

# Apontadores

**Programação I**  
**2021.2022**

*Teresa Gonçalves*  
[tcg@uevora.pt](mailto:tcg@uevora.pt)

Departamento de Informática, ECT-UÉ

# Reminder...



# Sumário

**Apontadores**

**Vetores e matrizes**

**Variáveis dinâmicas**

**Listas**

# Apontadores

# Variáveis: r-value e l-value

```
int k; k=2;
```

## R-value

Valor da variável

valor 2

## L-value

Endereço de memória onde a variável está armazenada

indicado por **&k**

# Apontador

## Variável que guarda um endereço de memória

### Exemplo

```
int *ptr
```

ptr é um apontador (indicado pelo \*) para uma variável de tipo int

ptr é uma variável que guarda um endereço de memória onde está um inteiro

# Exemplo

```
int k; int *ptr;
```

## Atribuição

```
ptr = &k;
```

&k : endereço de memória onde está a variável k

## Referência

```
printf( "%d\n", *ptr );
```

\*ptr : valor guardado na posição de memória referida pela variável ptr

# Operadores

**&**

operador unário que permite obter o endereço de memória de uma variável

**\***

operador unário que permite obter o valor apontado por um endereço (posição de memória)



# Utilização

```
int valor, outroValor; /* declara valor e outroValor */
int *valor_ptr;        /* declara apontador para um int */

valor                  /* um inteiro, por exemplo 4 */
&valor                 /* endereço/apontador para um inteiro */
valor_ptr              /* apontador para um inteiro */
*valor_ptr             /* um inteiro */
```

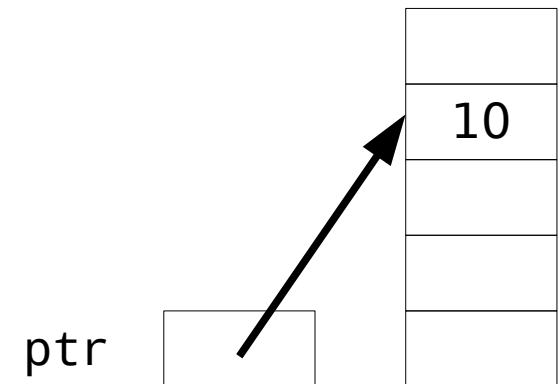
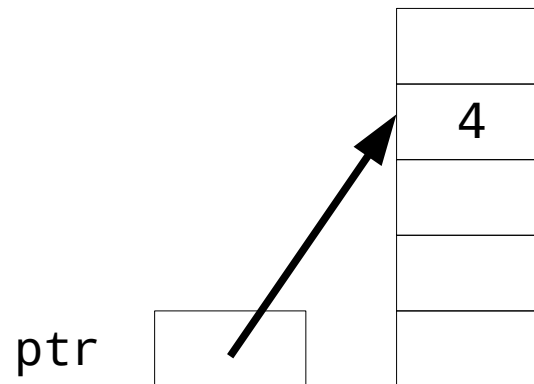
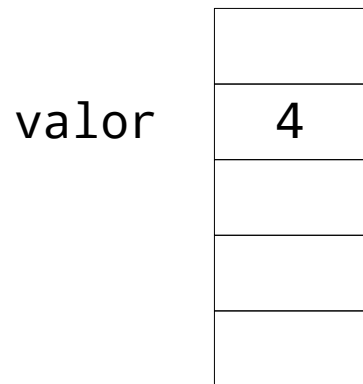
# Exemplo

...

```
valor = 4;
```

```
ptr = &valor; /* aponta para valor */
```

```
*ptr = 10;    /* atribui 10 a valor */
```



# Outro exemplo

...

```
int x = 1, y = 2;
```

```
int *ip;
```

```
ip = &x;
```

```
y = *ip;
```

```
x = (int) ip;
```

```
ip = 3; /* atencao */
```

...

	x	y	ip
endereço	0x1000	0x1004	0x1008
	1	2	?
			0x1000
		1	
	0x1000		
			3

*Está a apontar para o endereço de memória 3 (0x0003)*

# Inicialização e tipos

## Inicialização

Um apontador deve ser inicializado com valor **NULL**  
(não aponta para nada)

