CURSO SUPERIOR DE ADS

Ponteiros

Prof. Fernando Marlon Soares Figueiredo

Disciplina: Programação Estruturada

SECTRAS

ADS

Aula de Hoje

- Ponteiros em C.
- Equivalência com vetores.
- Questões objetivas.
- Exercícios.

Introdução

- Um dos principais diferenciais na utilização da linguagem C em relação a outras linguagens de programação atuais é o uso de ponteiros.
- Eles permite um maior entendimento e controle de como a memória e suas variáveis estão organizadas.

Conceitos Básicos

- Ponteiros são tipos de variáveis especiais, que **armazenam o valor de endereços de memória**.

Ao invés de conter inteiros, caracteres ou números reais, os ponteiros podem conter:

- endereços de outras posições de memória.
- onde se encontram outras variáveis.
- espaços ocupados ou desocupados.

Conceitos Básicos

- Quando um programa está executando, cada linha de código é lida e executada pelo processador. O processador interage com a memória para ler, escrever e buscar os valores das variáveis envolvidas na memória primária (RAM).
- Cada variável possui 4 características básicas: **endereço**, **identificador**, **valor** e **tipo**.
- Assim, o funcionamento da memória é similar a um vetor, onde o endereço do dado é necessário para que ele seja acessado.

Visão simplificada da memória até a linha

```
1 ...
2     int a, b;
3     float c = 5.6;
4
5     a = 10;
6     b = a * 5;
7     c = 7.8;
8 ...
```

Endereço	Identificador	Valor	Tipo
...	...	...	...
1000	*a*	?	*int*
1001	*b*	?	*int*
1002	*c*	5.6	*float*
1003	?	?	?
...	...	...	...

Visão simplificada da memória até a linha

```
1 ...
2     int a, b;
3     float c = 5.6;
4
5     a = 10;
6     b = a * 5;
7     c = 7.8;
8 ...
```

Endereço	Identificador	Valor	Tipo
...	...	...	...
1000	*a*	10	*int*
1001	*b*	50	*int*
1002	*c*	7.8	*float*
1003	?	?	?
...	...	...	...

Ponteiros

- A diferença entre variáveis não ponteiros e ponteiros está no que é armazenado no campo **valor** da memória.
- Variáveis não ponteiros armazenam valores relacionados ao tipo de dado, enquanto variáveis do tipo ponteiro armazenam endereços de memória, ou seja, **apontam para outros espaços da memória, incluindo para outras variáveis**.
- Dessa forma, **é possível acessar e modificar o conteúdo de uma variável por meio de outra variável do tipo ponteiro que possui seu endereço no campo valor**.

Tipo de dado do ponteiro

- É importante frisar que, assim como variáveis não ponteiros, **ponteiros também precisam de tipos de dados para operarem.**
- Por exemplo, não é possível manipular um ponteiro de float com uma variável int.

Para declarar uma variável do tipo ponteiro, utiliza-se a seguinte sintaxe:

```
tipo *nome_variável;
```

Acesso de variáveis

1. **Acesso ao campo valor**: válido para qualquer tipo de variável (inclusive ponteiro), permite acessar o campo valor. Basta utilizar o próprio nome da variável.
2. **Acesso ao campo endereço**: válido para qualquer tipo de variável (inclusive ponteiros), permite acesso ao campo endereço. Utiliza-se o símbolo & antes de uma variável.
3. **Acesso ao campo valor apontado**: válido apenas para variáveis do tipo ponteiro, permite acesso ao campo valor da variável apontada. Utiliza-se o símbolo * antes de uma variável do tipo ponteiro (não confundir com o símbolo * da declaração, que possui significado diferente).

3 formas de acesso a variável ponteiro

```c
#include <stdio.h>

int main(void) {
    int a, b;   //variáveis 'normais'
    int *ptr;   //variável ponteiro

    a = 5;
    ptr = &a;
    b = *ptr;

    printf("Valor: a = %d; b = %d; ptr = %p\n", a, b, ptr);
    printf("Endereço: &a = %p; &b = %p; &ptr = %p\n", &a, &b, &ptr);
    printf("Valor apontado: *ptr = %d\n", *ptr);

    return 0;
}
```

