LTP2

Ponteiros

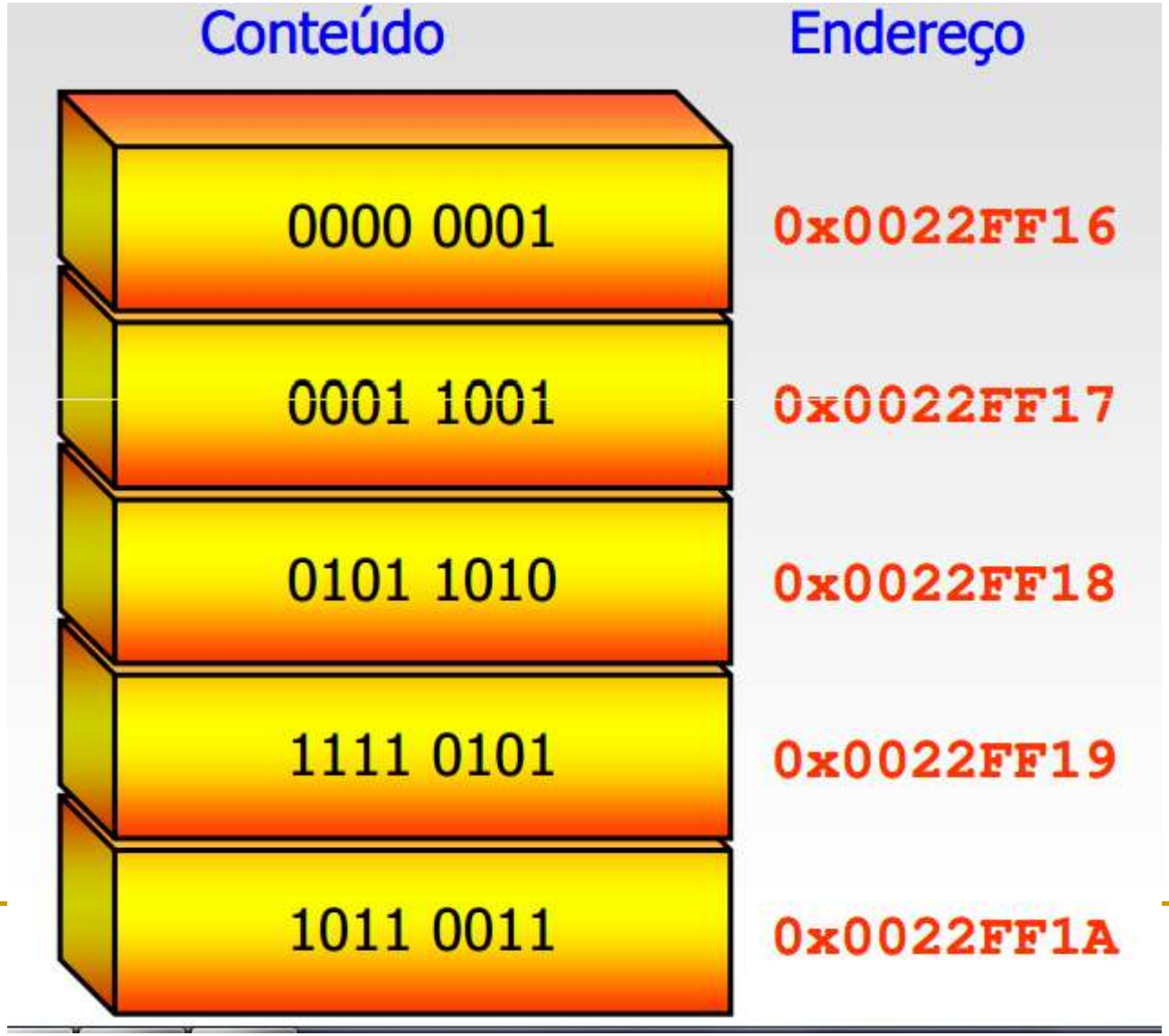
Profa.: Márcia Sampaio Lima

EST - UEA

Ponteiros

- A memória de um computador é uma seqüência de bytes.
- Casa byte pode armazenar um número inteiro entre 0 e 255.
- Cada byte, na memória, é identificado por um endereço numérico, independente do seu conteúdo.

