# MCTA028-15: Programação Estruturada

## Aula 8: Ponteiros
## (Segunda Parte)

*Wagner Tanaka Botelho*
*wagner.tanaka@ufabc.edu.br / wagtanaka@gmail.com*
*Universidade Federal do ABC (UFABC)*
*Centro de Matemática, Computação e Cognição (CMCC)*

# Ponteiros e *Arrays*

# Ponteiros e *Arrays*

- Como já estudamos, *arrays* são agrupamentos de dados do mesmo tipo na memória;
- Quando um *array* é declarado, o computador reserva uma quantidade de memória para armazenar os elementos do *array* de forma SEQUENCIAL:
  - Como resultado, o computador devolve um PONTEIRO que aponta para o COMEÇO dessa sequência de *bytes* na memoria.
- Na Linguagem C, o NOME de um *array* **SEM ÍNDICE** guarda o endereço para o começo do *array* na memória, ou seja, ele guarda o endereço do início de uma área de armazenamento dentro da memória:
  - Portanto, as **OPERAÇÕES** envolvendo *arrays* podem ser feitas utilizando PONTEIROS e ARITMÉTICA DE PONTEIROS.

# Acessando um *Array*

## Utilizando Ponteiros

Ex_01.c

```c
#include <stdio.h>
#include <stdlib.h>

void 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]);
    }
}
```

Notação de Colchetes para acessar os elementos da *array*.

Watches

| Function arguments | |
|---|---|
| Locals | |
| vet | |
| [0] | 1 |
| [1] | |
| [2] | |
| [3] | |
| [4] | 5 |
| p | 0x61fdf0 |
| i | 0 |
| &vet | (int (*)[5]) 0x61fdf0 |

p está apontando para o endereço de vet

## Utilizando Aritmética de Ponteiros

Ex_02.c

```c
#include <stdio.h>

void 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));
    }
}
```

Observa-se que NÃO foi necessário usar o & antes de y. Vetor ou matriz, ao fazer esse tipo de atribuição, o retorno padrão é o ENDEREÇO DE MEMÓRIA relativo à primeira posição, pois não foi indicada nenhuma posição.

Notação de Aritmética de Ponteiros para acessar os elementos da *array*.

- Para atribuir o endereço do `array` para o **ponteiro**, pode-se realizar de duas formas:

```
int *p = vet;
int *p = &vet[0];
```

Ex_01.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main(){
5      int vet[5]= {1,2,3,4,5};
6      int *p = vet;
7      int i;
8
9      for (i = 0;i < 5;i++){
10         printf("%d\n",p[i]);
11     }
12 }
```

**OU** `int *p = &vet[0];`

# Equivalências Entre *Arrays* e Ponteiros

Ex_03.c

```c
#include <stdio.h>

void main(){
    int vet[5] = {1,2,3,4,5};
    int *p = NULL, indice = 2;

    p = vet;

    printf("%d\n",*p);
    printf("%d\n",vet[0]);

    printf("%d\n",vet[indice]);
    printf("%d\n",*(p+indice));

    printf("%d\n",vet);
    printf("%d\n",&vet[0]);

    printf("%d\n",&vet[indice]);
    printf("%d\n",(vet+indice));
}
```

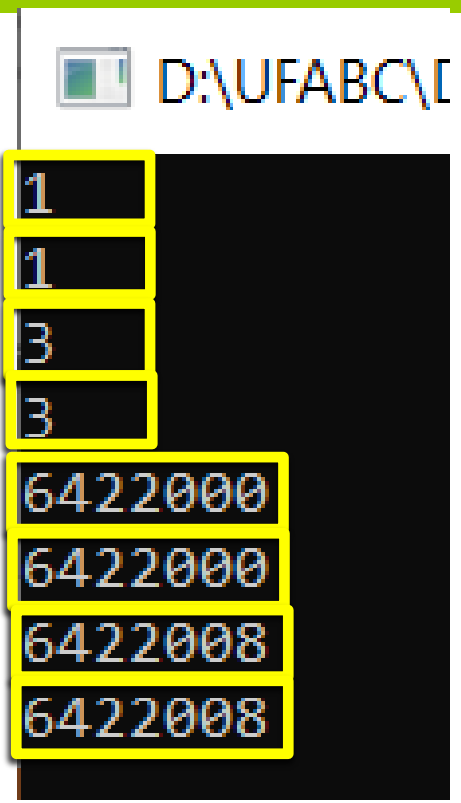
Pode ser: *p=vet; ou *p=&vet[0];

*p é equivalente a vet[0];

vet[indice] é equivalente a *(p+índice);

vet é equivalente a *vet[0];

&vet[indice] é equivalente a (vet+índice);



# Resumo

Ex_04.c

