



**ALGORITMOS E COMPLEXIDADE (ARA0174)**

**AULA 2** (Continuação da revisão de Estrutura de Dados)

**DATA: 24/02/2022**



# Heterogêneos

Conjunto de elementos que podem ser de mesmo tipo existente, inclusive outros agregados.

## Struct

Funciona como o registro. A variável é composta de vários elementos, que podem ser de tipos diferentes e ela possui espaço reservado na memória para armazenar os valores de todos os seus elementos simultaneamente.

### Definindo e declarando Registros

- *Sintaxe de definição de registros:*

```
struct <nome do registro> {  
    <tipo> <nome da variável 1 do registro>;  
    <tipo> <nome da variável 2 do registro>;  
    ...  
    <tipo> <nome da variável n do registro>;  
};
```

## Relembrando a declaração de struct em C.

- ❑ *struct* nada mais é que um conjunto, ou bloco, de variáveis.

```
struct Nome_de_sua_struct{  
    tipo_1 nome_dos_tipo_1;  
    tipo_2 nome_dos_tipo_2;  
    ...  
    tipo_N nome_dos_tipo_N;  
};
```

Os atributos não podem ser inicializados dentro da própria estrutura:

```
struct S {  
    // ERRADO!!!  
    float f = 2.5f;  
};
```

**Como fazer um código para cadastrar mais de um funcionário e ter os seguintes campos: matrícula, nome, salário.**



## EXEMPLO 1

```
#include <stdio.h>
#include <locale.h>

struct Funcionario {
    int matricula;
    char nome[50];
    float salario;
};

int main(){
    setlocale(LC_ALL, "portuguese");
    Funcionario f[2]; // max de 2 funcionarios
    int idx=0;

    printf("\n *** Cadastro *** ");

    for (idx=0; idx<2; idx++) {

        f[idx].matricula = idx+1;
        printf("\n\n Qual a matricula do funcionario?: %d", f[idx].matricula);
        printf("\n Qual o nome do funcionario?: ");
        scanf(" %s", f[idx].nome);
        printf(" Qual o salario do funcionario?: ");
        scanf("%f", &f[idx].salario);
    }

    int i;
    for(i = 0; i < idx; i++){
        printf("\n\n Funcionário\n");
        printf(" Matricula: %d\n", f[i].matricula);
        printf(" Nome: %s \n", f[i].nome);
        printf(" Salario: %.2f\n", f[i].salario);
        printf("=====\n");
    }
}
```

## COMPILANDO E EXECUTANDO O EXEMPLO 1 VAI APARECER A TELA ABAIXO:

```
*** Cadastro ***
```

```
Qual a matricula do funcionario?: 1
```

```
Qual o nome do funcionario?: Jorge
```

```
Qual o salario do funcionario?: 2591,33
```

```
Qual a matricula do funcionario?: 2
```

```
Qual o nome do funcionario?: Luciano
```

```
Qual o salario do funcionario?: 3011,93
```

```
Funcionário
```

```
Matricula: 1
```

```
Nome: Jorge
```

```
Salario: 2591,33
```

```
=====
```

```
Funcionário
```

```
Matricula: 2
```

```
Nome: Luciano
```

```
Salario: 3011,93
```

```
=====
```

## EXEMPLO 02

Utilizando o typedef struct

```
#include<iostream>
#include<iomanip>
#include<locale.h>
using namespace std;
typedef struct {
    int codigo;
    char nome[200];
    int dia, mes, ano;
} Aluno;
int main() {
    Aluno a;
    setlocale(LC_ALL, "portuguese");
    cout << "\n PREENCHA O FORMULÁRIO\n\n";
    cout << " Digite o código do aluno: ";
    cin >> a.codigo;
    cout << " Digite o nome do aluno: ";
    cin.ignore();
    cin.getline(a.nome, 200); /*Aceita espaço entre as palavras*/
    cout << " Digite o dia do nascimento do aluno: "; cin >> a.dia;
    cout << " Digite o mês do nascimento do aluno: "; cin >> a.mes;
    cout << " Digite o ano do nascimento do aluno: "; cin >> a.ano;
    cout << "\n\n  RESULTADO: ";

    /*Não apaga os ZEROS a esquerda* -> setfill('0') << setw(n) */
    cout << "\n\n    O código do aluno é: " <<
    setfill('0') << setw(3) << a.codigo;
    cout << "\n    O nome do aluno é: " << a.nome;
    cout << "\n    A data de nascimento do aluno é: " <<
    setfill('0') << setw(2) << a.dia << "/" <<
    setfill('0') << setw(2) << a.mes << "/" << a.ano;
    cout << "\n\n";
    system("pause");
    return(0);
}
```



## EXEMPLO 02

Utilizando o typedef struct

```
#include<iostream>
#include<iomanip>
#include<locale.h>
using namespace std;
```

```
typedef struct {
    int codigo;
    char nome[200];
    int dia;
    int mes;
    int ano;
} Aluno;
```

AO COMPILAR E EXECUTAR VAI APARECER O RESULTADO ABAIXO:

PREENCHA O FORMULÁRIO

Digite o código do aluno: 001

Digite o nome do aluno: CARLOS ROBERTO

Digite o dia do nascimento do aluno: 04

Digite o mês do nascimento do aluno: 07

Digite o ano do nascimento do aluno: 1990

RESULTADO:

O código do aluno é: 001

O nome do aluno é: CARLOS ROBERTO

A data de nascimento do aluno é: 04/07/1990

```
system("pause");
return(0);
}
```



## Definindo a struct, usando estruturas ANINHADAS

```
struct data  
{  
    int dia, mes, ano;  
};
```

## Definindo a struct, usando estruturas ANINHADAS

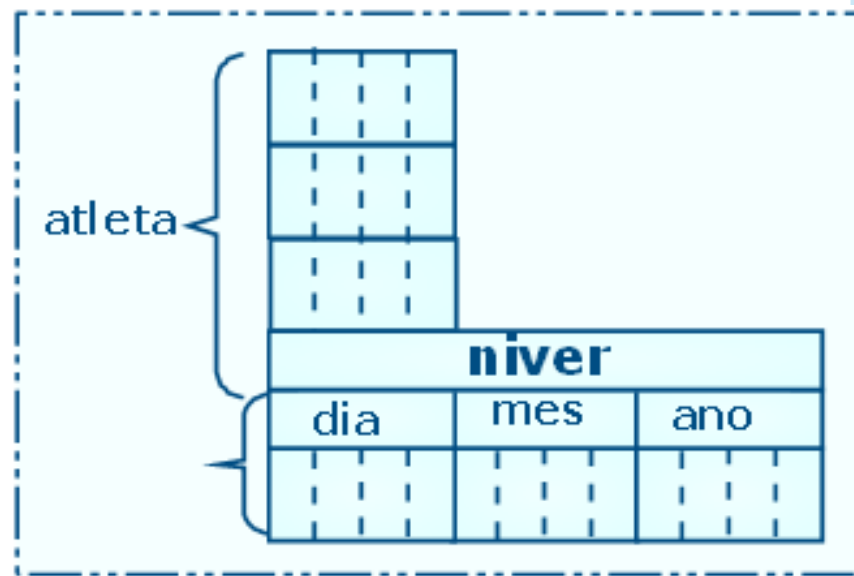
```
struct data
{
    int dia, mes, ano;
};
```

```
struct saude
{
    float peso, altura, IMC;
    data niver;
} atleta;
```

## Definindo a struct, usando estruturas ANINHADAS

```
struct data
{
    int dia, mes, ano;
};
```

```
struct saude
{
    float peso, altura, IMC;
    data niver;
} atleta;
```