Ponteiros


Conteúdo
Endereço
0000 0001
0x0022FF16
0001 1001
0x0022FF17
0101 1010
0x0022FF18
1111 0101
0x0022FF19
1011 0011
0x0022FF1A

Ponteiros

- Cada objeto (variáveis, strings, vetores, etc) que reside na memória do computador ocupa um certo número de bytes:
  - Inteiros: 4 bytes consecutivos.
  - Caracter: 1 byte
  - Float : 4 bytes consecutivos.

- Cada objeto tem um endereço.

Ponteiros

```
int x = 100;
```

- Na declaração acima temos associado:
  - Um nome (x)
  - Um endereço de memória (0xbfd267c4)
  - Um valor (100)
- Para acessar o endereço de uma variável usamos o operador `&`.

Ponteiro

- Um ponteiro é um tipo especial de variável:
  - Seu valor é um endereço de memória.
  - **`&x` representa o endereço de memória da variável `x`.**

- NULL
  - Valor especial que representa ausência de conteúdo em variáveis do tipo ponteiro.
    - Ou, “aponta pra nenhum lugar”
  - Definida na **`stdlib.h`**

Ponteiro

- Utilizadas por 3 razões específicas na programação:
  - Permitem a modificação de argumentos de funções:
    - Permitem que uma função altere valores de variáveis não globais e não locais a ela, através da referência ao endereço de memória da variável passada como parâmetro para a função.
  - Permitem o uso de rotinas de **alocação dinâmica de memória**:
    - alocação e liberação de memória em tempo de execução conforme a necessidade do programa.
  - Aumento de eficiência em determinadas rotinas.

Ponteiros

- Sintaxe de declaração de uma variável ponteiro é:

```
tipo  *nome_variável
```

  - onde *tipo* é o tipo de variável apontada pelo ponteiro.

  - Por exemplo:

```
float  *p;    //p é um ponteiro p/ float
```

- Ou seja, **`p`** apontará para uma região de memória que armazena um valor **`float`**.

Ponteiros

- Existem dois operadores para trabalhar com ponteiros. São eles:
  - * - conteúdo do endereço apontado por
  - & - endereço de

- ATENÇÃO: Quando * é usado na declaração de uma variável ele simplesmente indica que a variável é do tipo ponteiro. O significado de * descrito acima não serve para declaração de ponteiros!

Ponteiros

`*var`

- Representa o conteúdo do endereço de memória guardado na variável **`var`**.
- Ou seja, **`var`** não guarda nenhum valor, mas sim um endereço de memória.

Ponteiros

`*var`

- O símbolo * é chamado operador de indireção.
- Se aplicado a uma variável ponteiro, o símbolo indica o valor real armazenado no endereço contido no ponteiro.
- O operador unário *, ou conteúdo de, aplicado a variáveis do tipo ponteiro, acessa o conteúdo do endereço de memória armazenado pela variável ponteiro.

Ponteiros

`&var`

- Em contrapartida, o símbolo **&** é conhecido como operador endereço.
- Anexado no início de um nome de variável, esse operador representa o endereço na memória onde o valor da variável está armazenado.
- O operador &, ou endereço de, aplicado a variáveis resulta no endereço da posição de memória reservada para a variável.

Ponteiros

- Há vários tipos de ponteiros:
  - Ponteiro para caracter;
  - Ponteiro para inteiro;
  - Ponteiro para ponteiro;
  - Ponteiro para vetor;
  - Ponteiro para estrutura
  - ....

