FUNÇÕES

Arrays unidimensionais (vetores): Passagem para funções:

Em C, **dentro de uma função, não é possível saber com quantos elementos foi declarado um vetor que foi passado como parâmetro. Isso não tem qualquer importância, pois não interessa a dimensão do vetor que é passado a uma função, mas sim qual o tipo dos seus elementos.**

Pelo que, se tivermos dois (2) vetores de dimensão diferente não é necessário escrever duas funções diferentes que processem os seus elementos.

Suponhamos que queríamos escrever 2 vetores, com 10 e 20 elementos, respetivamente, e cujo valor seria 0 (zero).

Em vez de escrever duas funções distintas …

funções		Programa principal

void inic1 (int s[10])
{ int i;
for (i=0; i<10; i++)
s [i] = 0;
}
	void inic2 (int s[20])
	{ int i;
	for (i=0; i<20; i++)
	s [i] = 0;
	}
		…
		main()
		{ int s1 [10] , s2 [20];
		inic1 (s1);
		inic2 (s2);
		}

… podemos fazer com que a função receba um vetor de um determinado tipo (sem indicar o nº de elementos [1]) e uma variável do tipo dos elementos do vetor que irá indicar o nº de elementos a utilizar.

Assim fazemos …

```
void inic (int s[ ] , int n)
{ int i;
  for (i=0; i<n; i++)
     s [ i ] = 0;
}
```

```
main()
{ int s1 [ 10 ] , s2 [ 20 ];
 inic ( s1, 10 );
 inic ( s2, 20) ;
}
```

[1]Isto porque o nome de um vetor é igual ao endereço do seu 1.º elemento (o seu menor endereço), isto é, se V for um vetor, **V** corresponde a **&V[0]**, logo podemos omitir a sua dimensão ([ ]), na escrita do código, no parâmetro.

EXERCÍCIO:

13. Implementa a função **float Max (float v[ ], int n**), que recebe um vetor de reais e o nº de elementos a considerar. Devolve o maior nº entre os elementos do vetor.

Arrays multidimensionais: Passagem para funções:

A passagem de arrays (vetores) com mais do que uma dimensão para uma função é realizada indicando no cabeçalho destes, obrigatoriamente, o nº de elementos de cada uma das **n-1** dimensões à direita. Apenas a dimensão mais à esquerda pode ser omitida, colocando apenas **[ ]** ou um ***** .

É, no entanto, habitual colocar todas as dimensões dos vetores.

Exemplos:

1- void inic (char s[ ] [DIM]) { … } / * omitir uma dimensão */

2 - int v [3] [2] ;

Possíveis cabeçalhos de função que receberiam como parâmetro o vetor v atrás declarado.

g (int v[3] [2]) { … }

g (int v [ ] [2]) { … }

g (int *v [2]) { … } */ a usar na UFCD 0810*/

EXERCÍCIO:

14. Implementa a função **void Ler (int v[5] [2])** que procede à leitura dos elementos de um vetor bidimensional, em que o mesmo é passado como parâmetro.

Passagem de strings para funções

A passagem de strings para funções é exatamente igual à passagem de vetores para funções, uma vez que qualquer string é sempre um vetor de carateres.

Todas as strings devem ser passadas por referência para funções, uma função não pode devolver dados do tipo string, mas pode devolver o endereço de uma string (char *).

Algumas das funções de manipulação de strings da biblioteca <string.h>		
Função	**Finalidade**	**Exemplo**
int strlen (char *s) ou int strlen (char s[ ])	Devolve o comprimento de uma string (sem contar com o '\0')	srtlen ("") -> 0 strlen("aulas") ->5
char *strcpy(char *dest, char *orig) ou char *strcpy(char dest[ ], char orig[ ])	Copia a string **orig** para a string **dest**	strcpy(dest,"abc"); -> "abc"
char *strcat (char *s1, char *s2)	Concatena duas strings, colocando a string s2 imediatamente a seguir ao final da string s1	srtcat ("alfa", "beto") -> "alfabeto"
int isnull (char *s)	Verifica se uma string contém ou não algum caráter	isnull ("") -> TRUE isnull ("aulas") -> FALSE
int strcmp(char *s1, char *s2)	Compara as strings s1 e s2 alfabeticamente. Devolve um inteiro: <0 se s1 é alfab. Menor que s2; 0 se s1 é alfab. = a s2; >0 se s1 é alfab. Maior que s2 Útil para ordenar strings alfabeticamente.	strcmp("abc", "abxpo") Res: <0 .pois 'c'<'x' strcmp("Oláo", "Oláa") Res: >0 , pois 'o'>'a' strcmp("Olá", "Olá") Res: 0. pois são iguais
int strcountc (char *s, char ch)	Devolve o nº de ocorrências do caráter **ch** na string **s**	strcountc("abacate", 'a') -> 3
char *strrev (char *s)	Inverte uma string	strrev("olá") -> "álo"
int strcountd (char *s)	Devolve o nº de dígitos na string **s**	strcountd("12abacate") -> 2
char *strdelc (char *s, char ch)	Apaga todas as ocorrências do caráter ch em s	srtdelc("abacate", "a") ->"bcte"

EXERCÍCIOS:

15. Implementa a função **char *mygets (char *s)** que lê uma string do teclado e a coloca no parâmetro da função (isto é, implementa a função gets). A função deverá ainda devolver a string lida.

16. Escreve um programa que teste a função **x_isdigit** que verifica que um caráter é dígito ou não um dígito está compreendido entre os carateres '0' e '9').

DESAFIOS

- Cria um programa que permita ler uma lista de strings, ordená-la alfabeticamente e, então, imprimir a lista ordenada.
- Cria um programa que use uma função que simule a função **strdelc.**

PASSAGEM POR VALOR e PASSAGEM POR REFERÊNCIA

Os **parâmetros** de uma função podem ser passados **valor** e por **referência**.

PASSAGEM POR VALOR: neste caso os parâmetros funcionam como portas de entrada de valores a utilizar na função. **Todas as alterações efetuadas nos subprogramas** (funções e procedimentos) **não são efetivas nas variáveis declaradas no programa principal**. Na realidade, o que é enviado para a função são cópias dos valores de que esta necessita.

Exemplo: Passagem dos parâmetros a e b por valor

```
#include<stdio.h>
void troca (int a, int b);        /*protótipo da função*/
main()
{
   int n,k;
   puts(" Introduza 2 nºs inteiros"); scanf("%d%d", &n, &k);
   printf("Antes da troca  n = %d e k = %d\n", n, k);
   troca (n, k);
   printf("Depois da troca n = %d e k = %d\n", n, k);
}
void troca (int a, int b)
{
    int aux;
    aux = a;
    a = b;
    b = aux;
} …
```

Output:

```
Introduza 2 nºs inteiros
2 7
Antes da troca n = 2  k  = 7
Depois da troca n = 2  k  = 7
```

PASSAGEM POR REFERÊNCIA: Os parâmetros funcionam como portas de entrada e saída de valores a utilizar na função. **Todas as alterações efetuadas nos subprogramas** (funções e procedimentos) **são efetivas em todo o programa**. O que é enviado para a função não é uma cópia do valor da variável, mas sim a própria variável ou uma referência a esta.

- A passagem por valor permite utilizar, dentro de uma função, o valor de uma variável ou expressão. O valor da variável que é invocado nunca é alterado.
- A passagem por referência permite alterar o conteúdo das variáveis de invocação.

- **Em C só existe passagem por valor**, embora a passagem por referência seja possível utilizando apontadores (a lecionar posteriormente).

EXERCÍCIOS:

17. Escreve uma função chamada MEDIA que retorne a média de 3 valores reais (X, Y e Z) passados como parâmetros.

18. Escreve um programa que, para um número indeterminado de alunos, faça para cada um deles:

- Ler as 3 notas do aluno (a leitura do valor 0 indica o fim dos dados);
- Calcule a média do aluno (usando a função MEDIA) e exiba a média do aluno.

19. Considera a seguinte função:

```
void MEDIA1(float a, float b)
{
      a =(a+b)/2;
      printf ("Valor de a no sub : %.2f ", a);
}
```

a) Tendo em conta a função anterior escreve o output correspondente ao código seguinte:

```
…
  float a=1.0, b=2.0;
  MEDIA1(a,b);
  printf("Valor de a no programa : %.2f", a);
…
```

DESAFIO

Implementa a função **Exp** que calcula o valor da seguinte expressão, sendo todos os intervenientes do tipo inteiro.

$$\sum_{i=1}^{n} \left(a + \frac{n}{i} \right)^2$$

20. Dadas as funções **Ping** e **Pong**

```
void Ping (int i)
{
   switch (i)
   {
      case 1:
      case 2:
      case 3: while (i - - ){
                    printf(" \n%d", - - i);}
                 break;
     case 25: Pong (3);
                  break;
     default: printf("\nJá passei a C");
                 Pong(123);
   }
}
```

```
void Pong (int i)
{
  int j=0;
   switch (i)
   {
      case 1:
      case 2: Ping(i);
      case 3: j=5;
                 j+ + ;
                 return;
      default: printf("\nOlá");
                 return;
    }
     printf("\nVou sair");
}
```

Qual o output das seguintes chamadas:

a) Pong (3);

b) Ping (-4);

c) Ping (25);

d) Pong (2);

e) Pong (1).