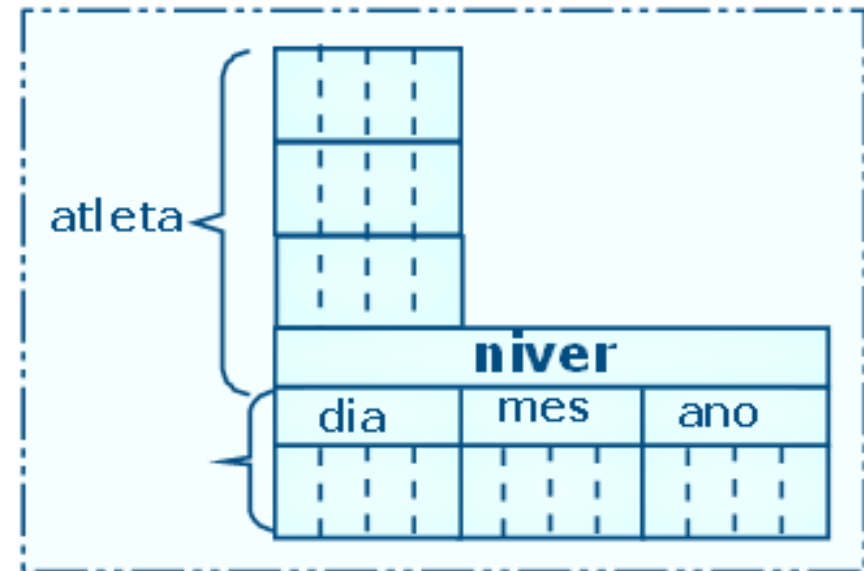


## Definindo a struct, usando estruturas ANINHADAS

```
struct data
{
    int dia, mes, ano;
};
```

```
struct saude
{
    float peso, altura, IMC;
    data niver;
} atleta;
```

### ACESSANDO



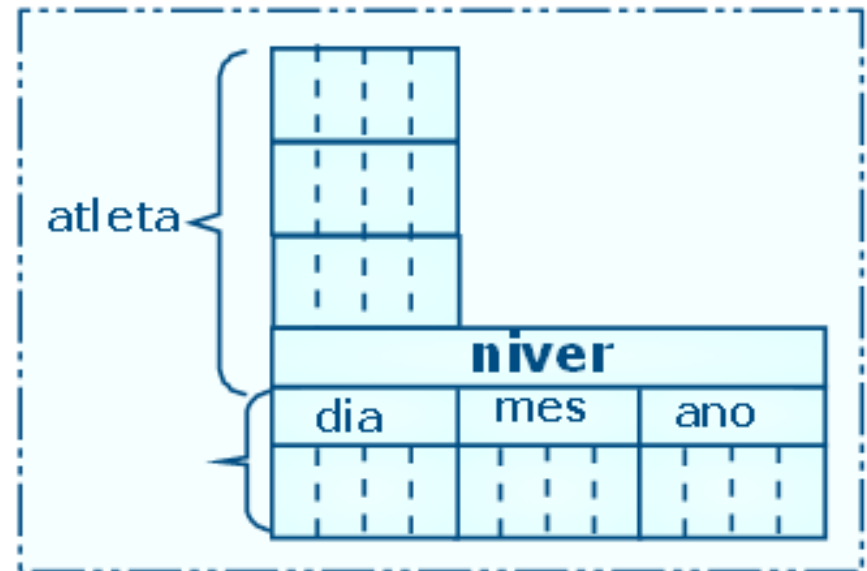


## Definindo a struct, usando estruturas ANINHADAS

```
struct data
{
    int dia, mes, ano;
};
```

```
struct saude
{
    float peso, altura, IMC;
    data niver;
} atleta;
```

# ACCESSANDO



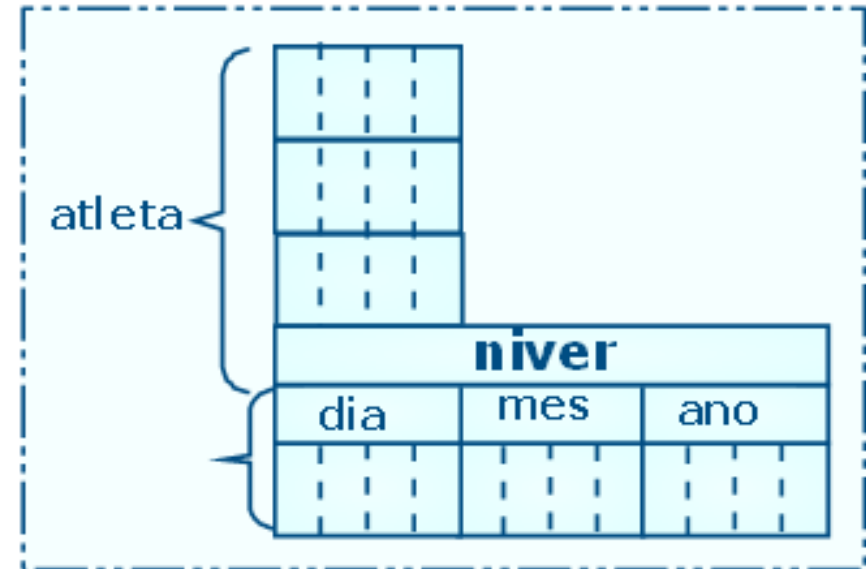
...atleta • peso...

## Definindo a struct, usando estruturas ANINHADAS

```
struct data
{
    int dia, mes, ano;
};
```

```
struct saude
{
    float peso, altura, IMC;
    data niver;
} atleta;
```

### ACESSANDO



...atleta • peso...

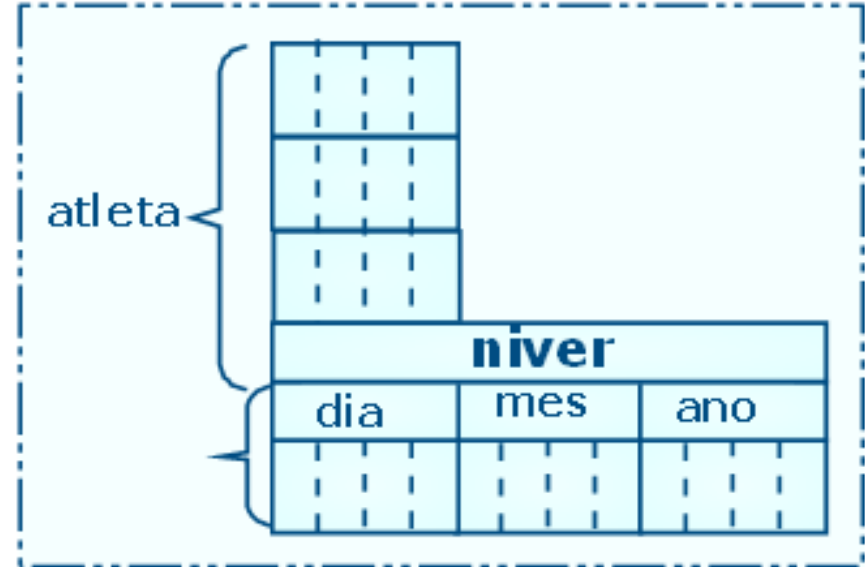
...atleta • niver • dia...

## Trabalhando com **struct** aninhadas

```
struct data
{
    int dia, mes, ano;
};
```

```
struct saude
{
    float peso, altura, IMC;
    data niver;
} atleta;
```

## ACESSANDO



...atleta.peso...

...atleta.niver.dia...