Ponteiros

```
Int       *ap_int   //apontador para int
Char      *ap_char  //apontador para char
Float     *ap_float//apontador para float
Double    *ap_double //apontador para double
```

- //ponteiro para ponteiro

```
Int    **ap_ap_int
```

Ponteiros - resumo

- Forma geral declaração de ponteiros:

```
tipo_do_ponteiro *nome_da_variável;
```

- É o asterisco (*) faz o compilador saber que a variável não vai guardar um valor mas sim um endereço para aquele tipo especificado.

- Exemplos de declarações:

```
int *pt;  // ponteiro para um inteiro
char *temp, *pt2; //ponteiro para char
```

Ponteiros - cuidado

- ```
  int *pt;  // ponteiro para um inteiro
  char *temp, *pt2; //ponteiro para char
  ```

- **`pt, temp e temp2`** ainda não foram inicializados (apenas declarada).

- Significa que eles apontam para um lugar indefinido.
  - Exemplo: Porção da memória reservada ao SO.
  ```

Ponteiros - cuidado

- Usar o ponteiro sem inicializar pode levar a um travamento do micro, ou a algo pior.

O ponteiro deve ser inicializado (apontado para algum lugar conhecido) antes de ser usado!
Isto é de suma importância!

Ponteiros – atribuindo valores

- Para atribuir um valor a um ponteiro poderíamos igualá-lo a um valor de memória.

- Mas, como saber a posição na memória de uma variável do nosso programa?
  - Os endereços são determinados pelo compilador e realocados na execução.
  - Para saber o endereço de uma variável basta usar o operador **&**.

Ponteiros – atribuindo valores

- Exemplo:

```
int count; //count como sendo do tipo inteiro
int *pt; //pt como sendo é ponteiro para inteiro
pt = &count; // &count nos dá o endereço de count
```

//pt está “liberado” para ser usado.

Ponteiros

- Alterar o valor de **count** usando **pt:**
  - usar o operador “inverso” do operador **&:** o operador *.
  - No exemplo:
    - **pt=&count**
    - a expressão ***pt** é equivalente ao próprio **count**.
  - Após a inicialização da variável **ponteiro** com o endereço da variável inteira, esta variável pode ser referenciada pelo seu nome ou pelo ponteiro que contém seu endereço.

Ponteiros

- Isto significa que, se quisermos atribuir um valor para **count** podemos fazer de duas formas.

```
count = 5;
```

ou

```
*pt = 5;
```

- As duas formas são equivalentes e têm o mesmo resultado. Qualquer modificação feita utilizando-se ***pt**, causará uma modificação na variável **count**.

Ponteiros - manipulando

- Os ponteiros são variáveis, logo podem ser manipulados tal como as variáveis.

- Se **py** e **px** são **ponteiros** para **inteiros**, então podemos fazer :

```
py = px;
```

Ponteiros

```
main( ) {

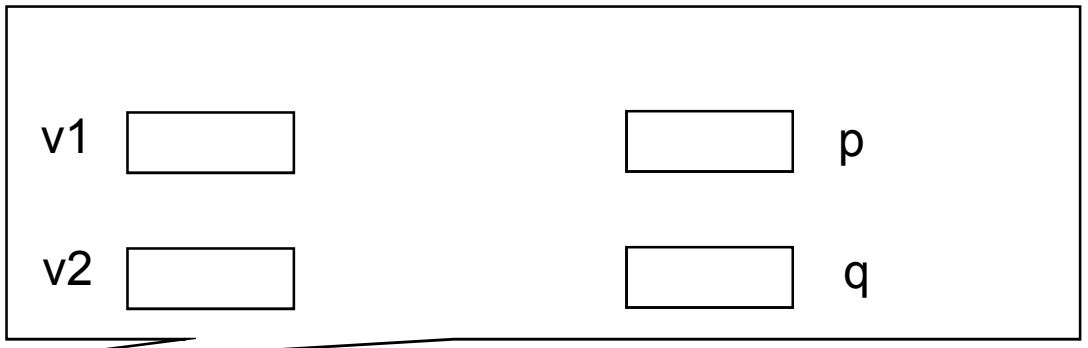
   int v1, v2, *p, *q;

   v1 = 3;
   v2 = 12;

