



UNIVERSIDADE FEDERAL DO PIAUÍ - UFPI
CAMPUS SENADOR HELVÍDIO NUNES DE BARROS - CSHNB
CURSO DE SISTEMAS DE INFORMAÇÃO
PICOS - PI

VETORES E PONTEIROS

Prof. Ma. Luana Batista da Cruz
luana.b.cruz@nca.ufma.br

Roteiro

2

- Ponteiros e vetores
- Passagem de vetores como parâmetros de funções
- Retornar ponteiro pela função
- Atividade

Vetores e ponteiros

3

- ❑ Para o C, um vetor é um apontador para a sua primeira posição (índice 0):
- ❑ O **nome de um vetor** é sinônimo do endereço da posição inicial do vetor
- ❑ `int v[10];`

Vetores e ponteiros

4

- ❑ Para o C, um vetor é um apontador para a sua primeira posição (índice 0):
- ❑ `int v[10];`
 - **Atribuir:** `v` ou `&v[0];`



Vetores e ponteiros

5

- ❑ Para o C, um vetor é um apontador para a sua primeira posição (índice 0):
- ❑ `int v[10];`
 - **Acessar:** `*v` ou `v[0];`



Vetores e ponteiros


6

- ❑ Para o C, um vetor é um apontador para a sua primeira posição (índice 0):
- ❑ O **nome de um vetor** é sinônimo do endereço da posição inicial do vetor
- ❑ Para o C, `int v[]` e `int *v` são sinônimos

Passagem de vetores como parâmetros de funções

7

- ❑ Para passar vetores como parâmetros de funções, sempre declaramos o tipo do parâmetro como um apontador para o tipo do vetor. O C não tem como saber o tamanho do vetor
- ❑ Se for preciso, temos que passar o tamanho em um parâmetro separado



```
int f_media (int *v, int quantidade){  
    int i, soma;  
    for(i = 0; i < quantidade; i++)  
        soma +=v[i];  
    media = soma/quantidade;  
    return media;  
}
```

Ponteiros e vetores

8

- ❑ Existe uma **estreita relação** entre **ponteiros** e **vetores**. Considere este fragmento:

```
int vet[80], *p1;
```

```
p1 = vet;
```

- ❑ Aqui, **p1** foi inicializado com o endereço do primeiro elemento do vetor **vet**. Para acessar o **quinto** elemento em **vet**, teria que ser escrito

```
vet[4]
```

ou

```
*(p1+4)
```


Ponteiros e vetores

9

- ❑ Existe uma **estreita relação** entre **ponteiros** e **vetores**. Considere este fragmento:

```
int vet[80], *p1;  
p1 = vet;
```

- ❑ Aqui, **p1** foi inicializado com o endereço do primeiro elemento do vetor **vet**. Para acessar o **quinto** elemento em **vet**, teria que ser escrito

vet[4]
ou
*(p1+4)



Portanto, C fornece **dois métodos** para acessar elementos em vetores:

- Indexação de vetores
- Aritmética de ponteiros

Ponteiros e vetores

10

- ❑ Podemos somar ou subtrair posições em um ponteiro
- ❑ A soma irá incrementar a próxima posição do vetor
- ❑ Os elementos do vetor estão contíguos na memória

```
int v[100];
```

```
int *pv;
```

```
pv = &v[0]; // também pode ser pv = v
```

```
*pv = 0; // é o mesmo que v[0] = 0;
```

```
*(pv+i) = 0; // é o mesmo que v[i] = 0;
```

```
for (i=0; i<100; i++)
```

```
    *(pv+i) = 0; // zera o vetor v
```

Ponteiros e vetores

11

- ❑ O nome do vetor também é um ponteiro:

```
int v[100];
```

```
*v = 33; é o mesmo que v[0] = 33;
```

```
*(v+2) = 44; é o mesmo que v[2] = 44;
```

```
for (i = 0; i < 100; i++)
```

```
    *(v+i) = 0; //zera o vetor v
```