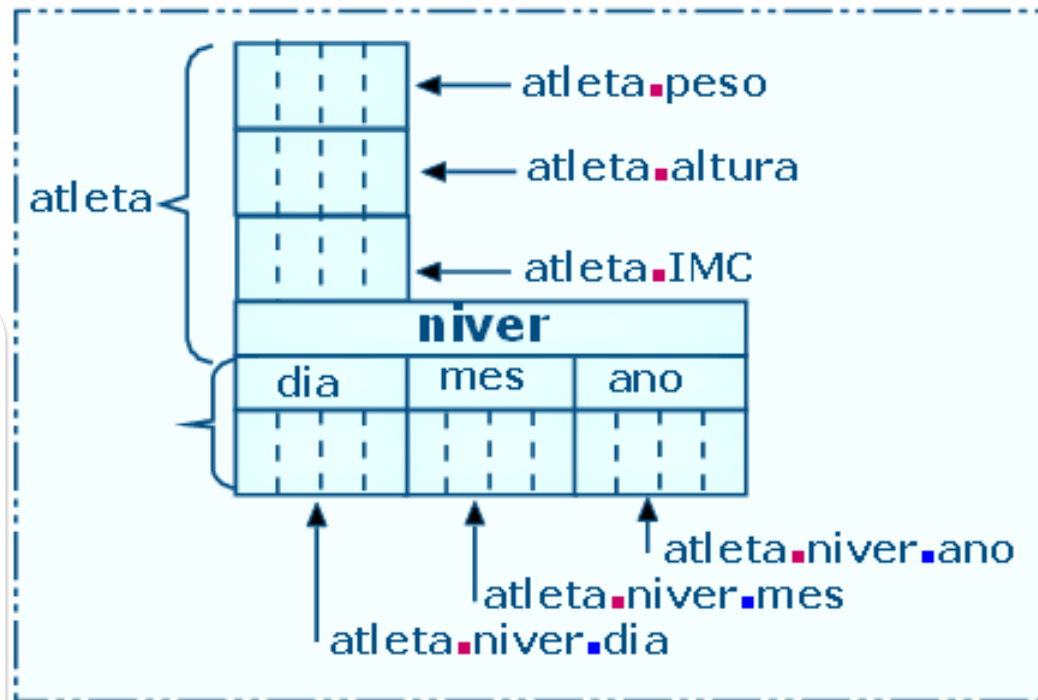
...atleta.niver.mes..

## Trabalhando com **struct** aninhadas

```
struct data
{
    int dia, mes, ano;
};
```

```
struct saude
{
    float altura, peso, IMC;
    data niver;
} atleta;
```

### ACESSANDO





## Definindo a struct, usando estruturas ANINHADAS - array

```
struct data
{
    int dia, mes, ano;
};
```

ACESSANDO

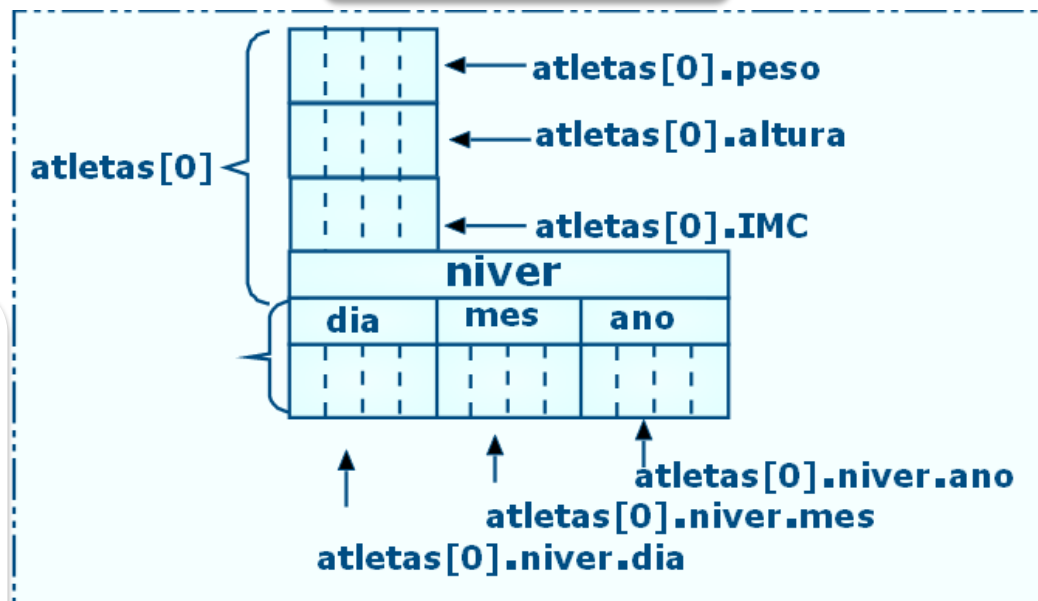
```
struct saude
{
    float altura, peso, IMC;
    data niver;
} atletas[500];
```

## Definindo a struct, usando estruturas ANINHADAS - array

```
struct data
{
    int dia, mes, ano;
};
```

```
struct saude
{
    float altura, peso, IMC;
    data niver;
} atletas[500];
```

### ACESSANDO

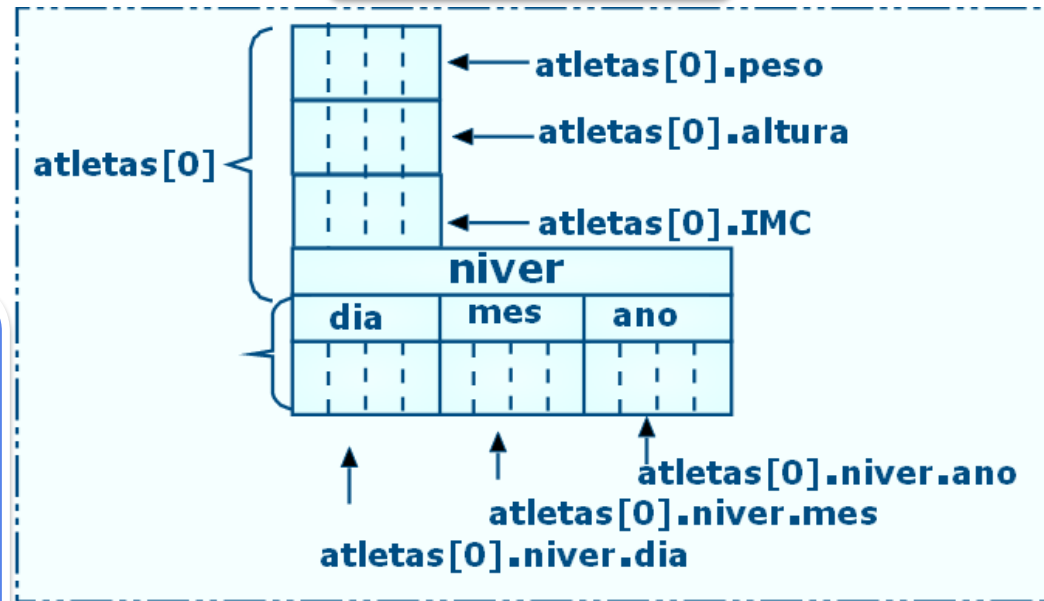


## Definindo a struct, usando estruturas ANINHADAS - array

```
struct data
{
    int dia, mes, ano;
};
```

```
struct saude
{
    float altura, peso, IMC;
    data niver;
} atletas[500];
```

