mais do que arrays de caracteres, e como tal, equivalentes a um apontador para o endereço de memória onde se situa o primeiro caracter da string.

```
char *hello = "hello, world";
```
É equivalente a

```
char hello[] = { 'h', 'e', 'l', 'l', 'o', ',', ' ', 'w', 'o', 'r', 'l', 'd' };
```

Este acesso directo à memória encontra-se mais presente em linguagens de mais baixo nível como C, ao contrário de linguagens como Java ou C#, onde a gestão de memória é em grande parte feita automaticamente.

Este acesso directo à memória confere uma grande vantagem de eficiencia a linguagens da família de C, mas com grande poder vem grande responsabilidade, que neste caso se manifesta como riscos de *segmentation fault, stack overflow*, *heap corruption,* entre outros.

Exercícios em sala de aula:

a) Pedir ao utilizador para preencher um array tamanho N e imprimi-lo em ordem inversa.

```
insert array size (<256): 8
insert element of index 0: 5
insert element of index 1: 2
insert element of index 2: 8
insert element of index 3: 4
insert element of index 4: 3
insert element of index 5: 9
insert element of index 6: 1
insert element of index 7: 2
reverse of array is:
2 1 9 3 4 8 2 5
```

b) Encontrar o máximo e o mínimo do array inserido:

```
insert array size (<256): 6
insert element of index 0: 2
insert element of index 1: 5
insert element of index 2: 76
insert element of index 3: 23
insert element of index 4: 12
insert element of index 5: 67
min: 2, max: 76
```

c) Preencher e mostrar um array de tamanho N com valores aleatórios entre uma certa gama

C possui um gerador de numeros pseudo-aleatorios acessível pela função

```
int rand(void);
```

Como indicado na assinatura, esta função não aceita argumentos e devolve um valor inteiro.

Este valor encontra-se entre 0 e a constante RAND_MAX. Definida em <limits.h>

Começe por usar esta função para criar outra função que devolve um inteiro entre um mínimo e um máximo:

```
int rand_between(int min, int max) {
    return rand() % (max – min) + min;
}
```

Dica: O resto da divisão (operador módulo) % devolve sempre um valor inferior ao seu argumento;

X % N < N

```
insert array size (<256): 20
insert range (min max): 0 20
1 7 14 0 9 4 18 18 2 4 5 5 1 7 1 11 15 2 7 16
```

d) Fazer a média deste array

```
insert array size (<256): 20
insert range (min max): 0 20
1 7 14 0 9 4 18 18 2 4 5 5 1 7 1 11 15 2 7 16
average: 7.350000
```

e) Determinar os elementos repetidos

```
insert array size (<256): 20
insert range (min max): 0 20
1 7 14 0 9 4 18 18 2 4 5 5 1 7 1 11 15 2 7 16
average: 7.350000
1 has 2 duplicates
7 has 2 duplicates
4 has 1 duplicate
18 has 1 duplicate
2 has 1 duplicate
5 has 1 duplicate
1 has 1 duplicate
7 has 1 duplicate
```

Tarefa única (para avaliação na aula seguinte (1.5 valores)):

Aceitar uma frase e desenhar o histograma das frequências das letras.

Neste exercício pretende-se que o utilizador insira uma frase e seja desenhado um gráfico que represente o número de vezes que cada letra aparece.

Recomenda-se o uso da função **fgets**, ao invés de **scanf** devido ao facto de scanf parar a leitura ao encontrar espaços, o que não se pretende.

A frase *The quick brown fox jumps over the lazy dog* é notória por conter pelo menos uma vez todas as letras do alfabeto: