Os parâmetros de uma função, como já foi referido, 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 n, int k);        /*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. Esta é a estratégia a adotar para conseguir alterar o valor de uma variável dentro de uma função:

    - Não podemos passar à função o valor que queremos alterar, mas podemos passar o seu endereço usando o operador **&;**
    - O endereço da variável é recebido dentro da função por uma variável do tipo apontador para o tipo dessa variável;
    - Dentro da função podemos alterar os locais para os quais os apontadores apontam e que correspondem às variáveis originais

Exemplo: Passagem dos parâmetros a e b por referência

```
#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);     /* invocação utilizando o endereço das var. */
   printf("Depois da troca n = %d e k = %d\n", n, k);
}
 void troca (int *a, int *b)/* recebe como parâmetros 2 apontadores*/
{
    int aux;
    aux = *a;          /* aux recebe o apontado por a (2) */
    *a = *b;          /* coloca na casa nº x o armazenado na casa nº y */
   *b = aux;        /* coloca em k o valor armazenado em aux */
}
```

Output:

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

Passagem de vetores para funções

Sempre que invocamos uma função, e lhe passamos um vetor como parâmetro, esta na realidade não recebe o vetor na sua totalidade, mas apenas o endereço inicial do vetor. Nesse caso a variável que recebe esse endereço terá que ser um apontador para o tipo de elementos do vetor em causa (ver página 2).
Por esta razão é que no cabeçalho de uma função que recebe um vetor poderá aparecer um apontador a receber o respetivo parâmetro:

```
void inic( int *s, int n)   /*  cabeçalho equivalente a (int s[ ] , int n) */
{ int i;
  for (i=1; i<n; i++)
     s [ i ] = 0;
}
```

Os vetores são sempre passados às funções sem usar o & pois, como foi dito, o nome de um vetor é por si só um endereço (o do 1º elemento do vetor), apenas as variáveis que não sejam vetores e que tenham que ser alteradas dentro das funções é que têm que ser precedidas de &.

Exemplo: Implementa um programa que imprima o maior e o menor elemento de um vetor inicializado com 9 elementos, recorrendo a uma função para determinar o maior e o menor dos elementos do vetor.

```
# include <stdio.h>
void calc ( float *v, int num, float *xmin, float *xmax)
{
  int i ;
  *xmin= *xmax=*v ;  /* ficam com o valor de v [ 0 ] */

  for( i =1; i< num; i++)

   {
     if (v[ i ] < *xmin)
         *xmin = v [ i ] ;
      if ( v [ i ] > *xmax )
         *xmax = v [ i ] ;
    }
}
main()
{   float v [ ] = {10, 20, 30 ,41, 34, 5, -5, 50, -30} ; float m, n ;
     calc (v, 9, &m, &n) ;
     printf(" Maior: %.1f\n Menor: %.1f\n", m,n);
}
```

EXERCÍCIOS

1. Indica os vários outputs, tendo em conta a seguinte declaração:

```
Int a [ ] = {0,1,2,3,4}, i, *p ;
```

a)
```
for (i=0; i<4; i++)
    printf ("%d   ", a [ i ] ) ;
```

b)
```
for (p = &a [ 0 ]; p < &a [ 4 ] ;p++)
    printf ("%d   ", *p) ;
```

c)
```
for (p = &a [ 0 ], i=1; i<5 ;i++)
    printf ("%d   ", a [ i ]) ;
```

d)
```
for (p = a , i=0; p+i<=a+4 ; p++, i++)
    printf ("%d   ", * (p + i)) ;
```

e)
```
for (p = a +4; p>=a ; p--)
    printf ("%d   ", * p ) ;
```

2. Considera o seguinte programa:

```
include <stdio.h>
main()
{
 int i,  v[ ]={92, 81, 70, 66, 51};
for (i=0; i<3; i++)
   printf("%d\n",v[i]);
}
```

Implementa o mesmo programa, mas agora recorrendo a ponteiros para v.

3. Escreve um programa que leia um vetor de 5 elementos inteiros e que recorra a uma função para determinar o maior elemento do vetor e que o imprima no ecrã (usa apontadores para funções para simular a passagem por referência).

4. Sabendo que Tab[ ] é um vetor de inteiros, quais as expressões que referenciam o valor do 3.º elemento do vetor? Tab[2] ou *(Tab+2)?

