Ponteiros

Algoritmos e Programação 2 – Ciências da Computação – UFJ

Professora: Ana Paula Freitas Vilela Boaventura

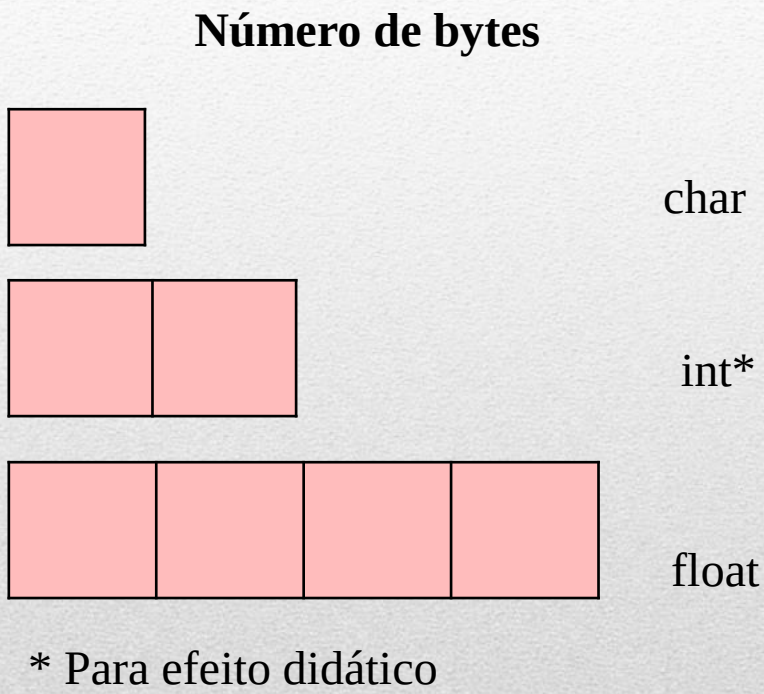
Memória: Local em que os dados são armazenados;
Programação: Memória RAM;
Assim, os **dados** a serem manipulados na execução dos **programas** ficam também na **memória RAM**;

Alocação de memória

Alocação de memória

A **menor parcela** de **informação** processada por um computador é o **bit**;

1 Byte = 8 bits e alocamos as **variáveis** em termos de **Bytes**;



* Para efeito didático

Como ver quantos *bytes* uma variável ocupa?

```
int main()
{ int numI;
   char carac;
   float numF;
   double numD;

    printf("Caractere: %d\n", sizeof(carac));
    printf("Inteiro: %d\n", sizeof(numI));
    printf("Real: %d\n", sizeof(numF));
    printf("Double: %d\n", sizeof(numD));
}
```

```
int main()
{ int numI;
   char carac;
   float numF;
   double numD;

    printf("Caractere: %d\n", sizeof(carac));
    printf("Inteiro: %d\n", sizeof(numI));
    printf("Real: %d\n", sizeof(numF));
    printf("Double: %d\n", sizeof(numD));
}
```

