Pontifícia Universidade Católica
do Rio de Janeiro

INF 1007 – Programação II

Aula 02 - Ponteiros

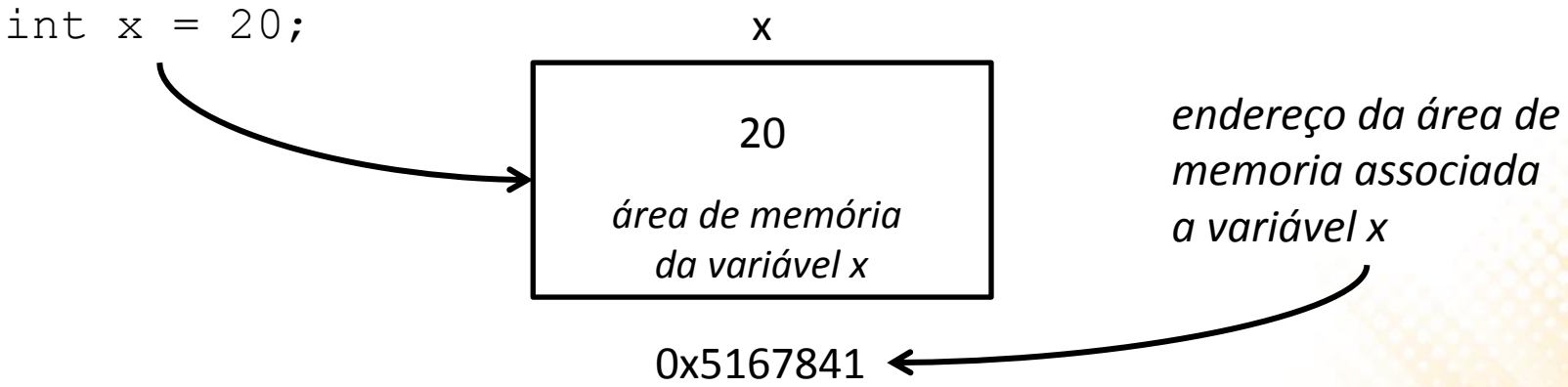
Edirlei Soares de Lima
<elima@inf.puc-rio.br>

visionlab
visualization laboratory

Endereço de uma Variável

- Toda variável definida em um programa ocupa uma área de memória;

- A cada área de memória está associado um endereço;
  - Um endereço é uma sequência de bits codificados da mesma forma que um número inteiro sem sinal;



Ponteiros

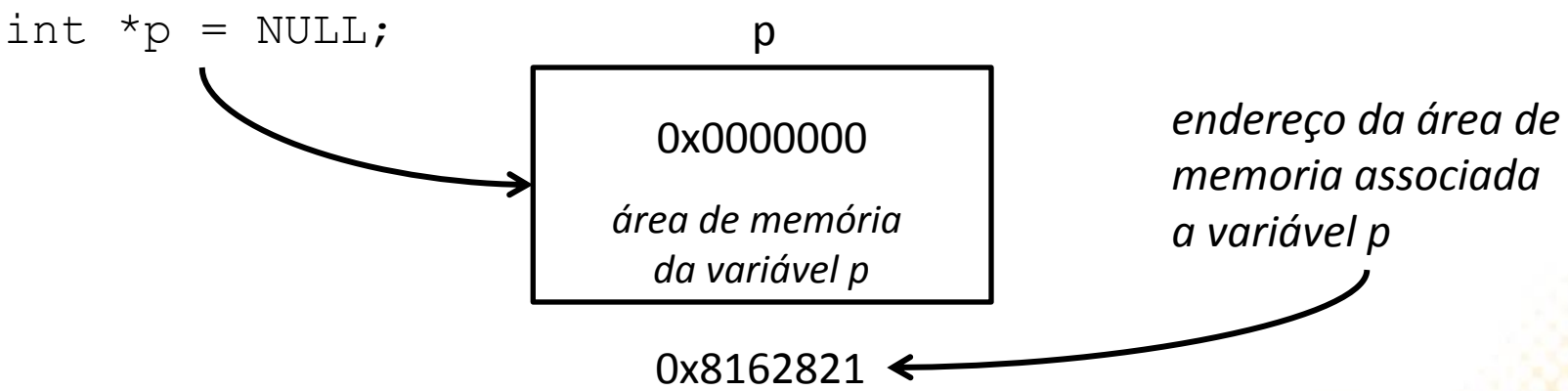
- Um ponteiro é uma **variável que armazena um endereço de memória**;
- Em linguagens tipadas, como C, um ponteiro declara o tipo da informação por ele apontada;
- Um ponteiro é declarado colocando-se um **asterisco** entre o tipo da variável e o nome da mesma;
  - Exemplo:

```
int *p;
```

