

Estruturas de Dados 1

Funções

Mailson de Queiroz Proença

Funções

O que é:

- São rotinas que tem como objetivo, executar trechos de códigos de forma modular, melhorando a organização do programa e evitando repetição de código. As funções são reutilizáveis dentro de um programa.



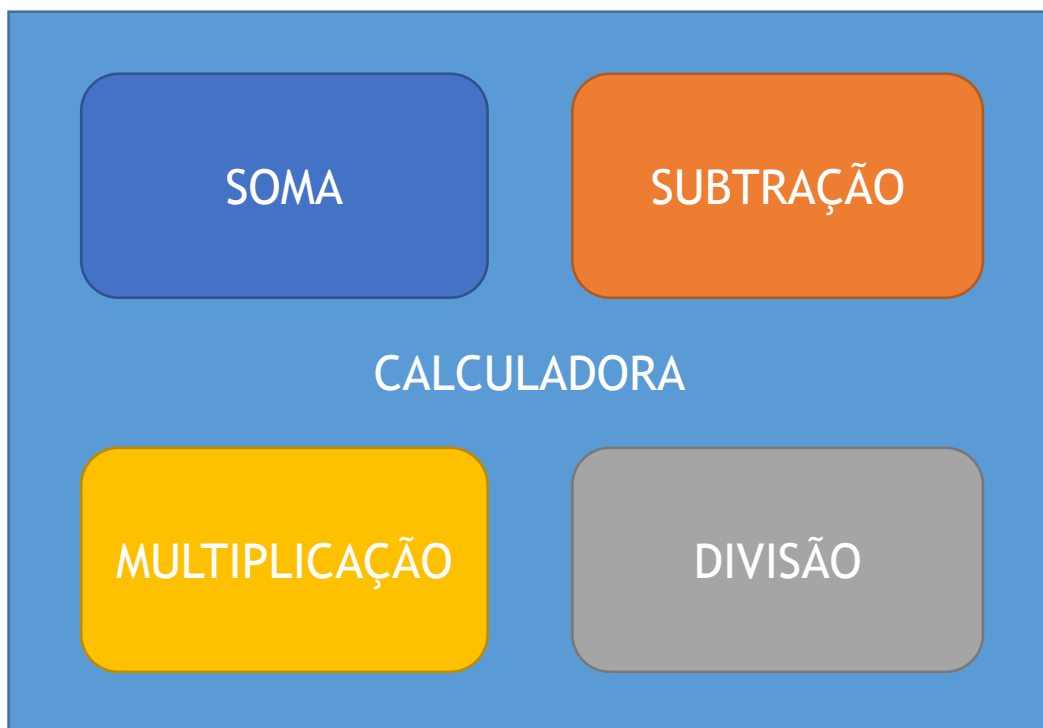
O que é?

- Uma função é um conjunto de instruções desenhadas para cumprir determinada tarefa e agrupadas em uma unidade com um nome para referi-la.
- Qualquer sequência de instruções que apareça mais de uma vez no programa é candidata a ser uma função;
- O código de uma função é agregado ao programa uma única vez e pode ser executado muitas vezes no decorrer do programa.

Funções

Para pensar:

Em uma calculadora quais são as suas 4 principais operações?



Conseguimos definir as 4 principais operações da calculadora.

No desenvolvimento do código da calculadora vamos precisar criar as 4 funções de acordo com as operações definidas.

Funções

Como criar funções:

- Análise quais tarefas o programa vai executar;
- Identifique ações;
- Ações são **verbos** como inserir, alterar, excluir e etc;
- A partir das ações temos as funções.

Mais um exemplo:

Quais as funções da lista de contatos de um celular?

- **Inserir** um Contato;
- **Alterar** um Contato;
- **Excluir** um Contato;
- **Discar** para um Contato.



