3. Linguagem de Programação C

Prof. Renato Tinós

Departamento de Computação e Matemática
(FFCLRP/USP)

Principais Tópicos

3.6. Ponteiros

3.6.1. Introdução
3.6.2. Uso de Ponteiros
3.6.3. Ponteiros e Vetores
3.6.4. Vetores de Ponteiros
3.6.5. Ponteiros para Ponteiros

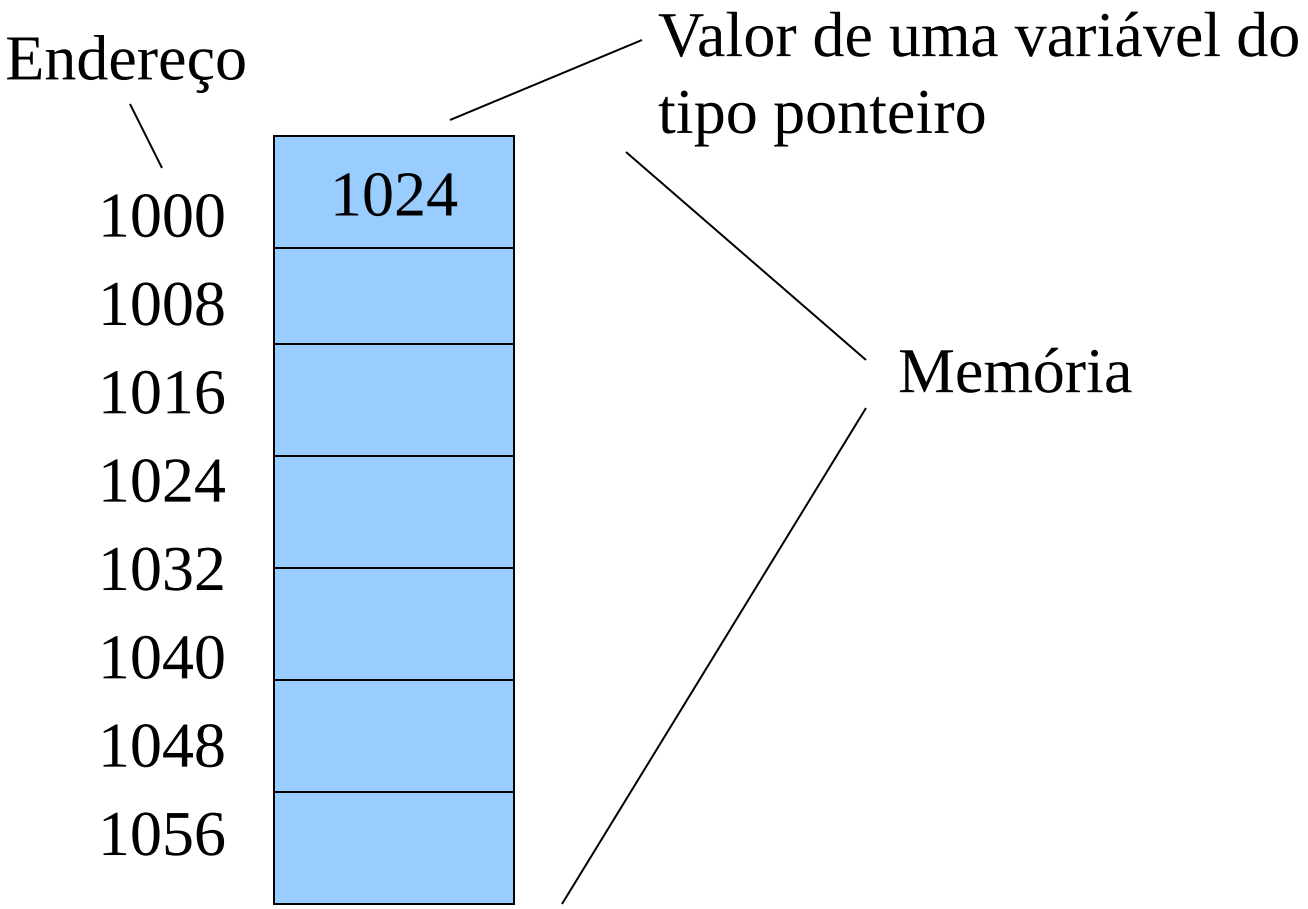
3.6.1. Introdução

- **Vetores e Ponteiros**
  - Um elemento do vetor é acessado na memória fornecendo: nome e índice
    - O nome indica o endereço inicial do vetor
    - Sabendo-se o tipo do vetor, o índice indica o número de endereços que devem ser percorridos para alcançar o elemento desejado
  - Assim, internamente, é necessário um indicador para o endereço da memória corrente
    - Devemos lembrar também que toda variável é relacionada a um endereço de memória
  - Endereços das variáveis podem ser manipulados diretamente por meio de ponteiros