declara que p é um ponteiro para uma área de memória que deve armazenar um valor inteiro

Ponteiros

```
int *p;
```



Operadores de Ponteiros

- Existem dois **operadores** relacionados a ponteiros:
  - O operador **&** retorna o endereço de memória de uma variável:

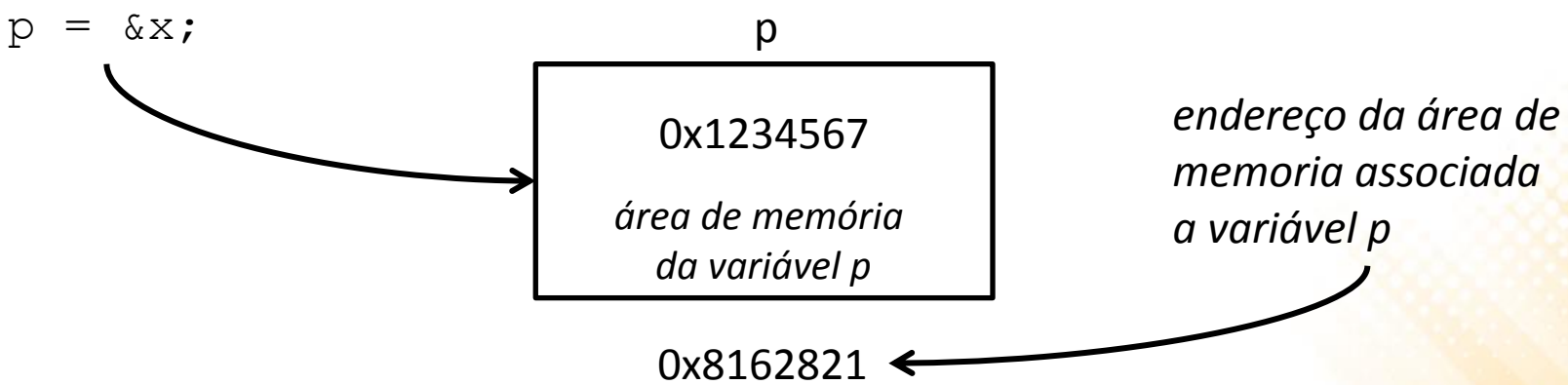
```
int *p;
int a = 40;
p = &a;
```

  - O operador ***** retorna o conteúdo do endereço indicado pelo ponteiro:

```
*p = 100;
printf("%d", *p);
```