Ponteiros e vetores

12

- ❑ Podemos também usar um ponteiro como um vetor (indexando)

```
int v[100];
```

```
int *pv;
```

```
pv = v; //também pode ser pv = &v[0];
```

```
for (i = 0; i < 100; i++)
```

```
    pv[i] = 0;
```

Atividades

13

- ❑ Considere a função zera a seguir e responda o que se pede

```
void zera (int a[], int n) {  
    int i;  
    for (i = 0; i < n; i++)  
        a[i] = 0;  
}
```

- ❑ O que acontece em cada chamada da função abaixo? Suponha int x[10]. Suponha também que o vetor x foi inicializado anteriormente
 - a) zera(x,5);
 - b) zera(&x[5], 5);
 - c) zera(x + 5, 5);

Atividades

14

- ❑ Considere a função zera a seguir e responda o que se pede

```
void zera (int a[], int n) {  
    int i;  
    for (i = 0; i < n; i++)  
        a[i] = 0;  
}
```

0	1	2	3	4	5	6	7	8	9
8	7	4	9	30	5	2	1	6	23

- ❑ int x[10]
- a) **zera(x,5);**
 - b) zera(&x[5], 5);
 - c) zera(x + 5, 5);

Atividades

15

- ❑ Considere a função zera a seguir e responda o que se pede

```
void zera (int a[], int n) {  
    int i;  
    for (i = 0; i < n; i++)  
        a[i] = 0;  
}
```

0	1	2	3	4	5	6	7	8	9
					5	2	1	6	23

- ❑ int x[10]
- a) **zera(x,5);**
 - b) zera(&x[5], 5);
 - c) zera(x + 5, 5);

Atividades

16

- ❑ Considere a função zera a seguir e responda o que se pede

```
void zera (int a[], int n) {  
    int i;  
    for (i = 0; i < n; i++)  
        a[i] = 0;  
}
```

0	1	2	3	4	5	6	7	8	9
8	7	4	9	30	5	2	1	6	23

- ❑ int x[10]
- a) zera(x,5);
 - b) **zera(&x[5], 5);**
 - c) zera(x + 5, 5);

Atividades

17

- ❑ Considere a função zera a seguir e responda o que se pede

```
void zera (int a[], int n) {  
    int i;  
    for (i = 0; i < n; i++)  
        a[i] = 0;  
}
```

0	1	2	3	4	5	6	7	8	9
8	7	4	9	30					

- ❑ int x[10]
- a) zera(x,5);
 - b) zera(&x[5], 5);**
 - c) zera(x + 5, 5);

Atividades

18

- ❑ Considere a função zera a seguir e responda o que se pede

```
void zera (int a[], int n) {  
    int i;  
    for (i = 0; i < n; i++)  
        a[i] = 0;  
}
```

0	1	2	3	4	5	6	7	8	9
8	7	4	9	30	5	2	1	6	23

- ❑ int x[10]
- a) zera(x,5);
 - b) zera(&x[5], 5);
 - c) **zera(x + 5, 5);**

Atividades

19

- ❑ Considere a função zera a seguir e responda o que se pede

```
void zera (int a[], int n) {  
    int i;  
    for (i = 0; i < n; i++)  
        a[i] = 0;  
}
```

0	1	2	3	4	5	6	7	8	9
8	7	4	9	30					

- ❑ int x[10]
- a) zera(x,5);
 - b) zera(&x[5], 5);
 - c) **zera(x + 5, 5);**

Atividades

20

❏ Assumindo que $V1[]$ é um vetor do tipo int. Quais das seguintes expressões mostram o valor do terceiro elemento de $V1$?

a) $*(V1 + 2)$

b) $*(V1 + 4)$

c) $V1 + 4$

d) $V1 + 2$

Atividades

21

- ❑ Assumindo que `V1[]` é um vetor do tipo `int`. Quais das seguintes expressões mostram o valor do terceiro elemento de `V1`?

a) `*(V1 + 2)`

b) `*(V1 + 4)`

c) `V1 + 4`

d) `V1 + 2`

Atividades

22

- ❑ Considere a declaração:

```
int vet[4], *p, x;
```

- ❑ Quais expressões são válidas?