### ACESSANDO



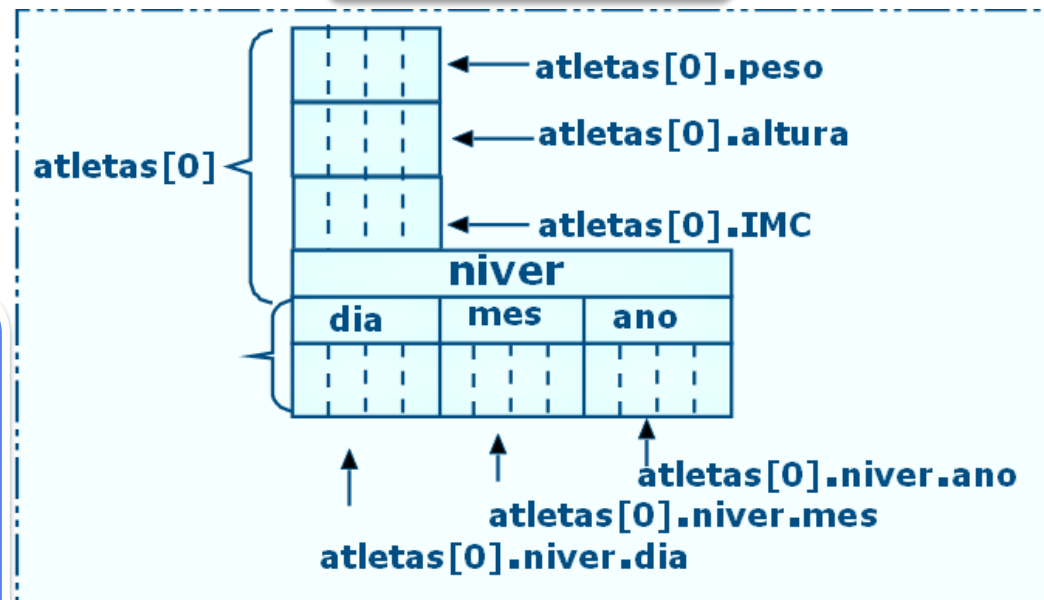
...atletas[0].**peso**...

## Definindo a struct, usando estruturas ANINHADAS - array

```
struct data
{
    int dia, mes, ano;
};
```

```
struct saude
{
    float altura, peso, IMC;
    data niver;
} atletas[500];
```

### ACESSANDO



...atletas[0].**peso**...

...atletas[0].**niver**.**dia**...

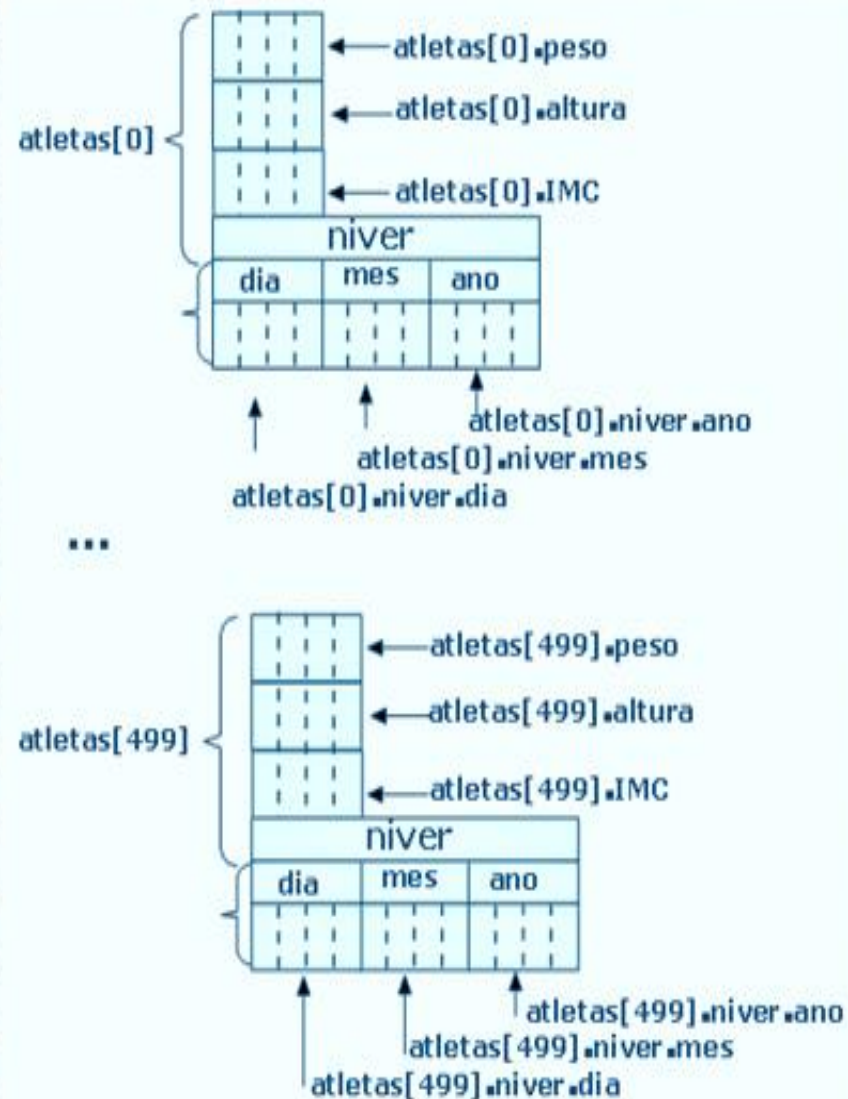


## Trabalhando com **struct** aninhadas

```
struct data
{
    int dia, mes, ano;
};
```

```
struct saude
{
    float altura, peso, IMC;
    data niver;
} atletas[500];
```

A  
C  
E  
S  
S  
A  
N  
D  
O



```
#include <iostream>
#include <cstdlib>
#include <locale.h>
using namespace std;
```

```
struct data {
    int dia, mes, ano;
};
```

```
struct saude {
    float altura, peso, IMC;
    data niver;
} atletas[500];
```

```
int main() {
    setlocale(LC_ALL, "portuguese");
    int x;
    char escolha;
    for (x=0; x<500; x++) {
```

```
        cout << "\n\nEntre com a ALTURA do atleta " << x + 1 << ": ";
        cin >> atletas[x].altura;
        cout << "\nEntre com o PESO do atleta " << x + 1 << ": ";
        cin >> atletas[x].peso;
        cout << "\nEntre com o IMC do atleta " << x + 1 << ": ";
        cin >> atletas[x].IMC;
```

Definindo a struct, usando estruturas  
ANINHADAS - array

## EXEMPLO 3

```
cout << "\nEntre com a DATA DE ANIVERSÁRIO do atleta " << x + 1 << ": \n";  
cout << "Dia: "; cin >> atletas[x].niver.dia;  
cout << "Mês: "; cin >> atletas[x].niver.mes;  
cout << "Ano: "; cin >> atletas[x].niver.ano;
```

denovo:

```
cout << "\n\nQuer continuar a inserção de dados? (Digite S ou N): ";  
cin >> escolha;
```

```
if (escolha=='S' | escolha=='s' ) {  
    goto inicio;
```

```
}
```

```
if (escolha=='N' | escolha=='n' ) {  
    system("cls");  
    goto fim;
```

```
}
```

```
if (escolha!='S' | escolha!='s' | escolha!='N' | escolha!='n' ) {  
    cout << "\n\nERRO ! Digite S ou N\n\n";  
    goto denovo;
```

```
}
```

```
inicio: system("cls");
```

```
}
```

```
fim: cout << "\n\n\nfim";
```

```
}
```

## CONTINUAÇÃO: EXEMPLO 3

Definindo a struct, usando estruturas ANINHADAS - array

# PONTEIRO

A utilização de ponteiros em linguagem C é uma das características que tornam a linguagem tão flexível e poderosa.

Ponteiros ou apontadores, são variáveis que armazenam o endereço de memória de outras variáveis.

Dizemos que um ponteiro “aponta” para uma variável quando contém o endereço da mesma.

Os ponteiros podem apontar para qualquer tipo de variável. Portanto temos ponteiros para int, float, double, etc.



# PONTEIRO

## Utilização dos ponteiros

Os ponteiros ou apontadores são muito úteis para acessar uma determinada variável em diferentes partes de um programa.