21. Escreve uma função, chamada **Situacao**, que determine se um aluno está *Aprovado* (média>=9.5) ou *Reprovado* (caso contrário), sendo a média fornecida como parâmetro.

Altera o Exercício 18 de modo a acrescentar ao programa a função *Situacao*.

22. Escreve uma função que determine se um nº inteiro dado é ou não primo e imprima o resultado.

Nota: Um nº é primo se tiver apenas 2 divisores o 1 e ele próprio.

RECURSIVIDADE

A recursividade é a capacidade que uma linguagem tem de permitir que uma função possa invocar-se a ela própria.

É uma forma de implementar um ciclo através de chamadas sucessivas à mesma função.

RECURSIVIDADE DIRETA – quando uma função se invoca a ela mesma no seu corpo da função.

RECURSIVIDADE INDIRETA – Quando uma função **f** invoca uma outra função **g** que, por sua vez, volta a invocar a função **f**.

REGRAS PARA A ESCRITA DE FUNÇÕES RECURSIVAS

1º – A 1ª instrução de uma função recursiva deve ser a implementação do critério de paragem, isto é, qual a condição ou condições que se devem verificar para que a função pare se se invocar a ela própria.

2º – Só depois de escrito o critério de paragem é que se deverá escrever a chamada recursiva da função, sempre relativa a um subconjunto (por exemplo fatorial de **N**, a sua chamada recursiva é realizada com **N-1**)

Notas: A recursividade é uma técnica de desenvolvimento particularmente útil na implementação de alguns algoritmos de pesquisa, poupando tempo aos programadores. Isto pela menor quantidade de código que é escrita e pela maior legibilidade. Perde-se, no entanto, em termos de performance pois uma função recursiva é, em geral, mais lenta que a sua correspondente iterativa.

Exemplos:

1- Implementa a função fatorial que calcula o valor de:

n! = n *(n-1) * (n-2) * … * 2 * 1 , sabendo que **0! = 1**

Sem recursividade (f)	**Com recursividade (f)** Critério de paragem: n!=0	**Rastreio para n=3** (com recursividade)

int fact (int n)
{
int i, res = 1;
if (n!= 0)
for (i= 1; i<=n ; i++)
res*= i;
else
res=1;
return res;
}
	int fact (int n)
	{
	if (n!= 0)
	return n*fact(n-1);
	else
	return 1;
	}
		fact(3) = 3 * fact(2)
		fact(2)= 2 * fact(1)
		fact(1)=1 * fact(0)
		fact(0)=1
		
		isto é…
		
		fact(3)=3 * 2 * 1 * 1 = 6

Programa completo

#include <stdio.h>
(f)
main()
{
int n;
while (1)
{
printf(“Fatorial de ?”);
scanf(“%d”, &n);
printf(“\n %d ! = %d \n”, n, fact (n));
}

2 - Implementa, de forma recursiva, as funções UP e DOWN que escrevem no ecrã os primeiros números inteiros por ordem crescente e decrescente, respetivamente.
Nota que ambas as funções têm 1 único parâmetro e não devolvem qualquer valor.

UP 1, 2,… n-2, n-1, n **(f)** Critério de paragem: quando n <1	DOWN n, n-1, n-2, … 2, 1 **(g)** Critério de paragem: quando n <1

void UP (int n)
{
if (n<1)
return;
UP(n-1);
printf("%d, ", n);
}
	void DOWN (int n)
	{
	if (n<1)
	return;
	printf("%d, ", n);
	down(n-1);
	}

Programa completo

```
#include <stdio.h>
(f)
(g)
main()
{
   int n;
   printf(" quantos nºs a escrever ?"); scanf("%d", &n);
   fflush(stdin);
   printf("\nOrdem decrescente\n");
   DOWN(n);
   printf("\nOrdem crescente\n");
   UP(n);
   getchar();
}
```

EXERCÍCIO:

Implementa de forma recursiva as seguintes funções:

23. **pot** que devolve o valor de x^n ($x^0 = 1.0$, $x^n = x*x*\ldots*x$ (n vezes))

DESAFIO

Implementa a função VAL, definida por:

$$VAL = \frac{x}{(1+t)} + \frac{x}{(1+t)^2} + \frac{x}{(1+t)^3} + \ldots + \frac{x}{(1+t)^n}$$