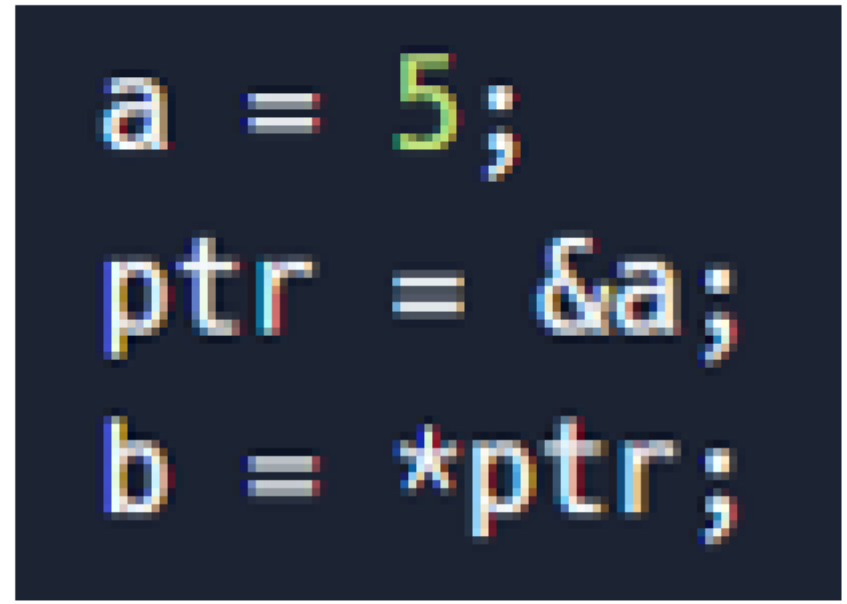
```
Valor: a = 5; b = 5; ptr = 0x7ffc3e5b95e4
Endereço: &a = 0x7ffc3e5b95e4; &b = 0x7ffc3e5b95e0; &ptr = 0x7ffc3e5b95d8
Valor apontado: *ptr = 5
```

Visão da Memória do Computador

Endereço	Identificador	Valor	Tipo
...	...	...	...
0x7f 1e0	*a*	5	*int*
0x7f 1e4	*b*	5	*int*
0x7f 1e8	*ptr*	0x7f 1e0	**int*
...	...	...	...

Resumindo...

- A variável a recebe em seu campo **valor** o inteiro 5.
- A variável ptr recebe em seu campo **valor** o **endereço** da variável a (ptr está agora apontando para a).
- Por fim, a variável b recebe o **valor** apontado por ptr, ou seja, b recebe o número que está no campo **valor** da variável a (o valor 5).



%p

Observe que, para imprimir endereços em hexadecimal, que é seu formato mais usual para representar endereços, utiliza-se o campo %p com a função printf.

Equivalência com Vetores

- Quando criamos um vetor com N posições, o espaço alocado seria equivalente à criação de N variáveis, mas garantidamente contíguas e referenciadas por um único identificador.
- Na verdade, a variável de um vetor quando criada, consiste em um ponteiro que aponta para seu próprio endereço, permitindo acessar outros espaços contíguos pelo deslocamento de posições de memória.
- Esse endereço é equivalente ao primeiro endereço do vetor.

