
# PCS3111

# Laboratório de Programação Orientada a Objetos para Engenharia Elétrica

**Aula 2**: Ponteiros, Testes e Depuração

Escola Politécnica da Universidade de São Paulo

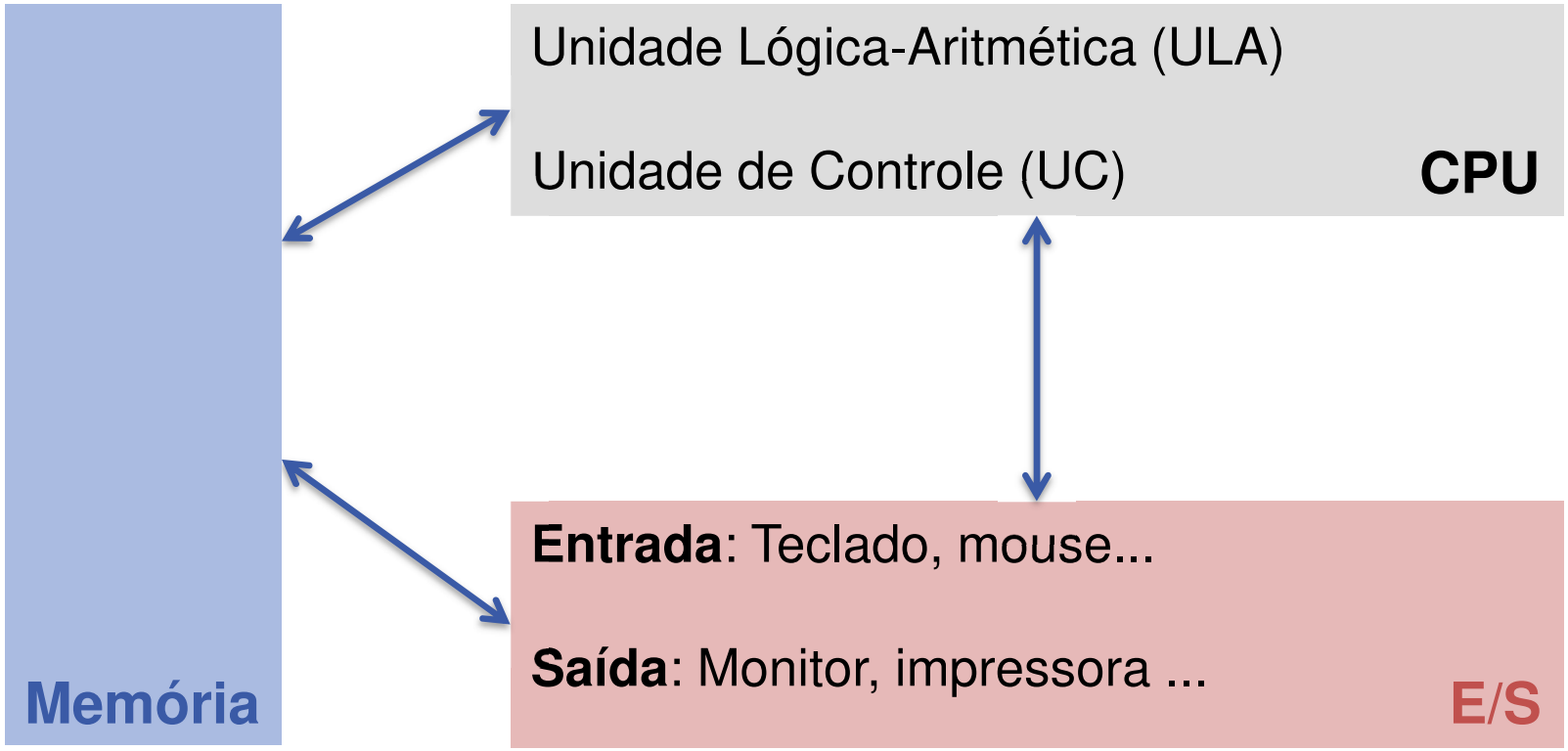
# Agenda

1. Arquitetura de von Neumann
2. Ponteiros
   - Ponteiros e vetores
   - Passagem de parâmetro em C++
3. Testes e depuração
4. **Qualidade de código (ler em casa)**

