

Ponteiros

Disciplina: Estrutura de Dados

Universidade Federal do Pampa – Unipampa – Campus Bagé

lucianobrum@unipampa.edu.br e marinagomes@unipampa.edu.br

pointer

70043573

Tópicos



- Endereçamento de memória;
- Ponteiros:
 - Declaração;
 - Operadores;
 - Operações aritméticas;
- Passagem de ponteiros para funções;
 - Variáveis globais e estáticas;
- Ponteiros e Vetores;
- Ponteiros e Matrizes.

Endereçamento de Memória

- A memória RAM dos computadores em geral é uma sequência de bytes. Cada byte armazena um de 256 possíveis valores. Os bytes são numerados sequencialmente e o número de um byte é o seu **endereço** (*address*).
- Cada objeto na memória do computador ocupa um certo número de bytes consecutivos. Um char ocupa 1 byte. Um int ocupa 4 bytes e um double ocupa 8 bytes em muitos computadores.
- O número exato de bytes de um objeto é dado pelo operador **sizeof**: a expressão `sizeof(int)`, por exemplo, dá o número de bytes de um int no seu computador.

Endereçamento de Memória

- Cada objeto na memória tem um **endereço**. Na maioria dos computadores, o endereço de um objeto é o endereço do seu primeiro byte.

- Por exemplo, depois das declarações:

```
char c;  
int i;  
struct{  
    int x, y;  
} ponto;  
int v[4];
```

- Os endereços das variáveis poderiam ser os seguintes:

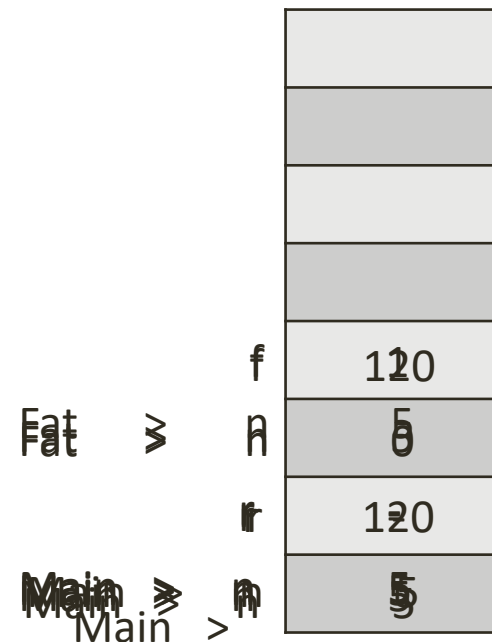
c	89421
i	89422
Ponto	89426
v[0]	89434
v[1]	89438
v[2]	89442

Endereçamento de Memória

- Exemplo didático: Esquema representativo da memória do computador (modelo de pilha).

```
#include<stdio.h>
int fat(int n);
int main(void){
    int n = 5;
    int r;
    r = fat(n);
    printf("Fat de %d = %d.\n",n,r);
    return 0;
}
int fat(int n){
    int f = 1;
    while(n!=0){
        f*=n;
        n--;
    }
    return f;
}
```

1- Declaração de função só pode ser feita uma vez.



Endereçamento de Memória

➤ Problema:

➤ Sabemos que:

- Uma função pode retornar **APENAS** um valor, de qualquer tipo, através do comando ***return***.
- Uma função pode receber **nenhum**, **um** ou **vários** argumentos de quaisquer tipos.
- A possibilidade de retornar um valor **nem sempre é satisfatória**. Muitas vezes precisamos transferir **mais de um resultado** para a função que chama, o que não é possível com retorno explícito de valores.

Endereçamento de Memória

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

```
void somaproduct(int a, int b, int c, int d){
```

```
    c = a + b;
```

```
    d = a * b;
```

```
}
```

```
int main(void){
```

```
    int s, p;
```

```
    somaproduct(3,5,s,p);
```

```
    printf("soma = %d produto = %d", s, p);
```

```
    return 0;
```

```
}
```

Qual o problema do código ao lado?