1 Byte

4 Bytes

4 Bytes

8 Bytes

```
int main()
{ int vetor[100];

    printf("Caractere: %d\n", sizeof(vetor));

}
```

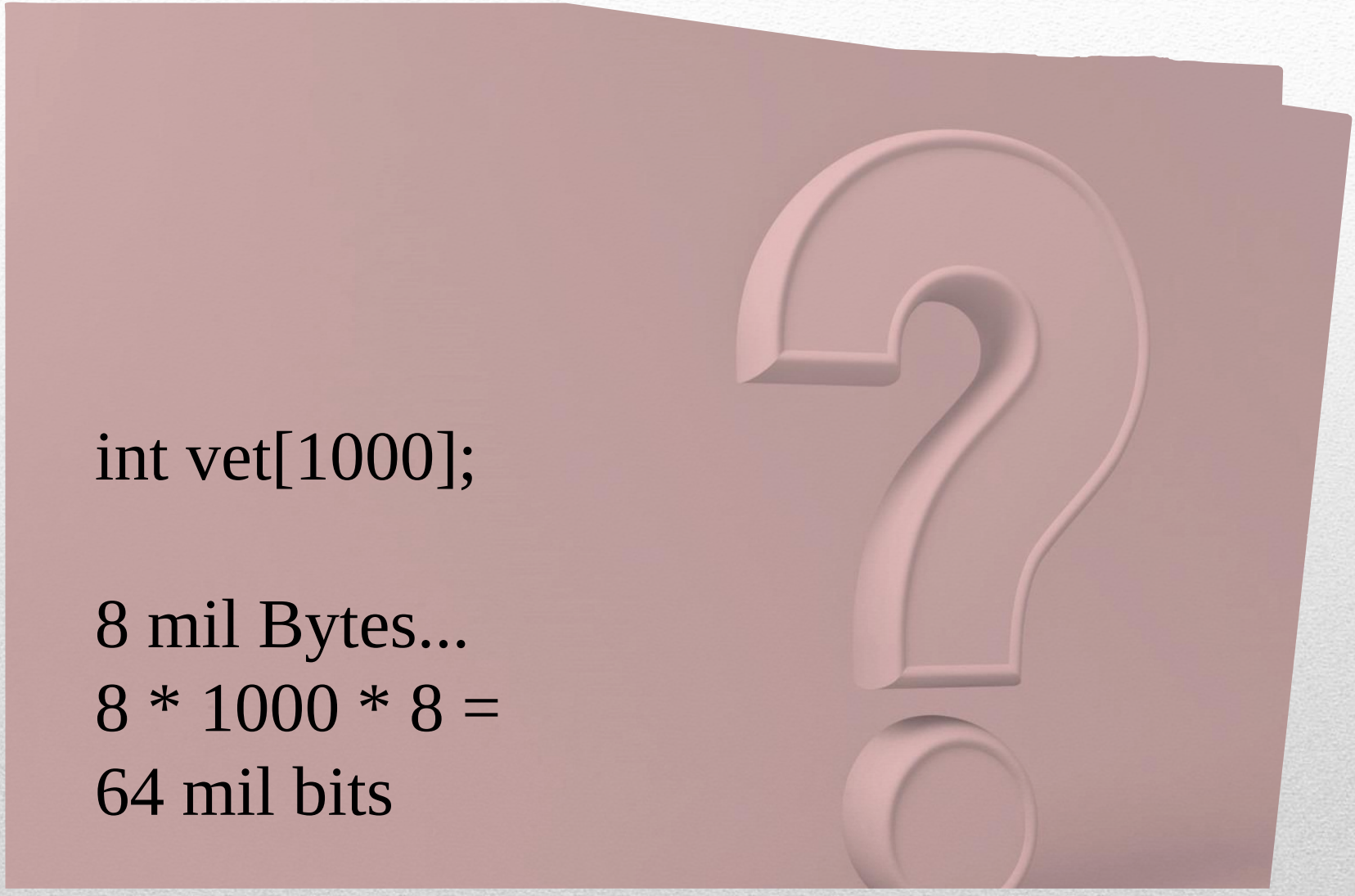
Variáveis compostas?

```
int main()
{ int vetor[100];

    printf("Caractere: %d\n", sizeof(vetor));

}
```