# Arquitetura de von Neumann

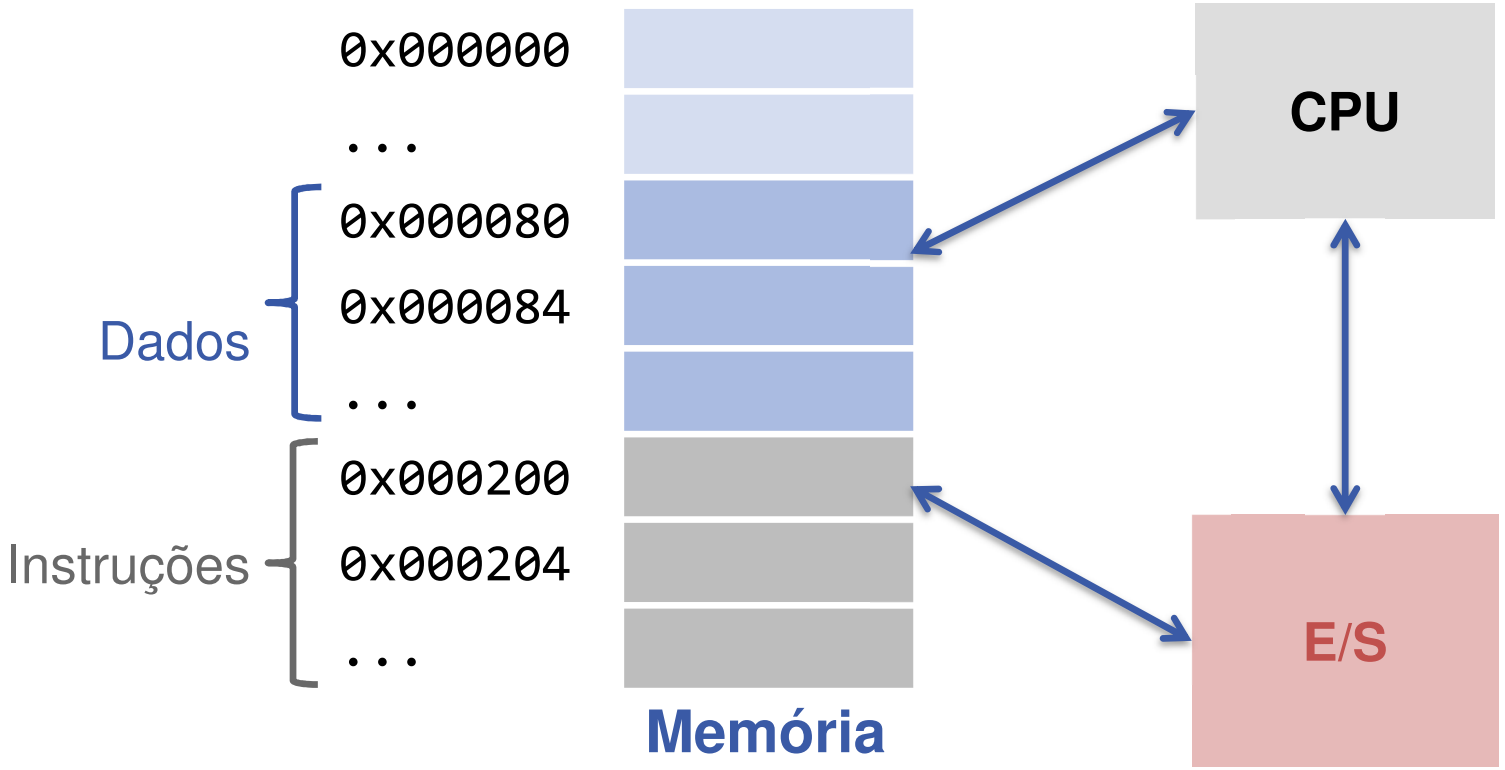
# Computador

- Arquitetura de von Neumann



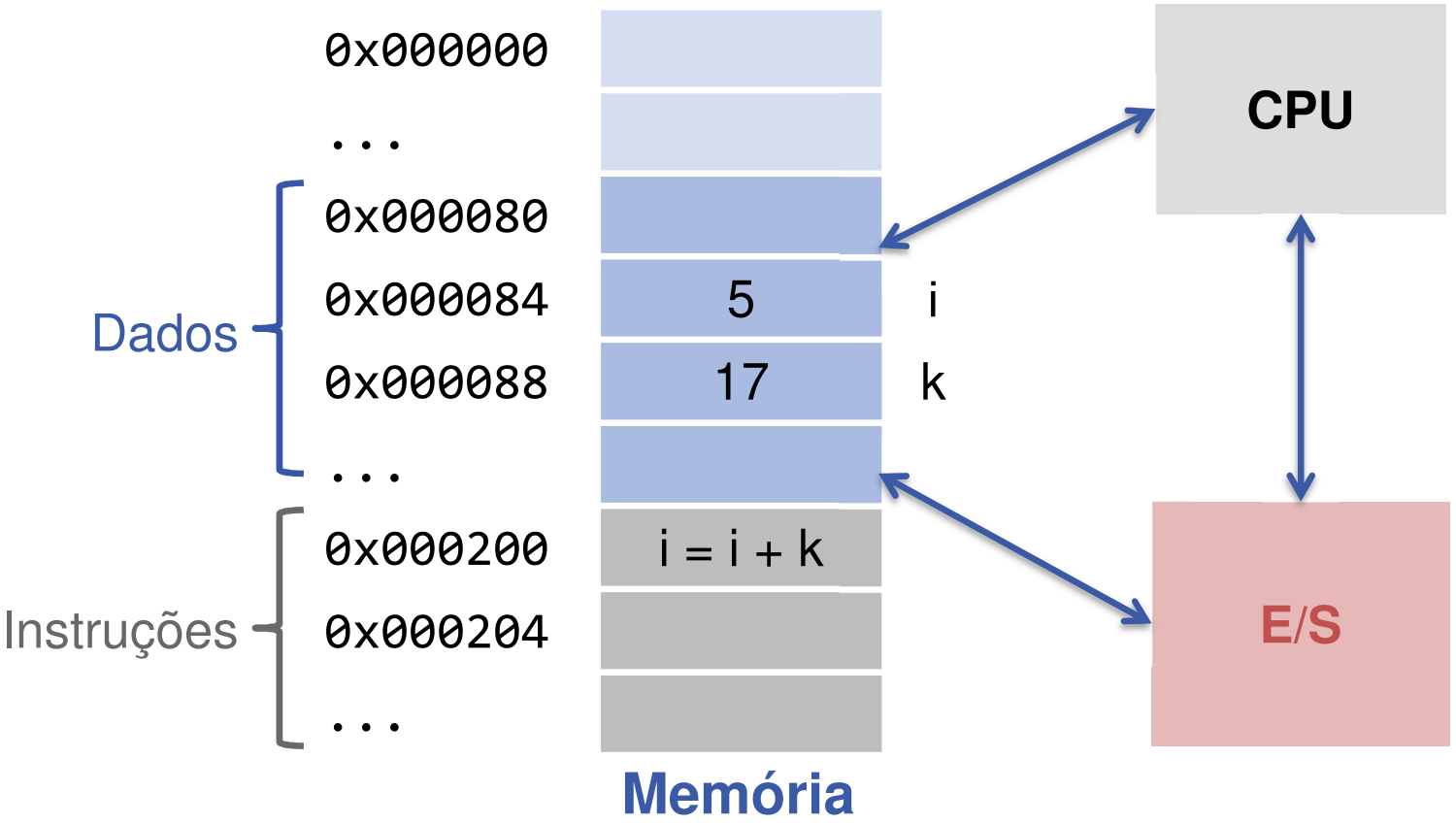
# Arquitetura de von Neumann

- A memória é uma sequência de bytes
  - 1 byte = 8 bits
  - Bytes são numerados sequencialmente
  - **Dados** e instruções (programa) ficam na memória