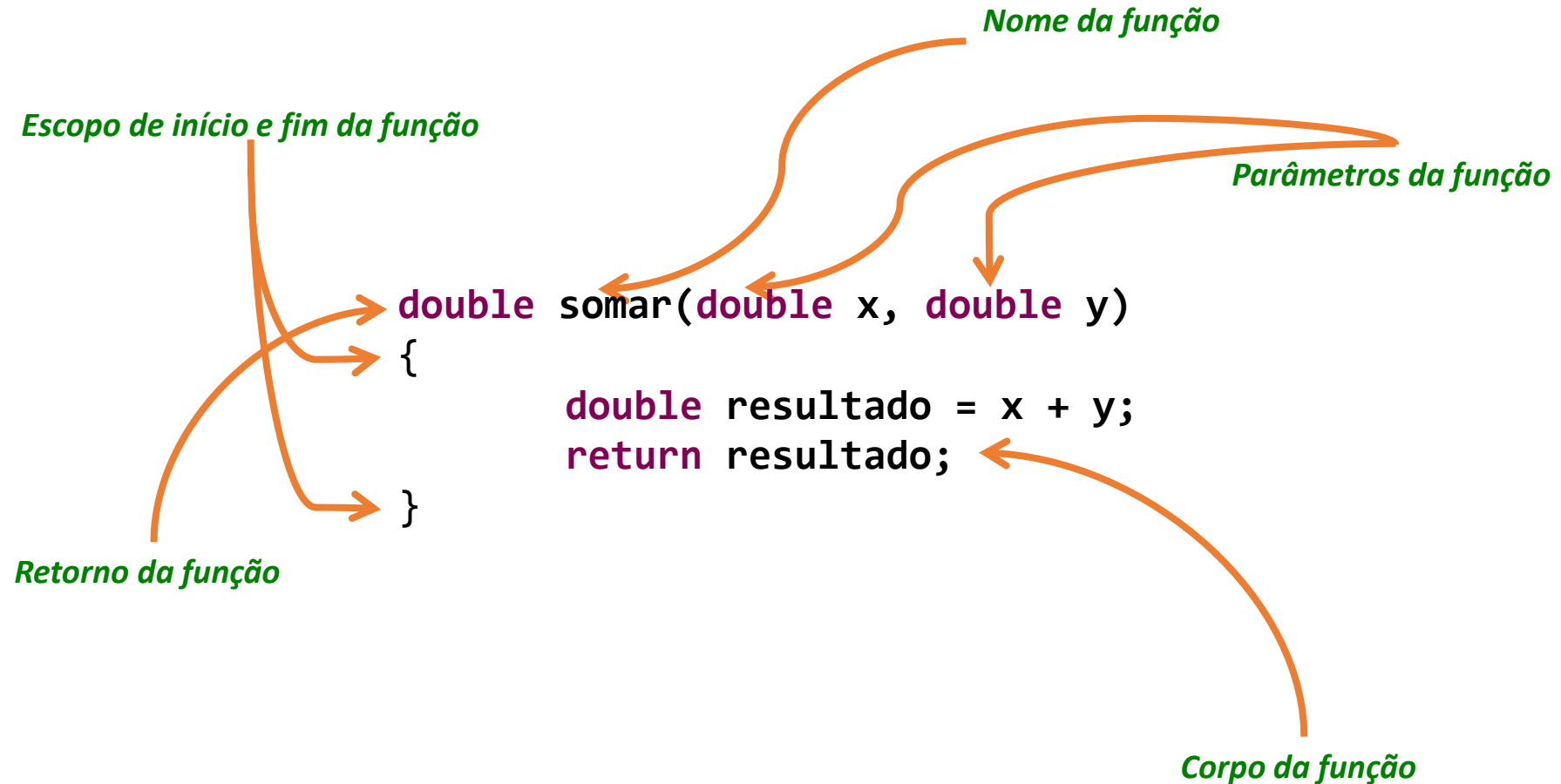
Definição da função

- O código C++ que descreve o que a função faz é chamado de **definição da função**. Sua forma geral é a seguinte:

```
tipo nome(parametros){  
    instruções;  
    return;  
}
```