   // p recebe o endereço de memória de v1
   p  = &v1;
   // copia o endereço guardado em p para q
   q = p;
   // altera o valor armazenado no endereço apontado por q
   *q = 44;

   cout << "valor de v1:" << v1 << endl;
   cout << "valor de v2:" << v2 << endl;
}
```

Ponteiros




```
main( ) {
   int v1, v2, *p, *q;

   v1 = 3;
   v2 = 12;

   // p recebe o endereço de memória de v1
   p  = &v1;
   // copia o endereço guardado em p para q
   q = p;
   // altera o valor armazenado no endereço apontado por q
   *q = 44;
}
```

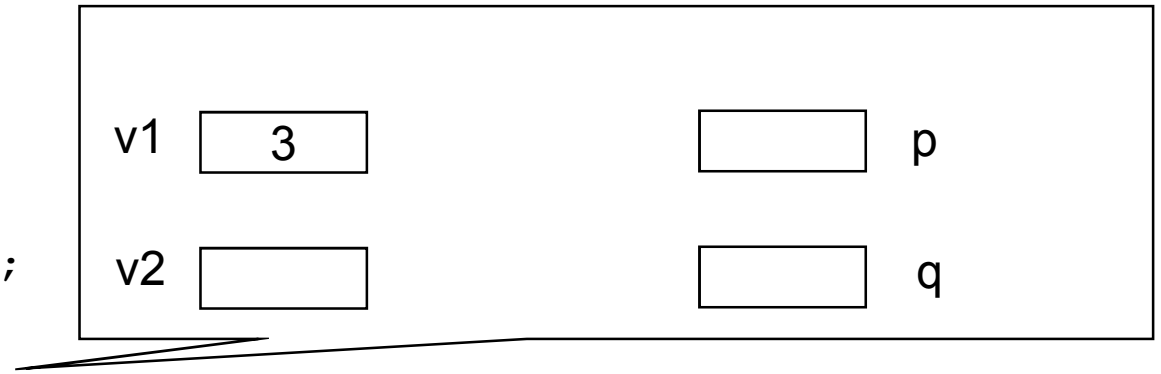
Ponteiros

```
main( ) {

   int v1, v2, *p, *q;

   v1 = 3;
   v2 = 12;

   // p recebe o endereço de memória de v1
   p  = &v1;
   // copia o endereço guardado em p para q
   q = p;
   // altera o valor armazenado no endereço apontado por q
   *q = 44;
}
```



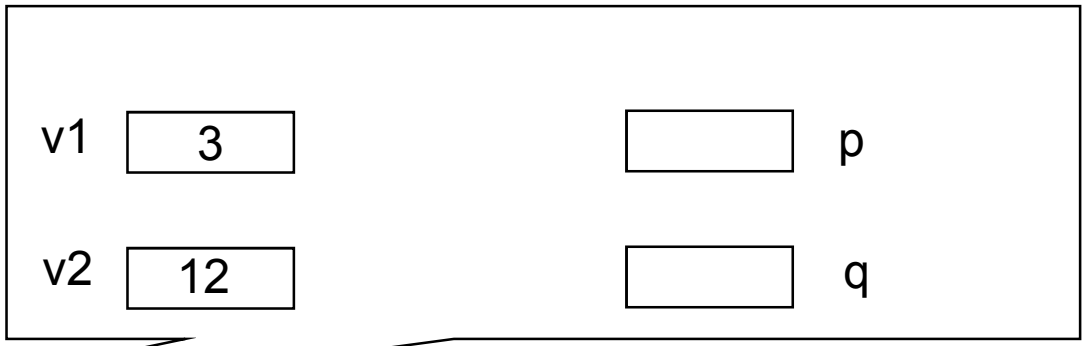
Ponteiros

```
main( ) {

   int v1, v2, *p, *q;

   v1 = 3;
   v2 = 12;