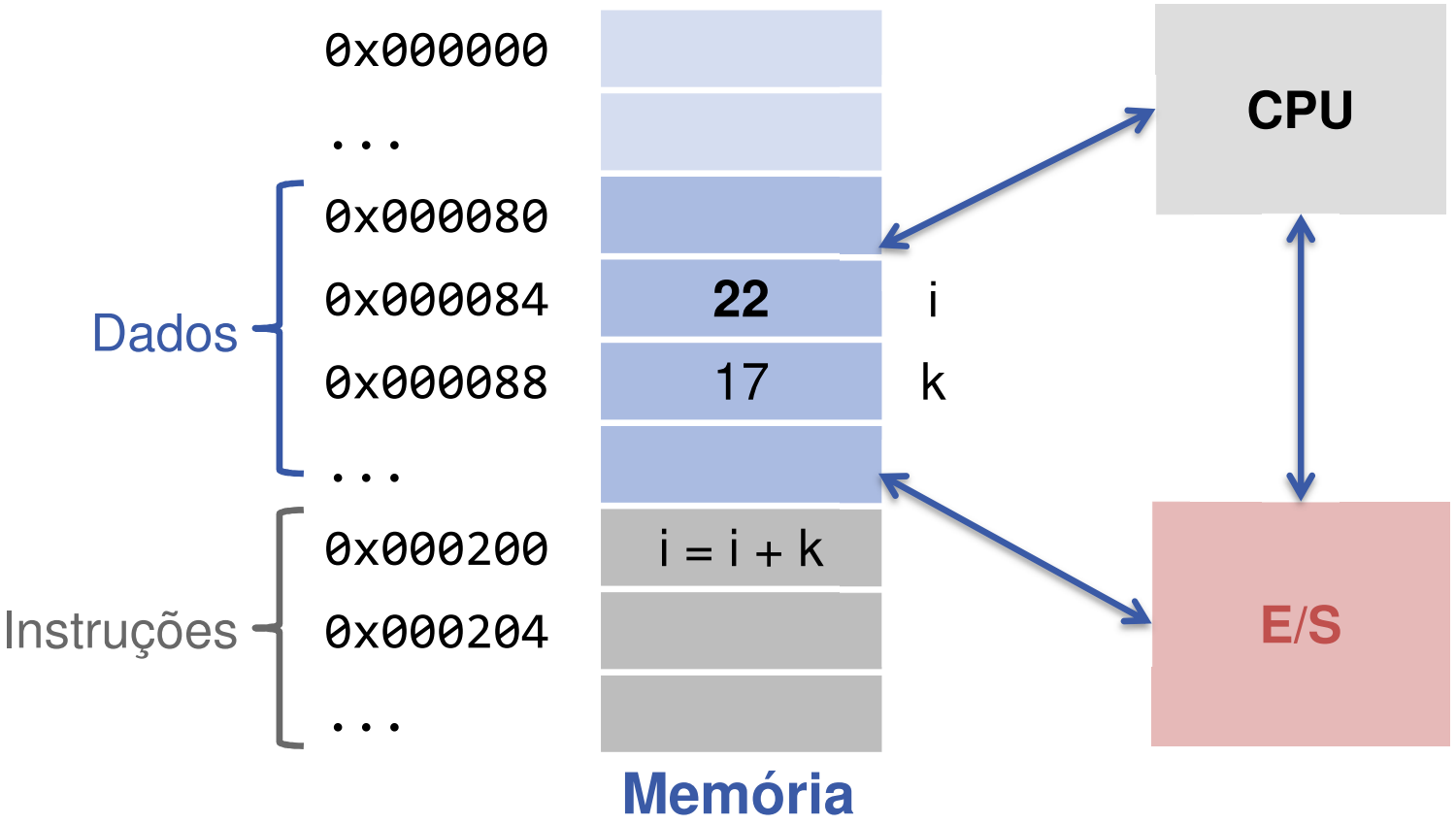
# Arquitetura de von Neumann

- Exemplo: antes da execução



# Arquitetura de von Neumann

- Exemplo: depois da execução

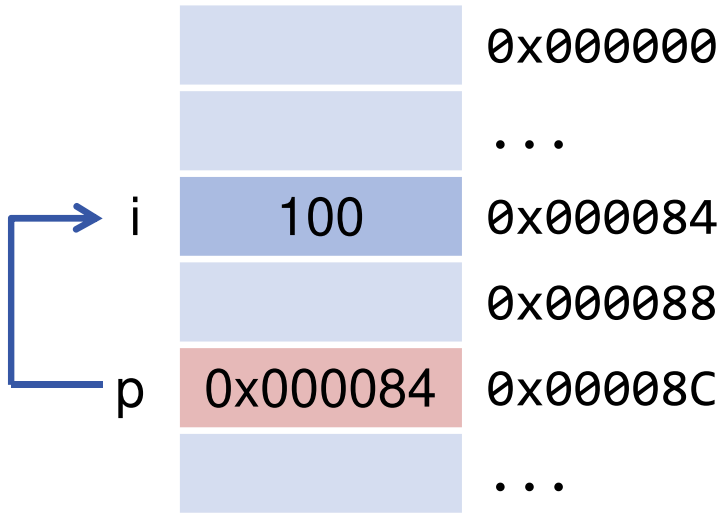


# Ponteiros

# Ponteiros

- Variável especial que referencia um endereço da memória
  - Também chamado de *apontador*

```
int i = 100;
```




  - "p" é um ponteiro que aponta para o valor de "i"