Funções

Estrutura de uma função:



Funções - Estrutura

- Tipos de retorno da função
 - double, float, int, char, void, vetores e outros tipos;
- Parâmetros da função
 - Cada parâmetro é composto pelo tipo, nome e separados por virgulas;
- Retorno da função
 - Quando uma função deve retornar um valor, devemos usar a palavra reservada return seguido de um valor, variável ou operação do mesmo tipo de retorno;
- Corpo da função
 - Código fonte com a funcionalidade que a função deve executar;
- Protótipo
 - As funções possuem protótipos para definir sua estrutura

A função main()

```
#include <iostream>
```

```
using namespace std;
```

```
int main() {  
    cout << "Olá mundo";  
    return 0;  
}
```

Os parênteses após o nome das funções

- Permite que o compilador saiba que se trata de uma função;
- Podem estar vazios ou não. Trataremos deste assunto nos próximos slides.

Chaves

- Toda função C++ deve começar com uma chave de abertura de bloco { e terminar com uma chave de fechamento de bloco }. As chaves delimitam o corpo da função.

Por que usar funções?

- Permite que outros programadores utilizem em seus programas;
- É como contratar uma mão-de-obra externa para executar uma certa tarefa;
- Reduz o tamanho do programa;

O nome das funções

- Para dar um nome a uma função você deve seguir as mesmas orientações determinadas para os nomes de variáveis;
- O nome de uma função pode ser qualquer um, com exceção de **main**, reservado para a função que inicia a execução do programa;
- Em todo programa deve existir uma e uma única função chamada de **main**;
- O programa termina quando for encerrada a execução da função **main**.

Funções

Como Utilizar em C++:

```
#include <iostream>
```

```
using namespace std;
```

```
double somar(double x, double y);  
double subtrair(double x, double y);  
double multiplicar(double x, double y);  
double dividir(double x, double y);
```

```
int main()  
{
```

```
    double x, y;
```

```
    cout << "Digite os valores para somar:\n";  
    cin >> x >> y;  
    cout << somar(x, y);
```

```
    cout << "Digite os valores para subtrair:\n";  
    cin >> x >> y;  
    cout << subtrair(x, y);
```

```
    cout << "Digite os valores para multiplicar:\n";  
    cin >> x >> y;  
    cout << multiplicar(x, y);
```

```
    cout << "Digite os valores para dividir:\n";  
    cin >> x >> y;  
    cout << dividir(x, y);  
    return 0;
```

```
}
```

Protótipo das funções

As quatro funções da calculadora

```
double somar(double x, double y)  
{  
    double resultado = x + y;  
    return resultado;  
}
```

```
double subtrair(double x, double y)  
{  
    double resultado = x - y;  
    return resultado;  
}
```

```
double multiplicar(double x, double y)  
{  
    double resultado = x * y;  
    return resultado;  
}
```

```
double dividir(double x, double y)  
{  
    double resultado = x / y;  
    return resultado;  
}
```

Chamando uma função

Funções

Chamando uma função:

```
#include <iostream>
```

```
using namespace std;
```

```
double somar(double x, double y);
```

*Protótipo da
função*



```
int main()
```


```
{
```

```
    double valor;
```

```
    valor = somar(10,15);
```

```
}
```

*Retornando o valor para
a variável “valor”*



...

*Chamando a função “somar”
passando os parâmetros 10 e 15*



Funções

- Chamando uma função:
 - Para executar uma função, devemos colocar o nome da função seguido dos parâmetros;
 - Nos parâmetros podemos passar um valor específico ou uma variável;
 - Devemos respeitar o tipo passado nos parâmetros das funções. Ex: se o parâmetro for int, devemos passar um tipo int. Se for passado o tipo incorreto no parâmetro, o programa não compila;
 - Caso a função retorne algum valor, podemos atribuir diretamente a uma variável;
 - Quando uma função não tem parâmetros, abrimos e fechamos parênteses;
 - void faz a função retornar nenhum valor.