```c
#include<stdio.h>

void main(){
    int *pNum;
    int vNum[4];

    pNum = &vNum[0];
    *pNum = 3;

    pNum = vNum;
    pNum[2] = 9;

    printf("Pos. 0: %d \n", vNum[0]);
    printf("Pos. 2: %d \n", vNum[2]);
}
```

Declarando o ponteiro (pNum)

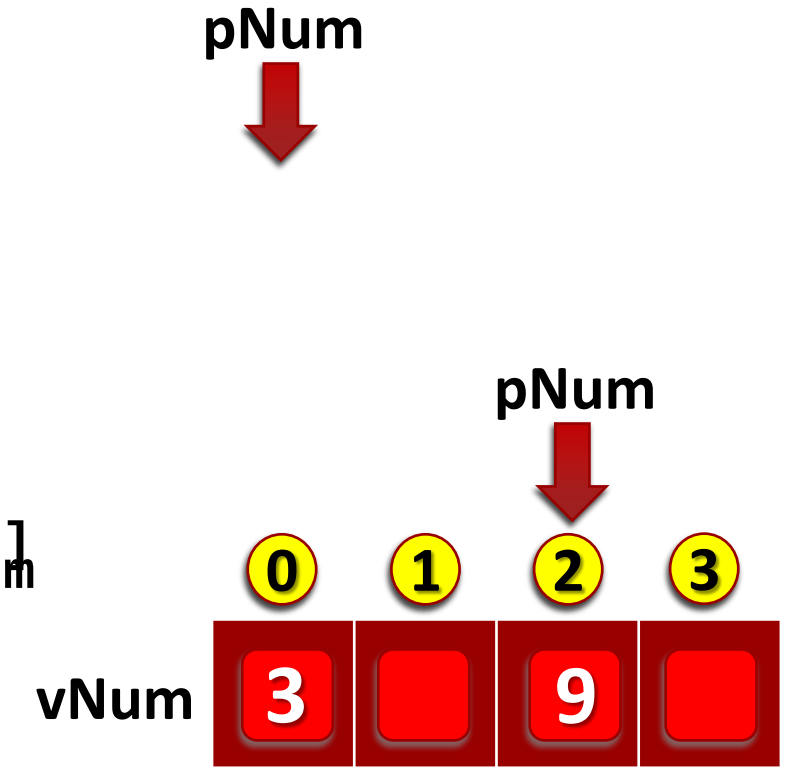
Declarando o vetor (vNum)

Aponta o **pNum** para a **POSIÇÃO 0** do **vNum**

Atribui o **3** na posição do **vNum** apontada por **pNum**

Aponta **pNum** para o **vNum[0]**

Atribui o **9** na posição do **vNum** apontada por **pNum**




# *Arrays* e Sequência de Caracteres (`Strings`)

# *Arrays* e `Strings`

- `String`:
  - Nome utilizado para definir uma sequência de caracteres adjacentes na memória do computador.
  - PALAVRA ou FRASE armazenada na memória do computador na forma de um *ARRAY* do tipo **CHAR**.
- Trabalhar com *arrays* de caracteres é simples:
  - Deve-se utilizar os conceitos já apresentados, por exemplo, em um *array* de inteiros.

Ex_05.c

