

# Estruturas de Dados

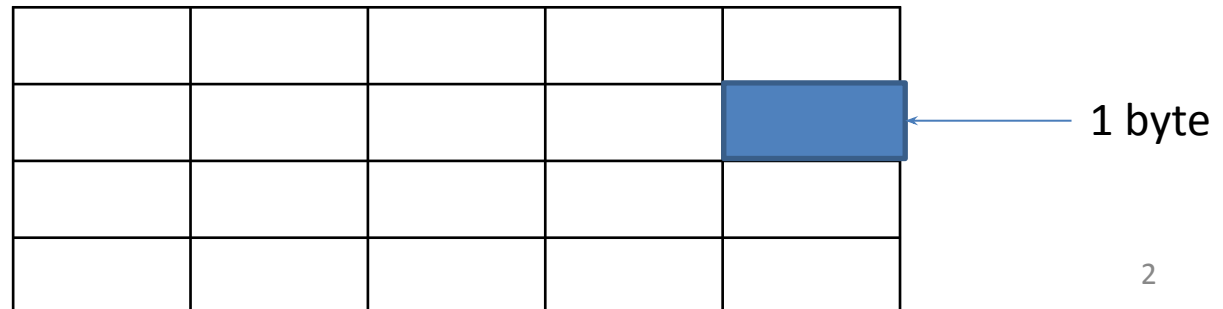
---

## Ponteiros

# Armazenamento de informações

- Toda variável utilizada por um programa ocupa espaço na memória do computador;
- Pode-se dizer que a memória é dividida em posições, que são identificadas por endereços únicos;
- As variáveis são, então, associadas aos endereços das posições de memória que ocupam;
- Através desse endereço podemos recuperar dados que estão armazenados no computador.

