

INF01202

Algoritmos e Programação

Modalidade Ensino a Distância

Linguagem C
Tópico 14: Ponteiros

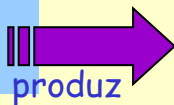
Variáveis e endereços

Toda e qualquer variável utilizada por um programa reside em determinado **endereço** de memória.

O acesso ao endereço de uma variável pode ser feito simbolicamente através de seu nome. Essa referência é **simbólica** porque o endereço de memória propriamente dito NÃO é especificado.

A declaração:

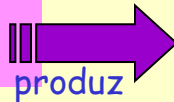
`int x;`



Variável	Valor da Variável	Endereço de Memória
x	?	AFA1

O comando:

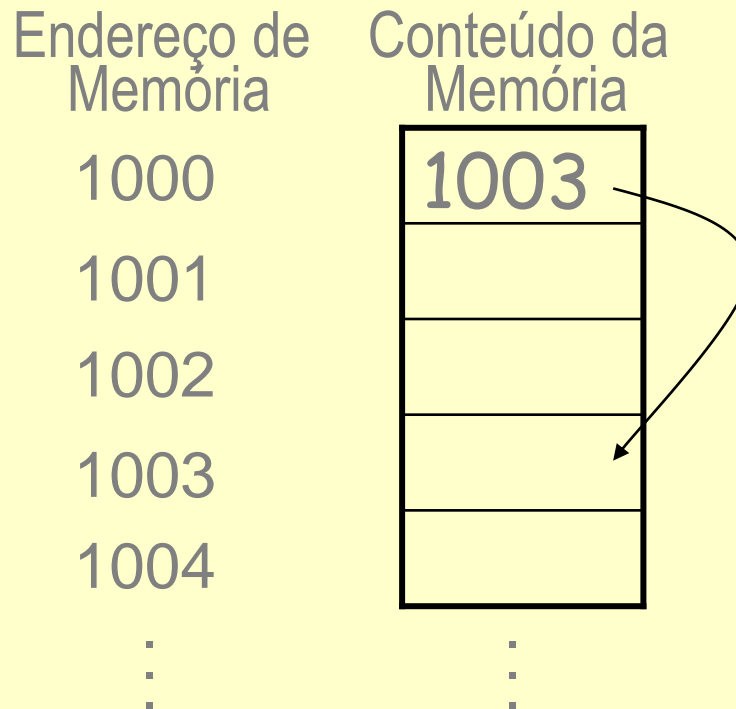
`x = 3;`



Variável	Valor da Variável	Endereço de Memória
x	3	AFA1

O que são ponteiros?

Ponteiros são variáveis que armazenam endereços de memória.



O endereço é a **posição** de uma outra variável na memória.
Se uma variável *a* armazena o endereço de uma variável *b*,
dizemos que *a* **aponta** para *b*.

Ponteiros em C

- A Linguagem C é **ALTAMENTE** dependente de *ponteiros*.
- Portanto, para ser um **BOM PROGRAMADOR** em C, é fundamental que ter um bom domínio sobre eles.

Declaração de Ponteiros

tipo *nome; → nome da variável ponteiro

asterisco: operador especial para denotar que a variável armazena um endereço de memória.

qualquer tipo válido em C

Exemplos:

`float *f;` // f eh um ponteiro para variaveis do tipo *float*

`int *p;` // p eh um ponteiro para variaveis do tipo inteiro

`char a, b, *p, c, *q;` //podem ser declarados junto com outras
//variaveis de mesmo tipo

O asterisco é o mesmo da operação de multiplicação, mas não provoca confusão uma vez que seu significado depende do contexto em que é usado!

Operadores de Ponteiros

Há dois operadores:

& operador unário que devolve o endereço de memória de seu operando.

***** operador unário que devolve o **valor** da variável localizada no endereço que o segue. Também chamado operador de indireção.


Operadores de Ponteiros (cont.)

& operador unário que devolve o endereço de memória de seu operando.

Exemplo:

```
int count, q, *m;
```

```
m = &count; //coloca em m o endereço de memória da  
            //variavel count
```



- Lê-se: "*m* recebe o endereço de *count*"

- Se a variável *count* está armazenada na posição de memória 2000, o valor de *m* será 2000.