Funções

Retorno da função:

Quando o tipo de retorno da função é diferente do tipo **void**, a função devolve algum valor para alguma variável. Para efetuar este tipo de operação utilizamos a palavra reservada **return** seguido de um valor ou uma variável.

```
#include <iostream>

using namespace std;

int par_ou_impar(int numero);

int main()
{
    int numero, resultado;

    cout << "Digite um número:\n";
    cin >> numero;
    resultado = par_ou_impar(numero);

    if (resultado == 0)
        cout << "Par";
    else
        cout << "Impar";
    return 0;
}
```

```
int par_ou_impar(int numero)
{
    int valor = numero % 2;
    return valor;
}
```

Retorno da função

*Retorno da função
na variável
"resultado"*

Chamando uma função

- Várias funções são desenvolvidas por programadores e fornecidas gratuitamente pela linguagem C++.
- Ex.:
- `Getche()`, `system()`, `printf()`, `setw()`,

Primeira função

- `celsius()`
- Vamos começar mostrando uma função que converte a temperatura de graus Fahrenheit para graus Celsius:

Função

```
#include <iostream>
```

```
using namespace std;
```

```
int celsius(int fahr){  
    int c;  
    c = (fahr - 32) * 5 / 9;  
    return c;  
}
```

```
int main()  
{  
    int c, f;  
    cout << "Digite a temperatura em graus Fahrenheit: ";  
    cin >> f;  
    c = celsius(f);  
    cout << "Celsius: " << c << endl;  
    return 0;  
}
```

Função que
converte a
temperatura de
graus Fahrenheit
para graus
Celsius.

O protótipo de funções

- O propósito principal de escrita de protótipos de funções é o de fornecer ao compilador as informações necessárias sobre o tipo da função, o número e o tipo dos argumentos.

Cuidado para não esquecer o ponto e vírgula!!



```
tipo_retorno nome_Função(lista_de_parametros);
```

O protótipo de funções

- Deve preceder sua definição e sua chamada;

`tipo_retorno nome_Função(lista_de_parametros);`

- É a declaração de uma função;
- É colocado, em geral, no início do programa;
- Estabelece o tipo da função e os argumentos que ela recebe.

Protótipo externo e local

- Externo:
 - Escrito antes de qualquer função. É feita uma única vez e é visível para todas as funções que a chamam.
- Local:
 - É escrito no corpo da função que a chama e antes da sua chamada.

Protótipo externo e local

```
#include <iostream>
using namespace std;

int main()
{
    int celsius(int fahr);
    int c, f;
    cout << "Digite a temperatura em graus Fahrenheit: ";
    cin >> f;
    c = celsius(f);
    cout << "Celsius: " << c << endl;
    return 0;
}
```

```
int celsius(int fahr)
{
    int c;
    c = (fahr - 32) * 5 / 9;
    return c;
}
```


Eliminando o protótipo de funções

- Se a função for escrita antes da instrução de sua chamada, seu protótipo não será obrigatório.
- O exemplo anterior poderia ser escrito da seguinte maneira:

Função sem protótipo

```
#include <iostream>
```

```
using namespace std;
```

```
int celsius(int fahr){  
    int c;  
    c = (fahr - 32) * 5 / 9;  
    return c;  
}
```

```
int main()  
{  
    int c, f;  
    cout << "Digite a temperatura em graus Fahrenheit: ";  
    cin >> f;  
    c = celsius(f);  
    cout << "Celsius. " << c << endl;  
    return 0;  
}
```

Tipos de funções

- É definido pelo valor que ela retorna por meio do comando **return**.
- Uma função é do tipo **int** quando retorna um valor do tipo **int**.
- Em C++, os tipos de funções são os mesmos de variáveis, exceto quando a função não retorna nada. Nesse caso, ela é do tipo **void**.