```
1  #include<stdio.h>
2  void main(){
3      char *pC;
4      char vC[6]={"UFABC"};
5      int i=0;
6
7      pC = &vC[1];
8      *pC = 'X';
9
10     pC = vC;
11     pC[3] = 'X';
12
13     while(pC[i]!='\0'){
14         printf("%c ", pC[i]);
15         i++;
16     }
17 }
```

Declarando o ponteiro (pC)

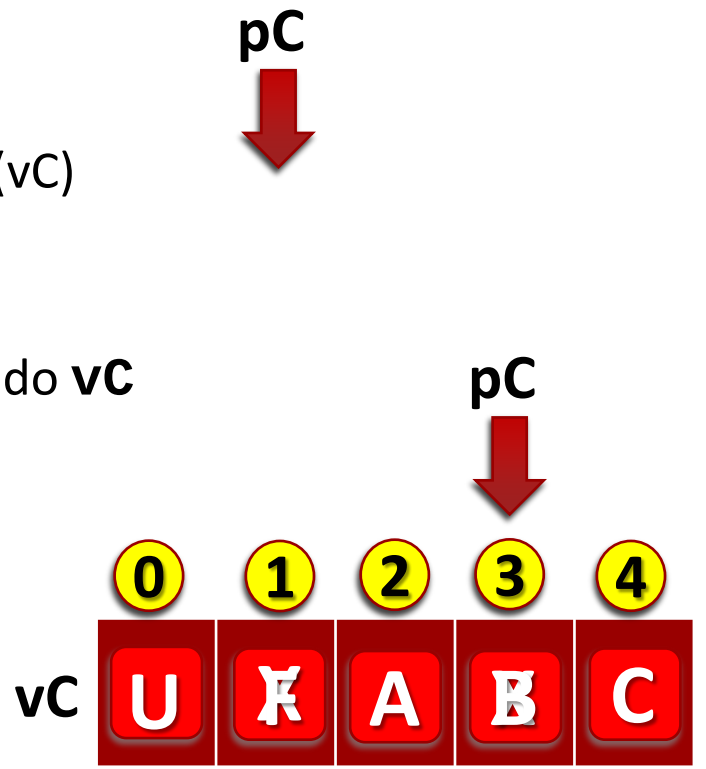
Declarando o vetor (vC)

Aponta o **pC** para a **POSIÇÃO [1]** do **vC**

Atribui o **'X'** na posição [1] do **vC** apontada por **pC**

Aponta **pC** para o **vC**

Atribui o **'X'** na posição [3] do **vC** apontada por **pC**




# *Arrays* Multidimensionais

# Ponteiros e Arrays Multidimensionais

- Apesar de terem o comportamento de estruturas com mais de uma dimensão, os dados dos *arrays* multidimensionais são armazenados LINEARMENTE na memória;

- O uso dos COLCHETES cria a impressão de estarmos trabalhando com MAIS de uma DIMENSÃO:
  - Por exemplo, a matriz `int mat[3][3]`, apesar de ser bidimensional, é armazenada como um **SIMPLES** *array* na memória:

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 5 | 7 | 5 |
| 1 | 1 | 9 | 7 |
| 2 | 0 | 2 | 1 |

# Acessando os Elementos do *Array*

- Os elementos podem ser acessados:
  - Usando a notação tradicional (**`mat[linha][coluna]`**);
  - Notação por ponteiros.
- Ponteiros permitem percorrer várias dimensões de um *array* multidimensional como se existisse apenas **UMA** dimensão;
- **NOTAÇÃO** por ponteiros:

```
*(X + coluna)
```

## Usando *Array*

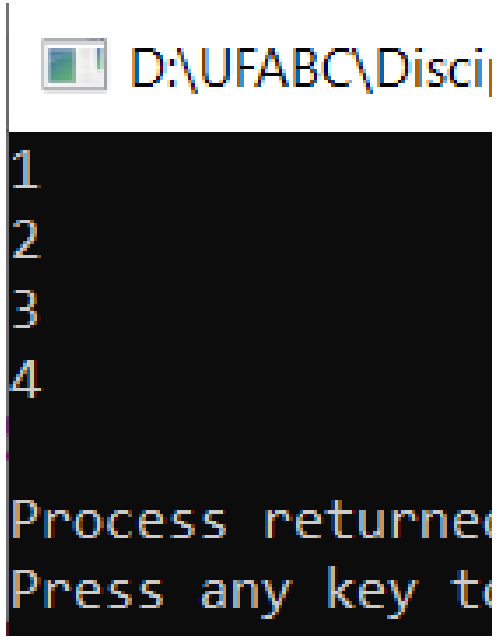
Ex_06.c