# Ponteiros

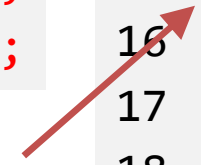
- Declaração `<Tipo> *p;`
  - O ponteiro é específico para um tipo de variável

```
12    int *p1;
13    double *p2;
```

- Operador &
  - Obtêm o endereço de uma variável

```
 6  int i = 100;
 7  double j = 5.5;
 8
 9  cout << &i << endl;
10  cout << &j << endl;
```

p1 aponta para i



```
12  int *p1;
13  double *p2;
14
15  p1 = &i;
16  p2 = &j;
17
18  cout << p1 << endl;
19  cout << p2 << endl;
```

**EX01**

**Saída**

```
0x6afef4
0x6afee8
0x6afef4
0x6afee8
```

# Ponteiros

- Operador * (*desreferenciação*)
  - Permite obter o valor apontado pelo ponteiro

EX02

```
 6    int i = 5;
 7    int *p = &i;
 8
 9    cout << i << endl;
10    cout << *p << endl;
```

**Saída**

```
5
5
```

  - É possível usá-lo para alterar o valor

```
12    *p = 10;
13    cout << *p << endl;
14    cout << i << endl;
```

**Saída**

```
10
10
```

# Usando um ponteiro

- O valor inicial de um ponteiro é indefinido

EX03

```
6    int *p1; // endereço indefinido
7
8    cout << *p1 << endl; // Ops… Problema!
```

- NULL
  - Representa que o ponteiro aponta para nenhum valor

EX03

```
10   int *p2; // endereço indefinido
11   p2 = NULL; // nenhum valor
12
13   if (p2 == NULL) { É possível testar
14     cout << "Null" << endl;
15   }
```

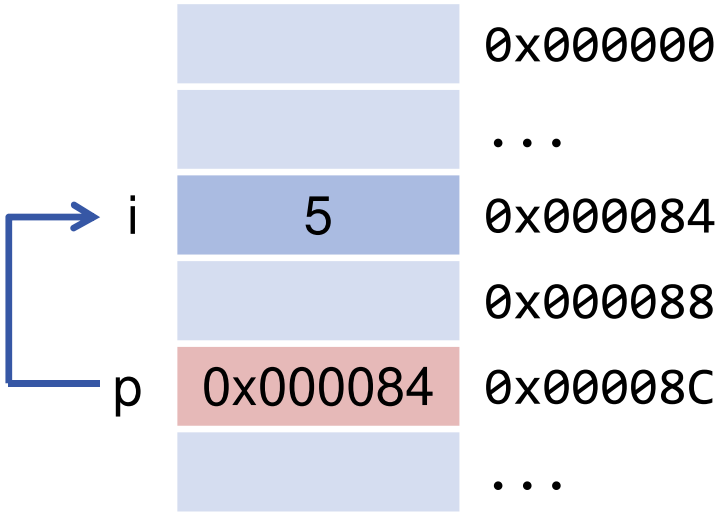
  - NULL está definido em várias bibliotecas
    - Em `iostream`, por exemplo

# Usando um ponteiro

- Na verdade, o ponteiro também está na memória

```
6     int i = 5;
7     int *p = &i;
```

EX02




- Então qual é o valor de:

`&i` `p` `*p` `&p` `*&i`

# Usando um ponteiro

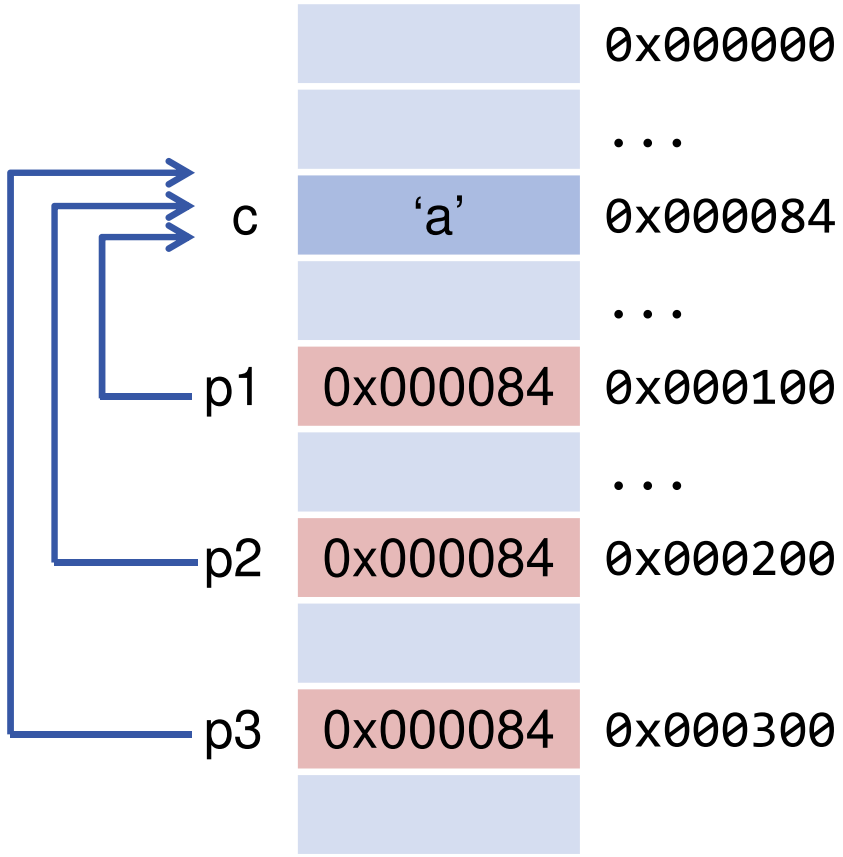
- Vários ponteiros podem apontar para o mesmo valor

```
 6    char c = 'a';
 7
 8    char *p1 = &c;
 9    char *p2 = &c;
10    char *p3 = p1;
11
12    cout << *p3 << endl;
13
14    *p3 = 'b';
15
16    cout << c << endl;
```