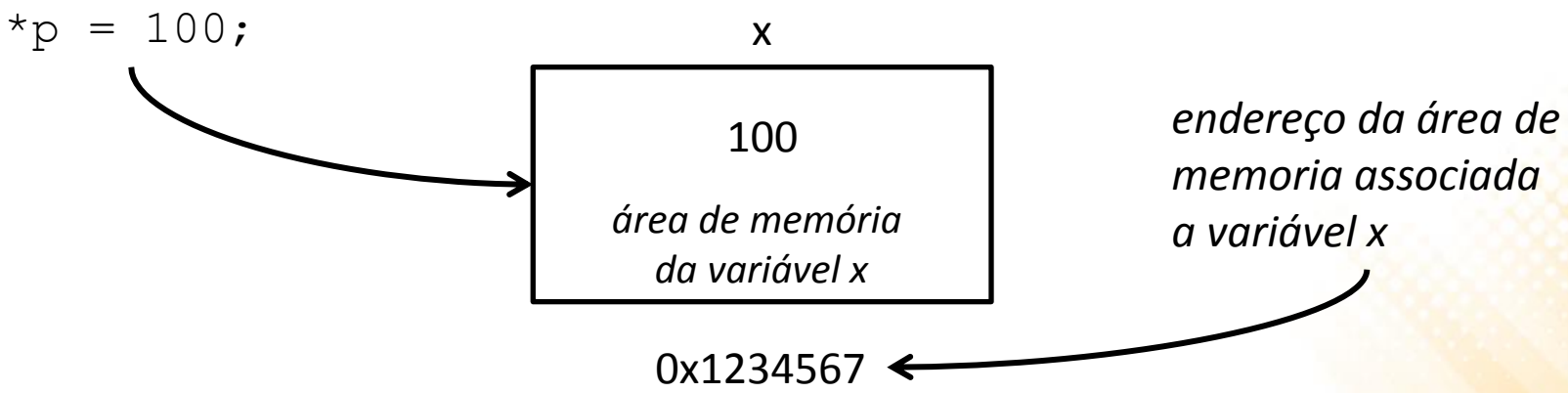
Operadores de Ponteiros

```
int main(void)
{
   int *p;
   int x = 40;
   p = &x;
   *p = 100;
   printf("%d", *p);
   return 0;
}
```



Operadores de Ponteiros

```
int main(void)
{
   int *p;
   int x = 40;
   p = &x;
   *p = 100;
   printf("%d", *p);
   return 0;
}
```



Operadores de Ponteiros

```c
#include <stdio.h>

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

SAIDA:

2

Operadores de Ponteiros

```c
#include <stdio.h>

int main(void)
{
  int a;
  int *p = &a; /*declaração e inicialização*/
  *p = 10;
  printf("%d", a);
  return 0;
}
```

SAIDA:

```
10
```

Operadores de Ponteiros

```
#include <stdio.h>

int main(void)
{
  int num, q = 1;
  int *p;
  num = 100;
  p = &num;
  q = *p;
  printf("%d", q);
  return 0;
}
```

SAIDA:

100

Cuidados com Ponteiros

- Não se pode atribuir um valor ao endereço apontado pelo ponteiro, sem antes ter certeza de que o endereço é válido:

```
int a;
int *c;

*c = 20; /* armazenará 20 em qual endereço? */
```

- O correto seria:

```
int a;
int *c;
c = &a;
*c = 13;
```

Passagem de Parâmetros

- Os parâmetros de uma função são o mecanismo utilizado para passar a informação de um trecho de código para o interior da função.

- **Ha dois tipos de passagem de parâmetros:**
  - Passagem por valor.
  - Passagem por referência.

Passagem de Parâmetros por Valor

```
#include <stdio.h>
void troca(int a, int b)
{
  int tmp = b;
  b = a;
  a = tmp;
}
int main (void)
{
  int a = 10, b = 20;
  troca(a, b);
  printf("A=%d B=%d", a, b);
}
```

SAIDA:

A = 10 B = 20

Passagem de Parâmetros por Referência

- A linguagem C permite a **passagem de ponteiros para funções**.
- Isso permite que as funções possam **alterar o conteúdo das variáveis** indiretamente pelo seu endereço de memória.
- Se uma função g chama uma função f:
  - f não pode alterar diretamente valores de variáveis de g, porém
  - se g passar para f os valores dos endereços de memória onde as variáveis de g estão armazenadas, f pode alterar, indiretamente, os valores das variáveis de g.

Passagem de Parâmetros por Referência