```c
#include <stdio.h>

void 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]);
        }
    }
}
```

Estudamos na aula sobre matrizes: `i` percor-a linha e `j` a coluna.




## Usando Ponteiro

Ex_07.c

```c
#include <stdio.h>

void main(){
    int mat[2][2] = {{1,2},{3,4}};
    int *p = &mat[0][0];

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

Somente UM `for` para percorrer as linhas e colunas da matriz?

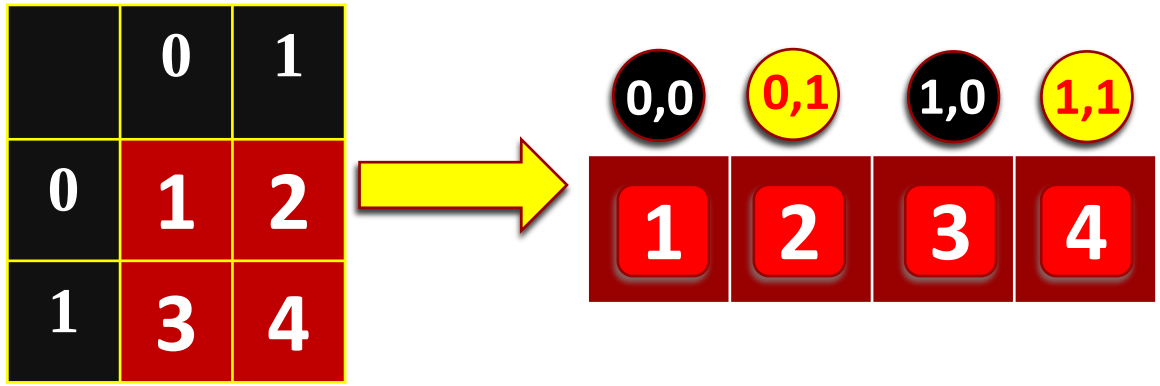
Ex_18.c

# Usando Ponteiro

```c
#include <stdio.h>

void 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));
    }
}
```



# *Arrays* de Ponteiros

# *Array* de Ponteiro

- A Linguagem C permite que *arrays* de ponteiros sejam declarados, assim como qualquer outro tipo de dado;
- A declaração segue a seguinte forma:

```
tipo_dado *nome_array[tamanho];
```

- Por exemplo, a declaração de um *array* de ponteiros para **INTEIROS** de tamanho **10**:

```
int *p[10];
```

# *Array* de Ponteiro

- Para atribuir o endereço de uma variável **x** a uma posição do *array* de ponteiros:

```
p[indice] = &x;
```

- Para retornar o **CONTEÚDO** guardado na posição de memória:

```
*p[índice]
```

Ex_08.c

```c
#include <stdio.h>

void main(){
    int *pvet[2];
    int x = 10;
    int y[2] = {20,30};

    pvet[0] = &x;
    pvet[1] = y;

    printf("Endereco pvet[0]: %p\n",pvet[0]);
    printf("Endereco pvet[1]: %p\n",pvet[1]);

    printf("Conteudo em pvet[0]: %d\n",*pvet[0]);
    printf("Conteudo pvet[1][1]: %d\n",pvet[1][1]);
}
```

Declarando um array de ponteiro (pVet)

Declarando uma variável (x) do tipo inteiro

Declarando vetor y do tipo inteiro

Atribui ao pvet o **ENDEREÇO** da variável x (int)

Atribui ao pvet o **ENDEREÇO** da array y

Observa-se que NÃO foi necessário usar o & antes de y. Vetor ou matriz, ao fazer esse tipo de atribuição, o retorno padrão é o ENDEREÇO DE MEMÓRIA relativo à primeira posição, pois não foi indicada nenhuma posição.

