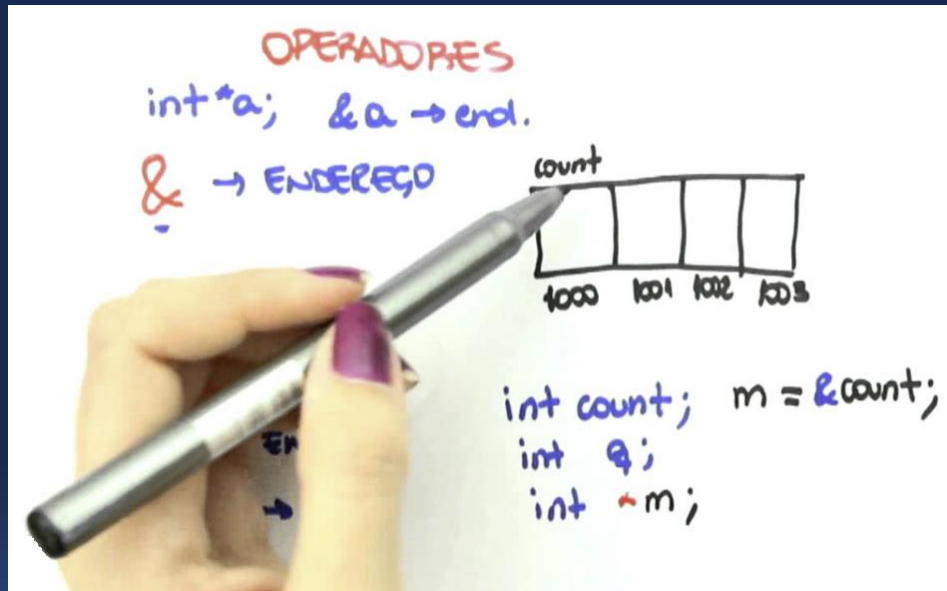




# Algoritmos e Estrutura de Dados I

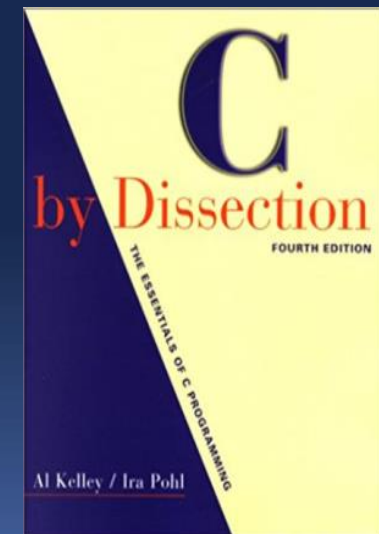
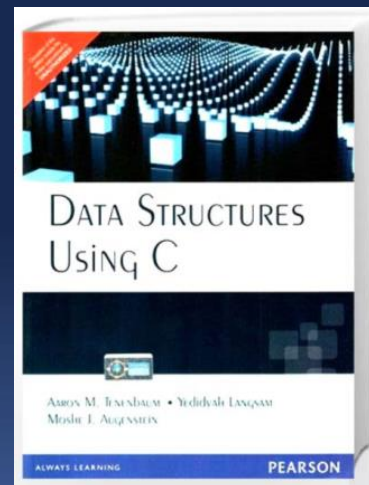
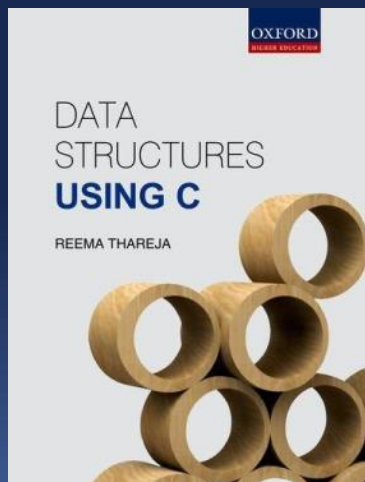
## Unidade 9 – Ponteiros



Prof. Aparecido V. de Freitas  
 Doutor em Engenharia  
 da Computação pela EPUSP  
[aparecido.freitas@online.uscs.edu.br](mailto:aparecido.freitas@online.uscs.edu.br)  
[aparecidovfreitas@gmail.com](mailto:aparecidovfreitas@gmail.com)