```
#include <stdio.h>
void troca(int *a, int *b)
{
  int tmp = *b;
  *b = *a;
  *a = tmp;
}
int main (void)
{
  int a = 10, b = 20;
  troca(&a, &b);
  printf("A=%d B=%d", a, b);
}
```

SAIDA:

A = 20 B = 10

Passagem de Parâmetros por Referência

- Vantagens da utilização de parâmetros por referência:
  - **Mais eficiência:** as funções recebem os endereços para as variáveis já inicializadas e o tamanho do endereço é sempre o mesmo, assim não há problema envolvendo cópias e inicialização.
  - **Mais liberdade:** possibilita a criação de funções que podem retornar mais do que um valor.

Vetores – Revisão Rápida

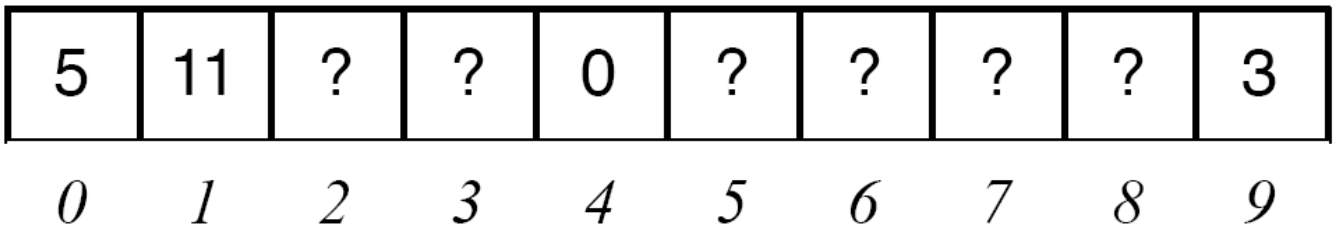
- **Declaração de um vetor:**

```
int meu_vetor[10];
```

  - Reserva um espaço de memória para armazenar 10 valores inteiros no vetor chamado meu_vetor.

- **Inicialização de algumas posições do vetor meu_vetor:**

```
meu_vetor[0] = 5;
meu_vetor[1] = 11;
meu_vetor[4] = 0;
meu_vetor[9] = 3;
```



Vetores Passados para Funções

- Quando passamos um vetor como parâmetro para uma função, a função chamada recebe uma **referência para o vetor, ou seja, um ponteiro para o primeiro elemento do vetor**.
  - Quando a função chamada acessa os elementos do vetor, ela acessa as **mesmas posições de memória** que a função que declarou vetor.
  - Se atribuirmos um valor a um elemento do vetor passado como parâmetro, este elemento **também é alterado no vetor original**.

Vetores Passados para Funções

- **Exemplo:** armazenar as notas dos alunos de uma turma em um vetor e em seguida calcular a média da turma.

- **Podemos dividir o programa em funções:**
  - Uma função para ler os valores e armazená-los em um vetor;
  - Uma função para calcular a média;

ManipulaDados.h

```
void ler_dados(float *vet, int num);

float calcula_media(float *vet, int num);
```

ManipulaDados.c

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

void ler_dados(float *vet, int num)
{
  int i;
  for(i=0;i<num;i++)
  {
    printf("Entre com o valor %d: ", i+1);
    scanf("%f", &vet[i]);
  }
}
```

Equivalente a:
void ler_dados(float vet[], int num)

ManipulaDados.c

[continuação...]

```
float calcula_media(float *vet, int num)
{
  float soma = 0.0;
  int i;
  for(i=0;i<num;i++)
    soma = soma + vet[i];
  return soma/num;
}
```

Principal.c

```
#include <stdio.h>
#include "ManipulaDados.h"
#define NUM_ALUNOS 6

int main (void)
{
  float notas[NUM_ALUNOS];
  ler_dados(notas, NUM_ALUNOS);
  printf("Media: %.2f\n.", calcula_media(notas, NUM_ALUNOS));
  return 0;
}
```