Os ponteiros são úteis em várias situações, por exemplo:

- Alocação dinâmica de memória.
- Manipulação de arrays.
- Para retornar mais de um valor em uma função.
- Referência para listas, pilhas, árvores e grafos.

# PONTEIRO

Na linguagem C/C++, sabe-se que cada variável tem um nome, um tipo, um valor e um endereço.

Exemplo:

```
int x = 5;  
char c = 'D';
```

Pode-se observar a alocação dos valores das variáveis x e c na memória.

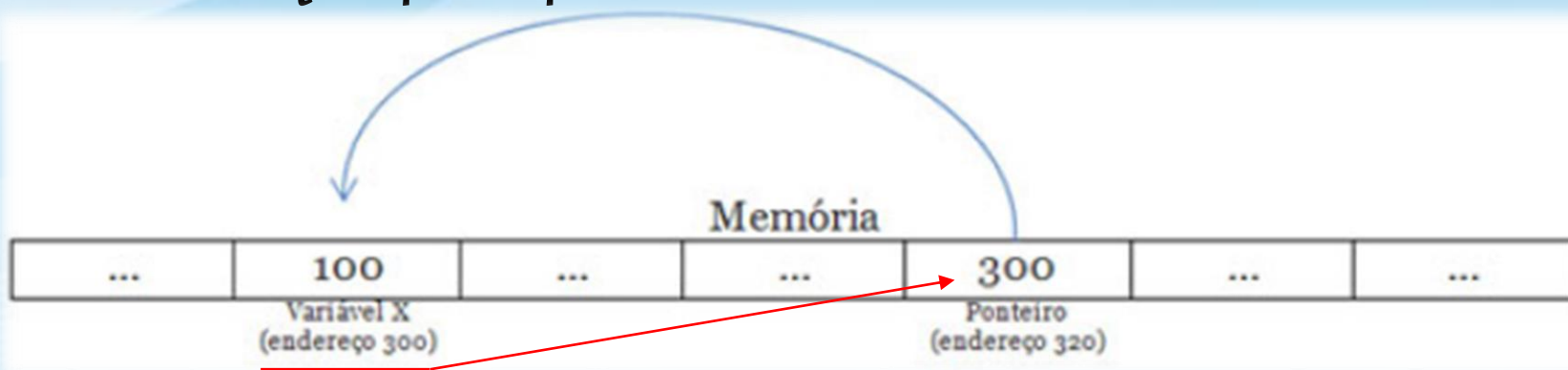


Alocação da variável "x" e "c" na memória.

# PONTEIRO

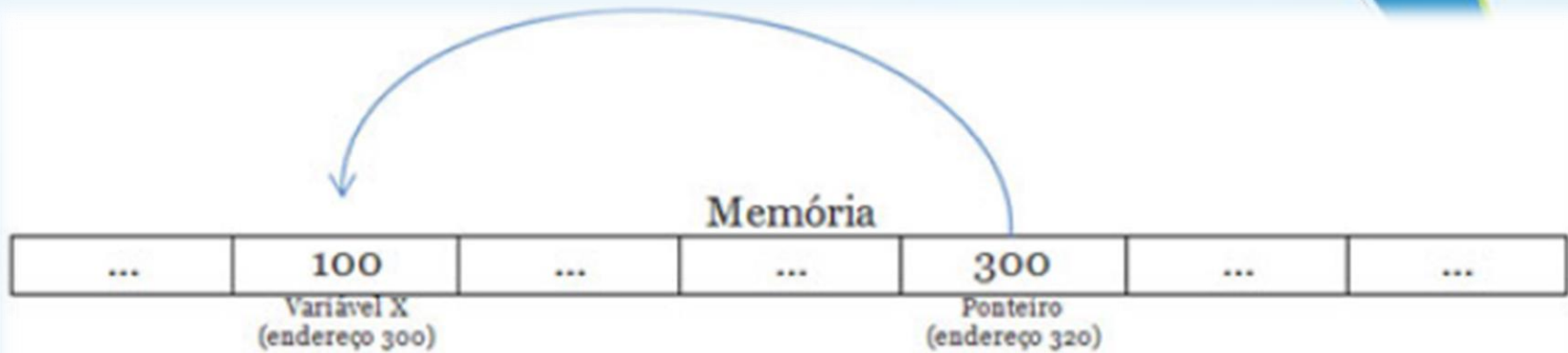
Todo dado armazenado em memória possui um endereço e que a definição de um ponteiro é uma variável que guarda o endereço de memória, ou seja, a localização do dado.

**Sendo assim, um ponteiro armazena o endereço de outra variável, isto é, uma variável que aponta para outra.**



A figura acima apresenta uma memória que armazena a variável “x” e o ponteiro para a variável “x”. Como o conteúdo do ponteiro é um endereço, a seta indica a relação entre o ponteiro e a variável que ele aponta.

# PONTEIRO



No exemplo, o valor da variável “x” é 100 e esse dado está armazenado no endereço de memória 300. Como o ponteiro também ocupa espaço, o endereço de memória da variável “x” está armazenado no endereço 320 da memória, que é a posição de memória alocada para o ponteiro.

Uma das características importantes de um ponteiro é que, utilizando ponteiros, podemos realizar o acesso indireto a outras variáveis, isto é, podemos ler ou alterar o conteúdo de uma variável sem utilizar o nome desta variável.

# PONTEIRO

## DECLARAÇÃO

Os ponteiros são declarados pelo símbolo “ \* ” entre o tipo e o nome da variável. A forma geral da declaração é:

```
Tipo_da_variável * Nome_da_Variável;
```

*// Exemplos:*

```
int *p;  
float *q;  
char *r;
```

*// As variáveis p, q e r são apontadores (ponteiro)  
// para um inteiro, float e caractere, respectivamente.*



# PONTEIRO

## Operadores

Vejamos dois tipos de operadores:

- **Operador unário “&” ou ponteiro constante** — Tem a função de obter o endereço de memória de uma variável.
- **Operador unário “\*” de indireção** — É usado para fazer a deferência.

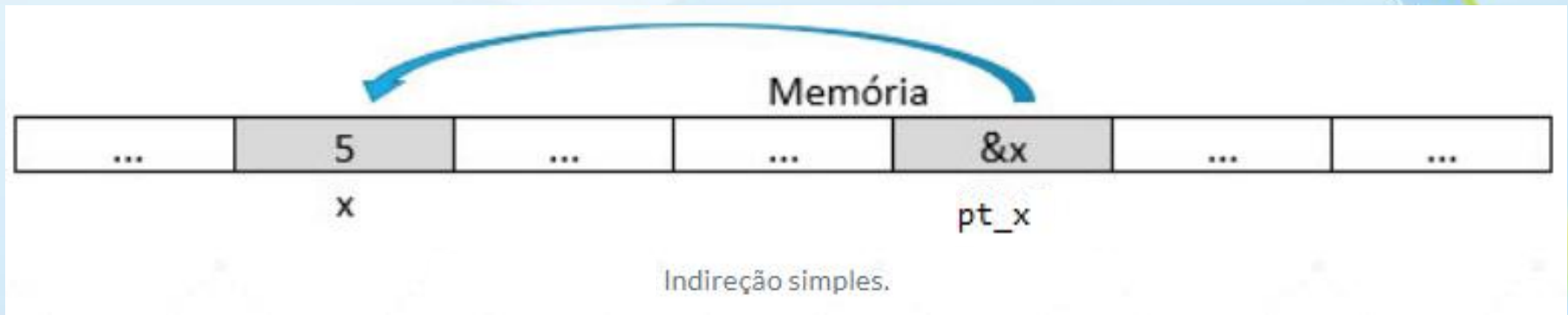
Para entendermos melhor, observe o Exemplo a seguir. Neste trecho de código, temos a definição de uma variável inteira e de um ponteiro para o tipo de dados inteiro.

```
int x = 5;  
int *pt_x;  
  
/* Pontoeiro pt_x recebe o endereço de memória da variável x */  
pt_x = &x;
```

Exemplo: Trecho de código.