# Bibliografia

- ✓ Data Structures using C - Oxford University Press - 2014
- ✓ Data Structures Using C - A. Tenenbaum, M. Augensem, Y. Langsam, Pearson 1995
- ✓ C By Dissection - Kelley, Pohh - Third Edition - Addison Wesley



# Introdução

## ○ Variável

- É um espaço reservado de memória usado para guardar um *valor* que pode ser modificado pelo programa;

## ○ Ponteiro

- É um espaço reservado de memória usado para guardar o *endereço de memória* de uma outra variável.
- Um ponteiro é uma variável como qualquer outra do programa – sua diferença é que ela não armazena um valor inteiro, real, caractere ou booleano.
- Ela serve para armazenar endereços de memória (são valores inteiros sem sinal).

# Declaração de Ponteiros



- Como qualquer variável, um ponteiro também possui um tipo

```
//declaração de variável  
tipo_variável nome_variável;
```

```
//declaração de ponteiro  
tipo_ponteiro *nome_ponteiro;
```

- É o **asterisco** (\*) que informa ao compilador que aquela variável não vai guardar um valor mas sim um endereço para o tipo especificado.

```
int x;  
float y;  
struct ponto p;
```

```
int *x;  
float *y;  
struct ponto *p;
```



# Declaração de Ponteiros



## Exemplos de declaração de variáveis e ponteiros

```
int main() {  
    //Declara um ponteiro para int  
    int *p;  
    //Declara um ponteiro para float  
    float *x;  
    //Declara um ponteiro para char  
    char *y;  
    //Declara um ponteiro para struct ponto  
    struct ponto *p;  
    //Declara uma variável do tipo int e um ponteiro para int  
    int soma, *p2,;  
  
    return 0;  
}
```



# Declaração de Ponteiros



- Na linguagem C, quando declaramos um ponteiro nós informamos ao compilador para que tipo de variável vamos apontá-lo.
  - Um ponteiro int\* aponta para um inteiro, isto é, int;
  - Esse ponteiro guarda o endereço de memória onde se encontra armazenada uma variável do tipo int



# Inicialização de Ponteiros



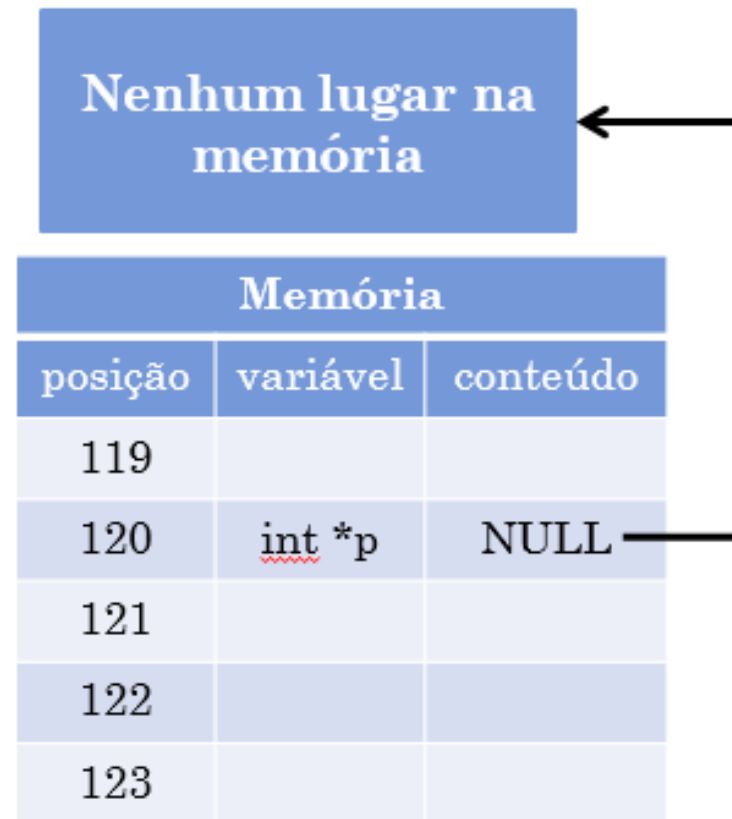
- Ponteiros apontam para uma posição de memória.
  - **Cuidado:** Ponteiros não inicializados apontam para um lugar indefinido.
- Exemplo
  - int \*p;