O comando return

- Em funções do tipo **void**, o comando **return** não é obrigatório;
- O valor de retorno de uma função é acessado na instrução de chamada por meio do nome da função seguido de parênteses contendo ou não argumentos. Ex.:

```
c = celsius(f); //Chamada à função
```

- Esse valor poder ser atribuído a uma variável ou fazer parte de alguma expressão;
- Várias instruções **return** podem fazer parte de uma função, mas apenas uma será executada.

```

#include <iostream>

using namespace std;

int operacao(char op)
{
    switch(op)
    {
        case '+':
            return (10 + 5);
            break;
        case '-':
            return 10 - 5;
            break;
        case '*':
            return 10 * 5;
            break;
        case '/':
            return 10 / 5;
            break;
    }
    return 0;
}

int main()
{
    char o;
    int resultado;
    cout << "Digite o tipo do operador(+ ou - ou * ou /): ";
    cin >> o;
    resultado = operacao(o);
    cout << "Resultado: " << resultado << endl;
    return 0;
}

```

Limitações do comando return

- O comando `return` pode retornar somente **um único valor**.

Funções que não retornam nada: Tipo void

- Uma função que não retorna nada é do tipo **void**.
Como exemplo, escreveremos uma função que desenha uma linha com um certo número de asteriscos:

Funções

Tipo void:

A palavra reservada **void** não efetua nenhum retorno para a função.

```
#include <iostream>

using namespace std;

void imprime_idade(int idade);

int main()
{
    int idade;
    cout << "Digite a sua idade:\n";
    cin >> idade;
    imprime_idade(idade);
}

void imprime_idade(int idade)
{
    cout << "Sua idade é: " << idade;
}
```