## Tipos

```
int *ip;      /* apontador para inteiro */  
char *cp;     /* apontador para char */
```

# Operações sobre apontadores

## É possível incrementar/decrementar um apontador

fá-lo avançar/recuar para a variável seguinte ou anterior (na memória)

Útil para manipulação de vetores/matrizes

## Exemplos

```
int *ip;  
int i=5, j=6;  
ip = &i;  
ip++; ip--;  
ip = ip + 2; ip = ip - 3;
```

# Exercício

Ler um número inteiro e escrever os valores dos seus bytes constituintes recorrendo a apontadores

```
#include <stdio.h>
main(){
    int i;
    int *pi;
    char *pc;
    printf("Introduza um número inteiro: ");
    scanf("%d", &i);
    pi = &i;
    pc = (char *) pi;
    /* um int ocupa 4 bytes, um char ocupa um byte */
    /* primeiro é guardado o byte menos significativo */
    printf("%d:%d:%d:%d\n", *(pc+3), *(pc+2), *(pc+1), *pc);
}
```

# sizeof()

**Função que indica o tamanho de uma variável/tipo**

**Tamanho de tipo**

```
sizeof(int);
```

**Tamanho de variáveis**

```
int i;
```

`sizeof(i)` → espaço ocupado pela variável i (inteiro)

```
char c;
```

`sizeof(c)` → espaço ocupado pela variável c (char)

```
int *ip;
```

`sizeof(ip)` → espaço ocupado pela variável ip (apontador)

```
char *cp;
```

`sizeof(cp)` → espaço ocupado pela variável cp (apontador)

**`sizeof(ip)` é igual a `sizeof(cp)`** ←

*O espaço que um apontador ocupa não depende do tipo de dados para onde aponta!*

# Erros comuns

## Não é possível atribuir um endereço a uma variável que não é apontador

```
int k;  
&k = 0x1000;
```

*Erro de compilação*

## Um apontador tem de ser sempre inicializado antes de ser utilizado

```
#include <stdio.h>  
main(){  
    int k;  
    int *p;  
    *p = 10;  
    printf("%d\n", *p);  
}
```

*Compilação sem erros. Em execução (pode) aparecer a mensagem de erro "Segmentation fault (core dumped)"*

Como **p** não tem um valor conhecido, não se sabe onde será colocado o valor 10!



# Vetores e matrizes

# Vetores e apontadores

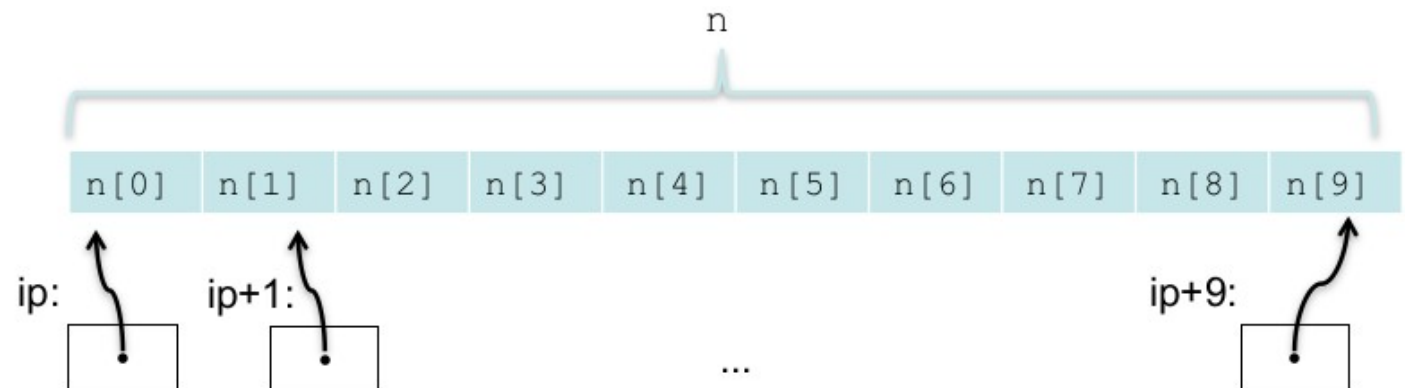
## Vetor

Os elementos do vetor são armazenados em posições contíguas da memória. Primeiro elemento está na posição 0 (zero) do vetor.