Memória		
posição	variável	conteúdo
119		
120	<u>int</u> *p	????
121		
122		
123		



# Inicialização de Ponteiros

- Um ponteiro pode ter o valor especial NULL que é o endereço de nenhum lugar.
- Exemplo
  - `int *p = NULL;`






# Inicialização de Ponteiros

- Os ponteiros devem ser inicializados antes de serem usados.
- Assim, devemos apontar um ponteiro para um lugar conhecido
  - Podemos apontá-lo para uma variável que já exista no programa.


Memória		
posição	variável	conteúdo
119		
120	<u>int</u> *p	122
121		
122	<u>int</u> c	10
123		



# Ponteiros

- O ponteiro armazena o endereço da variável para onde ele aponta.
  - Para saber o endereço de memória de uma variável do nosso programa, usamos o operador **&**.
  - Ao armazenar o endereço, o ponteiro estará apontando para aquela variável.

Memória		
posição	variável	conteúdo
119		
120	<u>int</u> *p	122
121		
122	<u>int</u> c	10
123		



```

int main() {
    //Declara uma variável int contendo o valor 10
    int c = 10;
    //Declara um ponteiro para int
    int *p;
    //Atribui ao ponteiro o endereço da variável int
    p = &c ;

    return 0;
}
    
```

# Uso de Ponteiros

- Tendo um ponteiro armazenado um endereço de memória, como saber o valor guardado dentro dessa posição?

# Uso de Ponteiros

- Para acessar o **valor** guardado dentro de uma posição na memória apontada por um ponteiro, basta usar o operador ***asterisco*** “\*” na frente do nome do ponteiro

# Uso de Ponteiros

```
int main() {  
    //Declara uma variável int contendo o valor 10  
    int c = 10;  
    //Declara um ponteiro para int  
    int *p;  
    //Atribui ao ponteiro o endereço da variável int  
    p = &c;  
    printf("Conteudo apontado por p: %d \n", *p); // 10  
    //Atribui um novo valor à posição de memória apontada por p  
    *p = 12;  
    printf("Conteudo apontado por p: %d \n", *p); // 12  
    printf("Conteudo de count: %d \n", c); // 10  
  
    return 0;  
}
```

# Uso de Ponteiros

- **\*p** :conteúdo da posição de memória apontado por **p**;
- **&c**: o endereço na memória onde está armazenada a variável **c**.

```
int main() {  
    //Declara uma variável int contendo o valor 10  
    int c = 10;  
    //Declara um ponteiro para int  
    int *p;  
    //Atribui ao ponteiro o endereço da variável int  
    p = &c;  
    printf("Conteudo apontado por p: %d \n", *p); // 10  
    //Atribui um novo valor à posição de memória apontada por p  
    *p = 12;  
    printf("Conteudo apontado por p: %d \n", *p); // 12  
    printf("Conteudo de count: %d \n", c); // 12  
  
    return 0;  
}
```

# Uso de Ponteiros

- De modo geral, um ponteiro só pode receber o endereço de memória de uma variável do mesmo tipo do ponteiro
  - Isso ocorre porque diferentes tipos de variáveis ocupam espaços de memória de tamanhos diferentes.
  - Na verdade, nós podemos atribuir a um ponteiro de inteiro (int \*) o endereço de uma variável do tipo float.
  - No entanto, o compilador assume que qualquer endereço que esse ponteiro armazene obrigatoriamente apontará para uma variável do tipo **int**.