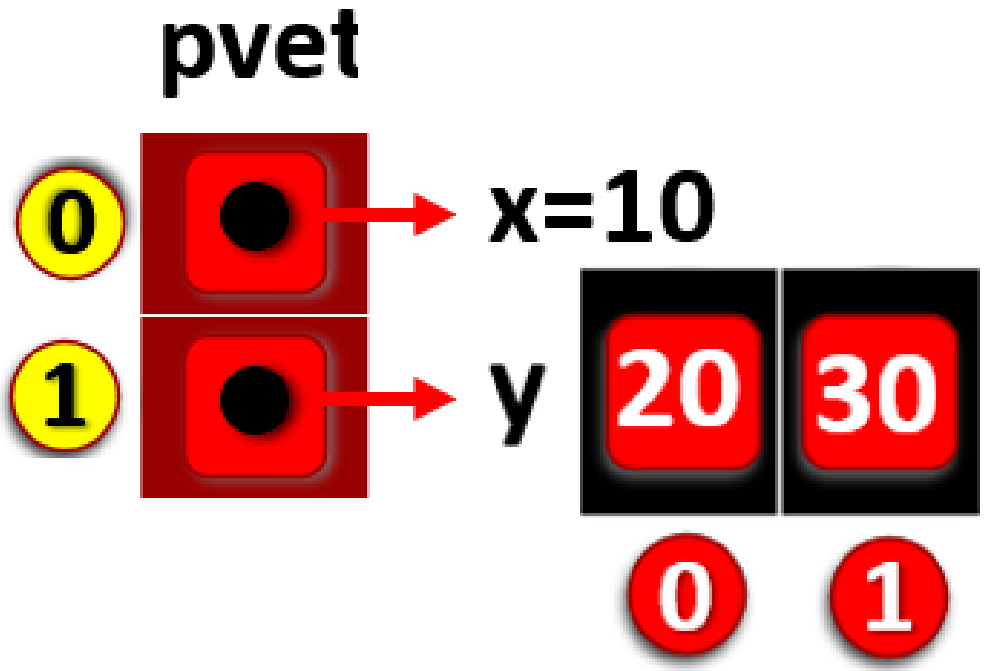
```
D:\UFABC\Disciplinas\2021-2025\Q1\PE\A
Endereco pvet[0]: 000000000061FE0C
Endereco pvet[1]: 000000000061FE04
Conteudo em pvet[0]: 10
Conteudo pvet[1][1]: 30
```

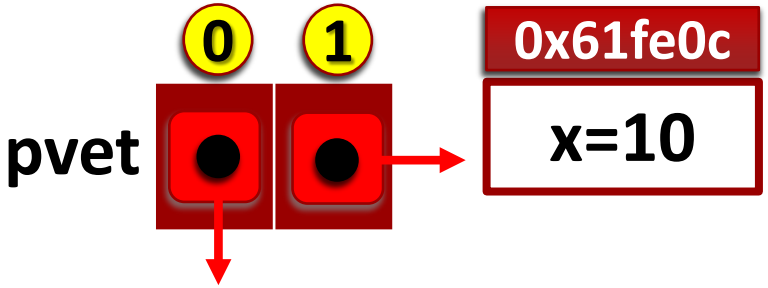
*Ex_09.c

```c
#include <stdio.h>

void main(){
    int *pvet[2];
    int x = 10;
    int y[2] = {20,30};

    pvet[0] = &x;
    pvet[1] = y;
}
```