Ponteiros

- Um ponteiro (= apontador = *pointer*) é um tipo especial de variável que armazena endereços de memória em que existem valores do tipo correspondente. Um ponteiro pode ter o valor especial NULL que não é endereço de lugar algum.
- Se um ponteiro 'p' armazena o endereço de uma variável 'i', dizemos que "p aponta para i" (em termos mais abstratos, diz-se que 'p' é uma referência à variável 'i').

- Quando fazemos:

- `int a;`

Reserva-se um espaço de memória de 4 bytes para armazenar um inteiro.

- `int *p;`

Reserva-se um espaço de memória para armazenar um endereço, no qual existe um número inteiro.



Declaração de Ponteiros

- Há vários tipo de ponteiros: ponteiros para caracteres, ponteiros para inteiros, ponteiros para ponteiros para inteiros, ponteiros para registros (*structs*), etc. O computador precisa saber o tipo de ponteiro que você está falando.

- Para declarar um ponteiro p para inteiro:

```
int *p;
```

- Para declarar um ponteiro p para um registro reg:

```
struct reg *p;
```

- Um ponteiro r para um ponteiro que apontará um inteiro é declarado assim:

```
int **r;
```

Operadores

- O operador unário `&` (leia-se “endereço de”): quando aplicado a uma variável resulta no endereço de memória reservado para esta variável.
- O operador unário `*` (também chamado de operador *indirection* ou *dereferencing* – no sentido de ser indireto, de ser derivativo de referenciar) e que pode ser lido como “conteúdo de”: quando aplicado a um ponteiro, ele acessa o conteúdo da variável que ele aponta.

Operadores

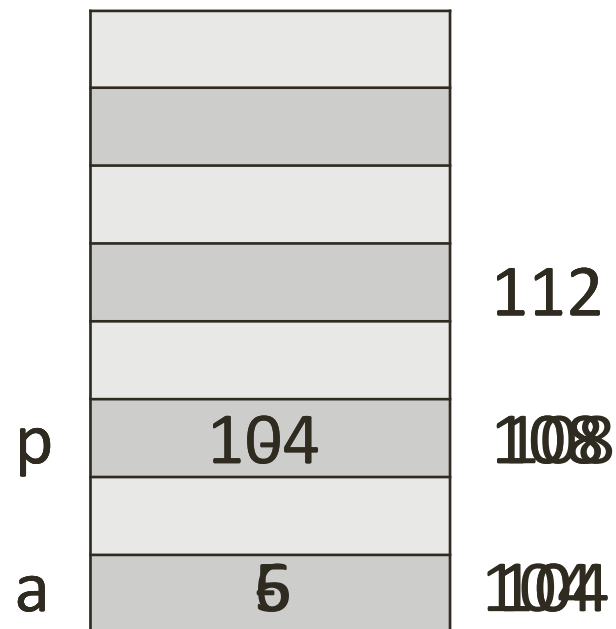
```
int a;
```

```
int *p;
```

```
a = 5;
```

```
p = &a;
```

```
*p = 6;
```



Operadores

```
int main (void) {  
    int a;  
    int *p;  
    p = &a; //int *p = &a;  
    *p = 2;  
    printf(" %d ", a);  
    return 0;  
}
```

```
int main (void) {  
    int a, b, *p;  
    a = 2;  
    *p = 3;  
    b = a + (*p);  
    printf(" %d ", b);  
    return 0;  
}
```

Operadores

```
int a, b, c;  
int *p;  
p = &a;
```

	Valor	Endereço
int a	10	3620
int b	20	3616
int p	3620	3612
int c	30	3608

Quais os valores de:

a =

b =

c =

&a =

*p =

**&p =

Operações Aritméticas com Ponteiros

```
int a = 5;
int *p;
p = &a;
int b = *p;      // b = 5
int x = *p + 1;  // x = 6, mas a continua 5
*p += 1;         // *p = *p + 1 = 6, a passa a se 6 tambem
++*p;           // *p = 7 = a
(*p)++;         // *p = 8 = a
*p++;           /* significa *(p++), i. e. p = p+1 contem o
                 proximo endereco e a fica inalterado e
                 igual a 8 */
p++;            // aumenta o endereco
*p = *p - 2;    //diminui o conteudo apontado pelo endereco
p = p - 2;      /* diminui o endereco em dois, ou seja,
                 volta para a */
```