EX04

**Saída**

```
a
b
```

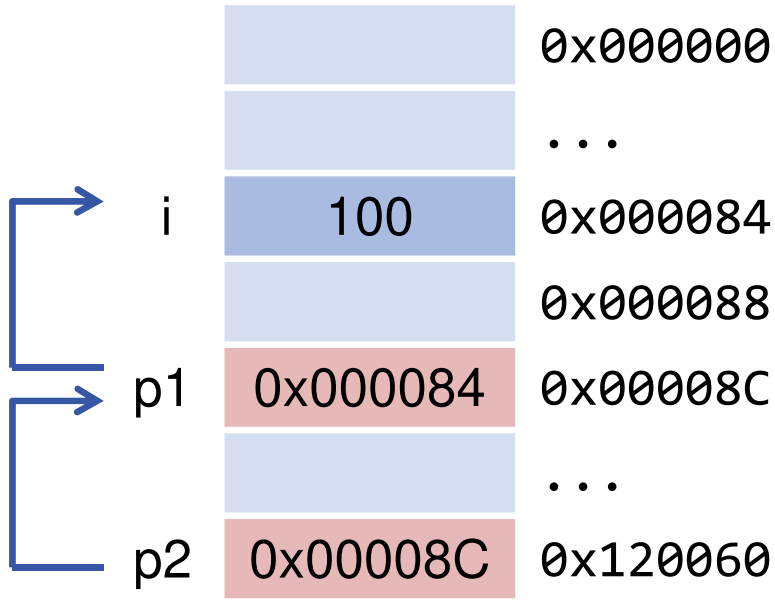


# Usando um ponteiro

- É possível ter ponteiros de ponteiros

EX05

```
 6  int i = 100;
 7  int *p1 = &i;
 8
 9  int **p2;
10  p2 = &p1;
11
12  cout << i << endl;
13  cout << *p1 << endl;
14  cout << **p2 << endl;
```




**Saída**

```
100
100
100
```

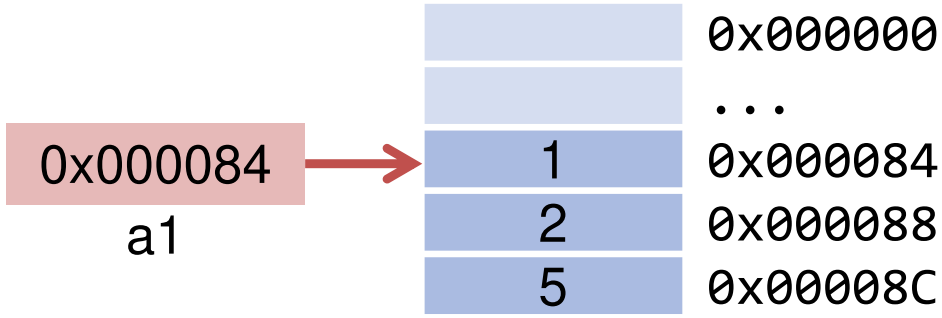
p2 é um ponteiro para ponteiro de inteiros

# Ponteiros e vetores

- Um vetor *funciona* como um ponteiro
  - A variável aponta para a primeira posição do vetor

```
6    int a1[] = {1, 2, 5};
```

**EX06**




  - Um ponteiro para o vetor pode ser usado como um vetor

```
 8    int *p = a1;
 9
10    cout << p[0] << endl;
11    cout << p[2] << endl;
```

**Saída**

```
1
5
```

**Observação**: em C++ não é possível retornar vetores. Mas é possível retornar um *ponteiro*.

# Passagem de parâmetro

- Como funciona a passagem de parâmetros?
  - Qual é a saída em tela?

**EX07**

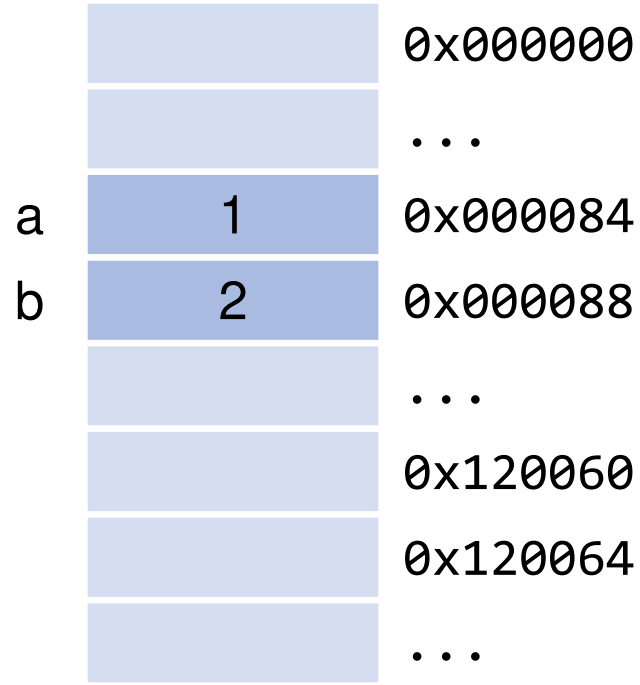
```
 5  void trocar(int a, int b) {
 6    int temp = a;
 7    a = b;
 8    b = temp;
 9  }
...
24  int main() {
25    int a = 1;
26    int b = 2;
27
28    trocar(a, b);
29    cout << "a: " << a << " b: " << b << endl;
...
```

# Passagem de parâmetro

- O C++ passa argumentos por valor
  - Ou seja, é feita uma cópia do valor da variável

```
 5  void trocar(int a, int b) {
 6    int temp = a;
 7    a = b;
 8    b = temp;
 9  }
...
24  int main() {
25    int a = 1;
26    int b = 2;
27
28    trocar(a, b);
...
```