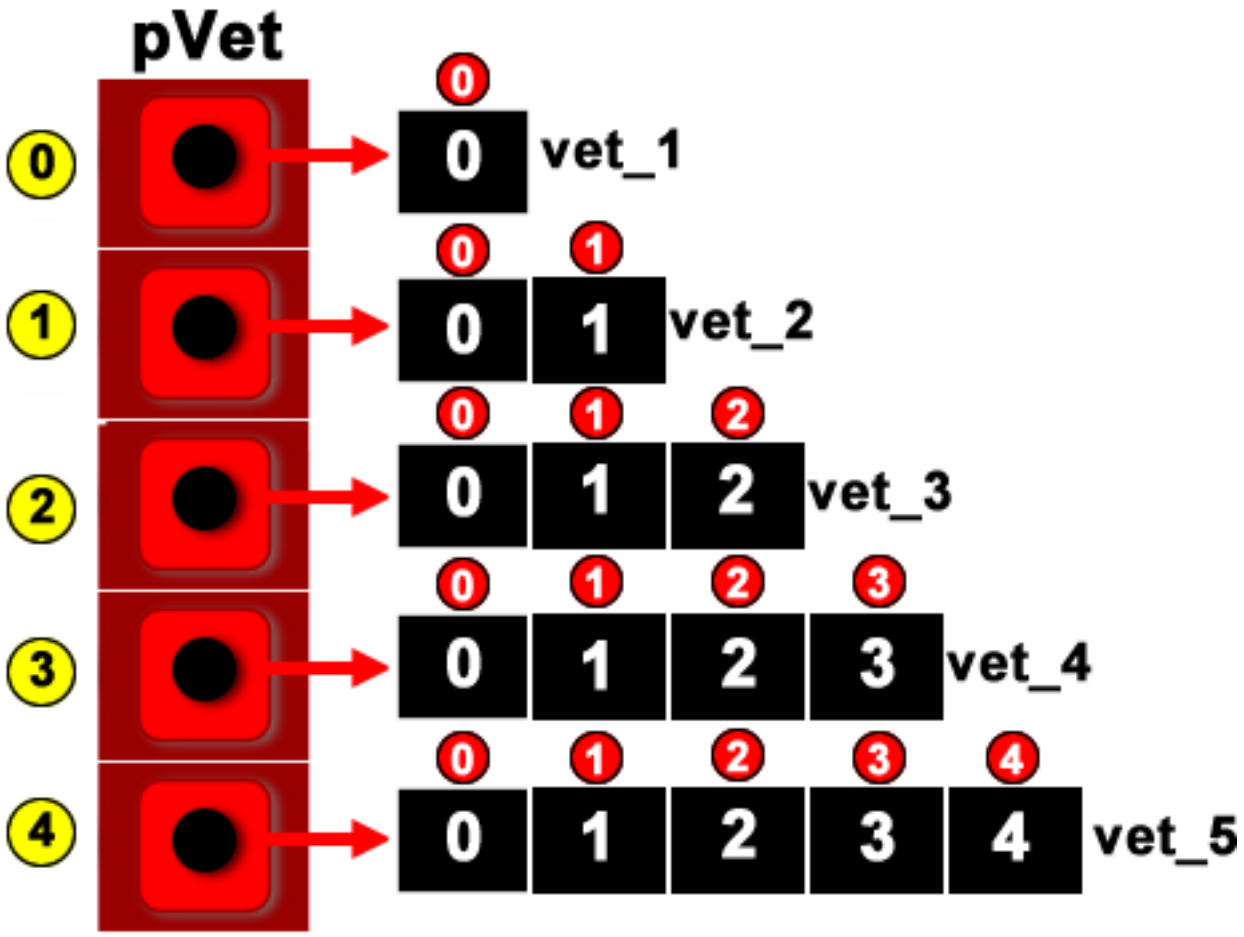
Watches

| | |
|---|---|
| [1] | 30 |
| &x | (int *) 0x61fe0c |
| &y | (int (*)[2]) 0x61fe04 |

```c
printf("%d\n",pvet[1][0]);
```

```c
printf("%d\n",pvet[1][1]);
```