Operações Aritméticas com Ponteiros

```
int main (void){  
    int b = 13;  
    int a = 15;  
    int *p;  
    p = &a;
```

	Valor	Endereço
int b	13	2293620
int a	15	2293616
int p	2293616	2293612

Qual a saída da função printf:


```
(*p)++;  
printf("%d %d %d %d %d", a, *p, &b, &a, &p);
```

```
p++;  
printf("%d %d %d %d %d", a, *p, b, &a, p);
```

```
p++;  
(*p)--;  
printf("%d %d %d %d %d", a, *p, b, &a, p);
```

Ponteiros e Funções

➤ Conforme visto anteriormente, não é possível alterar o valor das variáveis do main na função na passagem de argumentos por valor.



unipampa
Universidade Federal do Pampa

Endereçamento de Memória

```
#include <stdio.h>

void somaprod(int a, int b, int c, int d){
    c = a + b;
    d = a * b;
}

int main(void){
    int s, p;
    somaprod(3,5,s,p);
    printf("soma = %d produto = %d", s, p);
    return 0;
}
```

Qual o problema do código ao lado?

05/01/2018

[8]

➤ Vimos que é possível alterar o conteúdo de posições de memória de determinadas variáveis utilizando ponteiros...

➤ Como resolver o problema anterior utilizando ponteiros e funções?

Ponteiros e Funções

```
#include<stdio.h>

void somaproduct(int a, int b, int *c, int *d){
    *c = a + b;
    *d = a * b;
}

int main(void){
    int s, p;
    somaproduct(3,5,&s,&p);
    printf("soma = %d produto = %d", s, p);
    return 0;
}
```

Endereçamento de Memória

```
#include<stdio.h>
void somaproduct(int a, int b, int c, int d){
    c = a + b;
    d = a * b;
}
int main(void){
    int s, p;
    somaproduct(3,5,s,p);
    printf("soma = %d produto = %d", s, p);
    return 0;
}
```

1

Qual o problema do código ao lado?

05/01/2018

17/01/2018

[8]

[17]

Ponteiros e Funções

➤ Qual a saída do programa abaixo?

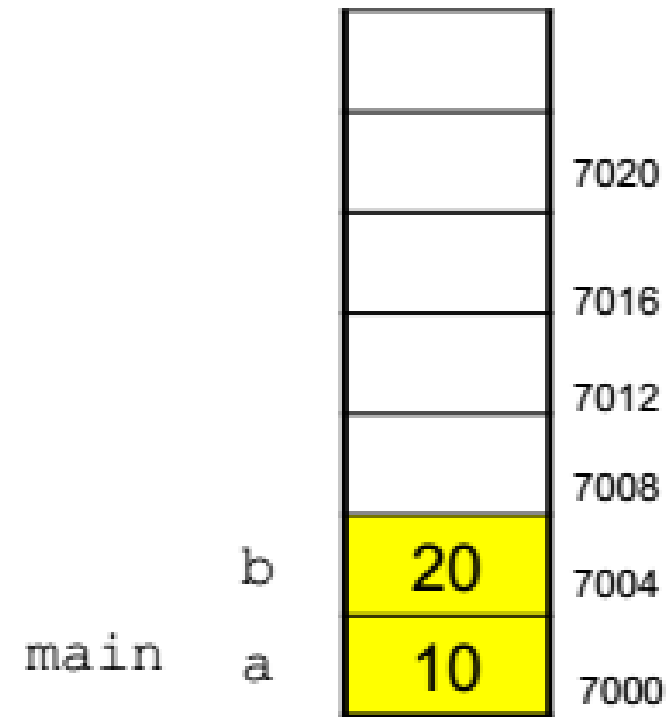
```
#include <stdio.h>

void troca(int a, int b);

int main (void)
{
    int a=10, b=20;
    troca(a,b);
    printf(" a=%d  b=%d\n",a,b);
}

void troca(int a, int b) {
    int tmp=b;
    b=a;
    a=tmp;
}
```

Pilha de memória ,



8107/T0/T

- 19

The diagram illustrates the state of memory stacks for three threads (main, troca, pb) at six different points in time, labeled 1 through 6. The stacks are labeled 'Pilha de memória' and contain variables 'a', 'b', 'pa', 'pb', and 'tmp' with their respective values and memory addresses.