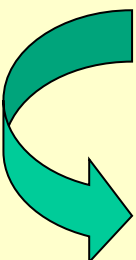
Operadores de Ponteiros (cont.)

- * operador unário que devolve o **valor** da variável localizada no endereço que o segue. Também chamado operador de indireção.

Exemplo:

```
int count, q, *m;
```

```
q = *m; //coloca em q o valor contido no endereço de  
        memória apontado por m
```



- Lê-se: “*q* recebe o valor que está no endereço *m*”

- Se a variável *m* aponta para um endereço que contém o valor 100, o valor de *q* será 100.

Esse operador permite desreferenciar um ponteiro, ou seja, acessar o valor armazenado no endereço contido no ponteiro.

Operadores de Ponteiros (cont.)

Exemplo 1

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

```
int main( )
{
```

```
    int count, q, *m;
    count = 100;
```

```
    m = &count; // m recebe o endereco de count
```

```
    q = *m; // q recebe o conteudo da posicao de memoria apontada por m
```

```
    printf("count = %d\n", count);
```

```
    printf("q = %d\n", q);
```

```
    printf("m = %p\n", m);
```

```
    // *m a seguir eh o conteudo da posicao de memoria apontada por m
```

```
    printf("conteudo da posicao de memoria apontada por m = %d\n\n", *m);
```

```
    system("pause");
```

```
    return 0;
```

```
}
```

```
C:\backupcida\LinguagemCPagina20082\aula18ex04aritmponte
```

```
count = 100
```

```
q = 100
```

```
m = 0022FF74
```

```
conteudo da posicao de memoria apontada por m = 100
```

```
Pressione qualquer tecla para continuar. . . _
```

%p é usado para imprimir ponteiros

Operadores de Ponteiros (cont.)

Exemplo 2 - Onde está o problema?

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
    float x;
    int *p;
    x = 100;
    p = &x;
    printf("x=%f\p=%p\n\n", x,p);
    system("pause");
    return 0;
}
```

Erro de compilação: tipos incompatíveis
p é um ponteiro para inteiros, logo não pode apontar para uma variável do tipo *float*.

Atribuições com ponteiros

Assim como qualquer variável, um ponteiro pode ser usado no lado direito de um comando de atribuição, para passar seu valor para um outro ponteiro.

Exemplos:

```
int x, *p1, *p2;
```

```
p1 = &x; //p1 aponta para x
```

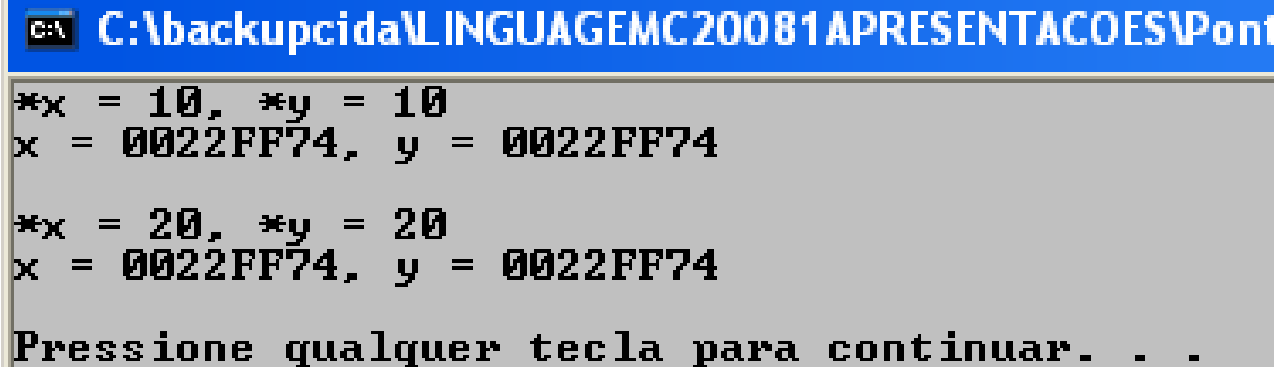
```
p2 = p1; //p2 recebe p1 e tambem passa a apontar para x
```