Equivalência com Vetores

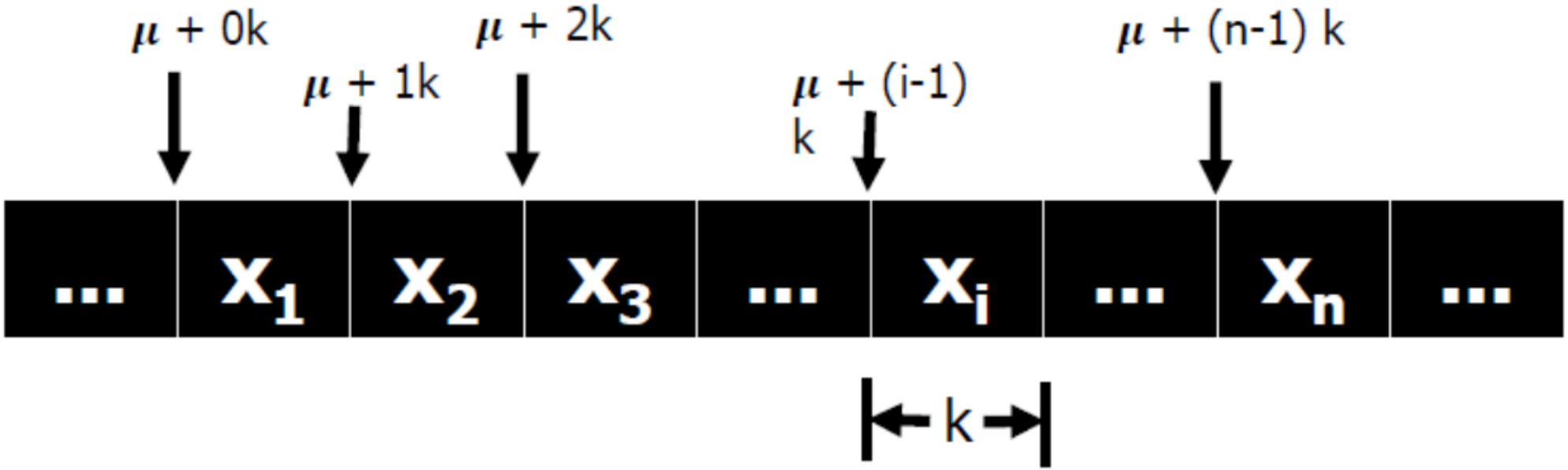
- Sendo v um vetor de inteiros com N posições, tem-se que &v, v e &v[0] correspondem ao mesmo endereço de memória, ou seja, o primeiro elemento do vetor.
- Assim, o valor do primeiro elemento pode ser obtido tanto por v[0] quanto por *v.

Vetores Utilizam uma Alocação Sequencial

- Após alocar um bloco de memória...
- Podemos supor que existe um item x_i que foi alocado na célula de endereço μ. Além disso, cada célula ocupa um tamanho fixo de ***k*** bytes na memória.
- Assim, podemos calcular rapidamente qualquer ***i-ésimo*** endereço da lista usando a formula:

- endereço $(x_i) = \mu + (i - 1) * k$

Esquema de Alocação Sequencial



μ – endereço de memória do começo da lista.

i – é a posição de um elemento na lista.

x_i – um item da lista.

k – quantidade de bytes.

Equivalência entre Ponteiros e Vetores

```c
#include <stdio.h>

int main(void) {
  int v[3] = {1, 3, 5 };
  int *ptr;

  printf("&v = %p; v = %p; &v[0] = %p\n", &v, v, &v[0]);

  ptr = v;
  *ptr = 10;
  ptr = ptr + 2;
  *ptr = 20;

  for (int i=0; i<3; i++) {
    printf("v[%d] : %d\n", i, v[i]);
  }

  return 0;
}
```

```
&v = 0x7ffc10289b68; v = 0x7ffc10289b68; &v[0] = 0x7ffc10289b68
v[0] : 10
v[1] : 3
v[2] : 20
```

Questões Objetivas

Questão 1

O que é um ponteiro em linguagem C?

a) Um tipo de dado que armazena números inteiros.

b) Uma variável que contém uma referência à posição de memória de outra variável.

c) Um operador utilizado para realizar operações matemáticas.

d) Uma função que aloca memória dinamicamente.

Questão 1

Resposta: b

O que é um ponteiro em linguagem C?

a) Um tipo de dado que armazena números inteiros.

b) Uma variável que contém uma referência à posição de memória de outra variável.

c) Um operador utilizado para realizar operações matemáticas.

d) Uma função que aloca memória dinamicamente.

Questão 2

Qual é a operação usada para obter o endereço de memória de uma variável em C?

a) &

b) *

c) /

d) +

Questão 2

Resposta: a

Qual é a operação usada para obter o endereço de memória de uma variável em C?

a) &

b) *

c) /

d) +

Questão 3

Qual é o operador usado para acessar o valor apontado por um ponteiro?

a) *

b) &

c) /

d) ->

Questão 3

Resposta: a

Qual é o operador usado para acessar o valor apontado por um ponteiro?

a) *

b) &

c) /

d) ->

Questão 4

Qual das seguintes declarações define corretamente um ponteiro para um inteiro em C?

a) int *ptr;

b) ptr int;

c) ptr *int;

d) pointer(int) ptr;

Questão 4

Resposta: a

Qual das seguintes declarações define corretamente um ponteiro para um inteiro em C?

a) int *ptr;

b) ptr int;

c) ptr *int;

d) pointer(int) ptr;

Questão 5

```c
#include <stdio.h>

int main() {
    int x = 10;
    int *ptr1 = &x;
    int *ptr2 = ptr1;

    (*ptr2)++;

    printf("Valor de x: %d\n", x);

    return 0;
}
```

Qual o valor de x?

a) 9

b) 10

c) 11

d) Isso causará um erro de compilação.