400 Bytes

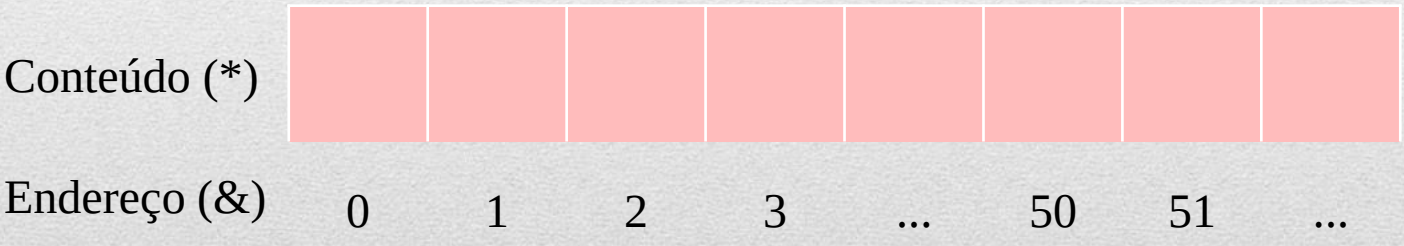


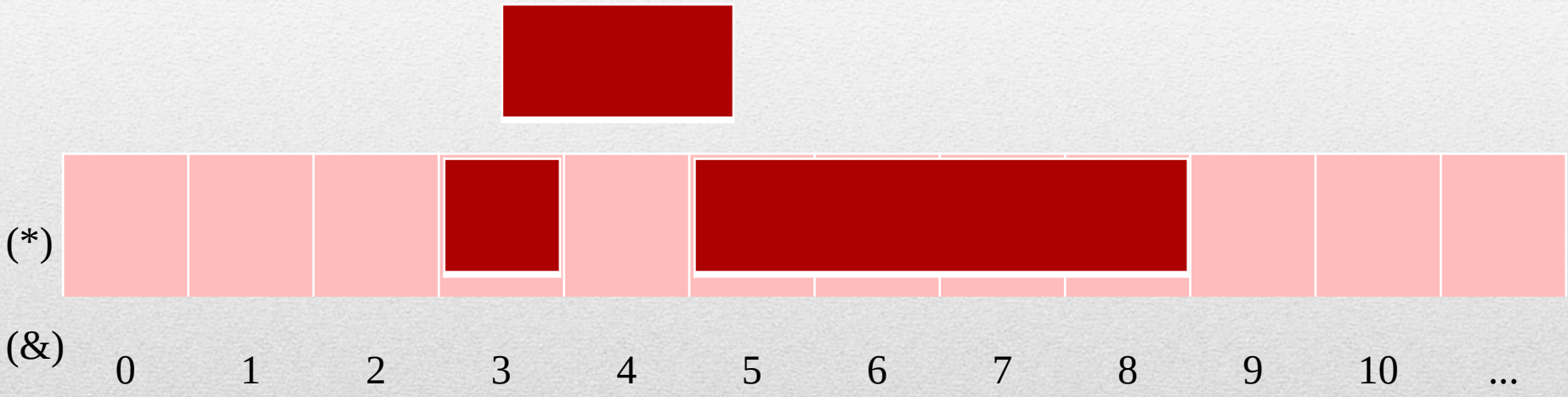
int vet[1000];

8 mil Bytes...
8 * 1000 * 8 =
64 mil bits

Alocação de memória

Cada variável ocupa uma posição na memória;
Posição é identificada por um número único que varia entre 0 e a totalidade de bytes que ocupam a memória RAM do computador.

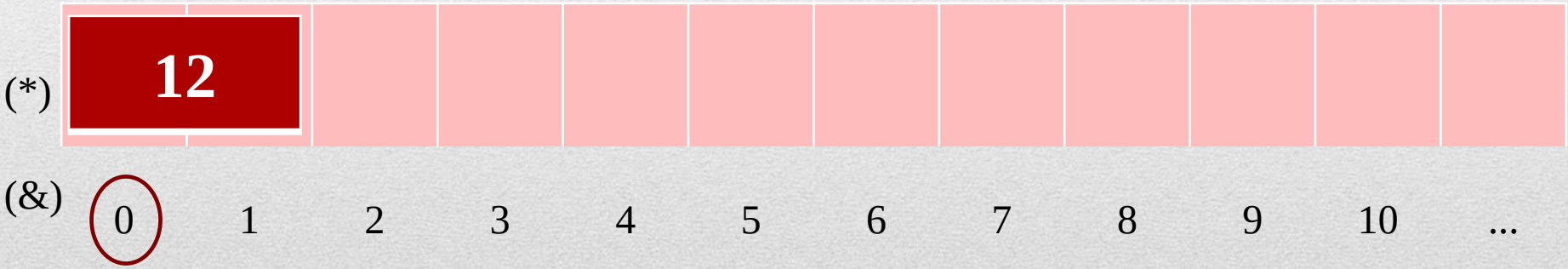




Alocação de memória

int a=12

a	&
12	0



Alocação de memória

Ponteiros

- **Variáveis estáticas** em C: *char*, *int*, *float*, *double*, *struct*;
- **Ponteiro** é um tipo de variável que **armazena o endereço de outra variável**, que por sua vez, é um número;
- Se uma variável do tipo **ponteiro** armazena um endereço (número), logo ele também **ocupa um local na memória**;