**EX07**

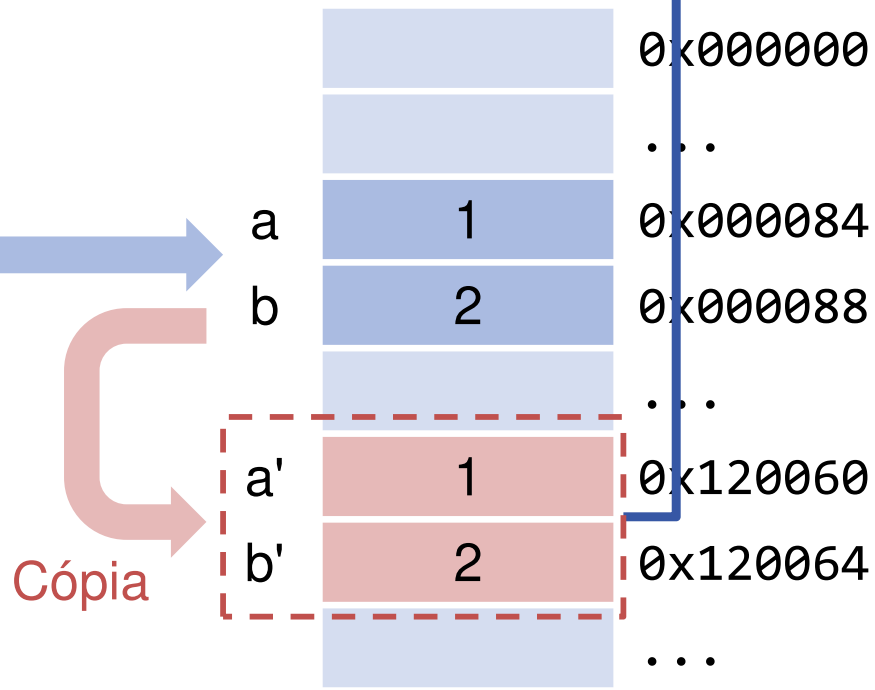


# Passagem de parâmetro

- O C++ passa argumentos por valor
  - Ou seja, é feita uma cópia do valor da variável

```
 5  void trocar(int a, int b) {
 6    int temp = a;
 7    a = b;
 8    b = temp;
 9  }
...
24  int main() {
25    int a = 1;
26    int b = 2;
27
28    trocar(a, b);
...
```

**EX07**

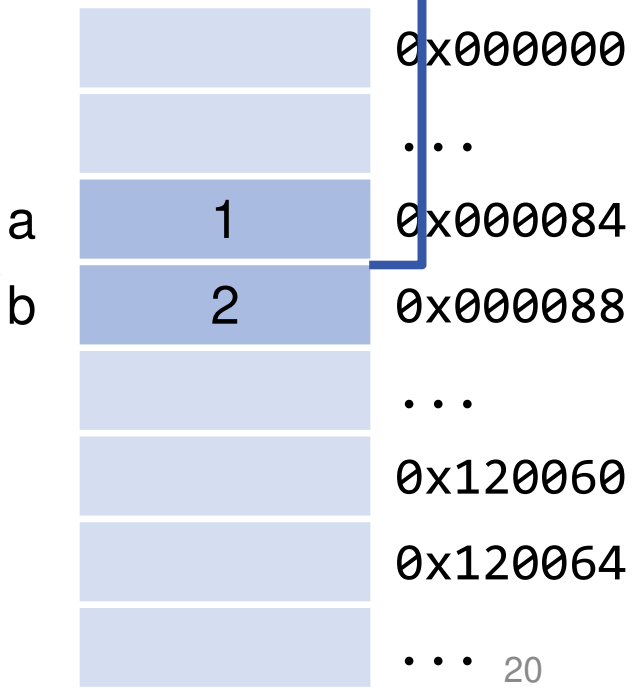


# Passagem por referência

- C++ permite parâmetros por referência
  - Nesse caso a referência ao valor é passada como parâmetro
  - Usar o símbolo & na declaração do parâmetro

```
11  void trocar1(int& a, int& b) {
12    int temp = a;
13    a = b;
14    b = temp;
15  }
...
24  int main() {
25    int a = 1;
26    int b = 2;
...
33    trocar1(a, b);
...
```

Por referência




**EX07**

# Passagem usando ponteiros

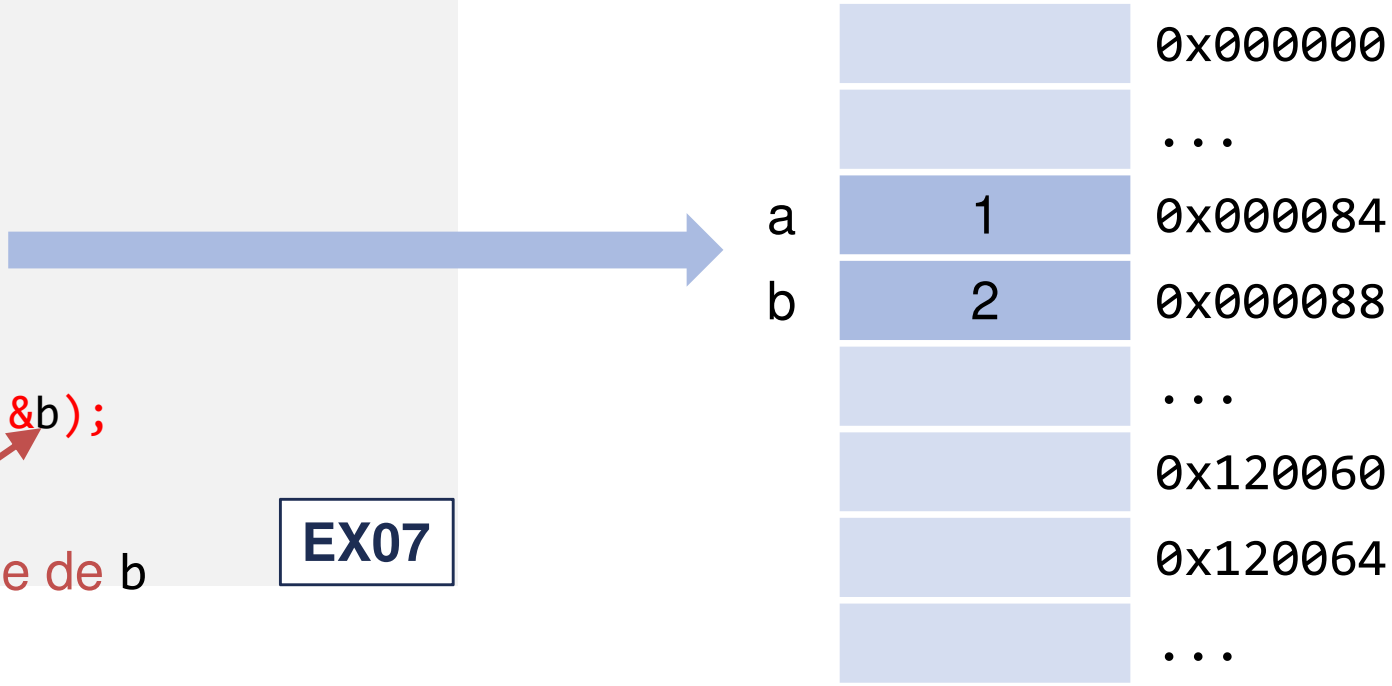
- Uma outra forma é usar ponteiros

```
17  void trocar2(int* a, int* b) {
18    int temp = *a;
19    *a = *b;
20    *b = temp;
21  }
...
24  int main() {
25    int a = 1;
26    int b = 2;
...
38    trocar2(&a, &b);
...
```