3.6.1. Introdução

- **<u>Ponteiro: variável que contém um endereço de memória</u>**
  - Este endereço é geralmente a posição de uma outra variável na memória
  - Quando uma variável contém o endereço de uma outra, diz-se que a primeira variável aponta para a segunda

3.6.1. Introdução



3.6.1. Introdução

- **Por que usar?**
  - Permitem que as funções possam modificar seus argumentos de chamada
  - São usados para suportar rotinas de alocação de memória
    - permitem que novos vetores sejam alocados durante a execução do programa
  - Podem substituir as matrizes proporcionando aumento da eficiência

3.6.1. Introdução

- **Por outro lado, o uso incorreto de ponteiros pode causar sérios problemas na execução do programa**
  - Podem causar o travamento do sistema quando
    - Não-inicializados
    - São utilizados incorretamente
  - Podem causar bugs nos programas quando são utilizados incorretamente

3.6.2. Uso de Ponteiros

- **Variáveis ponteiros devem ser declaradas como tal**

*tipo_base *nome-da-variavel*

  - O tipo base do ponteiro define que tipo de variáveis o ponteiro pode apontar
    - Tecnicamente, qualquer tipo de ponteiro pode apontar para qualquer lugar na memória
    - Aritmética de ponteiros é feita pelo tipo base, assim é importante declarar o ponteiro corretamente

3.6.2. Uso de Ponteiros

- **Cada tipo base ocupa um tamanho de memória diferente**
  - *char: 8 bits (1 byte)*
  - *int:* 16 ou 32 bits *(2 ou 4 bytes)*
  - *short int:16 bits (2 bytes)*
  - *long int:* 32 bits *(4 bytes)*
  - *float 32 bits (4 bytes)*
  - *double 64 bits (8 bytes)*
  - *long double 80 bits (10 bytes)*

3.6.2. Uso de Ponteiros

- **Operadores de ponteiros**
  - Operador unário **&** fornece o endereço na memória de seu operando (variável ou constante)
    - Ex.: m = &val;
    - & pode ser lido como “endereço de”
  - Operador unário ***** é o complemento de &
    - Devolve o valor da variável armazenada no endereço que o segue
    - Ex.: x = *val;
    - * pode ser lido como “no endereço”

3.6.2. Uso de Ponteiros

```
// Programa: ponteiro 1
#include <stdio.h>

main( ){
  int x, y;
  int *p1;

  x = 100;
  p1 = &x;                // pega o endereço de x
  y = *p1;                / / pega o valor de x
  printf("%d", y);        // escreve o valor de x

}
```

3.6.2. Uso de Ponteiros

```
// Programa: ponteiro 2
#include <stdio.h>

void main(void){
  int x;
  int *p1, *p2;

  p1 = &x;
  p2 = p1;

  printf("%p", p2);       /* escreve o valor
                              hexadecimal do endereco de x! */
}
```

3.6.3. Ponteiros e Vetores

- **Relacionamento entre ponteiros e vetores**
  - Em C, o nome de um vetor tem como valor o endereço do primeiro elemento deste vetor
    - Ex.: *int lista[5];* // lista é idêntico a &lista[0]
    - Nome de um vetor equivale a um ponteiro para o primeiro elemento do vetor
    - Nome de um vetor pode ser atribuído a uma variável do tipo ponteiro
      - Usado quando um vetor é passado de uma função para outra

3.6.3. Ponteiros e Vetores

- **Diferença entre ponteiros e vetores ocorre na alocação de memória durante a declaração**
  - Ex.: *int vet[5];*
    - Reserva 5 posições consecutivas, cada uma capaz de armazenar um valor do tipo inteiro
  - Ex.: *int *p;*
    - Reserva apenas uma palavra
    - Suficiente para armazenar um endereço de memória