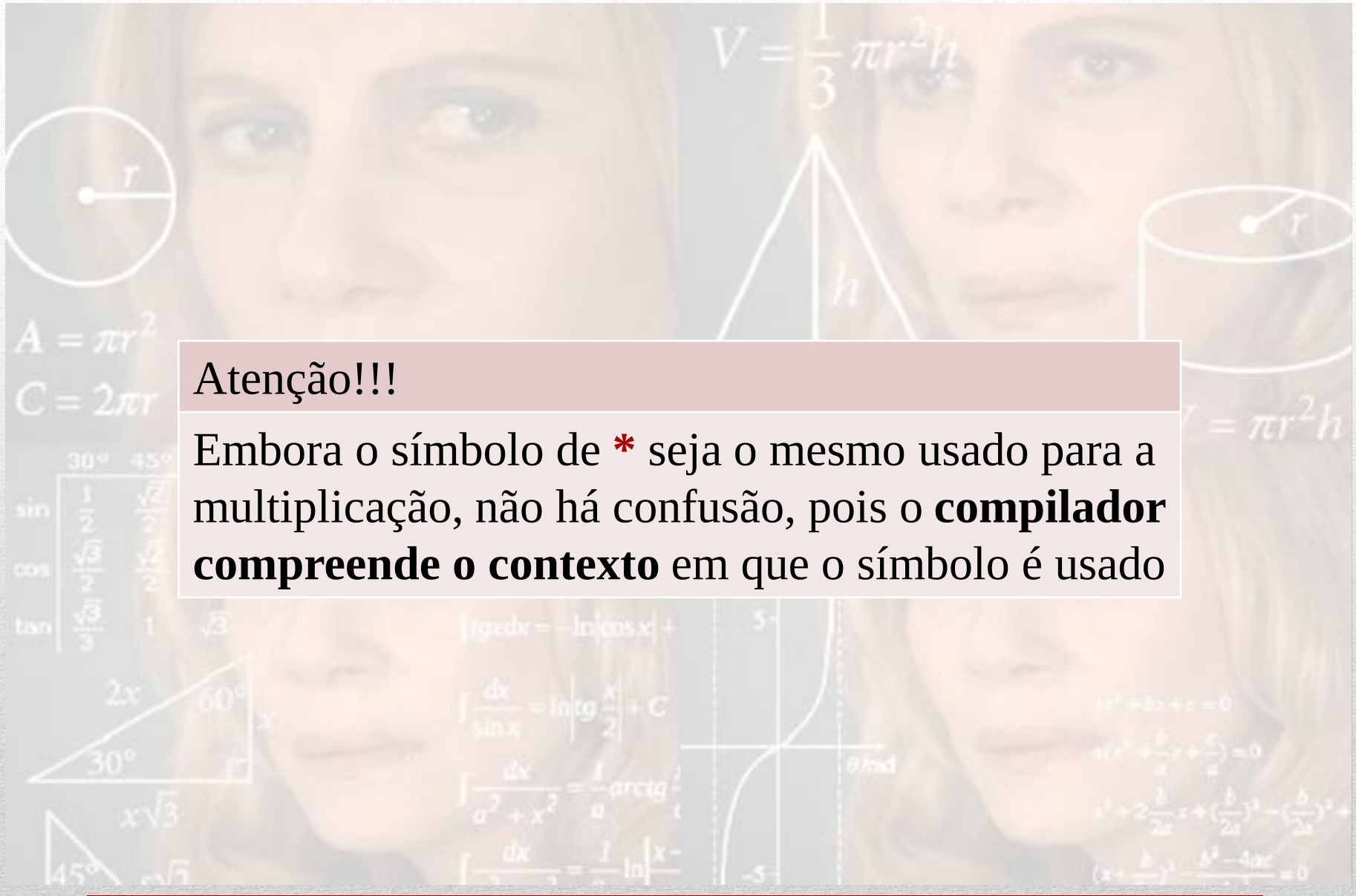
Sintaxe

<tipo> * nome_variável_ponteiro;

tipo: int, char, double, float, void
*: indica que trata-se de uma variável ponteiro
nome_variável_ponteiro: mnemônico da variável (nome da variável)

Exemplo:

```
char a, *p, *q;
int idade, *ptridade;
```



Atenção!!!

Embora o símbolo de * seja o mesmo usado para a multiplicação, não há confusão, pois o **compilador compreende o contexto** em que o símbolo é usado

https://pt.wikipedia.org/wiki/Ficheiro:Nazar%C3%A9_Confusa.jpg

Carga inicial

A **carga inicial** do ponteiro se faz através do operador de endereço **&**.

A variável do tipo **ponteiro** também pode ser **inicializada durante a declaração**. Ex.:

```
int x=15;
float pi=3.14;
int *ptrx=&x;
float *ptrpi=&pi;
```

Atenção!!!