Thread 1 (main):

- Variable 'a' is at address 7000, containing value 10.
- Variable 'b' is at address 7004, containing value 20.

Thread 2 (troca):

- Variable 'pa' is at address 7008, containing value 7000.
- Variable 'pb' is at address 7012, containing value 7004.
- Variable 'tmp' is at address 7016, containing value -.

Thread 3 (pb):

- Variable 'a' is at address 7000, containing value 10.
- Variable 'b' is at address 7004, containing value 20.
- Variable 'pa' is at address 7008, containing value 7000.
- Variable 'pb' is at address 7012, containing value 7004.
- Variable 'tmp' is at address 7016, containing value 20.

Thread 4 (main):

- Variable 'a' is at address 7000, containing value 10.
- Variable 'b' is at address 7004, containing value 10.
- Variable 'pa' is at address 7008, containing value 7000.
- Variable 'pb' is at address 7012, containing value 7004.
- Variable 'tmp' is at address 7016, containing value 20.

Thread 5 (troca):

- Variable 'pa' is at address 7008, containing value 7000.
- Variable 'pb' is at address 7012, containing value 7004.
- Variable 'tmp' is at address 7016, containing value 20.
- Variable 'a' is at address 7000, containing value 20.
- Variable 'b' is at address 7004, containing value 10.

Thread 6 (pb):

- Variable 'a' is at address 7000, containing value 20.
- Variable 'b' is at address 7004, containing value 10.

Variáveis globais e estáticas

- Outra forma de fazer comunicação entre funções consiste no uso de variáveis globais.
- Se a variável for declarada fora do corpo das funções, ela é global, pois será visível a todas as funções subsequentes.
- Tais variáveis não são armazenadas na pilha de execução, ou seja, não deixam de existir quando uma função termina de executar. Apenas deixam de existir quando o programa termina.

Variáveis globais e estáticas

- Variáveis globais não são boas porque qualquer parte do programa pode alterá-la.
- Isso é ruim porque ao dar manutenção é difícil saber onde ele é inicializada, para que serve, etc.
- Isso fica pior ainda se pensarmos em um ambiente *multithread*. Você pode simplesmente ler um valor em uma linha e na linha seguinte obter um valor completamente diferente.

Variáveis globais e estáticas

- Quando usar?
 - Quando um valor constante for necessário em todo o código (ex: π , constante de Planck, etc);
 - Quando for necessário um valor de referência para várias funções e métodos.

Variáveis globais e estáticas

- Variáveis estáticas: também não são armazenadas na pilha, mas sim numa área de memória estática que existe enquanto o programa é executado.
- Tais variáveis continuam existindo antes ou depois das funções serem executadas.
- Só é visível no escopo da função onde é declarada, porém, tal variável mantém seu valor mesmo após a função ter executado.

Variáveis globais e estáticas

- Exemplo prático de variável estática:

```
void imprime(float a){  
    static int n = 1;  
    printf(" %f ", a);  
    if((n%5)==0){  
        printf("\n");  
    }  
    n++;  
}
```

Cada vez que imprime é chamado, o valor de n é incrementado.

Se a variável estática não for inicializada explicitamente na declaração, é automaticamente setada em zero, assim como as globais.

Vetores e Ponteiros

- `int v[10];` //Cria um espaço de memória para dez números inteiros (40 bytes)
- Para acessar elementos do vetor: `v[0]`, `v[1]`, etc.
- `&v[i]` representa o endereço de memória do elemento do vetor na posição `i`.
- O '`v`' representa o endereço para o elemento inicial do vetor, o que nos permite usar aritmética de ponteiros.
- Se usarmos `(v+1)`, seria equivalente a `&v[1]`.