**Identificador de um vetor é um apontador para o seu primeiro elemento**

## Exemplo

```
int n[10], *ip;  
ip = n;
```



# Exemplo

```
#include <stdio.h>

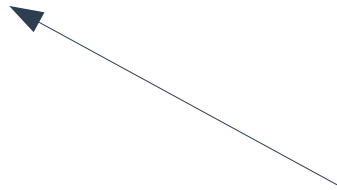
main(){
    float x[4] = {2.5, -1.0, -7.3, 5.9};
    float *px;
    px = x;
    printf("%.1f\n", *px );           /* escreve 2.5 */
    px++;                             /* px aponta para x[1] */
    printf("%.1f\n", *(px + 2));      /* escreve 5.9 */
    printf("%.1f\n", *(x + 2));       /* escreve -7.3 */
    px = x + 3;                       /* px aponta para x[3] */
    printf("%.1f\n", *(px - 2));      /* escreve -1.0 */
}
```

# Vetores e funções

## Exemplo

```
float somatorio( float v[], int n )
```

```
float somatorio( float *v , int n )
```



*São exatamente a mesma coisa!*

# Variáveis dinâmicas

# Variáveis

## Variáveis

Dimensão fixa

Ocupam uma área de memória invariável ao longo do programa

É o compilador que “cria” as variáveis e faz a gestão de memória

## Exemplo

```
int valor;          /* ocupa espaço correspondente a um inteiro */  
int vetor[10];      /* ocupa espaço correspondente a 10 inteiros */
```

# Alocação de memória

## Reserva de espaço de memória

### **void \*malloc( int size )**

Devolve um apontador para um espaço de memória para gestão pelo programador

### Exemplo

```
#include <stdio.h>
#include <stdlib.h>
main(){
    int *p;
    p = (int *) malloc(sizeof(int));
    *p = 10;
    printf( "%d\n", *p );
}
```

```
#include <stdio.h>
main(){
    int *p;
    int n;
    p = &n;
    *p = 10;
    printf( "%d\n", *p );
}
```

# malloc()

## Devolve NULL quando não há espaço disponível

É sempre necessário testar o resultado antes de utilizar

### Exemplo

```
char *p = (char *) malloc( 15*sizeof(char) );  
if(p == NULL)  
    printf("não tem espaço\n");  
else  
    strcpy(p, "Hello, world!");
```



# Libertação de memória

## **void \* free(void \* p)**

liberta o espaço de memória apontado pelo apontador p

## **Exemplo**

```
#include <stdio.h>
#include <stdlib.h>
main(){
    int *p;
    p = (int *) malloc(sizeof(int)); /* aloca espaço */
    *p = 10;
    printf("%d\n", *p);
    free(p);                        /* liberta espaço */
}
```

# Gestão de memória

**No fim do programa, a memória pedida ao sistema operativo é automaticamente devolvida**

## Atenção

Um espaço de memória reservado durante a execução de uma função não é libertado quando se retorna ao programa principal: o apontador deixa de existir, mas o apontado, não!

Como deixa de haver apontador, deixa de ser possível libertar o apontado!

## Boa prática

Libertar a memória quando já não for necessária

```
free(p);  
p = NULL;
```

# Outras funções

**`void *realloc( void *p, int new_size )`**

retorna um apontador para a realocação de um espaço de memória previamente alocado

**`void *calloc( int nelements, int size )`**

retorna um apontador para um espaço de memória que permite armazenar nelements de tamanho size

## Atenção

malloc() não inicializa a memória alocada, enquanto que calloc() inicializa a memória alocada a ZERO

`p = calloc(m, n);` é equivalente a

`p = malloc(m * n);  
for(i=0; i<m; i++) *(p+i)=0;`