Vejam a diferença entre os comandos

```
int x=15;
int *ptx=&x; //declaro e atribuo o valor (endereço) ao ponteiro
```

Vejam a diferença entre os comandos

```
int x=15, *ptx; //apenas declaro o ponteiro
ptx=&x; //atribuo a variável ponteiro a um determinado
//endereço de memória, sem o *
```

Carga inicial

Carga inicial

Para **evitar problemas** de programação, dê uma carga inicial aos ponteiros;

Entretanto, podem haver **situações** em que não deseja-se **inicializar as variáveis** do tipo ponteiro;

```
int * ptrx=NULL;
```

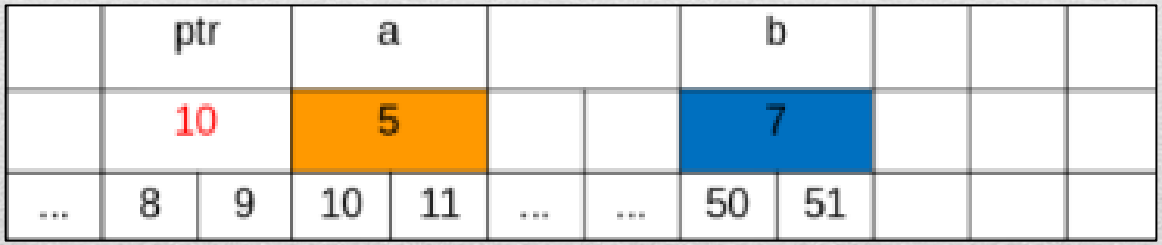
NULL: indica que não aponta para nenhuma variável;

Programação com ponteiros

O **ponteiro** guarda **endereços**, assim, os **tipos de operações** sobre inteiros também é válido para ponteiros.

```
int a=5, b=7, *ptr=NULL;
ptr=&a;
printf("%d", a);
printf("%d", ptr);
printf("%d", *ptr);
```

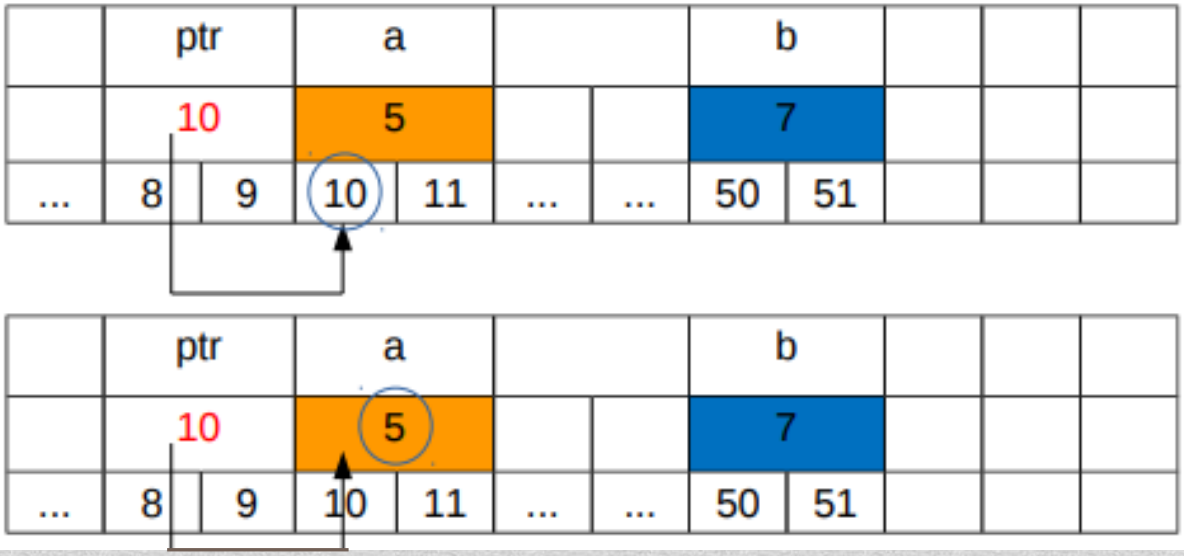
Variável	Saída
a	5
ptr	10
*ptr	5



Por que isso acontece?