**Conseguem imaginar que é possível implementar estruturas um pouco mais avançadas?**



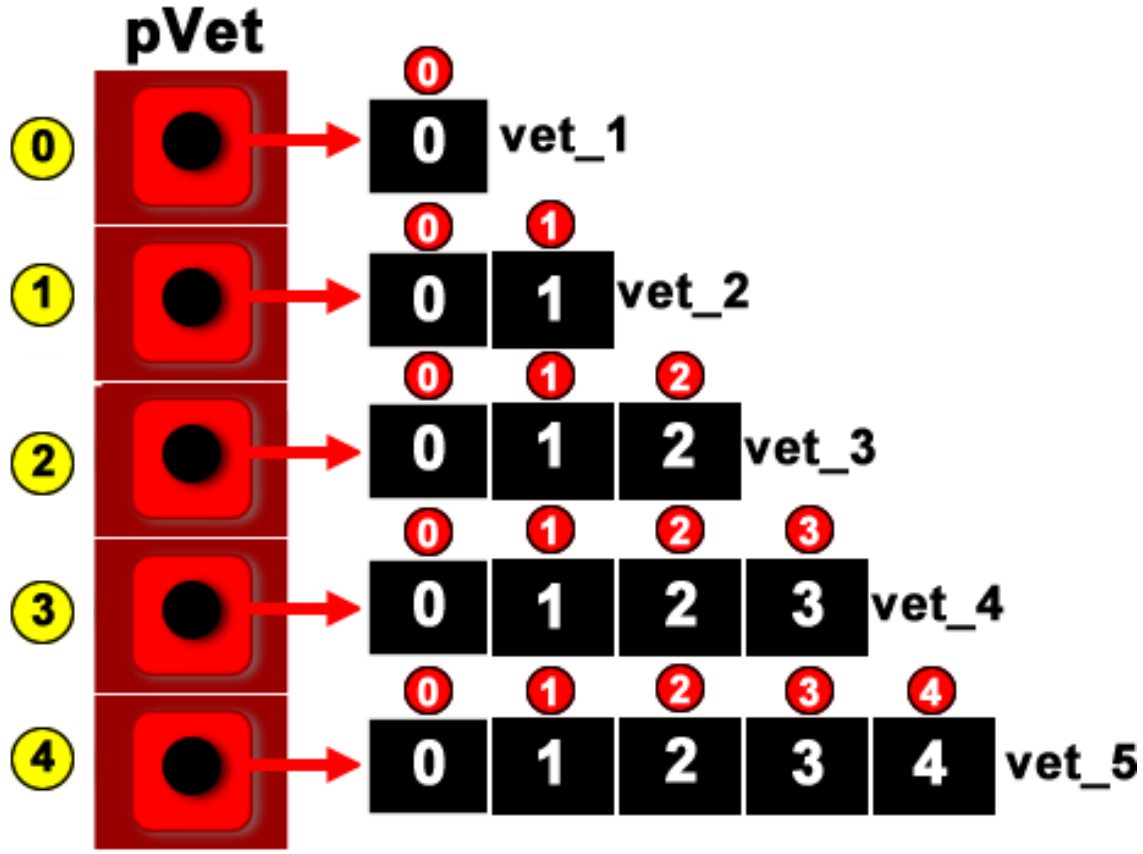
*Ex_10.c

```c
#include<stdio.h>

void main (){
    int *pVet[5];
    int vet_1[1] = {0};
    int vet_2[2] = {0,1};
    int vet_3[3] = {0,1,2};
    int vet_4[4] = {0,1,2,3};
    int vet_5[5] = {0,1,2,3,4};

    pVet[0] = &vet_1;
    pVet[1] = &vet_2;
    pVet[2] = &vet_3;
    pVet[3] = &vet_4;
    pVet[4] = &vet_5;

    printf("%d\n", pVet[0][0]);
    printf("%d\n", pVet[3][3]);
}
```



# Ponteiro para Ponteiro

# Ponteiro para Ponteiro

- A Linguagem C permite criar ponteiros com diferentes níveis de apontamento, ou seja, ponteiros que apontam para outros ponteiros;
- A declaração de um ponteiro para ponteiro é:

```
tipo_do_ponteiro **nome_do_ponteiro;
```

O `nome_do_ponteiro` **NÃO** vai guardar um VALOR, mas **SIM** um ENDEREÇO DE MEMÓRIA para **OUTRO** ENDEREÇO DE MEMÓRIA para aquele tipo especificado.