# Uso de Ponteiros

```
int main() {
    int *p, *p1, x = 10;
    float y = 20.0;
    p = &x;
    printf("Conteudo apontado por p: %d \n", *p);

    p1 = p;
    printf("Conteudo apontado por p1: %d \n", *p1);

    p = &y;
    printf("Conteudo apontado por p: %d \n", *p);
    printf("Conteudo apontado por p: %f \n", *p);
    printf("Conteudo apontado por p: %f \n", *((float*)p));

    return 0;
}
```

```
Conteudo apontado por p: 10
Conteudo apontado por p1: 10
Conteudo apontado por p: 1101004800
Conteudo apontado por p: 0.000000
Conteudo apontado por p: 20.000000
```



# Operações com Ponteiros



## ○ Atribuição

- p1 aponta para o mesmo lugar que p2;

```
int *p, *p1;  
int c = 10;  
p = &c;  
p1 = p; //equivale a p1 = &c;
```

- a variável apontada por p1 recebe o mesmo conteúdo da variável apontada por p2;

```
int *p, *p1;  
int c = 10, d = 20;  
p = &c;  
p1 = &d;  
  
*p1 = *p; //equivale a d = c;
```



# Operações com Ponteiros



- Apenas duas operações aritméticas podem ser utilizadas com no endereço armazenado pelo ponteiro: adição e subtração
- podemos apenas somar e subtrair valores INTEIROS
  - $p++$ ;
    - soma +1 no endereço armazenado no ponteiro.
  - $p--$ ;
    - subtrai 1 no endereço armazenado no ponteiro.
  - $p = p+15$ ;
    - soma +15 no endereço armazenado no ponteiro.



# Operações com Ponteiros



○ As operações de adição e subtração no endereço dependem do tipo de dado que o ponteiro aponta.

- Considere um ponteiro para inteiro, int \*.
- O tipo int ocupa um espaço de 4 bytes na memória.
- Assim, nas operações de adição e subtração são adicionados/subtraídos 4 bytes por incremento/decremento, pois esse é o tamanho de um inteiro na memória e, portanto, é também o valor mínimo necessário para sair dessa posição reservada de memória

Memória		
posição	variável	conteúdo
119		
120	<u>int</u> a	10
121		
122		
123		
124	<u>int</u> b	20
125		
126		
127		
128	char c	'k'
129	char d	's'
130		



# Operações ilegais com Ponteiros



- Dividir ou multiplicar ponteiros;
- Somar o endereço de dois ponteiros;
- Não se pode adicionar ou subtrair valores dos tipos float ou double de ponteiros.



# Operações com Ponteiros

- Já sobre seu conteúdo apontado, valem todas as operações
  - `(*p)++;`
    - incrementar o conteúdo da variável apontada pelo ponteiro `p`;
  - `*p = (*p) * 10;`
    - multiplica o conteúdo da variável apontada pelo ponteiro `p` por 10;

```
int *p;  
int c = 10;
```

```
p = &c;
```

```
(*p)++;  
*p = (*p) * 10;
```

# Operações com Ponteiros

## ○ Operações relacionais

- == e != para saber se dois ponteiros são iguais ou diferentes.
- >, <, >= e <= para saber qual ponteiro aponta para uma posição mais alta na memória.

```
int main() {  
    int *p, *p1, x, y;  
    p = &x;  
    p1 = &y;  
    if (p == p1)  
        printf("Ponteiros iguais\n");  
    else  
        printf("Ponteiros diferentes\n");  
  
    return 0;  
}
```

# Ponteiros Genéricos

- Normalmente, um ponteiro aponta para um tipo específico de dado.
  - Um ponteiro genérico é um ponteiro que pode apontar para qualquer tipo de dado.
- Declaração

```
void *nome_ponteiro;
```

# Ponteiros Genéricos

## ○ Exemplos

```
int main() {  
    void *pp;  
    int *p1, p2 = 10;  
    p1 = &p2;  
    //recebe o endereço de um inteiro  
    pp = &p2;  
    printf("Endereco em pp: %p \n", pp);  
    //recebe o endereço de um ponteiro para inteiro  
    pp = &p1;  
    printf("Endereco em pp: %p \n", pp);  
    //recebe o endereço guardado em p1 (endereço de p2)  
    pp = p1;  
    printf("Endereco em pp: %p \n", pp);  
  
    return 0;  
}
```