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

Variável	Saída
ptr	10
*ptr	5



É possível criar um ponteiro para uma *struct*, de forma semelhante à criação de ponteiro para outros tipos de dados.

Ponteiros e structs

```
void main( )
{
    struct tAluno
    {  int matricula ;
       float nota;
    } a1;
    struct tAluno *ptrAluno = &a1; // criamos o ponteiro *ptrAluno do tipo tAluno que recebe o endereço de a1;
    (*ptrAluno).nota = 8.5; //Usamos o * para dereferenciar o ponteiro e o . para acessar o membro da struct.
    ptrAluno -> matricula = 711;
    printf("\n %f e %d",a1.nota, a1.matricula);
    printf("\n %f e %d",ptrAluno -> nota,  ptrAluno -> matricula);
```

Ponteiros e structs

Ponteiros e vetores

- **Ponteiros** também são usados para manipular **vetores (matrizes) e *strings***;
- O nome de um vetor corresponde ao **endereço do seu primeiro elemento**, isto é, se v for um vetor v = = &v[0];
- Daí, tem-se que um **ponteiro fará referência ao endereço do primeiro elemento desse vetor**;

```
int V[3]={10,20,30};
int *ptr;
ptr=&V; //ptr=&V[0]
printf("%d",*ptr);
```

```
int v[3]={10,20,30};
int *ptr;
ptr=V; //ptr=&V[0]
printf("%d",*ptr)
```

Ponteiros e vetores

```
void main( )
{
    int V[3]= {10, 20, 30};
    int *ptr;
    ptr = V; //ptr = &V[0] OU ptr = &V
    printf("%d \t %d \t %d",*ptr,*ptr+1,*(ptr+2)); // saída 10, 11 e 30

}//fim main
```

Por quê?

Ponteiros e vetores

Aritmética de ponteiros

Lembrando de que **ponteiros** guardam posições de memória, é possível realizar **operações aritméticas**;
Incremento - o ponteiro **avança** a **dimensão (sizeof) bytes** do tipo do objeto para o qual ele aponta;
Decremento - o ponteiro **recua sizeof(xyz) bytes** do tipo do objeto para o qual ele aponta;
Comparação – entre dois ponteiros para o mesmo tipo, utilizando os **operadores relacionais** (<,<=,>,>=,= = e ! =);

Aritmética de ponteiro

Escreva um programa que mostre uma string pela ordem em que está escrita e pela ordem contrária, usando decremento de ponteiros.

Exemplo: caderno – onredac.

Aritmética de ponteiro

```
void main( ) {
    char v[100],*ptr=v;
    printf("Digite uma string:");
    gets(v);
    printf("\n%s",v);
    if(*ptr=='\0') return;
    printf("\nOrdem direta\n");
    while(*ptr!='\0’)
    {
        putchar(*ptr++);
    }
    printf("\nOrdem inversa\n");
    ptr--; // para evitar o \0
    while(ptr>=&v) {
        putchar(*ptr--); }
}//fim main
```

Ponteiro para ponteiro

- Uma vez que os ponteiros ocupam espaço em memória, é possível obter a sua posição através do operador endereço &;
- Assim, é possível armazenar o endereço de um ponteiro, ao declarar um ponteiro para ponteiro;

```
int x;
int *ptr_x;
int **ptr_ptr_x;
```

- É possível fazer essa declaração, sem limitação para o número de asteriscos;


```
int x=5;
int *ptr_x;
int **ptr_ptr_x;
ptr_x=&x;
ptr_ptr_x=&ptr_x;
printf("x = %d - &x %ld",x,&x);
printf("\nx = %d - &x= %ld",*ptr_x,ptr_x);
printf("\nx = %d - &x=%ld",**ptr_ptr_x,*ptr_ ptr_ x);
```

Ponteiro para ponteiro

- https://youtu.be/SJzd9x2S2yg (Aula 55: Ponteiros - Parte 1 - Conceitos)
- https://www.youtube.com/watch?v=cg1mnWupbTE (Aula 56: Ponteiros - Parte 2 - Operações)
- https://youtu.be/bqw-GebrvEU (Aula 57: Ponteiros - Parte 3 - Ponteiro Genérico)
- https://youtu.be/w_BBUJWS-50 (Aula 58 - Ponteiros e *Arrays*)
- https://youtu.be/2-GllOuAYFE (Aula 59 - Ponteiro para Ponteiro)

Assistam aos vídeos