Exercício 1

- Qual o valor de x, y e p no final da execução desse trecho de código?

```
int x, y, *p;
y = 0;
p = &y;
x = *p;
x = 4;
(*p)++;
x--;
(*p) += x;
```

Resultado:

x = 3 y = 4 p = endereço de y

Exercício 2

- O método `misterio(&i, &j)` tem um problema. Qual é? Antes da chamada do método, temos a seguinte linha de comando: `int i=6, j=10;`

```
void misterio(int *p, int *q)
{
  int *temp;
  *temp = *p;
  *p = *q;
  *q = *temp;
}
```

Problema:

O endereço apontado por *temp nunca foi definido

Exercício 3

- Qual é o resultado da execução desse programa?

```
void imprime_primeiro(int *vet)
{
  printf("Valor: %d\n", vet[0]);
}

int main(void)
{
  int vet[5] = {1, 2, 3, 4, 5};
  imprime_primeiro(vet);
  return 0;
}
```

Resultado:

Valor: 1

Exercício 4

- Qual é o resultado da execução desse programa?

```
void imprime_primeiro(int *vet)
{
  printf("Valor: %d\n", vet[0]);
}

int main(void)
{
  int vet[5] = {1, 2, 3, 4, 5};
  imprime_primeiro(&vet[2]);
  return 0;
}
```

Resultado:

Valor: 3

Exercício 5

- Qual é o resultado da execução desse programa?

```
int* metade_final(int *vet, int n)
{
  return &vet[(int)(n/2)];
}

int main(void)
{
  int vet[6] = {1, 2, 3, 4, 5, 6};
  int *v = metade_final(vet, 6);
  printf("Valor: %d\n", v[0]);
  return 0;
}
```

Resultado:

Valor: 4

Exercício 6

- Qual é o resultado da execução desse programa?

```
int main(void)
{
  int vet[6] = {1, 2, 3, 4, 5, 6};
  printf("Valor1: %d\n", vet);
  printf("Valor2: %d\n", *vet);
  printf("Valor3: %d\n", *(vet + 2));
  return 0;
}
```

Resultado:

Valor1: 3997056 -> endereço de memória de vet[0]

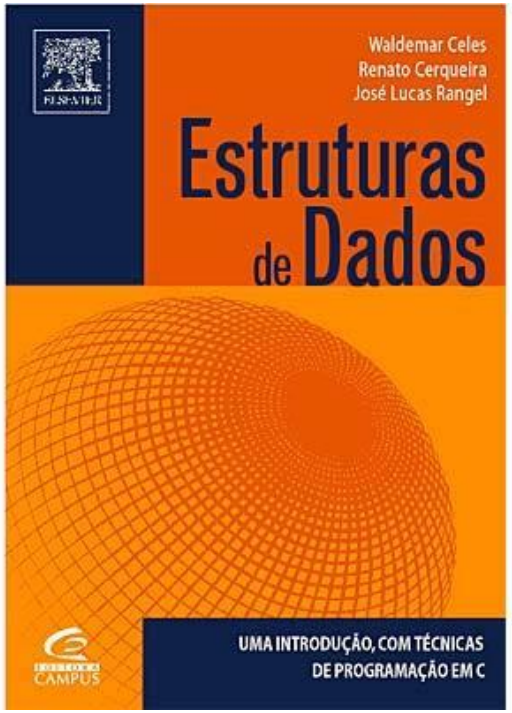
Valor2: 1

Valor3: 3

Leitura Complementar

- Waldemar Celes, Renato Cerqueira, José Lucas Rangel, **Introdução a Estruturas de Dados**, Editora Campus (2004).

- **Capítulo 4 – Ponteiros e Endereços de Variáveis**



Exercícios

Lista de Exercícios 01 – Ponteiros

http://www.inf.puc-rio.br/~elima/prog2/