   // p recebe o endereço de memória de v1
   p  = &v1;
   // copia o endereço guardado em p para q
   q = p;
   // altera o valor armazenado no endereço apontado por q
   *q = 44;
}
```



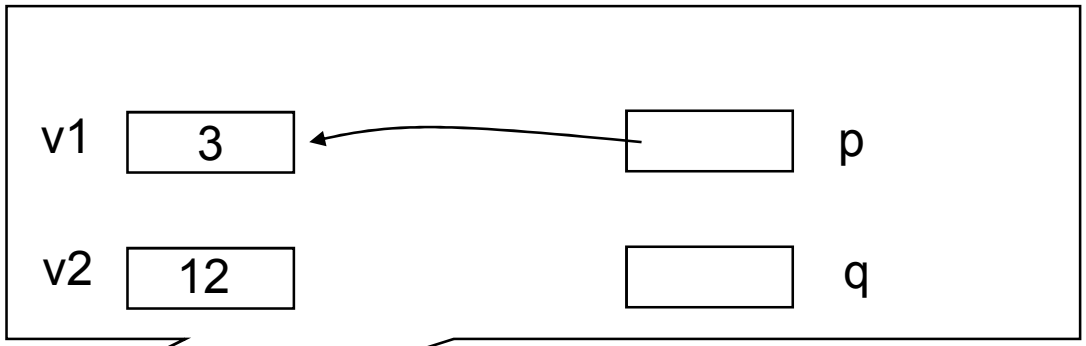
Ponteiros

```
main( ) {

   int v1, v2, *p, *q;

   v1 = 3;
   v2 = 12;

   // p recebe o endereço de memória de v1
   p  = &v1;
   // copia o endereço guardado em p para q
   q = p;
   // altera o valor armazenado no endereço apontado por q
   *q = 44;
}
```



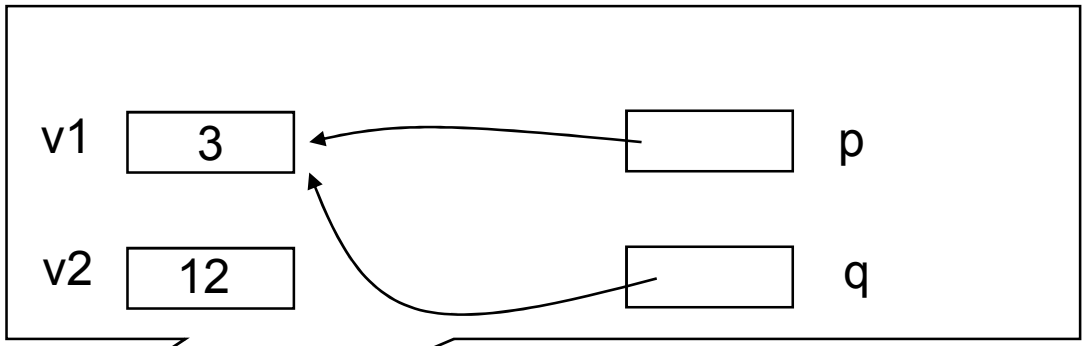
Ponteiros

```
main( ) {

   int v1, v2, *p, *q;

   v1 = 3;
   v2 = 12;

   // p recebe o endereço de memória de v1
   p  = &v1;
   // copia o endereço guardado em p para q
   q = p;
   // altera o valor armazenado no endereço apontado por q
   *q = 44;
}
```



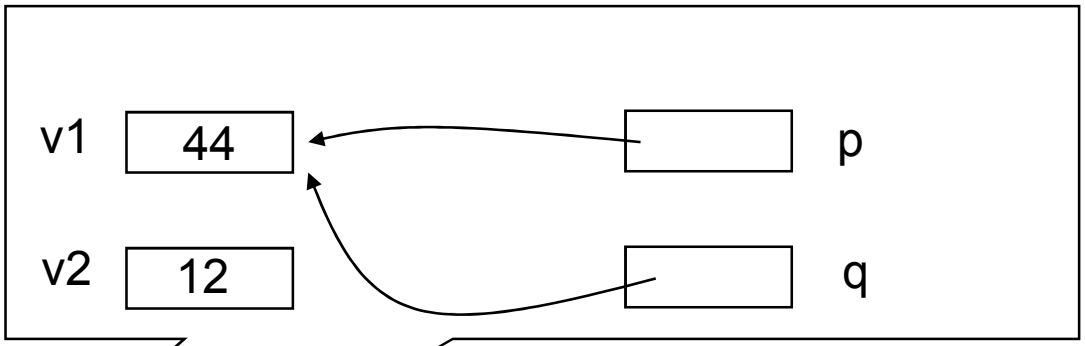
Ponteiros