Ponteiro

Endereço de a e de b

EX07



# Passagem usando ponteiros

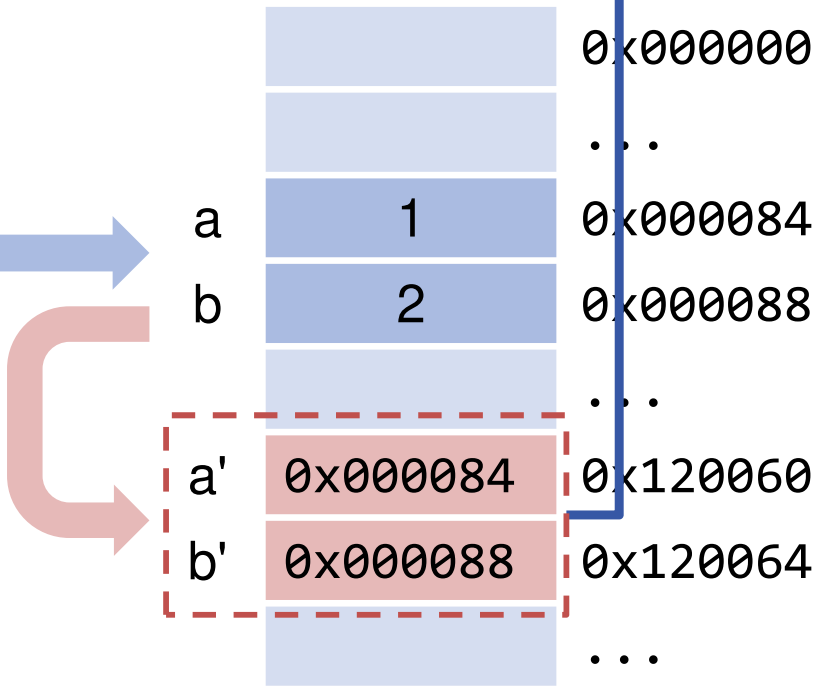
- Uma outra forma é usar ponteiros

```
17  void trocar2(int* a, int* b) {
18    int temp = *a;
19    *a = *b;
20    *b = temp;
21  }
...
24  int main() {
25    int a = 1;
26    int b = 2;
...
38    trocar2(&a, &b);
...
```

Ponteiro

Endereço de a e de b

EX07



# Testes e Depuração

# Testes e depuração

- Definições
  - **Teste**: processo de executar um programa com o objetivo de encontrar erros
  - **Depuração**: processo de localizar um *suposto* erro e corrigi-lo

- Saídas corretas em um teste não *garantem* que o software não tem erros
  - O teste pode não ter sido bom o suficiente!

- Existem abordagens de teste e de depuração
  - Engenharia de Software

# Exemplo: total de uma compra

- Se a soma dos preços dos produtos for maior que "limite" reais, o frete não deve ser cobrado

```
11  double totalDaCompra (double produtos[], int quantidade,
12                        double frete, double limite) {
13    double total = 0;
14    for (int i = 0; i < quantidade; i++)
15      total = total + produtos[i];
16    if (total > limite)
17      total = total + frete;
18    return total;
19  }
```

**EX08**

  - Como fazer testes?

# Exemplo: total de uma compra

- Se a soma dos preços dos produtos for maior que "limite" reais, o frete não deve ser cobrado

```
11  double totalDaCompra (double produtos[], int quantidade,
12                        double frete, double limite) {
13    double total = 0;
14    for (int i = 0; i < quantidade; i++)
15      total = total + produtos[i];
16    if (total > limite)
17      total = total + frete;
18    return total;
19  }
```

EX08

```
21  int main() {
22    double produtos[] = {5, 10, 5, 10};
23    double total = totalDaCompra (produtos, 4, 5.5, 25);
24
25    if (total != 30) cout << "Erro: " << total << endl;
```