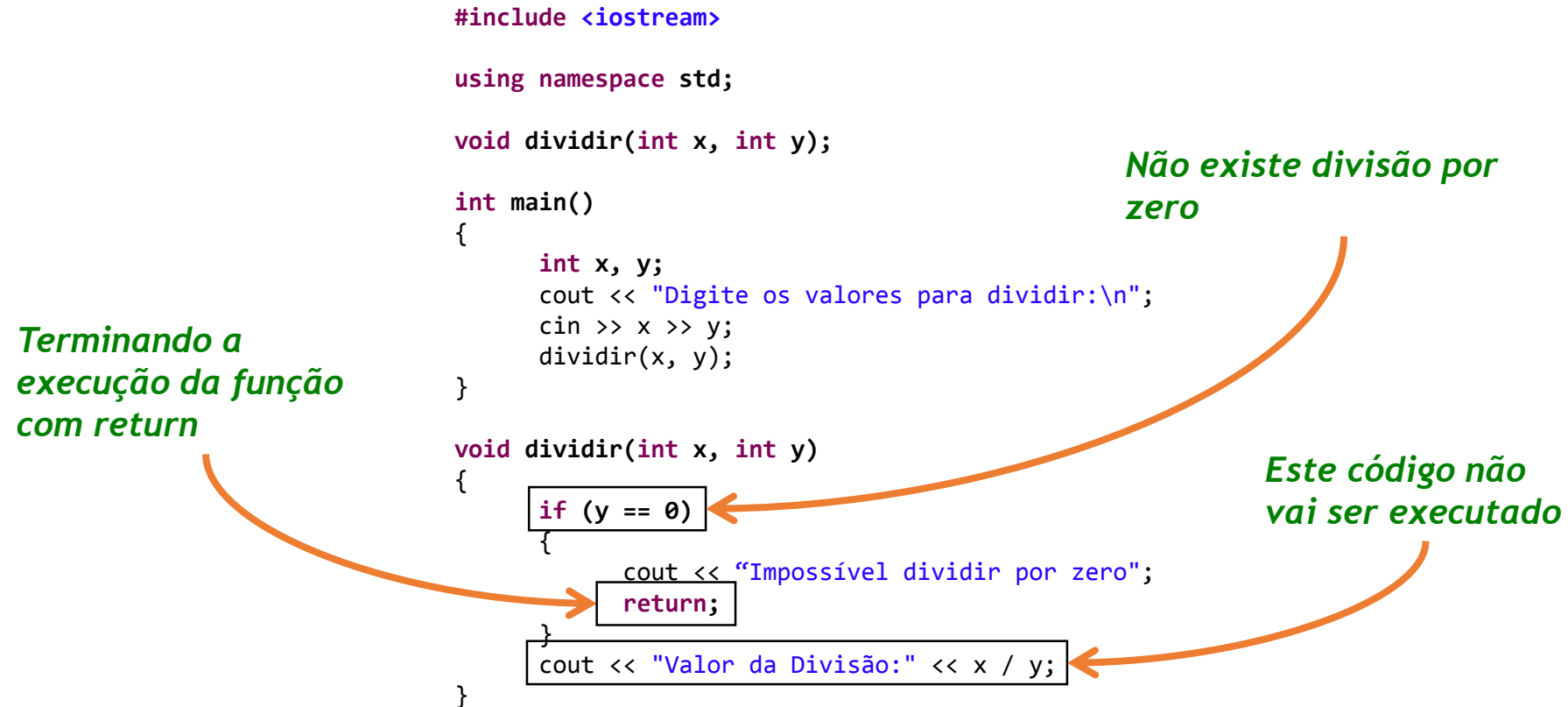
Retorno da função do tipo void

Função não retorna um valor

Funções

Usando return em uma função void:

O return em uma função void, interrompe a execução do código.



Passando vários argumentos

- Vários argumentos podem ser passados entre parênteses separados por vírgula, na chamada da função;
- A seguir, temos um exemplo de programa que passa dois argumentos para uma função:

Passando vários argumentos

Argumento 1

Argumento 2

```
#include <iostream>
using namespace std;
```

```
int areaDoRetangulo(int altura, int largura); //protótipo
```



```
int main() {

    int largura, altura, area;
    cout << "Digite a altura em centímetros: ";
    cin >> altura;
    cout << "Digita a largura em centímetros: ";
    cin >> largura;
    area = areaDoRetangulo(altura, largura);
    cout << "A area do retangulo e: "<< area << " centímetros";
    return 0;
}

int areaDoRetangulo(int altura, int largura){
    return largura * altura;
}
```

Chamadas a funções usadas como argumentos de outras funções

- Podemos também chamar uma função como argumento para outra função.
- A sintaxe é a mesma utilizada na chamada de variáveis como argumentos para funções.

Chamadas a funções usadas como argumentos de outras funções

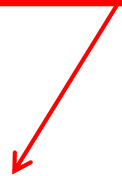
```
#include <iostream>
using namespace std;

float somarQuadrados(float m, float n){
    return m + n;
}

float quadrado(float n){
    return n * n;
}

int main() {
    float a, b;
    cout << "Digite dois numeros: ";
    cin >> a >> b;
    cout << "A soma dos quadrados e: " << somarQuadrados(quadrado(a),quadrado(b));
    return 0;
}
```

Usando outras funções
como argumentos



Parâmetros da função

- As variáveis que receberão as informações enviadas a uma função são chamadas de **parâmetros**;
- Devem ser declaradas entre parênteses, no cabeçalho de sua definição;
- Podem se utilizados livremente no corpo da função;
- São acessados somente pela função onde foram declarados.

Passagem de parâmetros por valor

- A função recebe apenas uma cópia do valor passado para ela e o guarda em uma variável local existente no seu corpo;
- O valor da variável original não é alterado porque a função não tem acesso a mesma.