```
printf("%p", p2); //escreve o endereço de x
```

Atribuições com ponteiros

Exemplo 3 - O que será impresso na tela?

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
    int a, *x,*y;
    a = 10;
    x = &a;
    y = x;
    printf("*x = %d, *y = %d \n",*x, *y);
    printf("x = %p, y = %p\n\n",x,y);
    *x = 20;
    printf("*x = %d, *y = %d \n",*x, *y);
    printf("x = %p, y = %p\n\n",x,y);
    system("PAUSE");
    return 0;
}
```



```
C:\backupcida\LINGUAGEMC20081\APRESENTACOES\Pont
*x = 10, *y = 10
x = 0022FF74, y = 0022FF74
*x = 20, *y = 20
x = 0022FF74, y = 0022FF74
Pressione qualquer tecla para continuar. . .
```

Inicialização de ponteiros

- Ponteiros não devem ser utilizados sem prévia inicialização. Por exemplo:

```
int x;  
float pi = 3.14;  
int *ptr_x = &x;  
float *ptr_to_pi = &pi;
```

- No entanto, podem existir situações em que não se quer que um ponteiro aponte uma variável. Nesse caso, deve-se inicializar o ponteiro com NULL. Exemplo:

```
int a = 5, b = 7;  
int *ptr = NULL;
```

Inicialização de ponteiros (cont.)

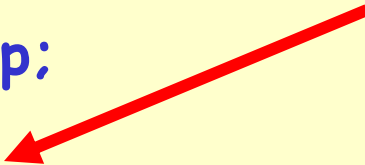
Problemas com Ponteiros não inicializados

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

int main( )
{
    int x, *p;
    x = 10;
    *p = x;
    system("PAUSE");
    return 0;
}
```

Problema:

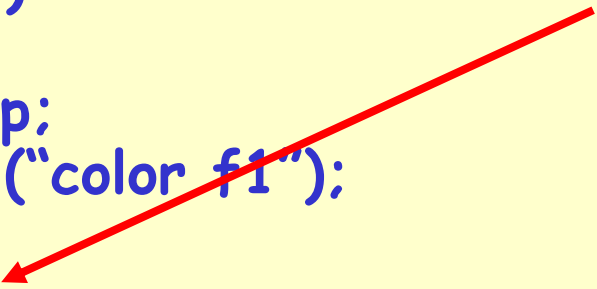
O ponteiro **p** nunca recebeu valor, por isso contém lixo. O programa atribui o valor 10 a uma posição de memória desconhecida.



Inicialização de ponteiros (cont.)

Problemas com Ponteiros não inicializados (cont.)

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
    int x, *p;
    system ("color f1");
    x = 10;
    p = &x;
    *p = 30;
    printf("\nConteudo de x = %d\n", x);
    system("PAUSE");
    return 0;
}
```



Solução:

Certifique-se de que o ponteiro está apontando para algo válido antes de usá-lo.

```
C:\backupcida\LinguagemCProgramas\Ponteiros\Aula
Conteudo de x = 30
Pressione qualquer tecla para continuar. . .
```

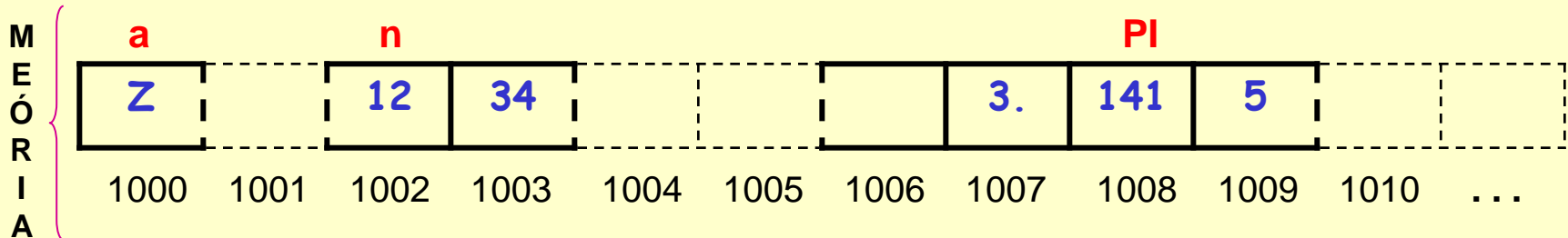
Por que os ponteiros têm que possuir um tipo?