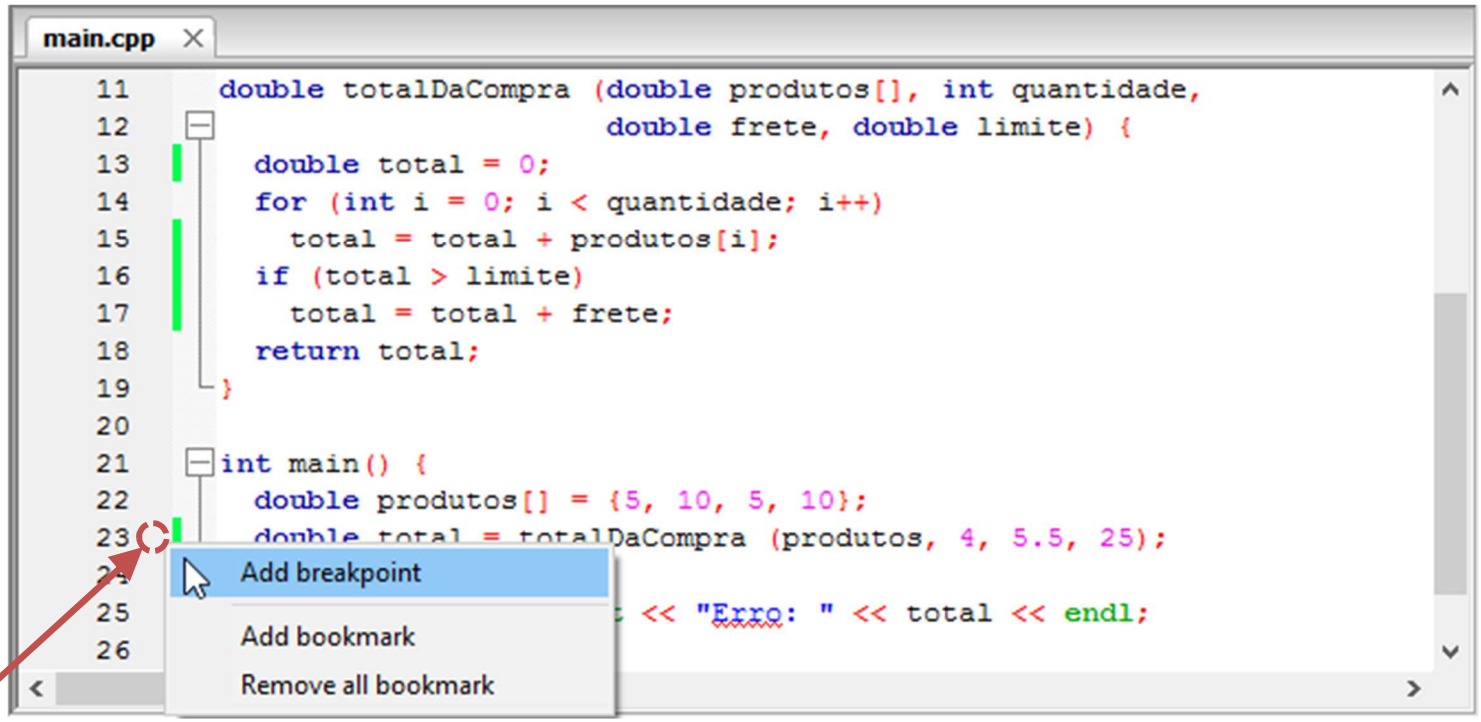
Teste

**Saída**

```
Erro: 35.5
```

# Depuração

- *Breakpoint*
  - Ponto de parada no programa para depuração




Inserir *breakpoint* na linha 23 usando botão direito **ou** 1 clique na linha 23

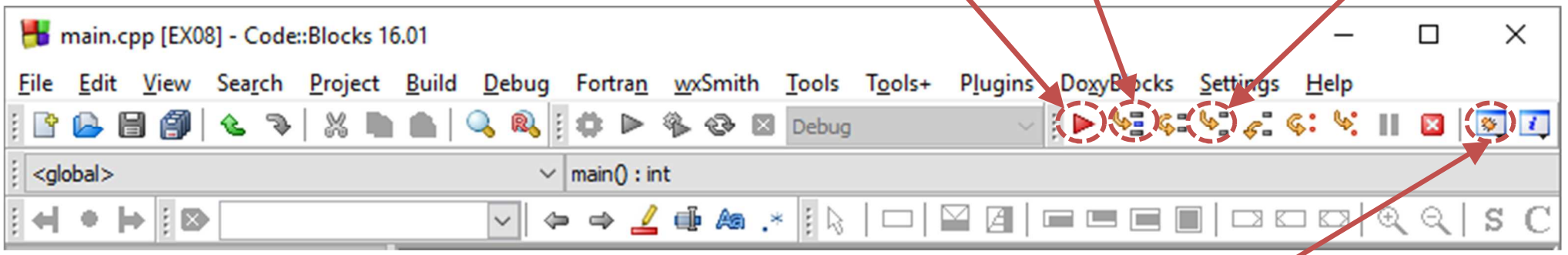
# Depuração

- Controle da depuração

Executar até o próximo *breakpoint*

Ir para a próxima linha

Entrar na função existente na linha atual




Janelas de depuração (a mais importante é a **watches**, que mostra os valores das variáveis)

Watches (new)

| | |
|---|---|
| Function arguments | |
| ⊟ Locals | |
| ⊟ produtos | |
| [0] | 5 |
| [1] | 10 |
| [2] | 5 |
| [3] | 10 |
| total | 1.9763250266786954e-307 |

# Observação

- Nunca compare `double` com == ou !=
  - Pontos flutuantes tem *problemas de precisão*

```
 1  #include <iostream>
 2  #include <cmath>
 3
 4  using namespace std;
..
21
22  int main() {
23    double produtos[] = {5, 10, 5, 10};
24    double total = totalDaCompra (produtos, 4, 5.5, 25);
25
26    if (abs(total - 30) > 0.01) cout << "Erro: " << total << endl;
27    return 0;
28  }
```

Inclua `cmath` para usar o `abs` e faça `using namespace std`

**EX08b**

Se a diferença entre o valor esperado e o obtido for maior que um *épsilon*, há um erro

# Bibliografia


- MYERS, G. J. **The Art of Software Testing**. John Wiley & Sons, 2ª edição, 2004.

- SAVITCH, W. **C++ Absoluto**. Pearson, 1st ed. 2003. Seção 10.1.