Vetores e Ponteiros

- $\&v[i]$ é o mesmo que $(v+i)$ e $v[i]$ é o mesmo que $*(v+i)$.
- Passar um vetor para uma função consiste em passar o endereço inicial do vetor, **e não uma cópia do vetor para a função!**
- Se passarmos o vetor para uma função e alterarmos esse vetor nela, as mudanças no vetor permanecerão após o encerramento da função?

Vetores e Ponteiros

```
#include<stdio.h>

void incr_vetor(int n, int *v){
    int i;
    for(i=0;i<n;i++){
        v[i]++;
    }
}

int main(void){
    int a[] = {1,3,5};
    incr_vetor(3,a);
    printf("%d %d %d \n",a[0],a[1],a[2]);//Qual a saída?
    return 0;
}
```

Vetores e Ponteiros

- Quais valores serão exibidos nos printf's??

```
int main (void){  
    int a[5] = {1,3,5,8,13};  
    printf("%d", *a);  
    printf("%d", *(a+0));  
    printf("%d", *(a+4));  
    printf("%d", *a+4);  
  
    int *p;  
    p = a + 2;  
    printf("%d", *(p+2));  
    printf("%d", *p+2);  
}
```

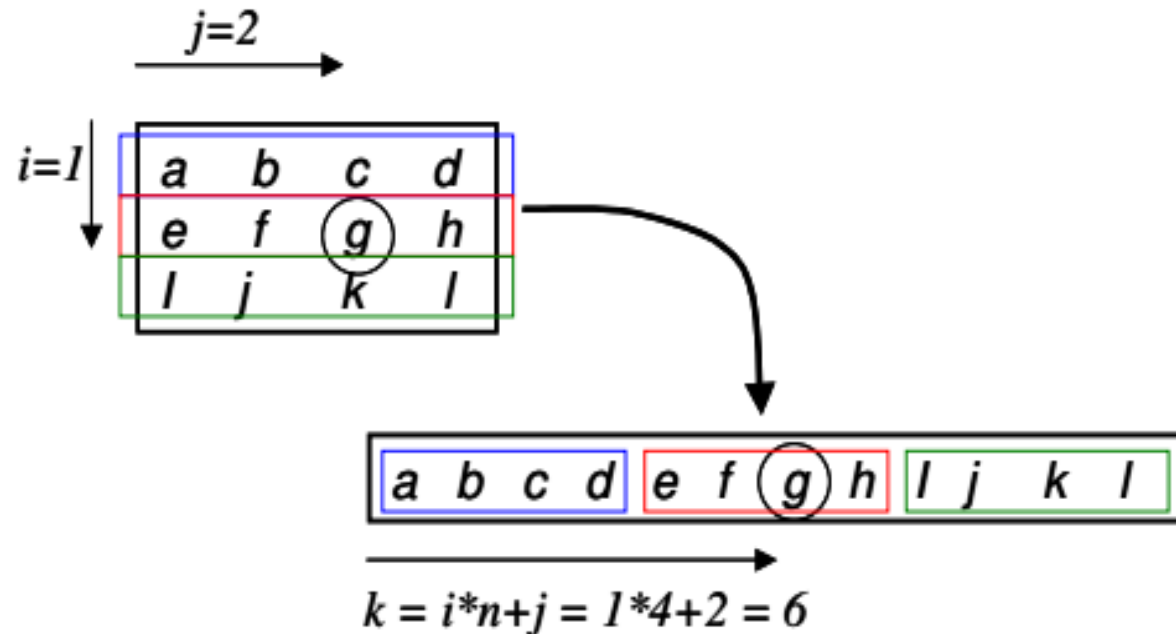
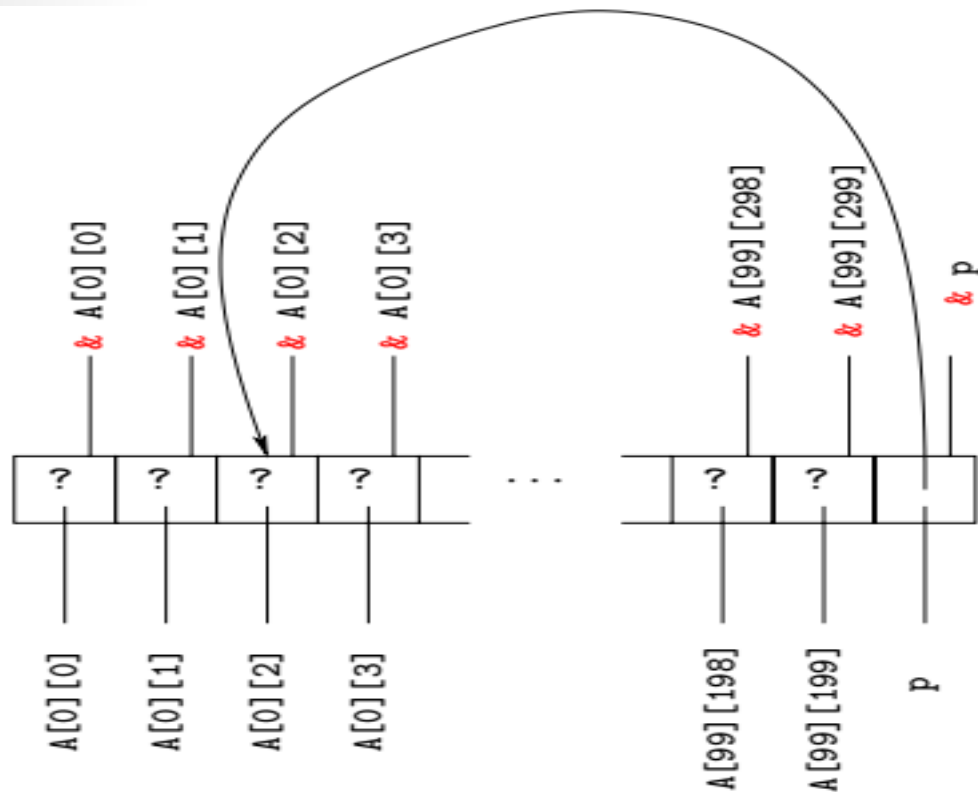
Variável	Valor	Endereço
a	1	30000
a	3	30004
a	5	30008
a	8	30012
a	13	30016