# Ponteiros Genéricos

- Para acessar o **conteúdo** de um ponteiro genérico é preciso antes convertê-lo para o tipo de ponteiro com o qual se deseja trabalhar
  - Isso é feito vai *type cast*

```
int main() {  
    void *pp;  
    int p2 = 10;  
    // ponteiro genérico recebe o endereço de um  
    // inteiro  
    pp = &p2;  
    //enta acessar o conteúdo do ponteiro genérico  
    printf("Conteudo: %d\n", *pp); //ERRO  
    // converte o ponteiro genérico pp para (int *)  
    // antes de acessar seu conteúdo.  
    printf("Conteudo: %d\n", *(int*)pp); //CORRETO  
  
    return 0;  
}
```

# Ponteiros e Arrays

- Ponteiros e arrays possuem uma ligação muito forte.
  - Arrays são agrupamentos de dados do mesmo tipo na memória.
  - Quando declaramos um array, informamos ao computador para reservar uma certa quantidade de memória a fim de armazenar os elementos do array de forma sequencial.
  - Como resultado dessa operação, o computador nos devolve um ponteiro que aponta para o começo dessa sequência de bytes na memória.


# Ponteiros e Arrays

- O nome do array (sem índice) é apenas um ponteiro que aponta para o primeiro elemento do array.

```
int vet[5] = {1, 2, 3, 4, 5};
int *p;

p = vet;
```

Memória		
posição	variável	conteúdo
119		
120		
121	<u>int</u> *p	123
122		
123	<u>int</u> <u>vet[5]</u>	1
124		2
125		3
126		4
127		5
128		



# Ponteiros e Arrays

- Os colchetes [ ] substituem o uso conjunto de operações aritméticas e de acesso ao conteúdo (operador “\*”) no acesso ao conteúdo de uma posição de um array ou ponteiro.
  - O valor entre colchetes é o deslocamento a partir da posição inicial do array.
  - Nesse caso, **p[2]** equivale a **\*(p+2)**.

```
int main () {  
    int vet[5] = {1,2,3,4,5};  
    int *p;  
    p = vet;  
  
    printf("Terceiro elemento: %d ou %d", p[2], *(p+2));  
  
    return 0;  
}
```

# Ponteiros e Arrays

## ○ Nesse exemplo

```
int vet[5] = {1, 2, 3, 4, 5};  
int *p;  
  
p = vet;
```

## ○ Temos que:

- **\*p** é equivalente a vet[0];
- vet[índice] é equivalente a **\*(p+índice)**;
- vet é equivalente a &vet[0];
- &vet[índice] é equivalente a (vet + índice);

# Ponteiros e Arrays

## Usando array

```
int main() {  
    int vet[5] = {1, 2, 3, 4, 5};  
    int *p = vet;  
    int i;  
    for (i = 0; i < 5; i++)  
        printf("%d\n", p[i]);  
  
    return 0;  
}
```