```
main( ) {

   int v1, v2, *p, *q;

   v1 = 3;
   v2 = 12;

   // p recebe o endereço de memória de v1
   p  = &v1;
   // copia o endereço guardado em p para q
   q = p;
   // altera o valor armazenado no endereço apontado por q
   *q = 44;
}
```



Ponteiros – exemplo 2

```
#include <stdio.h>
main( )
{
int x,*px,*py;
x = 9;
px = &x;
py = px;

// imprime o valor de x
printf("x= %d\n",x);
// endereço da variável x, que é igual ao conteúdo
de px
printf("&x= %d\n",&x);
```

Ponteiros – exemplo 2

```
// valor de px, que é o endereço de x
printf("px= %d\n",px);

// conteúdo da variável apontada por px, isto é,
valor de x
printf("*px= %d\n",*px);

// imprime valor de x, pois py = px
printf("*py= %d\n",*py);
}
```

Ponteiros - expressões

- Os ponteiros podem aparecer em expressões:
  - se **px** aponta para um inteiro **x**, então ***px** pode ser utilizado em qualquer lugar que **x.**
  - O operador * tem maior precedência que as operações aritméticas.
  - Expressão abaixo pega o conteúdo do endereço que **px** aponta e soma **1**.

```
y = *px+1; // y é uma variável do tipo de x
```

Ponteiros – expressões 2

- No caso abaixo somente o ponteiro será incrementado e o conteúdo da próxima posição da memória será atribuído a **y**.

  y = *(px+3); // y é uma variavel do tipo de px

Ponteiros – expressões 3

- Os incrementos e decrementos dos endereços podem ser realizados com os operadores ++ e --, que possuem precedência sobre o * e operações matemáticas e são avaliados da direita para a esquerda:

```
*px++; /* sobe uma posição na memória*/
*(px--); /* mesma coisa de *px-- */
```

```
# include <stdio.h>
main()
{
int x, *px;
x = 1;
px = &x;

printf("x= %d\n", x);
printf("px= %u\n", px);
printf("*px+1= %d\n", *px+1);
printf("px=%u\n", px);
printf("*px= %d\n", *px);
printf("*px+=1= %d\n", *px+=1);
printf("px= %u\n",px);
printf("(*px)++=%d\n", (*px)++);
```

```
# include <stdio.h>
main()
{
int x, *px;
x = 1;
px = &x;

printf("*(px++)= %d\n", *(px++));
printf("px= %u\n",px);
printf("*px++-=%d\n", *px++);
printf("px= %u\n", px);
}
```

Ponteiros - problemas

- O erro chamado de **ponteiro perdido** é um dos mais difíceis de se encontrar, pois a cada vez que a operação com o ponteiro é utilizada, poderá estar sendo lido ou gravado em posições desconhecidas da memória.
- Isso pode acarretar sobreposições em áreas de dados ou mesmo área do programa na memória.

Ponteiros - problemas

```
int *p;
x = 10;
*p = x;
```

Neste trecho de programa, o ponteiro **p** não foi inicializado, não recebeu nenhum endereço, portanto o valor **10** foi colocado em uma posição aleatória e desconhecida de memória. **A conseqüência desta atribuição é imprevisível**.

Ponteiros e matrizes

- Em C ponteiros e matrizes podem ser tratados da mesma maneira.
  - Versões com ponteiros são mais rápidas.
- Declarando uma matriz:

```
tipo nome_variável [tam1][tam2] ... [tamN];
```

Ponteiros e matrizes

- O compilador calcula o tamanho, em bytes, necessário para armazenar esta matriz:

```
tam1 x tam2 x tam3 x ... x tamN x tamanho_do_tipo
```

- Este número de bytes é alocado em um espaço livre de memória.
- O nome da variável é um ponteiro para o tipo da variável da matriz.

Ponteiros e matrizes

- Este conceito é fundamental:
  - Tendo alocado na memória o espaço para a matriz, ele toma o nome da variável (que é um ponteiro) e aponta para o primeiro elemento da matriz.

Ponteiros e matrizes

- Como é que podemos usar a seguinte notação?

```
nome_da_variável[índice]
```

- A notação acima é absolutamente equivalente a se fazer:

```
*(nome_da_variável+índice)
```

Ponteiros e matrizes

- A memória de um computador não tem arranjos multidimensionais:
  - pode ser descrita como uma extensa lista de posições de memória enfileiradas, cada uma com um endereço específico.
-

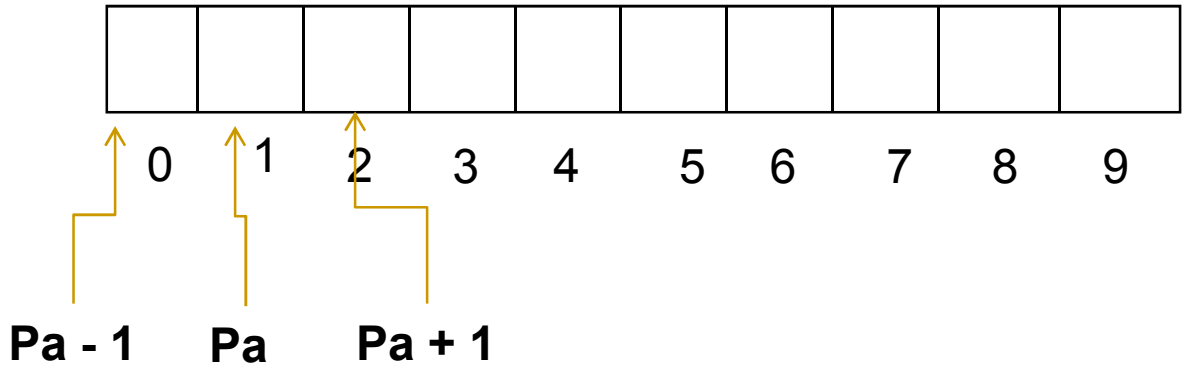
Ponteiros e matrizes

- A indexação de uma matriz começa com o valor zero.
  - Ao pegarmos o valor do primeiro elemento de uma matriz, queremos, de fato, `*nome_da_matriz` e então devemos ter um índice igual a zero, resultando `(*nome_da_matriz+0)`.
  - Logo: `*nome_da_matriz` é equivalente a `nome_da_matriz[0]`

Ponteiros e matrizes

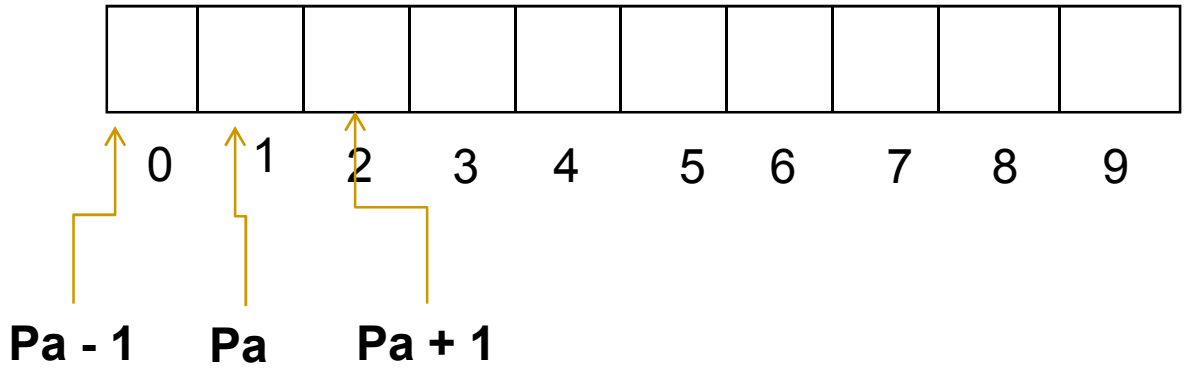
- **int a[10], x; //** declaração da matriz
- **int *pa; // pa** um ponteiro para inteiro
- pa = &a[0]; /*passa o endereço inicial do vetor a para o ponteiro pa */
- pa = a; /* é a mesma coisa de pa=&a[0];
- x = *pa; //passa o conteúdo de a[0] para x

Ponteiros e matrizes



- Se **pa** aponta para um elemento particular de um vetor **a**, então por definição **pa+1** aponta para o próximo elemento, e em geral **pa-i** aponta para **i** elementos antes de **pa** e **pa+i** para **i** elementos depois.

Ponteiros e matrizes



- Se **pa** aponta para **a[0]** então:
  - ***(pa+1)** aponta para **a[1];**
  - **pa+i** é o endereço de **a[i]** e
  - ***(pa+i)** é o conteúdo.