Ex_11.c

```c
#include <stdio.h>

void main(){
    int x = 10;
    int *p = &x;
    int **p2 = &p;

    printf("Endereco em p2: %p\n",p2);
    printf("Conteudo em *p2: %p\n",*p2);
    printf("Conteudo em **p2: %d\n",**p2);
}
```

Declarando um variável (x)

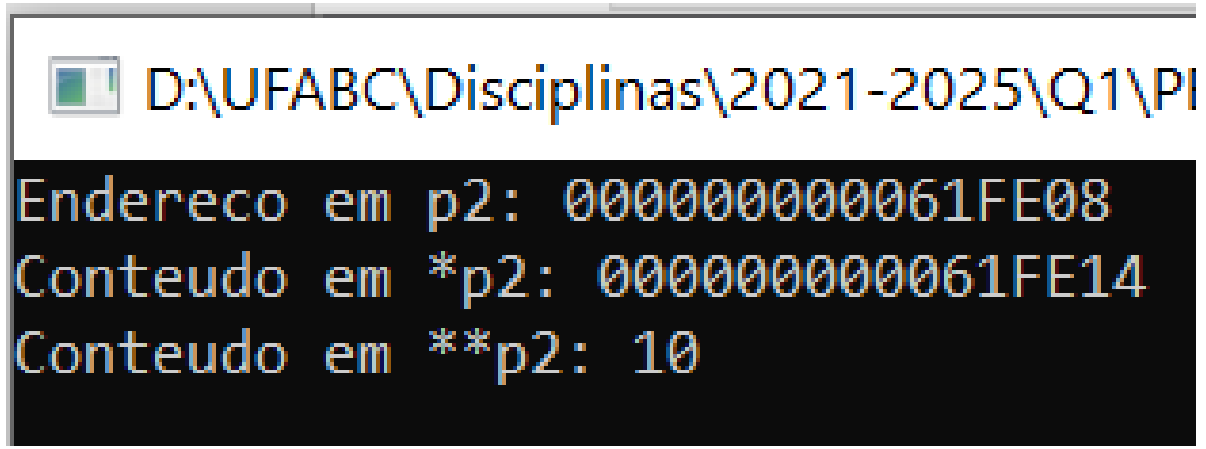
Atribui ao p o **ENDEREÇO** da variável x

Atribui ao p2 (ponteiro para ponteiro) o **ENDEREÇO** de p

Como *p2* é um **PONTEIRO** para **PONTEIRO**. Significa que o seu **CONTEÚDO** pode ser acessado **DUAS** vezes.

**MEMÓRIA**

| Endereço | Var | Conteúdo |
|---|---|---|
| 0x61fe14 | int x | 10 |
| 0x61fe08 | int *p | 0x61fe14 |
| 0x61fe18 | int **p2 | 0x61fe08 |



# Ponteiro para Ponteiro

- A Linguagem C permite criar um ponteiro que aponte para outro ponteiro, que aponte para outro ponteiro, etc:
  - Formando diferentes níveis de apontamento ou endereçamento.

*Ex_12.c

```
1  #include <stdlib.h>
2  void main(){
3      int x;
4      int *p1;
5      int **p2;
6      int ***p3;
7  }
```

# Referências


- Slides do Prof. Luiz Rozante;
- SALES, André Barros de; AMVAME-NZE, Georges. Linguagem C: roteiro de experimentos para aulas práticas. 2016;
- BACKES, André. Linguagem C Completa e Descomplicada. Editora Campus. 2013;
- SCHILDT, Herbert. C Completo e Total. Makron Books. 1996;
- DAMAS, Luís. Linguagem C. LTC Editora. 1999;
- DEITEL, Paul e DEITEL, Harvey. C Como Programar. Pearson. 2011.