Matrizes e Ponteiros

- Utilizamos a representação de matrizes até agora como $A[\text{linha}][\text{coluna}]$, mas podemos representa-la como um vetor $A[\text{linha} * \text{coluna}]$, onde sua dimensão é o tamanho total da matriz.
- Para trabalharmos com ponteiros como utilizamos no exemplo anterior, utilizando deslocamentos, é necessário que uma variável do tipo ponteiro aponte para o primeiro elemento desta matriz.
- E o mapeamento dos elementos da matriz dentro do vetor será realizado pelos índices linha e coluna.

Matrizes e Ponteiros

- Necessário que uma variável do tipo ponteiro aponte para o primeiro elemento desta matriz.
- E o mapeamento dos elementos da matriz dentro do vetor será realizado pelos índices de linha e coluna (i e j) e pelo número de colunas (n) .



17/01/2018



Matrizes e Ponteiros

- Qual a saída para o programa abaixo:

```
int main (void) {  
    int a[5] = {3, 5, 8, 11, 13};  
    int i;  
  
    for(i=0; i<5; i++) {  
        *(a+i) += 1;  
    }  
  
    for(i=0; i<5; i++) {  
        printf("%d", *(a+i));  
    }  
}
```


Matrizes e Ponteiros

- Escreva uma função `mm` que recebe um valor inteiro `v[10]` e os endereços de duas variáveis inteiras, digamos `min` e `max`, e deposite nessas variáveis o valor do elemento mínimo e o valor de um elemento máximo do vetor. Escreva também uma função `main` que use a função.
- Seja o código abaixo usando passagem por referência. Analise o código e explique o resultado mostrando passo a passo as alterações ocorridas no vetor `a`.

```
void incrementa(int *x, int *y){
    *x = *x + (*y);
    (*y)++;
}
int main(){
    int a[] = {1,2,3};
    for (int i=0; i<3; i++){
        incrementa(&a[i], &a[1]);
        printf("\n %d", a[i]);
    }
}
```