```
#include<stdio.h>
main(){

   int vet[10] =
   {10,20,30,40,50,60,70,80,90,100};
   int *pVet, valorVet;

   pVet = vet; // == pVet = &vet[0];

   printf("\nVersão Tradicional");
   for(int i = 0; i < 10; i++){
       printf("\nvet[%d] = %d",i,vet[i]);
   }
```

```
printf("\nVersão com ponteiro");
for(int i = 0; i < 10; i++){
    printf("\nvet[%d] = %d",i,*(pVet+i));
}

printf("\nVersão 2 com ponteiro");
for(int i = 0; i < 10; i++,pVet++){
    printf("\nvet[%d] = %d",i,*pVet);
}
```

```
    printf("\nVersão 3 com ponteiro");
    pVet = &vet[0];
    int i=0;
    while(i<10){

        printf("\nvet[] = %d",*pVet);
        pVet++;
        i++;
    }


}
```

Ponteiros

- Ponteiros para Estruturas:
  - Como outros tipos de dados, ponteiros para estruturas são declarados colocando-se o operador * na frente do nome da variável estrutura:

```
Typedef struct{
        char Nome[30];
        int idade;
        float coeficiente;
}aluno;
```

```
aluno *ap_aluno1;
```

Ponteiros

```
Typedef struct{
        char Nome[30];
        int idade;
        float coeficiente;
}aluno;
```

- Ponteiros para Estruturas:
  - Sintaxe para acessar uma estrutura com ponteiros:
    - Operador seta:
      - **<ponteiro_estrutura>-><campo>**
      - **ap_aluno1->nome;**

Ponteiros

- Faça um programa em C que declare a estrutura:

```
Typedef struct{
        char Nome[30];
        int idade;
        float coeficiente;
}aluno;
```

- Declare uma variável do tipo ponteiro para estrutura, efetue a operação de leitura e escrita dos dados desta variável.

```
typedef struct{

   char nome[30];
   int idade;
   float coeficiente;

}t_aluno;

main(){

   t_aluno *pAluno, aluno;
   pAluno = &aluno;
```

```
    puts("Informe o nome do aluno:");
    gets(pAluno->nome);
    puts("Informe a idade do aluno:");
    scanf("%d",&pAluno->idade);
    puts("Informe o coeficiente do aluno:");
    scanf("%f",&pAluno->coeficiente);

    printf("\nNome: %s",pAluno->nome);
    printf("\nIdade: %d",pAluno->idade);
    printf("\nCoeficiente: %f",pAluno-
    >coeficiente);
}
```