Questão 5

Resposta: c

```c
#include <stdio.h>

int main() {
    int x = 10;
    int *ptr1 = &x;
    int *ptr2 = ptr1;

    (*ptr2)++;

    printf("Valor de x: %d\n", x);

    return 0;
}
```

Qual o valor de x?

a) 9

b) 10

c) 11

d) Isso causará um erro de compilação.

Questão 6

```c
#include <stdio.h>

int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int *ptr = arr + 2;

    printf("%d\n", *ptr);

    return 0;
}
```

Qual será o resultado impresso para *ptr após a execução deste código?

a) 1

b) 2

c) 3

d) 4

Questão 6

Resposta: c

```c
#include <stdio.h>

int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int *ptr = arr + 2;

    printf("%d\n", *ptr);

    return 0;
}
```

Qual será o resultado impresso para *ptr após a execução deste código?

a) 1

b) 2

c) 3

d) 4

Exercícios

Exercício 1

- Declare uma variável inteira x e um ponteiro ptr que aponta para x.
- Atribua um valor a x e imprima o valor de x e o valor apontado por ptr.

```
Valor de x: 10
Valor apontado por ptr: 10
```

Exercício 1 - Resposta

```c
#include <stdio.h>

int main() {
    int x = 10;
    int *ptr = &x;

    printf("Valor de x: %d\n", x);
    printf("Valor apontado por ptr: %d\n", *ptr);

    return 0;
}
```

Exercício 2

- Declare uma variável x e um ponteiro ptr que aponta para x.
- Use o ponteiro para alterar o valor de x e imprima o novo valor de x.

```
Novo valor de x: 20
```

Exercício 2 - Resposta

```c
#include <stdio.h>

int main() {
    int x = 10;
    int *ptr = &x;

    *ptr = 20;

    printf("Novo valor de x: %d\n", x);

    return 0;
}
```

Exercício 3

- Declare uma variável x e um ponteiro ptr que aponta para x.
- Use o operador de incremento (++) para incrementar o valor de x por meio do ponteiro e imprima o novo valor de x.

```
Novo valor de x: 6
```

Exercício 3 - Resposta

```c
#include <stdio.h>

int main() {
    int x = 5;
    int *ptr = &x;

    (*ptr)++; // Incremento usando o ponteiro

    printf("Novo valor de x: %d\n", x);

    return 0;
}
```

Exercício 4

- Declare um array de inteiros e um ponteiro do tipo inteiro que aponta para este array.
- Atribua o endereço do primeiro elemento do array ao ponteiro e imprima os valores do array usando tanto o array quanto o ponteiro.

```
arr[0] = 1, ptr[0] = 1
arr[1] = 2, ptr[1] = 2
arr[2] = 3, ptr[2] = 3
arr[3] = 4, ptr[3] = 4
arr[4] = 5, ptr[4] = 5
```

Exercício 4 - Resposta

```c
#include <stdio.h>

int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int *ptr = arr;

    for (int i = 0; i < 5; i++) {
        printf("arr[%d] = %d, ", i, arr[i]);
        printf("ptr[%d] = %d\n", i, ptr[i]);
    }

    return 0;
}
```

Exercício 5

Faça um programa que peça para o usuário digitar dois valores inteiros e, em seguida, imprima o endereço da variável de maior valor.

Exercício 6

- Faça um programa que declare um array de inteiros com cinco posições, peça para o usuário preencher suas informações e imprima-as na tela.
- Todas as operações de acesso ao vetor (exceto sua declaração) devem ser feitas com notação de ponteiros.

```
int v[3] = {1, 3, 5};

int i, *ptr;
```

```
ptr = v;

*ptr = 10;

ptr = ptr + 2;

*ptr = 20;
```

Exercício 7

- Faça um programa que troque os dois primeiros valores de lugar em um vetor com três posições.
- O programa deve atribuir os valores das variáveis na declaração. Todas as operações só podem ser feitas através de dois ponteiros que são inicializados apontando para o primeiro elemento do vetor.
- Todas as operações devem ser feitas manipulando apenas as variáveis do tipo ponteiro, sem qualquer acesso ao vetor. Ao final, imprima os dois primeiros elementos do vetor.