# PONTEIRO

A Figura abaixo mostra a indireção do exemplo.



O ponteiro “pt\_x” recebe o endereço de memória da variável “x” pela instrução:

```
pt_x = &x;
```



# PONTEIRO

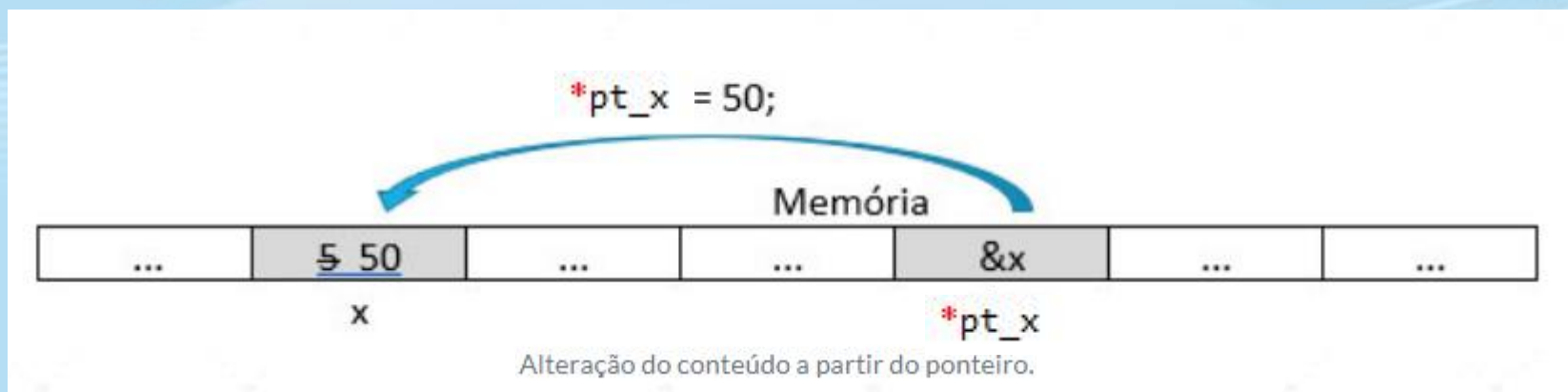
Agora, considere a instrução apresentada abaixo. Note que o valor da variável “x” é alterado pelo ponteiro “pt\_x”.

```
*pt_x = 50;
```

# PONTEIRO

```
*pt_x = 50;
```

Como visto na Figura abaixo, podemos observar que o conteúdo da variável “x” foi alterado de forma indireta, ou seja, não foi feita referência ao nome da variável “x”. Neste caso, o ponteiro é chamado de ponteiro variável, pois assim é possível armazenar qualquer endereço.



# PONTEIRO

REVISANDO O CONTEÚDO “PONTEIRO”

EXEMPLO:

```
int a, *pa;  
a=23;  
pa= &a;
```

```
int a=23, *pa=&a;
```

Vamos analisar um outro exemplo, que está destacado abaixo:

```
int ano =1989;  
//declaração de variável ponteiro  
int *ptrano;  
//atribuindo o endereço da variável  
"ano" ao ponteiro  
ptrano = &ano;
```

# PONTEIRO

## REVISANDO O CONTEÚDO "PONTEIRO"

EXEMPLO:

```
int a, *pa;  
a=23;  
pa= &a;
```

```
int a=23, *pa=&a;
```

Vamos analisar um outro exemplo, que está destacado abaixo:

```
int ano =1989;
```

```
//declaração de variável ponteiro
```

```
int *ptrano;
```

```
//atribuindo o endereço da variável  
"ano" ao ponteiro
```

```
ptrano = &ano;
```



```
int ano =1989;  
//declaração de variável  
ponteiro  
int *ptrano;  
//atribuindo o endereço da  
variável "ano" ao ponteiro  
ptrano = &ano;
```



## Entendendo

```
cout << "ano : " << ano;
```

```
cout << "\n&ano: " << &ano;
```

```
cout << "\nptrano: " << ptrano;
```

```
cout<<"\n&ptrano: "<<&ptrano;
```

```
cout << "\n*ptrano: "<<*ptrano;
```

```
int ano =1989;
//declaração de variável
//ponteiro
int *ptrano;
//atribuindo o endereço da
//variável "ano" ao ponteiro
ptrano = &ano;
```



## Entendendo

```
cout << "ano : " << ano;
```

1989

```
cout << "\n&ano: " << &ano;
```

```
cout << "\nptrano: " << ptrano;
```

```
cout<<"\n&ptrano: "<<&ptrano;
```

```
cout << "\n*ptrano: "<<*ptrano;
```

```
int ano =1989;
//declaração de variável
//ponteiro
int *ptrano;
//atribuindo o endereço da
//variável "ano" ao ponteiro
ptrano = &ano;
```



## Entendendo

```
cout << "ano : " << ano;
```

1989

```
cout << "\n&ano: " << &ano;
```

0x22ff74

```
cout << "\nptrano: " << ptrano;
```

```
cout<<"\n&ptrano: "<<&ptrano;
```

```
cout << "\n*ptrano: "<<*ptrano;
```



```
int ano =1989;
//declaração de variável
//ponteiro
int *ptrano;
//atribuindo o endereço da
//variável "ano" ao ponteiro
ptrano = &ano;
```



## Entendendo

<code>cout &lt;&lt; "ano : " &lt;&lt; ano;</code>	1989
<code>cout &lt;&lt; "\n&amp;ano: " &lt;&lt; &amp;ano;</code>	0x22ff74
<code>cout &lt;&lt; "\nptrano: " &lt;&lt; ptrano;</code>	0x22ff74
<code>cout&lt;&lt;"\n&amp;ptrano: "&lt;&lt;&amp;ptrano;</code>	
<code>cout &lt;&lt; "\n*ptrano: "&lt;&lt;*ptrano;</code>	

```
int ano =1989;
//declaração de variável
//ponteiro
int *ptrano;
//atribuindo o endereço da
//variável "ano" ao ponteiro
ptrano = &ano;
```



## Entendendo

<code>cout &lt;&lt; "ano : " &lt;&lt; ano;</code>	1989
<code>cout &lt;&lt; "\n&amp;ano: " &lt;&lt; &amp;ano;</code>	0x22ff74
<code>cout &lt;&lt; "\nptrano: " &lt;&lt; ptrano;</code>	0x22ff74
<code>cout&lt;&lt;"\n&amp;ptrano: "&lt;&lt;&amp;ptrano;</code>	0x22ff70
<code>cout &lt;&lt; "\n*ptrano: "&lt;&lt;*ptrano;</code>	

```
int ano =1989;
//declaração de variável
//ponteiro
int *ptrano;
//atribuindo o endereço da
//variável "ano" ao ponteiro
ptrano = &ano;
```



## Entendendo

<code>cout &lt;&lt; "ano : " &lt;&lt; ano;</code>	1989
<code>cout &lt;&lt; "\n&amp;ano: " &lt;&lt; &amp;ano;</code>	0x22ff74
<code>cout &lt;&lt; "\nptrano: " &lt;&lt; ptrano;</code>	0x22ff74
<code>cout&lt;&lt;"\n&amp;ptrano: "&lt;&lt;&amp;ptrano;</code>	0x22ff70
<code>cout &lt;&lt; "\n*ptrano: "&lt;&lt;*ptrano;</code>	1989