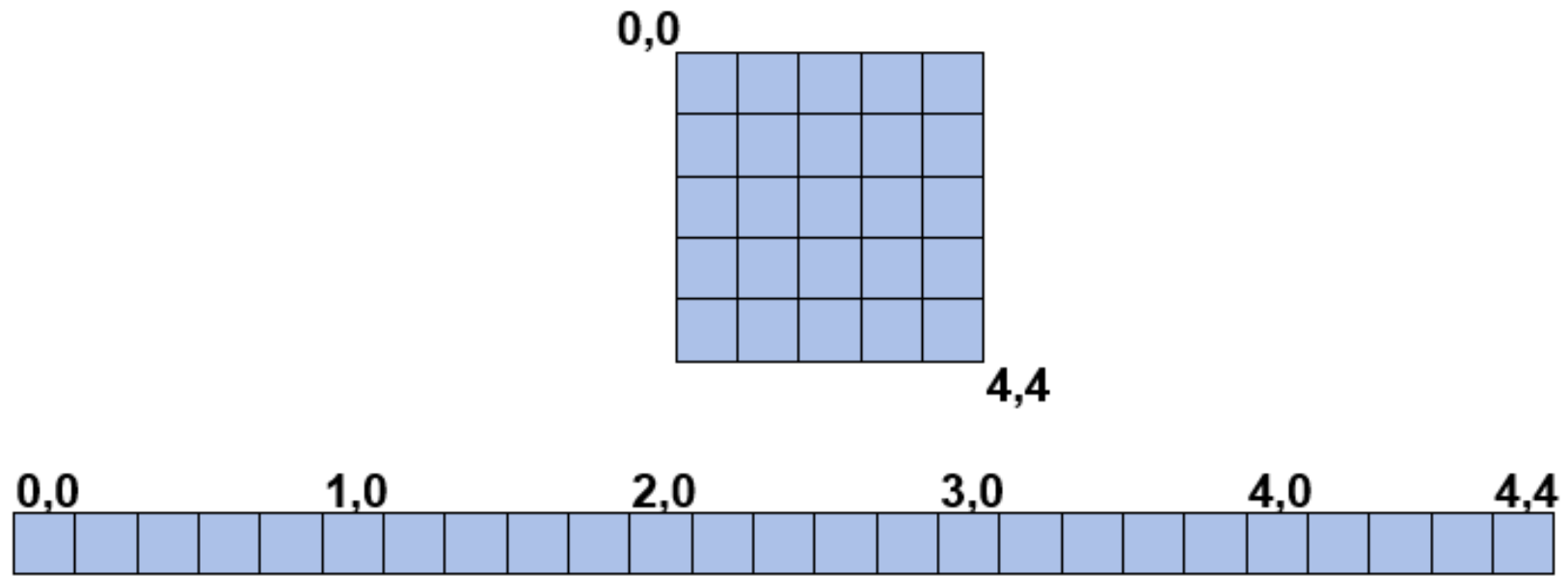
## Usando ponteiro

```
int main() {  
    int vet[5] = {1, 2, 3, 4, 5};  
    int *p = vet;  
    int i;  
    for (i = 0; i < 5; i++)  
        printf("%d\n", *(p+i));  
  
    return 0;  
}
```

# Ponteiros e Arrays

## ○ Arrays Multidimensionais

- Apesar de terem mais de uma dimensão, na memória os dados são armazenados linearmente.
- Ex.:
- int mat[5][5];



# Ponteiros e Arrays

- Pode-se então percorrer as várias dimensões do array como se existisse apenas uma dimensão. As dimensões mais à direita mudam mais rápido

## Usando array

```
int main() {
    int mat[2][2] = {{1,2},{3,4}};
    int i,j;
    for(i=0;i<2;i++)
        for(j=0;j<2;j++)
            printf("%d\n", mat[i][j]);

    return 0;
}
```

## Usando ponteiro

```
int main() {
    int mat[2][2] = {{1,2},{3,4}};
    int *p = &mat[0][0];
    int i;
    for(i=0;i<4;i++)
        printf("%d\n", *(p+i));

    return 0;
}
```



# Ponteiro para struct

- Existem duas abordagens para acessar o conteúdo de um ponteiro para uma struct
- Abordagem 1
  - Devemos acessar o conteúdo do ponteiro para struct para somente depois acessar os seus campos e modificá-los.
- Abordagem 2
  - Podemos usar o *operador seta* “->”
  - ponteiro->nome\_campo

```
struct ponto {  
    int x, y;  
};
```

```
struct ponto q;  
struct ponto *p;
```

```
p = &q;
```

```
(*p).x = 10;  
p->y = 20;
```

# Ponteiro para ponteiro

- A linguagem C permite criar ponteiros com diferentes níveis de apontamento.
  - É possível criar um ponteiro que aponte para outro ponteiro, criando assim vários níveis de apontamento
  - Assim, um ponteiro poderá apontar para outro ponteiro, que, por sua vez, aponta para outro ponteiro, que aponta para um terceiro ponteiro e assim por diante.

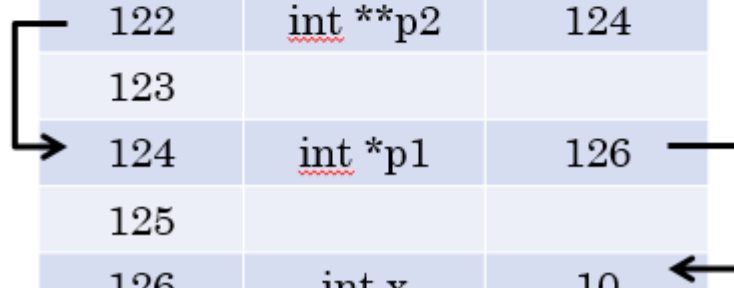
# Ponteiro para ponteiro

- Um ponteiro para um ponteiro é como se você anotasse o endereço de um papel que tem o endereço da casa do seu amigo.
- Podemos declarar um ponteiro para um ponteiro com a seguinte notação
  - tipo\_ponteiro \*\*nome\_ponteiro;
- Acesso ao conteúdo
  - \*\*nome\_ponteiro é o conteúdo final da variável apontada;
  - \*nome\_ponteiro é o conteúdo do ponteiro intermediário.