Em C, os tipos de dados ocupam diferentes números de *bytes* na memória. Assim, ao se declarar o tipo de uma variável:

```
char a = 'Z';           //1 byte ?  
int n = 1234;           //2 bytes ?  
float PI = 3.1415;      //4 bytes ?
```

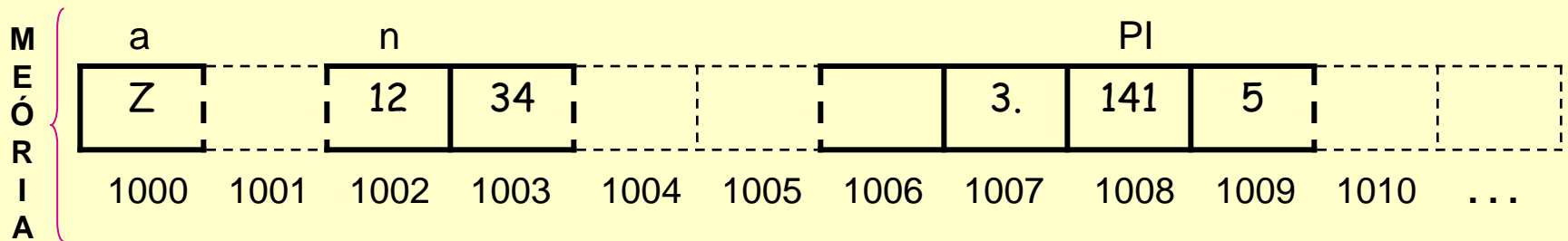
Depende da arquitetura !

se está indicando ao compilador qual o espaço que ele deverá reservar para cada uma dessas variáveis.



Qual o endereço de uma variável com mais de 1 *byte* de comprimento?

O endereço de uma variável é sempre o **MENOR** dos endereços que ela ocupa na memória!



Um ponteiro para o tipo *xyz* endereça sempre o número de *bytes* que esse tipo ocupa na memória!

Como determinar o tamanho de variáveis ou tipos?

Operador *sizeof*

O operador **sizeof** permite determinar o tamanho em *bytes* de tipos ou variáveis. É o meio mais seguro para obter esse tipo de informação.

Ele é substituído pelo tamanho do tipo ou variável *no momento da compilação*.

sizeof admite duas formas:

sizeof nome_da_variável

Ex.: variável **f** de tipo **float** --> **sizeof f**

sizeof (nome_do_tipo)

Ex.: tamanho do tipo **float** --> **sizeof(float)**.

Fonte:

<http://www.mtm.ufsc.br/~azeredo/cursoC/aulas/cb50.html>

Aritmética de Ponteiros

Sendo os ponteiros números que representam posições de memória, então podem ser realizadas algumas operações sobre eles:

Incremento (adição)

Decremento (subtração)

Diferença

Comparação

Aritmética de Ponteiros (cont.)

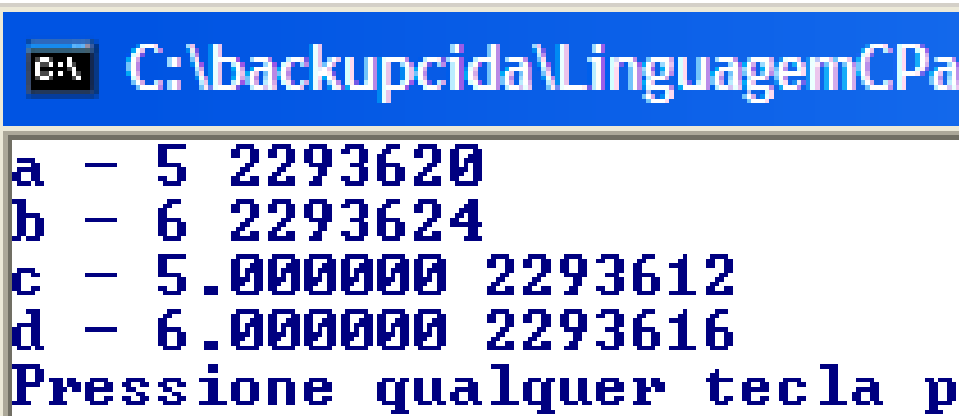
Adição e subtração de ponteiros:

- suponha-se que $p1$ é um ponteiro para inteiros com um valor inicial 2000 (contém o endereço 2000);
- em um sistema em que cada inteiro ocupa 4 *bytes*;
- após o comando $p1++$, o valor de $p1$ passa a ser 2004 ($p1$ passa a conter o endereço 2004), ou seja, $p1$ **passa a apontar para o próximo inteiro**;
- o mesmo vale para o decremento ($p1--$).

Aritmética de Ponteiros (cont.)

Incrementando ponteiros

Um ponteiro para o tipo *xyz* avança sempre *sizeof (xyz) bytes* por unidade de incremento.

Exemplo:	Valores impressos
<pre>#include <stdio.h> #include <stdlib.h> int main () { int x = 5, *px = &x; float y = 5.0, *py = &y; system("color f1"); printf ("a - %d %d\n", x, px); printf ("b - %d %d\n", x+1, (px+1)); printf ("c - %f %d\n", y, py); printf ("d - %f %d\n", y+1, (py+1)); system ("pause"); return 0; }</pre>	<p>Nesta arquitetura:</p> <ul style="list-style-type: none">→ o tipo int: 4 bytes→ o tipo float: 4 bytes 

Decrementando ponteiros

Um ponteiro para o tipo *xyz* recua sempre *sizeof (xyz) bytes* por unidade de incremento.

Exemplo:

Valores impressos

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main ( )
{ int len;
  char s[100] , char *ptr;
  system("color f1");
  printf ("Introduza um texto: ");
  fgets(s, sizeof(s), stdin);
  printf("Texto = %s", s);
  if (s[strlen(s) - 1] == '\n')
    s[strlen(s) - 1] = '\0';
  len = strlen(s) - 1;
  // em len definicao do indice maximo
  ptr = &s[len];
  while (ptr >= s)
    putchar (*ptr--);
  printf ("\n\n");
  system ("pause");
  return 0;
}
```

Este exemplo não testa *string* vazia!

C:\F:\PARACASA\aula18ex01aritmponteiros.exe

Introduza um texto: Algoritmos e Programacao
Texto = Algoritmos e Programacao
oacamargorP e sontiroglA

Pressione qualquer tecla para continuar. . .

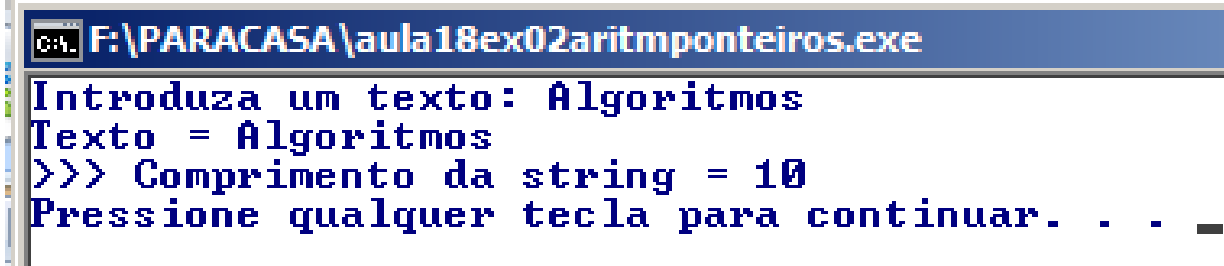
O nome de um arranjo é o mesmo que o endereço de seu elemento de índice 0.

Aritmética de Ponteiros (cont.)

- Ao serem **incrementados**, os ponteiros passam a apontar para a posição de memória do(s) elemento(s) **posterior(es)** do seu tipo base.
- Ao serem **decrementados**, os ponteiros passam a apontar para a posição de memória do(s) elemento(s) **anterior(es)** do seu tipo base.
- O valor do ponteiro irá aumentar ou diminuir um múltiplo do número de *bytes* que o tipo base ocupa.

Diferença entre dois ponteiros

A diferença entre dois ponteiros para elementos de **mesmo tipo** permite saber quantos elementos existem entre eles.

Exemplo:	Valores impressos
<pre>#include <stdio.h> #include <stdlib.h> #include <string.h> int main () { char s[100], *len1, *len2; system("color f1"); printf ("Introduza um texto: "); fgets(s, sizeof(s), stdin); printf("Texto = %s", s); if (s[strlen(s) - 1] == '\n') s[strlen(s) - 1] = '\0'; len1 = len2 = s; while (*len2 != '\0') len2++; printf (">>> Comprimento da string = %d\n" (len2-len1)); system ("pause"); return 0; }</pre>	<p><i>Este exemplo não testa string vazia!</i> <i>Faz o mesmo que a função <code>strlen</code> do C.</i></p> 

Aritmética de Ponteiros (cont.)

Somando ou subtraindo inteiros de ponteiros

Podemos somar ou subtrair inteiros de ponteiros.

Por exemplo:

```
p1 = p1 + 10;
```

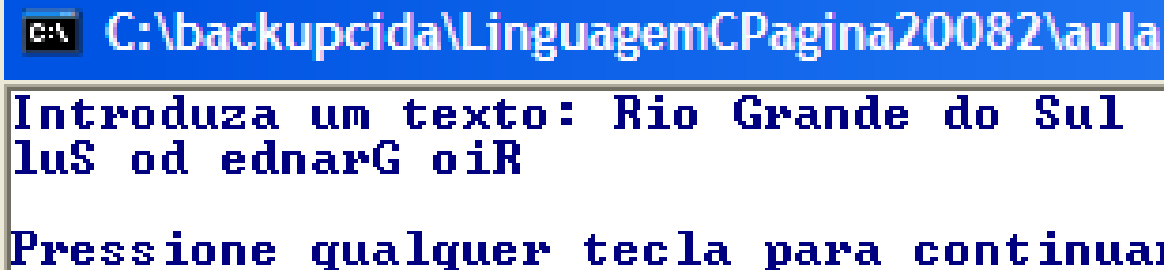
faz p1 apontar para o décimo elemento do tipo p1 (adiante do elemento atualmente apontado por p1).

Comparação entre ponteiros

É possível fazer a comparação entre dois ponteiros mediante o uso dos operadores relacionais (<, <=, >, >=, !=) .

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
main ( )
{
    int len;
    char s[100];
    char *ptr;
    system("color f1");
    printf ("Introduza um texto: ");
    fgets(s, sizeof (s), stdin);
    if (s[strlen(s) - 1] == '\n')
        s[strlen(s) - 1] = '\0';
    len = strlen (s)-1;
    ptr = &s[len];
    while (ptr >= s)
        putchar (*ptr--);
    printf ("\n\n");
    system ("pause");
    return 0;
}
```

Exemplo:



```
C:\backupcida\LinguagemCPagina20082\aula
Introduza um texto: Rio Grande do Sul
lu$ od ednarG oiR
Pressione qualquer tecla para continuar
```

Ponteiros e Vetores

Os ponteiros são normalmente utilizados na manipulação de vetores.

O nome de um vetor é um ponteiro que aponta para o **primeiro elemento** do vetor. Portanto,

Se v for um vetor Então $v == \&v[0]$

Considerando:

```
int v[3] = {10, 20, 30}; //vetor c/ 3 inteiros
```

```
int *ptr; //ponteiro p/ inteiro
```

Existem 2 formas para fazer com que ptr aponte para v :

```
ptr = &v[0];
```

```
ptr = v;
```

O ponteiro v não pode ser alterado durante a execução do programa, mas ptr PODE!

Ponteiros e Vetores (cont.)

Como acessar os elementos de um vetor através de ponteiros ?