# PONTEIRO

Exemplo de utilização de ponteiros:

```
#include<iostream>
#include<conio.h> // é con (console) io (entra e saída) ou seja, nela tem as função
                // para a entrada e saída de programas que usam o console (DOS)
#include<locale.h>
using namespace std;
int main() {
    setlocale(LC_ALL, "portuguese");

    //v_num é a variável que
    //será apontada pelo ponteiro
    int v_num = 10;

    //declaração de variável ponteiro
    int *ptr;

    //atribuindo o endereço da variável v_num ao ponteiro
    ptr = &v_num;

    cout << "Utilizando ponteiros\n\n";
    cout << "Conteúdo da variável v_num: " << v_num;
    cout << "\nEndereço da variável v_num: " << &v_num;
    cout << "\nConteúdo da variável ponteiro ptr: " << ptr;
    cout << "\nDessa forma mostra o conteúdo da variável v_num: " << *ptr;
    getch(); // lê um caracter do teclado.
    return(0); }
```



# PONTEIRO

Exemplo de utilização de ponteiros:

AO COMPILAR E EXECUTAR VAI APARECER O RESULTADO ABAIXO

```
#include<iostream>
#include<conio.h> // é con (console) io (entra e saída) ou seja, nela tem as função
                 // para a entrada e saída de programas que usam o console (DOS)
#include<locale.h>
using namespace std;
int main()
{
    setlocale(LC_ALL, "");

    //v_num
    //será o
    int v_num = 10;

    //declaração de variável ponteiro
    int *ptr;

    //atribuindo o endereço da variável v_num ao ponteiro
    ptr = &v_num;

    cout << "Utilizando ponteiros\n\n";
    cout << "Conteúdo da variável v_num: " << v_num;
    cout << "\nEndereço da variável v_num: " << &v_num;
    cout << "\nConteúdo da variável ponteiro ptr: " << ptr;
    cout << "\nDessa forma mostra o conteúdo da variável v_num: " << *ptr;
    getch(); // Lê um caracter do teclado.
    return(0); }
```

Utilizando ponteiros

Conteúdo da variável v\_num: 10

Endereço da variável v\_num: 62fe14

Conteúdo da variável ponteiro ptr: 62fe14

Dessa forma mostra o conteúdo da variável v\_num: 10\_

# PONTEIRO

## Indireção múltipla

O endereço do ponteiro pode ser obtido da seguinte forma:

```
&pt_x
```

Sendo assim, um ponteiro pode armazenar o endereço de outro ponteiro, ocasionando uma indireção múltipla.

Para entendermos melhor, vamos analisar a declaração de um ponteiro de indireção múltipla. Utiliza-se  $N$  vezes o operador  $*$ , sendo  $N$  o nível de indireção.

# PONTEIRO

No Exemplo, é declarado um ponteiro para ponteiro.

```
int ** pt2; /*ponteiro para ponteiro do tipo inteiro*/  
int * pt1; /*ponteiro para o tipo inteiro*/  
int x = 10;  
  
pt2 = &pt1;  
pt1 = &x;  
  
*pt1 = 30;  
**pt2 = 50;
```

Exemplo: Declarando um ponteiro para ponteiro.

pt1: Conteúdo de 'pt1', ou seja, o endereço de 'x', que foi recebido através do comando pt1 = &x;

- \*\*pt2: Conteúdo do endereço apontado, ou seja, o conteúdo de 'pt1'.
- \*pt1: Conteúdo do endereço apontado, ou seja, o conteúdo de 'x'.
- \*\*pt2: Acessa o conteúdo do endereço armazenado no ponteiro que é referenciado por 'pt2', ou seja, acessa 'x' indiretamente.



# PONTEIRO

```
#include<iostream>
#include<locale.h>
using namespace std;
int main() {
    setlocale(LC_ALL, "portuguese");
    int **pt2; /*ponteiro para ponteiro do tipo inteiro*/
    int *pt1; /*ponteiro para o tipo inteiro*/
    int x = 10;

    pt2 = &pt1;
    pt1 = &x;

    *pt1 = 30;
    **pt2 = 50;

    cout << "\nP O N T E I R O S: \n\n";
    cout << "O valor da variável x após o processamento no ponteiro *pt1: " << *pt1;
    cout << "\nO valor da variável x após o processamento no ponteiro para ponteiro **pt2: " << **pt2;
    cout << "\n\n";
    cout << "O endereço de memória da variável x contida no ponteiro *pt1: " << pt1;
    cout << "\nO endereço de memória do ponteiro *pt1 contido no ponteiro para ponteiro **pt2: " << pt2;
}
```

# PONTEIRO

AO COMPILAR E EXECUTAR VAI APARECER O RESULTADO ABAIXO

```
#include<iostream>
```

```
using namespace std;
```

```
P O N T E I R O S:
```

```
O valor da variável x após o processamento no ponteiro *pt1: 50
```

```
O valor da variável x após o processamento no ponteiro para ponteiro **pt2: 50
```

```
O endereço de memória da variável x contida no ponteiro *pt1: 0x6ffdfc
```

```
O endereço de memória do ponteiro *pt1 contido no ponteiro para ponteiro **pt2: 0x6ffe00
```

```
pt2 = &pt1;
```

```
pt1 = &x;
```

```
*pt1 = 30;
```

```
**pt2 = 50;
```

```
cout << "\nP O N T E I R O S: \n\n";
```

```
cout << "O valor da variável x após o processamento no ponteiro *pt1: " << *pt1;
```

```
cout << "\nO valor da variável x após o processamento no ponteiro para ponteiro **pt2: " << **pt2;
```

```
cout << "\n\n";
```

```
cout << "O endereço de memória da variável x contida no ponteiro *pt1: " << pt1;
```

```
cout << "\nO endereço de memória do ponteiro *pt1 contido no ponteiro para ponteiro **pt2: " << pt2;
```

```
}
```

# PONTEIRO

No caso das operações básicas de incremento e decremento, analisaremos o comportamento e o que ocorre com o endereço armazenado no ponteiro quando uma das operações aritméticas é utilizada.

# PONTEIRO

Na Figura abaixo, podemos observar um ponteiro e três variáveis do tipo inteiro.

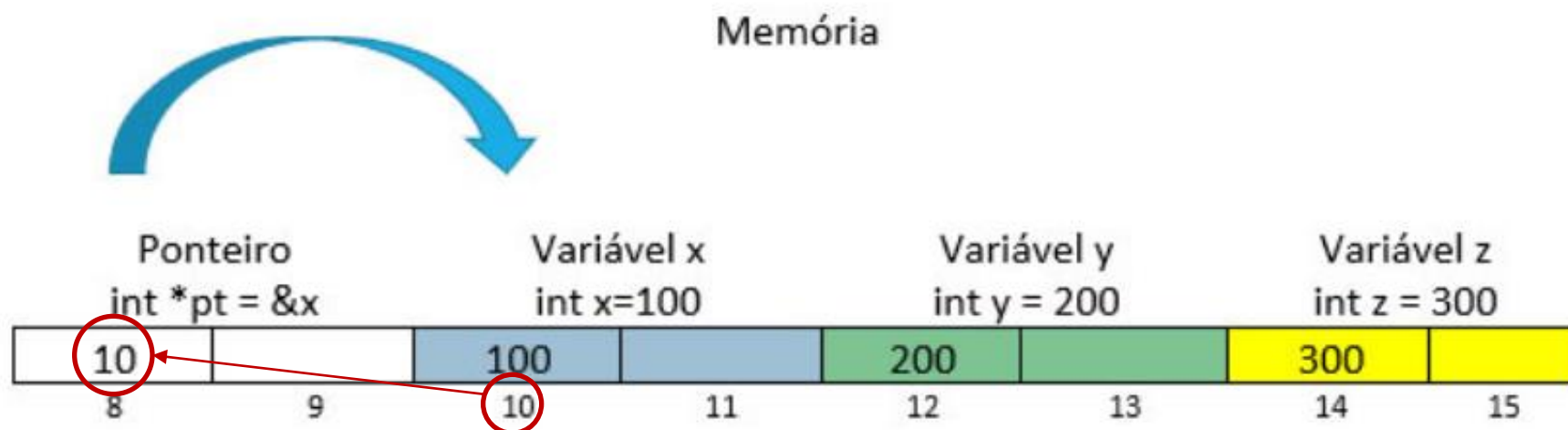


Figura 8: Ponteiro do tipo inteiro apontando para uma variável. Fonte: O Autor.