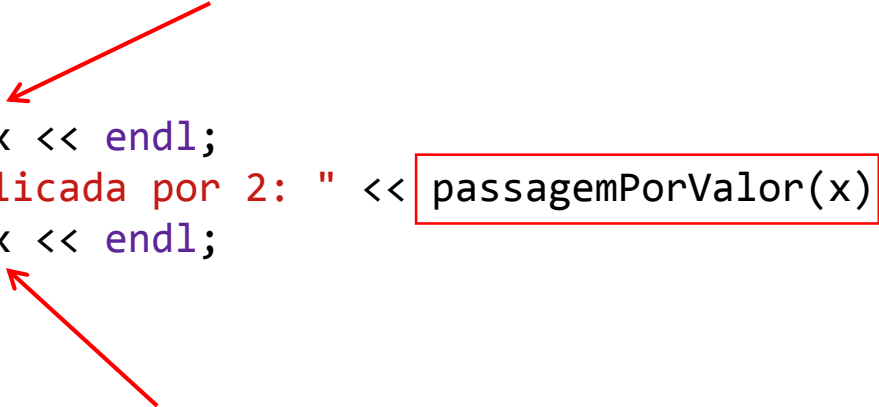
Passagem de parâmetros por valor

```
#include <iostream>
```

```
using namespace std;
```

```
int passagemPorValor(int copia)
{
    return copia * 2;
}
```

```
int main()
{
    int x = 10;
    cout << "Variavel x: " << x << endl;
    cout << "Copia de x multiplicada por 2: " << passagemPorValor(x) << endl;
    cout << "Variavel x: " << x << endl;
    return 0;
}
```



Passagem de parâmetros por valor

Exemplo:

```
#include <iostream>
```

```
using namespace std;
```

```
void troca(int a, int b);
```

```
int main()
```

```
{
```

```
    int a = 10;
```

```
    int b = 20;
```

```
    troca(a, b);
```

```
    cout << "Valor de A e B não foi alterado:" << a << b << endl;
```

```
}
```

```
void troca(int a, int b)
```

```
{
```

```
    a = b;
```

```
    b = a + 100;
```

```
    cout << "Valor de A e B: " << a << b << endl;
```

```
}
```

Variáveis a e b tem valor 10 e 20

Variáveis a e b continuam com o mesmo valor 10 e 20

Trocamos o valor de a com b dentro da função

Passagem de parâmetros por referência

O que é:

É quando uma variável é passada como parâmetro de uma função, e seu valor **original pode ser alterado**.



Passagem de parâmetros por referência

Exemplo:

```
#include <iostream>
```

```
using namespace std;
```

```
void troca(int &a, int &b);
```

```
int main()
```

```
{
```

```
    int a = 10;  
    int b = 20;
```

```
    cout << "Valor de A e B original:" << a << "-" << b << endl;
```

```
    troca(a, b);
```

```
    cout << "Valor de A e B FOI alterado:" << a << "-" << b << endl;
```

```
}
```

```
void troca(int &a, int &b)
```

```
{
```

```
    int temp;  
    temp = b;  
    b = a;  
    a = temp;
```

```
}
```

Referencia endereço de memória

Variáveis a e b tem valor 10 e 20

Variáveis a e b mudam de valor

Trocamos o valor de a com b dentro da função

Sobrecarga de funções

- Em C++ duas funções podem ter o mesmo nome se:
 - Tiverem um n° diferente de parâmetros e/ou
 - Se os parâmetros forem de tipos diferentes (ints floats,..)
- A função não pode ser sobrecarregadas apenas com diferentes tipo de retorno de função (ie, uma função retornar ints e a outra retornar floats) então os parâmetros é que interessam.

Parâmetros default (padrão)

- Pode acontecer que tenhamos que passar várias vezes o mesmo valor como parâmetro de uma função. Para simplificar a chamada a funções que variam pouco podemos definir um parâmetro default.
- Os parâmetros por default devem ser os últimos da lista, ou seja, mais à direita.

Parâmetros default (padrão)

```
#include <iostream>

using namespace std;

void funcao(int a,int b, int c = 100 )
{
    cout << "Meu Primeiro argumento: " << a<<endl;
    cout << "Meu segundo argumento: " << b<<endl;
    cout << "Meu terceiro argumento: " << c<<endl;
}

int main (void)
{
    funcao( 10, 30, 40);

    return 0;
}
```