Se o vetor v for um vetor já declarado, então:

escrever $v[0]$ é o mesmo que escrever $*(v)$

escrever $v[1]$ é o mesmo que escrever $*(v + 1)$

escrever $v[2]$ é o mesmo que escrever $*(v + 2)$

...

escrever $v[n]$ é o mesmo que escrever $*(v + n)$

E por conseguinte, se v for um vetor e v aponta para o primeiro elemento, então o elemento índice n é $v[n]$ ou $*(v + n)$.

Ponteiros e Vetores (cont.)

Leitura e escrita de vetores usando notação de ponteiros:

Seja o vetor:

```
int vet[MAX];
```

Leitura

```
for (i = 0; i < MAX; i++)  
{  
    printf ("Valor %d: ", i);  
    scanf ("%d", vet + i);  
}
```

Escrita

```
for (i = 0; i < MAX; i++)  
    printf ("%d\t", *(vet+i));
```

Ponteiros e Vetores (cont.)

Existe uma diferença fundamental entre declarar um conjunto de dados como um vetor ou através de um ponteiro.

Na declaração de **vetor**, o compilador automaticamente **reserva um bloco** de memória para que o vetor seja armazenado.

Quando apenas um ponteiro é declarado, a única coisa que o compilador faz é **alocar espaço para o ponteiro**, sem que qualquer espaço adicional seja reservado.

Ponteiros e Vetores (cont.)

Em C, strings são **vetores** de caracteres. Logo toda a discussão anterior relativa a vetores vale para o processamento de strings.

```
char str[80], *p1;
```

```
p1 = str; //atribui a p1 o endereço do 1º. elemento de str
```

```
str[4] ou
```

```
*(p1+4) //devolvem o 5º. elemento de str
```

Ponteiros e Vetores: resumo

Há uma estreita relação entre ponteiros e vetores.

O nome de um vetor é um ponteiro que aponta para a **primeira posição** do vetor e todas as operações já mencionadas para ponteiros podem ser executadas com um nome de vetor.

Assim, declarado o vetor `int v[100]`, as colunas abaixo são equivalentes:

notação de vetor

equivale a

notação de ponteiro

`v[i]` ou `*(v+i)`
`&v[i]`

`*(v+i)` ou `v[i]`
`v+i`

Ponteiros em C

Possibilitam a passagem de parâmetros por endereço para as funções, sobretudo **vetores e matrizes**.

Permitem um acesso mais ágil a elementos de arranjos através da **aritmética de ponteiros**.

São fundamentais em estruturas de dados **não alocadas em blocos contíguos de memória**, como listas encadeadas, árvores ou grafos (objetos de estudo nas disciplinas Estruturas de Dados e Classificação e Pesquisa de Dados).

Exercícios

Exercício 1

Modifique o programa abaixo para imprimir o conteúdo do vetor usando ponteiros.

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

int main( )
{
    float v[ ] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0};
    int i;

    for (i=0; i<9; i++)
        printf("%.1f ", v[i]);

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

Solução

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

int main()
{
    float v[] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0};
    int i;

    for (i=0; i<9; i++)
        printf("%.1f ", *(v+i));

    system("pause");
}
```

Exercício 2

Faça um programa que contenha o seguinte vetor como dado inicial: `int v [8] = {10,5,20,9,4,1,15,1}`. A seguir, crie dois ponteiros para inteiros que deverão apontar para as posições do vetor `v` que contêm o **maior** elemento e o **menor** elemento. Ao final, imprima o conteúdo dos elementos apontados por esses ponteiros, que corresponderão ao **menor** e ao **maior** elemento do vetor (1 e 20).

Solução

```
//Inserir cabeçalho ...
#include <stdio.h>
#include <windows.h>
int main( )
{
    int v[8] = {10,5,20,9,4,2,15,1};
    int *max,*min;
    int i;

    max = v;
    min = v;

    for (i = 1; i < 8; i++)
    {
        if (v[i] > *max)
            max = &(v[i]);

        if (v[i] < *min)
            min = &(v[i]);
    }
    printf("Menor elemento: %d\n",*min);
    printf("Maior elemento: %d\n",*max);
    system("PAUSE");
}
```