memória

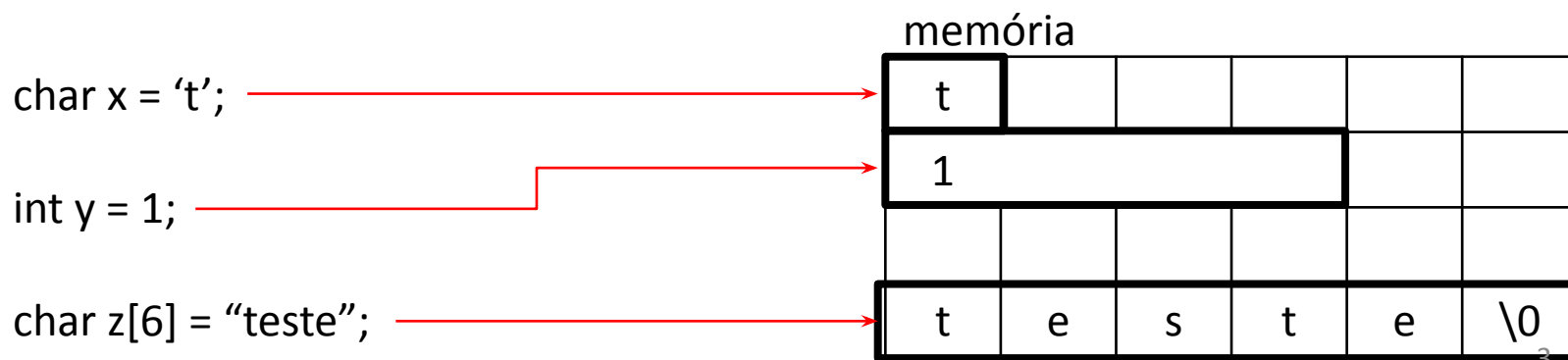




# Alocação de memória

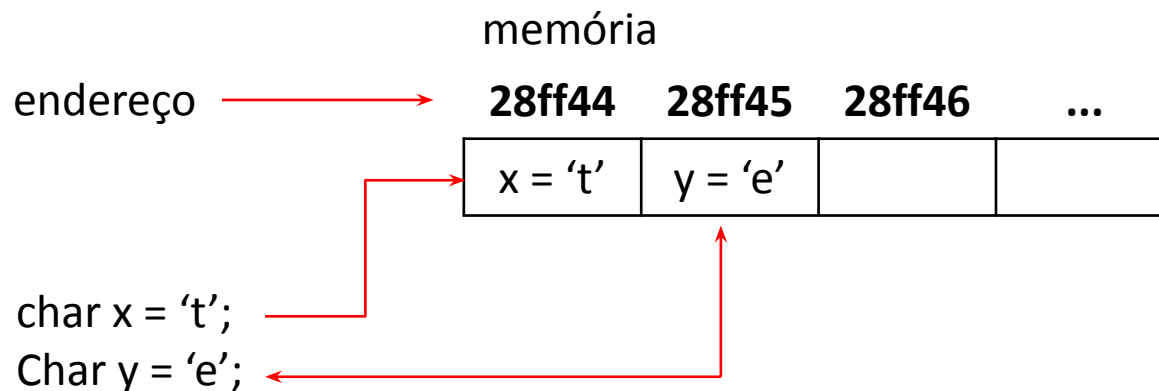
- Cada tipo de dado ocupa um espaço, de tamanho específico, na memória do computador.

Tipo de dado	Tamanho
char	1 byte
int	4 bytes
float	4 bytes
double	8 bytes



# Endereço de memória

- Sempre que uma variável é declarada, ela é associada a um *endereço de memória*.





# Ponteiros

- Os **endereços** de memória utilizados por variáveis podem ser armazenados em PONTEIROS.
- Ponteiro é um tipo de **variável que armazena apenas endereço de memória** de uma outra variável.
- O valor do ponteiro indica em que posição da memória a variável está alocada.
- Sua função é *apontar* para um endereço de memória.



# Por que usar ponteiros?

- Funções que precisam retornar mais de um valor (passagem por referência);
- Alocação dinâmica de memória;
- Estruturas de dados complexas: listas encadeadas e árvores.

memória

**&2000**

**&2001**

**&2002**

**&2003**

**&2004**

x = 't'	y = 'e'			
---------	---------	--	--	--

Variável x = 't';

Endereço da variável x = 2000.



# Declaração de ponteiros

- Sintaxe:

***tipo*** \*nome\_do\_ponteiro;

Onde:

***tipo***: tipo de dado do ponteiro.

***\*nome\_do\_ponteiro***: nome da variável precedido de um asterisco, indicando que ela é um ponteiro.

- Exemplo:

int \*idade;

float \*ponteiro;

char \*sexo;

\* O asterisco indica que a variável armazena um endereço de memória para o tipo indicado, e **não um valor!!!**



# Manipulação de ponteiros

- Ponteiros podem ser manipulados de duas formas:
  - Por meio do endereço de uma variável;
  - Por meio do conteúdo de um endereço apontado pelo ponteiro.
- Operadores:
  - &: retorna o endereço de memória que está sendo manipulado.
  - \*: retorna o conteúdo do endereço (valor armazenado naquele local).





# Operador de endereço (&)

- Permite saber qual endereço de memória está sendo usado por uma variável.
- Pode ser impresso na tela através da função *printf*, usando o operador de conversão *%x* (ou *%p*).
- Exemplo:

```
11 int main()
12 {
13     char a = 's';
14     printf("conteudo de c: %c \n", a);
15     printf("endereco de c: %p \n", &a); // ou %x
16     return 0;
17 }
```

**Valor da variável != valor do  
endereço de memória**

```
conteudo de a: s
endereço de a: 22ff47
```

# Inicialização de ponteiros

- Como mover o endereço de memória de uma variável para um ponteiro?
  - Através do operador de endereços (&)



```
10
28ff44
28ff44
Pressione q
```

- Exemplo:

1. Declara uma variável **ponteiro** - *pldade*
2. Atribui o **endereço** de memória usado por *idade* para a variável *pldade*
3. Imprime o **valor** armazenado na variável *idade*
4. Imprime o **endereço** de *idade* através de &.
5. Imprime o **conteúdo** armazenado em *pldade* (que refere-se a um endereço de memória)

```
int main()
{
    int idade = 10;
    1 int *pIdade;
    2 pIdade = &idade;

    3 printf("%d \n", idade);
    4 printf("%x \n", &idade);
    5 printf("%x \n", pIdade);

    return 0;
}
```



# Operador de conteúdo (\*)

- Também conhecido como *operador de referência*;
- Usado na declaração de um ponteiro:  
`int *p;`
- Usado para manipular o conteúdo armazenado no endereço para o qual o ponteiro aponta;
- Um ponteiro armazena um endereço de memória. O operador de conteúdo permite acessar e alterar o valor armazenado neste endereço.



# Operador de conteúdo

- Exemplo:

```
int main()
{
    int idade = 10;
    int *pIdade;
    pIdade = &idade;

    printf("%d \n", idade);
    printf("%x \n", &idade);
    printf("%x \n", pIdade);
    printf("%d \n", *pIdade);
}
```

Qual valor será impresso?

Resposta: 10.



# Exercício 1

- Escreva um programa que contenha uma variável **num** para receber um valor inteiro.
- Crie um ponteiro que receba o endereço de memória dessa variável.
- Através do operador de conteúdo do ponteiro, altere o valor de **num** para 1 e apresente o resultado na tela.



# Exercício 1: solução

- Escreva um programa que contenha uma variável **num** para receber um valor inteiro.
- Crie um ponteiro que receba o endereço de memória dessa variável.
- Através do operador de conteúdo do ponteiro, altere o valor de **num** para 1 e apresente o resultado na tela.

```
1 #include<stdio.h>
2 #include<conio.h>
3
4 int main() {
5     int num, *p;
6     printf(" Informe um numero inteiro: ");
7     scanf("%d", &num);
8     p = &num;
9     *p = 1;
10    printf("\n Novo valor: %d", num);
11    return 0;
12 }
```



## Exercício 2

- Escreva um programa que contenha duas variáveis (inteira e real), atribua um valor a elas e crie dois ponteiros, cada um apontando para uma das variáveis.
- Mostrar na tela:
  - O valor das variáveis
  - O endereço de memória das variáveis
  - O valor dos ponteiros
  - O endereço de memória dos ponteiros
  - O valor apontado pelos ponteiros



## Exercício 2: solução

```
1 #include<stdio.h>
2 #include<conio.h>
3
4 int main(){
5     int a, *p1;
6     float b, *p2;
7     a = 11;
8     b = 22.55;
9     p1 = &a;
10    p2 = &b;
11    printf("\n Valor de a:      %d", a);
12    printf("\n Endereco de a:   %p", &a);
13    printf("\n Valor de p1:      %p", p1);
14    printf("\n Endereco de p1: %p", &p1);
15    printf("\n Valor apontado por p1: %d", *p1);
16    printf("\n");
17    printf("\n Valor de b:      %.2f", b);
18    printf("\n Endereco de b:   %d", &b);
19    printf("\n Valor de p2:      %d", p2);
20    printf("\n Endereco de p2: %d", &p2);
21    printf("\n Valor apontado por p2: %.2f", *p2);
22
23    printf("");
24    return 0;
25 }
```





# Operações com ponteiros



# Operações com ponteiros

- Atribuição de ponteiros:
  - Como qualquer variável, um ponteiro pode ser usado no lado direito de uma instrução, para atribuir o seu valor para outro ponteiro.
- Exemplo:

```
#include<stdio.h>
int main ()
{
    int x, *p1, *p2;
    x = 123;
    p1 = &x;
    p2 = p1;
    printf("Endereço da variável: %x", p2); // imprime o endereço de x
    printf("Valor da variável: %d", *p2); // imprime o valor de x
    return 0;
}
```