# Ponteiro para ponteiro

```
int x = 10;
int *p1 = &x;
int **p2 = &p1;
//Endereço em p2
printf("Endereco em p2: %p\n", p2);
//Conteúdo do endereço
printf("Conteúdo em *p2: %p\n", *p2);
//Conteúdo do endereço do endereço
printf("Conteúdo em **p2: %d\n", **p2);
```

Memória		
posição	variável	conteúdo
119		
120		
121		
122	<u>int</u> **p2	124
123		
124	<u>int</u> *p1	126
125		
126	<u>int</u> x	10
127		



# Ponteiro para ponteiro

- É a quantidade de asteriscos (\*) na declaração do ponteiro que indica o número de níveis de apontamento que ele possui.

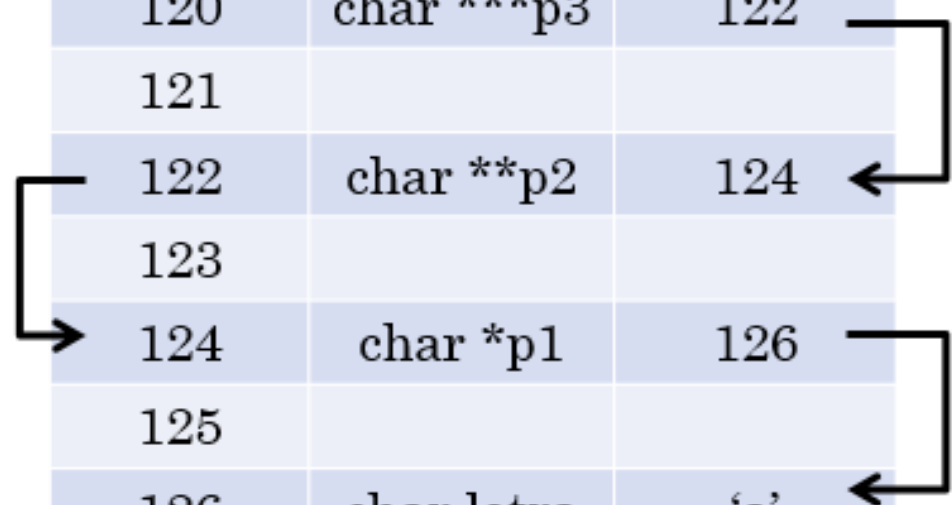
```
//variável inteira
int x;
//ponteiro para um inteiro (1 nível)
int *p1;
//ponteiro para ponteiro de inteiro (2 níveis)
int **p2;
//ponteiro para ponteiro para ponteiro de inteiro (3 níveis)
int ***p3;
```

# Ponteiro para ponteiro

```
char letra = 'a';
char *p1;
char **p2;
char ***p3;
```

```
p1 = &letra;
p2 = &p1;
p3 = &p2;
```

Memória		
posição	variável	conteúdo
119		
120	char ***p3	122
121		
122	char **p2	124
123		
124	char *p1	126
125		
126	char letra	'a'
127		