3.6.3. Ponteiros e Vetores

- **Como usar um ponteiro como um vetor:**
  - Atribuir ao ponteiro o endereço base do vetor
    - Ex. *p = vet;* // *p* e *vetor* se tornam "sinonimos"
  - Ponteiro pode ser inicializado para uma nova memória que ainda não foi utilizada
    - Permite a criação de novos vetores durante execução do programa (alocação dinâmica)

3.6.3. Ponteiros e Vetores

- **Aritmética de ponteiros**
  - Em C, operadores + e - podem ser aplicados a ponteiros
  - Se *p* é um ponteiro para o elemento inicial de um vetor *vet* e *k* é um inteiro, então:

    (p+k) é definido como *&vet[k]*

    (p+k) corresponde ao elemento *k* de *p*

    – *Em C, pode-se usar p[k] ao invés de (p+k)*

    **(p + k) é o valor armazenado no endereço (p + k)*

    **(p + 4) = vet[4]*

3.6.3. Ponteiros e Vetores

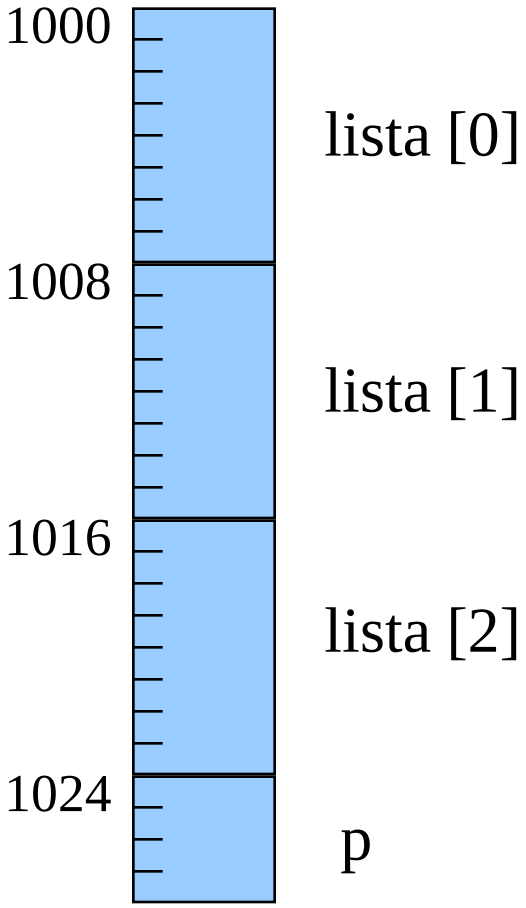
Ambas as operações atribuem o valor 20 ao sexto elemento de val

```
...
int *p, val[10];

p = val;

p[5] = 20;          // atribui usando indice

*(p + 5) = 20       // atribui usando aritmética de ponteiros
...
```

3.6.3. Ponteiros e Vetores

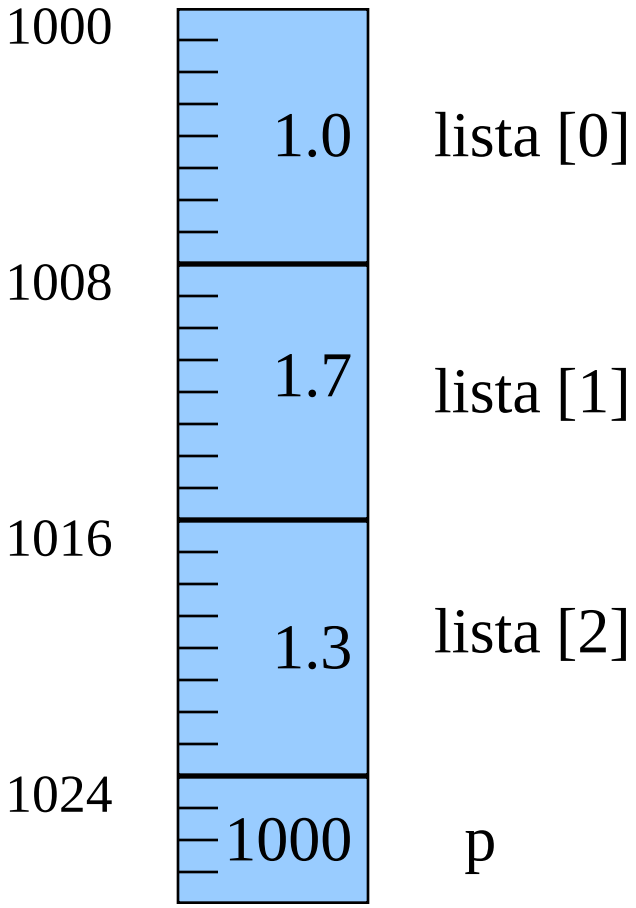
Sejam as declarações:

- *double lista [3];*
- *double *p;*



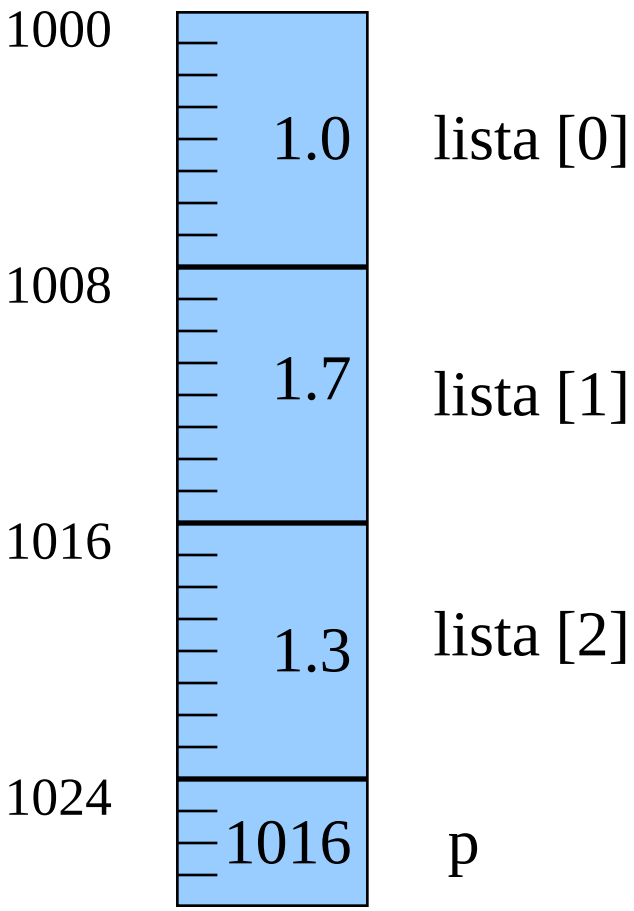
3.6.3. Ponteiros e Vetores

- **Sejam as atribuições:**
  - *lista [0] = 1.0;*
  - *lista [1] = 1.7;*
  - *lista [2] = 1.3;*
  - *p = &lista [0];*



3.6.3. Ponteiros e Vetores

- **Seja a operação:**
  - *p = p + 2;*
    - p passa a apontar para o elemento que aparece dois elementos depois de *&lista [0];*
    - Para cada unidade adicionada a p, seu valor numérico interno é acrescido de 8
    - Aritmética leva em conta o tamanho do tipo base



3.6.3. Ponteiros e Vetores

- **Aritmética de ponteiros**
  - Operações /, * e % não podem ser usadas com operandos do tipo ponteiro
  - Ponteiros **não** podem ser somados
  - Ponteiros podem ser subtraídos
    - A operação de subtração ocorre de forma semelhante à subtração de inteiros
    - Operação retorna o número de elementos entre os dois ponteiros

3.6.4. Vetores de Ponteiros

- **Assim como qualquer outro tipo de dados, ponteiros podem ser organizados em vetores**

  – Ex.: int *x[10];

```
x[2] = &var  // atribui o endereco de var a  x[2]
*x[2]        // retorna o valor de var
```

3.6.5. Ponteiros para Ponteiros

- **Um ponteiro pode indicar outro ponteiro**
  - O primeiro ponteiro contém o endereço do segundo ponteiro, que aponta para uma variável com o valor desejado
  - Declaração

    tipo **nome;

  - Pode ser usado em matrizes

3.6.5. Ponteiros para Ponteiros

```
// Programa: ponteiro para ponteiro
#include <stdio.h>

void main(void){
  int  x, *p, **q;

  x = 10;
  p = &x;
  q = &p;

  printf("%d", **q);     // imprime o valor de x
}
```