Consideremos que essas variáveis estão alinhadas sequencialmente na memória e que o ponteiro (`pt`) está armazenando o endereço da variável “`x`” (endereço 10).



# PONTEIRO

Para incrementar o ponteiro: `pt++`;

Ao realizar a operação, o ponteiro passará a apontar para o próximo dado do seu tipo. Ou seja, ao incrementar o ponteiro, o seu valor é alterado conforme o tamanho do tipo de dado que ele aponta. Diante disso, o próximo valor `pt` é o endereço 12, logo, o ponteiro aponta para a variável `y`. Esta situação é ilustrada abaixo.

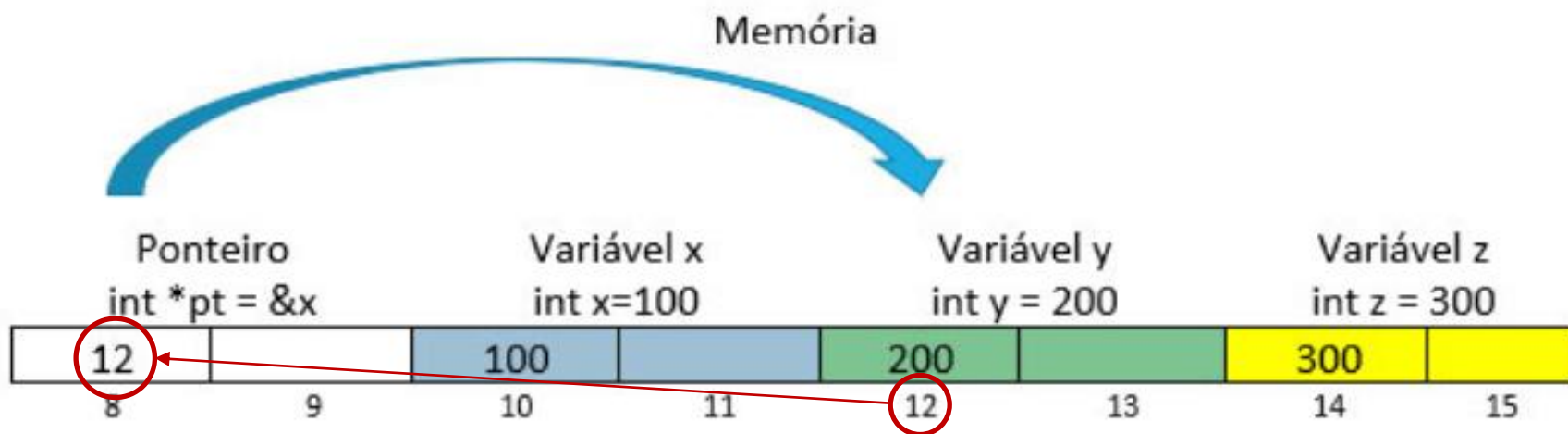


Figura 9: Operação de incremento. Fonte: O Autor.

# PONTEIRO

*Para decrementar o ponteiro: **pt--;***

Neste caso, o ponteiro passa a apontar para o elemento anterior. Ou seja, ao decrementar o ponteiro (pt), ele irá apontar novamente para a variável x que ocupa a posição 10.

De maneira geral, qualquer operação aritmética de ponteiro será realizada conforme o seu tipo base. Por exemplo:

# OPERADOR SETA ( **->** )EM C++



# OPERADOR SETA ( -> ) EM C++

Sabemos que o operador “ . ” é utilizado para acessar um membro de uma estrutura (STRUCT), conforme foi trabalhado em materiais anteriores.

*Mas esse operador “ . ” pode ser manipulado de forma eficiente com PONTEIROS?*

Nesse caso entra o operador de seta ( -> ) em C++ para ser utilizado para acessar um membro de uma estrutura que é referenciada pelo ponteiro em questão.

# OPERADOR SETA ( -> ) EM C++

**Observe o exemplo a seguir com  
atenção !**

# OPERADOR SETA ( -> ) EM C++

Vamos definir uma estrutura conforme apresentado abaixo:

```
struct TESTE {
```

```
    int a;
```

```
    int b;
```

```
};
```

E uma variável para ela:

```
struct TESTE x;
```

Se usa o ponto para acessar o membro dela:

```
x.a = 200;
```

Basicamente a linguagem C calcula o deslocamento em memória de onde está o membro **a** da variável **x**.

**OK !. ISSO JÁ É CONHECIDO ATÉ O MOMENTO DE ACORDO COM O CONTEÚDO DADO ATÉ AQUI !.**

# OPERADOR SETA ( -> ) EM C++

Mas quando se cria **UM PONTEIRO** para a **ESTRUTURA**?

# OPERADOR SETA ( -> ) EM C++

## VAMOS PARA ALGUMAS OBSERVAÇÕES:

```
struct TESTE *p;
```

**p** NÃO É DO TIPO *struct TESTE*, **p** é do tipo PONTEIRO.

**TENTAR FAZER O QUE ESTÁ MOSTRADO ABAIXO NÃO FUNCIONA COM PONTEIRO:**

```
p.a = 300;
```

Vai dar ERRO, pois **p** NÃO É ESTRUTURA e não tem membro **a** (**p** APONTARÁ para uma estrutura).

Mesmo que você tente fazer:

```
p = &x; (agora p aponta para a minha variável x).
```

**AINDA ASSIM EU NÃO POSSO FAZER **p.a****

# OPERADOR SETA ( **->** )EM C++

**E AGORA ?**



# OPERADOR SETA ( -> ) EM C++

Como *p* é PONTEIRO, *SEMPRE* que eu desejo acessar o conteúdo, ou seja, o que tem dentro de *p*, devo usar asterisco, certo? **LEMBRA?**

Logo para acessar o membro *a* devo fazer:

*(\*p).a = 400;*

Nessa formato funciona. Ou seja, a posição de memória para onde *p* aponta, no seu campo *a* receberá o valor 400.

**OPERADOR SETA ( -> ) EM C++**

**LEGAL. ENTENDI.**

**E OPERADOR SETA ( -> ) ?**

# OPERADOR SETA ( -> ) EM C++

PARA ENTENDER A FORMA DE UTILIZAÇÃO VEJA O CÓDIGO EXEMPLO

```
#include<iostream>
using namespace std;

struct TESTE {
int a;
int b;
};

int main ()
{
struct TESTE *p, x;

p = &x;

(*p).a = 200;
(*p).b = 400;

cout << x.a << "\n\n" << x.b;

}
```

**OPERADOR SETA ( -> ) EM C++**

**SIM.**

**E OPERADOR SETA ( -> ) ?**

# OPERADOR SETA ( -> ) EM C++

PARA ENTENDER A FORMA DE UTILIZAÇÃO VEJA O CÓDIGO EXEMPLO

```
#include<iostream>
using namespace std;
```

```
struct TESTE {
int a;
int b;
};
```


```
int main ()
{
struct TESTE *p, x;
```

```
p = &x;
```

```
p->a = 200;
p->b = 400;
```

```
cout << x.a << "\n\n" << x.b;
}
```

```
(*p).a = 200;
(*p).b = 400;
```



# OPERADOR SETA ( -> ) EM C++

APÓS COMPILAR E EXECUTAR TEM-SE:

```
200
400
-----
Process exited after 3.001 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```