Resumo da Aula

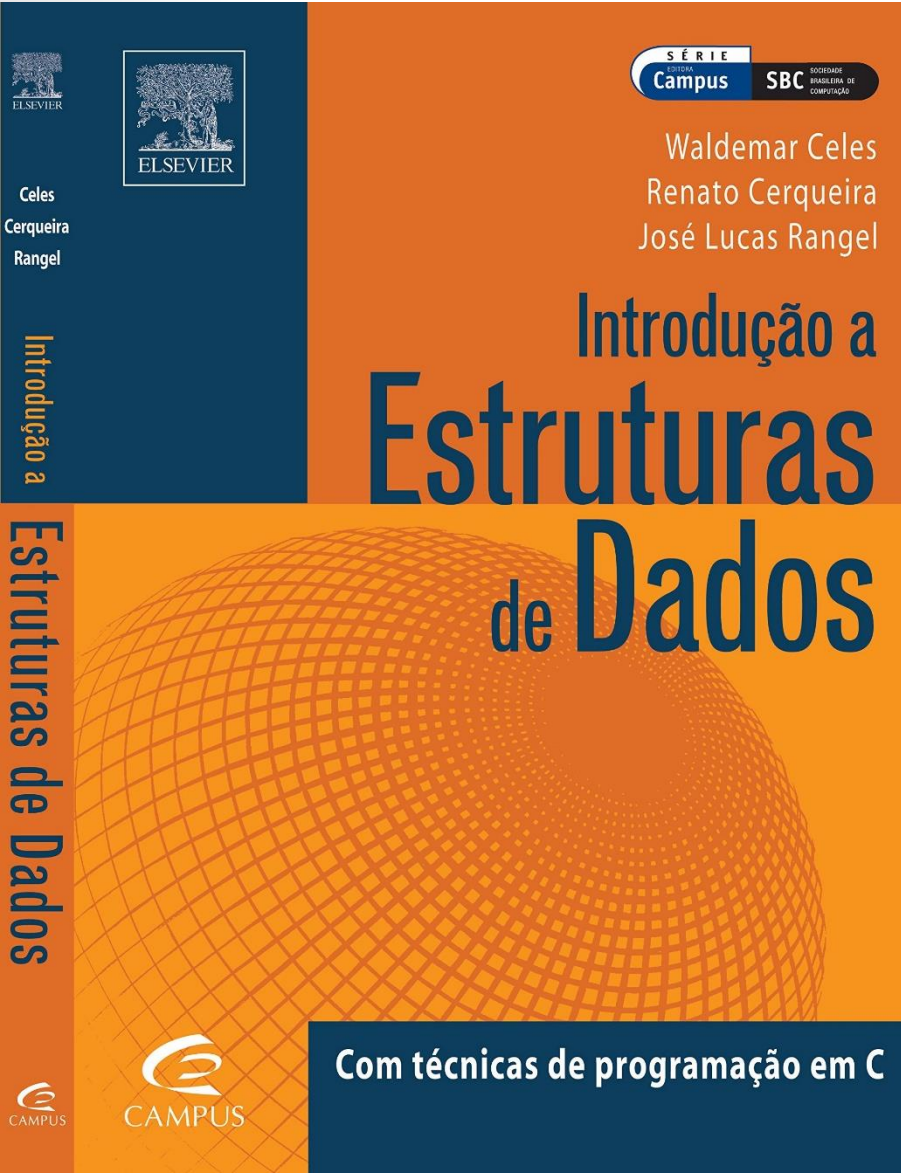
➤ Vimos hoje:

- Funcionamento básico dos endereços de memória em C;
- Uso, declaração e inicialização de ponteiros em C e seu significado, além de aplicações;
- Uso de ponteiros com funções e matrizes;
- Uso de variáveis globais e estáticas;

8107/T0/T

- 35

Referências



- CELES, Waldemar; CERQUEIRA, Renato; RANGEL, José Lucas. Introdução a Estruturas de Dados com técnicas de programação em C. Rio de Janeiro: Elsevier (Campus), 2004. 4ª Reimpressão. 294 p.

Dúvidas ?