# Operações com ponteiros

- É possível realizar operações com adição, subtração, incremento, decremento sobre ponteiros.
- Exemplo: Para um ponteiro p do tipo inteiro:

`p++;`      `/* O ponteiro passa a apontar para o endereço de memória do próximo elemento do tipo inteiro */`

```
1  #include<stdio.h>
2
3  int main()
4  {
5      int a, *p;
6      a = 11;
7      p = &a;
8      printf("Valor de p: %x \n", p);
9      p++;
10     printf("Proxima posicao: %x \n", p);
11     return 0;
12 }
```

C:\Users\Leandro\Desktop\C³/4digos em C\Ponteiros1.exe

```
Valor de p: 9ffe44
Proxima posicao: 9ffe48
Pressione qualquer tecla para continuar. . .
```

**Note que o incremento não ocorre byte a byte!**



# Operações com ponteiros

- Adição:  
 $p = p + 2;$  // faz com que o ponteiro aponte para **dois elementos**  
// **além** da variável para a qual o ponteiro está apontando.
- Subtração  
 $p = p - 2;$  // faz com que o ponteiro aponte para **dois elementos**  
// **antes** da variável para a qual ponteiro está apontando.
- Incremento  
 $p++;$  // faz com que o ponteiro aponte para o **próximo**  
// **elemento** da variável para a qual ele aponta atualmente.
- Decremento  
 $p--;$  // faz com que o ponteiro aponte para o **elemento anterior**  
// da variável para a qual ele aponta atualmente.

**Exemplo:** suponha o ponteiro **p** armazene o endereço de memória (&90) de uma variável inteira.  
Logo:  **$p = p + 2$**  equivale a:  $(\&90 + 2 * 4 \text{ bytes}) = \&98$



## Exercício 3

- Crie uma variável do tipo inteiro e atribua um valor qualquer;
- Crie um ponteiro que aponte para esta variável (receba seu endereço de memória);
- Utilizando ponteiros, imprima na tela o valor contido na variável;
- Imprima o conteúdo das próximas 20 posições inteiras na memória.



## Exercício 3 - solução

```
1 #include<stdio.h>
2 #include<conio.h>
3
4 int main() {
5     int i, a, *p;
6     a = 11;
7     p = &a;
8
9     for (i = 0; i < 20; i++) {
10         printf("\n Valor de p1 (endereço): %d", p);
11         printf("\n Conteúdo do endereço p1: %d ", *p);
12         printf("\n");
13         p++;
14     }
15     return 0;
16 }
```



# Operações com ponteiros

- ==
  - Verifica se os ponteiros possuem o mesmo endereço
- !=
  - Verifica se os ponteiros possuem endereços diferentes
- >, <, >=, <=
  - Verifica qual endereço aponta para a posição mais alta na memória



## Exercício 4

- Escreva um programa que contenha duas variáveis inteiras com dois valores quaisquer. Crie dois ponteiros que apontam para essas variáveis.
- Compare os dois ponteiros e mostre na tela qual o ponteiro está mais “adiante” na memória.





## Exercício 4 - solução

```
1 #include<stdio.h>
2 #include<conio.h>
3
4 int main(){
5     int a, b,*p1, *p2;
6     a = 11;
7     p1 = &a;
8     b = 22;
9     p2 = &b;
10
11     if (p1 > p2){
12         printf("\n p1 esta mais adiante");
13         printf("\n Endereço de a: %d", p1);
14         printf("\n Endereço de b: %d", p2);
15     }
16     else{
17         printf("\n p2 esta mais adiante");
18         printf("\n Endereço de a: %d", p1);
19         printf("\n Endereço de b: %d", p2);
20     }
21     return 0;
22 }
```