- a) `p = vet + 1;`
- b) `p = vet++;`
- c) `x = (*vet)++;`

Atividades

23

- ❏ Considere a declaração:

```
int vet[4], *p, x;
```

- ❏ Quais expressões são válidas?

- a) **p = vet + 1;**
- b) **p = vet++;**
- c) **x = (*vet)++;**

Atividades

24

- ❑ Considere a declaração:

```
int vet[4], *p, x;
```

- ❑ Quais expressões são válidas?

a) $p = \text{vet} + 1;$

$p = \text{vet}[1];$

b) $\text{bp} = \text{vet}++;$

c) $x = (*\text{vet})++;$

Atividades

25

- ❑ Considere a declaração:

```
int vet[4], *p, x;
```

- ❑ Quais expressões são válidas?

a) `p = vet + 1;`

b) `p = vet++;`

`p = vet;`

`vet = vet + 1;` ❌

c) `x = (*vet)++;`

Atividades

26

- ❏ Considere a declaração:

```
int vet[4], *p, x;
```

- ❏ Quais expressões são válidas?

a) `p = vet + 1;`

b) `p = vet++;`

c) `x = (*vet)++;`

`x = *vet;`

`*vet = *vet + 1;`

Retornar ponteiro pela função

27

- ❑ Uma função pode retornar qualquer tipo de dado válido em C, portanto pode retornar um ponteiro! Uma função que retorna um ponteiro deve declarar explicitamente qual o tipo de ponteiro que está retornando

```
int *divide_vetor(int a[], int n, int indice) {  
    if (indice >= n)  
        return NULL;  
  
    return &a[indice];  
}
```

Retornar ponteiro pela função

28

- ❑ Uma função pode retornar qualquer tipo de dado válido em C, portanto pode retornar um ponteiro! Uma função que retorna um ponteiro deve declarar explicitamente qual o tipo de ponteiro que está retornando

```
int *divide_vetor(int *a, int n, int indice) {  
    if (indice >= n)  
        return NULL;  
  
    return a+indice;  
}
```

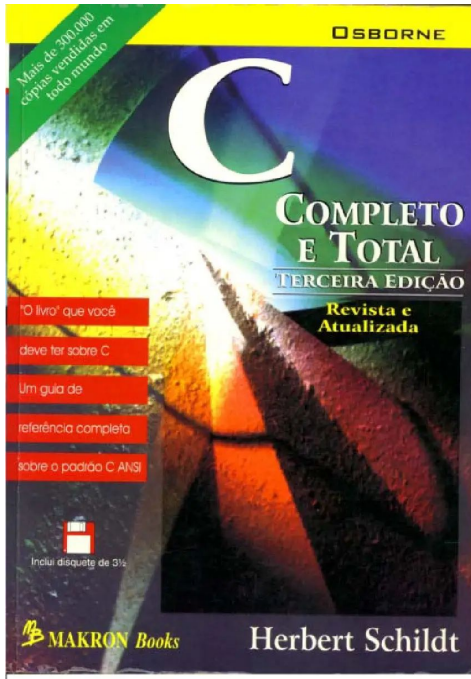
Atividade

29

- ❑ Construa um programa em C que faça uso da função “divide_vetor”
 - a) Construa a função main
 - b) Informe um vetor, o tamanho do vetor (n) e o índice de divisão do vetor que será passado para a função “divide_vetor”
 - c) Construa um função que imprima o vetor
- void imprimir_vetor(int *a, int n)**

Referências

30



SCHILD, Herbert. **C completo e total**. Makron, 3ª edição revista e atualizada, 1997.



SZWARCHFITER, J. **Estruturas de Dados e seus algoritmos**. 3 ed. Rio de Janeiro: LTC, 2010.