5. O que faz o programa?

```
include <stdio.h>
main()
{
 int M[ ]={1,2,3}, j,  *p;
p=M;
for (j=0; j<3; j++)
{
        printf("%d\n",*p);
        p++;
  }
}
```

6. Implementa um programa que leia e imprima um vetor de strings recorrendo a apontadores.

Passagem de estruturas para funções

A passagem de estruturas para funções faz-se indicando no parâmetro o tipo associado à estrutura (*ou typedef*).

Exemplo: Escreve uma função que permita escrever no ecrã os valores existentes numa estrutura recebida como argumento.

```
#include <stdio.h>
typedef struct {int dia, mes, ano;} DATA ;
typedef struct PESSOA
{
   char nome[50];
   int idade;
   DATA nasc;
} P ;
void MOSTRAR (struct PESSOA x )  /* ou …(P x)  */
{
        printf("Nome        : %s \n", x.nome);
        printf("Idade : %d \n", x.idade);
        printf("Data de nascimento        :    %d/%d/%d    \n",    x.nasc.dia,    x.nasc.mes,
x.nasc.ano);
}
main()
{
 struct PESSOA  ps={"Carlos", 23, {25, 1,1970}} ; /* ou  P ps={"Carlos", 23, {25, 1,1970}} */
 MOSTRAR( ps);
}
```

Nota: a passagem de parâmetros é feita por valor, pois é colocada em x uma cópia da variável que lhe é enviada. Desta forma não podemos alterar a variável enviada como argumento à função, a menos que lhe enviemos o endereço da variável, tal como fazíamos com as variáveis simples.
Para alterar os valores presentes na estrutura (passagem de parâmetros por referência), teremos que passar a esta função o endereço da estrutura que queremos ler.

Exemplo:

```
#include <stdio.h>
typedef struct {int dia, mes, ano;} DATA ;
typedef struct PESSOA
{
   char nome[50];
   int idade;
   DATA nasc;
} P ;
/* Carrega a estrutura passada como parâmetro */
void LER (P *x)
{
        printf("Nome          :"); gets((*x) . nome);
        printf("\n Idade       :");scanf("%d", &(*x) . idade);
        printf("\nData de nascimento      : "); scanf("%d %d %d  ", &(*x) . nasc . dia,
                                                    &(*x) . nasc . mes, &(*x) . nasc . ano);
}

/* Mostra a estrutura passada como parâmetro */

void MOSTRAR (P x)
{
        printf("Nome          : %s \n", x.nome);
        printf("Idade : %d \n", x.idade);
        printf("Data de nascimento      :   %d/%d/%d   \n",   x.nasc.dia,   x.nasc.mes,
x.nasc.ano);
}

main()
{
  P ps;
 LER (&ps);
 puts("\n");
 MOSTRAR( ps);
}
```

Notas:

Verifica-se que o operador Ponto (.) tem maior precedência que o operador apontador (*). Pelo que é necessário colocar um parêntesis de forma a aceder aos campos da estrutura, em que ***x** é um apontador para uma estrutura e **(*x).nome** permite o acesso ao campo nome da estrutura para o qual x aponta.

Em C existe o operador **->** (sinal menos seguido do sinal de maior) que permite simplificar a expressão (*x) . nome.

Assim se x for um apontador para uma estrutura e nome um campo apontado por x, então as expressões **(*x) . nome** e **x ->nome** são equivalentes.

7- O exercício mostra um exemplo de função que retorna um apontador. A função acha retorna o ponteiro pnome.
Qual o output do programa, sabendo que foi inserida a string "Rui Unas"?

```
#include <stdio.h>
char * ache (char nome [ ])
{
char * pnome ;
int i = 0;
while ( nome [i] != ' ')
{
i++;
}
i++;
pnome = & nome [i];
return pnome ;
}
int main (void)
{
char nomeCompleto [80];
puts (" Entre com o seu nome e um apelido.");
gets ( nomeCompleto );
puts (ache ( nomeCompleto ));
}
```

8- Comenta o programa seguinte indicando o seu output:

```
#include < stdio .h>
int main (void)
{
int i = 10, j = 20;
int temp ;
int *p1 , *p2;     /*                                                    */
p1 = &i;           /*                                                    */
p2 = &j;           /*                                                    */
temp = *p1;        /*                                                    */
*p1 = *p2;         /*                                                    */
*p2 = temp ;       /*                                                    */
printf ("%d %d\n", i, j);          /* output:                             */

}
```

9- Escreve um programa que leia e imprima no ecrã uma strings, com no máximo 20 carateres, recorrendo a apontadores.