# Exemplos

```
/* ----- */
/* --- Exercícios com Ponteiros --- */
/* ----- ponteiro_01.c ----- */
/* ----- */

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

int main() {

    setlocale(LC_ALL, "Portuguese");
    printf("----- Início do Programa ----- \n\n");
    int x;
    double y;
    char z;

    int * p1;
    double * p2;
    char * p3;

    x = 10;
    y = 5.8;
    z = 'A';

    p1 = &x;
    p2 = &y;
    p3 = &z;

    printf ("x = %d\n\n", *p1);
    printf ("y = %.1f\n\n", *p2);
    printf ("z = %c\n\n", *p3);

    printf("----- Fim do Programa ----- \n\n");

}
```



```
/* ----- */
/* --- Exercícios com Ponteiros --- */
/* ----- ponteiro_02.c ----- */
/* ----- */

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

int main() {

    setlocale(LC_ALL, "Portuguese");
    printf("----- Início do Programa -----\\n\\n");
    int x, y, resp ;

    int * p1;
    int * p2;
    int * p3;

    x = 10;
    y = 20;

    p1 = &x;
    p2 = &y;

    resp = *p1 * *p2;

    p3 = &resp;

    printf ("resp = %d\\n\\n", *p3);

    printf("----- Fim do Programa -----\\n\\n");

}
```

```
/* --- Exercícios com Ponteiros --- */
/* ----- ponteiro_03.c ----- */
/* ----- */

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

int main() {

    setlocale(LC_ALL, "Portuguese");
    printf("----- Início do Programa -----\\n\\n");
    int x, y, resp ;

    int * p1;
    int * p2;
    int * p3;

    x = 10;
    y = 20;

    p1 = &x;
    p2 = &y;

    p3 = p1;
    p2 = p3;

    resp = *p3 * *p2;

    p3 = &resp;

    printf ("resp = %d\\n\\n", *p3);

    printf("----- Fim do Programa -----\\n\\n");

}
```

```
/* --- Exercícios com Ponteiros -- */
/* ----- ponteiro_04.c ----- */
/* ----- */

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

int main() {

    setlocale(LC_ALL, "Portuguese");
    printf("----- Início do Programa -----\\n\\n");
    int x, y, z, resp ;

    int * p1;
    int * p2;
    int * p3;

    x = 10;
    y = 20;
    z = 5;

    p1 = &x;
    p2 = &y;
    p3 = &z;

    *p2 = *p1;
    *p3 = *p2;

    resp = *p3 + (*p2 * *p3) + ++(*p1);

    p3 = &resp;

    printf ("resp = %d\\n\\n", *p3);

    printf("----- Fim do Programa -----\\n\\n");

}
```

```
/* ----- */
/* --- Exercícios com Ponteiros --- */
/* ----- ponteiro_05.c ----- */
/* ----- */

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

int main() {

    setlocale(LC_ALL, "Portuguese");
    printf("----- Início do Programa -----\\n\\n");
    int x, y, z, resp ;

    int vet[5] = { 1,2,3,4,5}, i;
    int *p;
    p = vet;

    for (i = 0; i < 5; i++)
        printf ("%d\\n\\n", *(p + i) );

    printf("----- Fim do Programa -----\\n\\n");

}
```

```
/* ----- */
/* --- Exercícios com Ponteiros --- */
/* ----- ponteiro_06.c ----- */
/* ----- */

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

int main() {

    setlocale(LC_ALL, "Portuguese");
    printf("----- Início do Programa -----\\n\\n");
    int x, y, z, resp ;

    int vet[5] = { 1,2,3,4,5}, i;
    int *p;
    p = vet;

    for (i = 0; i < 5; i++) {
        printf ("%d\\n\\n", *p );
        p++;
    }

    printf("----- Fim do Programa -----\\n\\n");

}
```

```
/* ----- */
/* --- Exercícios com Ponteiros --- */
/* ----- ponteiro_07.c ----- */
/* ----- */

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

int main() {

    setlocale(LC_ALL, "Portuguese");
    printf("----- Início do Programa -----\\n\\n");
    int x, y, z, resp ;

    int mat[2][2] = { {1,2} , {3,4} }, i;
    int *p;
    p = mat;

    for (i = 0; i < 4; i++) {
        printf ("%d\\n\\n", *p );
        p++;
    }

    printf("----- Fim do Programa -----\\n\\n");

}
```

```
/* ----- */
/* --- Exercícios com Ponteiros -- */
/* ----- ponteiro_08.c ----- */
/* ----- */

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

int main() {

    setlocale(LC_ALL, "Portuguese");
    printf("----- Início do Programa -----\\n\\n");

    int numero = 10, resp;

    int * p1 = &numero;

    int ** p2 = &p1;

    resp = **p2;

    printf ("%d\\n\\n", resp);

    printf("----- Fim do Programa -----\\n\\n");

}
```