

# Estruturas de Dados 3

## Revisão

Mailson de Queiroz Proença

# Estrutura de um programa C++

```
#include <iostream>
```

*Bibliotecas de funções*

```
using namespace std;
```

*Identificador da função*

```
int main()
```

*Função principal do programa*

```
{
```

```
    cout << "Olá Mundo !!!";
```

```
    return 0;
```

```
}
```

# Variáveis

```
#include <iostream>

using namespace std;

int main()
{
    const float numero_pi = 3.14;
    char genero = 'M';
    int idade = 31;
    float nota_final = 8.5;
    double salario = 1200.12;

    cout << "Variáveis:" << genero << "," << idade << "," << nota_final
          << "," << salario << "," << numero_pi << endl;

    double nota1, nota2;
    cout << "Digite as notas:" << endl;
    cin >> nota1 >> nota2;
    cout << "Notas: " << nota1 << "-" << nota2;
    return 0;
}
```

# Operadores

```
#include <iostream>

using namespace std;

int main()
{
    int x = 10 + 5; // soma
    int y = 4 - 20; // subtração
    int j = 34 * 160; // multiplicação
    int i = 6 / 2; // divisão
    int p = 150 % 2; // resto da divisão

    int acelerar = 100; // incremento pós-fixado
    acelerar++;

    int desacelerar = 100; // decremento pós-fixado
    desacelerar--;
    return 0;
}
```

# Operadores

```
#include <iostream>

using namespace std;

int main()
{
    int correr = 20; // incremento pré-fixado
    ++correr;

    int andar = 30; // decremento pré-fixado
    --andar;

    int a = 1; int b = 2; int c = 3; int d = 4; int e
    = 5;

    a += 1; // atribuição soma
    b -= 1; // atribuição subtração
    c *= 1; // atribuição multiplicação
    d /= 1; // atribuição divisão
    e %= 1; // atribuição resto da divisão
    return 0;
}
```

# Desvios Condicionais

```
#include <iostream>

using namespace std;

int main()
{
    int idade;
    cout << "Digite sua idade:";
    cin >> idade;

    if (idade >= 21)
        cout << "Maior de idade";
    else
        cout << "Menor de idade";
    return 0;
}
```

# Desvios Condicionais

```
#include <iostream>

using namespace std;

int main()
{
    int opcao;
    cout << "Informe uma opção (1,2,3):";
    cin >> opcao;

    switch (opcao)
    {
        case 1:
            cout << "Opção 1 Selecionada";
            break;
        case 2:
            cout << "Opção 2 Selecionada";
            break;
        case 3:
            cout << "Opção 3 Selecionada";
            break;
        default:
            cout << "Nenhuma Opção Selecionada";
            break;
    }
    return 0;
}
```

# Laço (Loops)

```
#include <iostream>

using namespace std;

int main()
{
    for (int i=0;i<=10;i++)
        cout << i << "\n";

    int j = 0;
    while (j <=10)
    {
        cout << j << "\n";
        j++;
    }

    int k = 0;
    do
    {
        cout << k << "\n";
        k++;
    }
    while (k <= 10);
    return 0;
}
```



# Modularização e Funções

## O que é Modularização:

- É o processo de decompor um programa em partes menores.
  - Facilitando a manutenção e o entendimento pelos desenvolvedores;
  - Rotinas reutilizáveis para outros programas e aplicações;
  - Efetuar atividades paralelas na construção do código-fonte.

# Modularização e Funções

**Veja o exemplo abaixo:**

Um carro é composto por vários componentes (portas, rodas, motor e etc). Esses componentes tornam o carro modular.

*Cada parte tem uma finalidade, todas juntas formam o carro.*

*Caso uma peça tenha problema, fazemos o ajuste neste local sem afetar o restante do carro.*



# Modularização e Funções

- Benefícios da Modularização em um programa:
  - Componentes menores formam um programa;
  - Programa fica mais legível para compreensão;
  - Facilidade na manutenção da aplicação;
  - Funcionalidade pode ser reutilizada em outros programas;
  - Previne duplicação de código e retrabalho.

# Funções

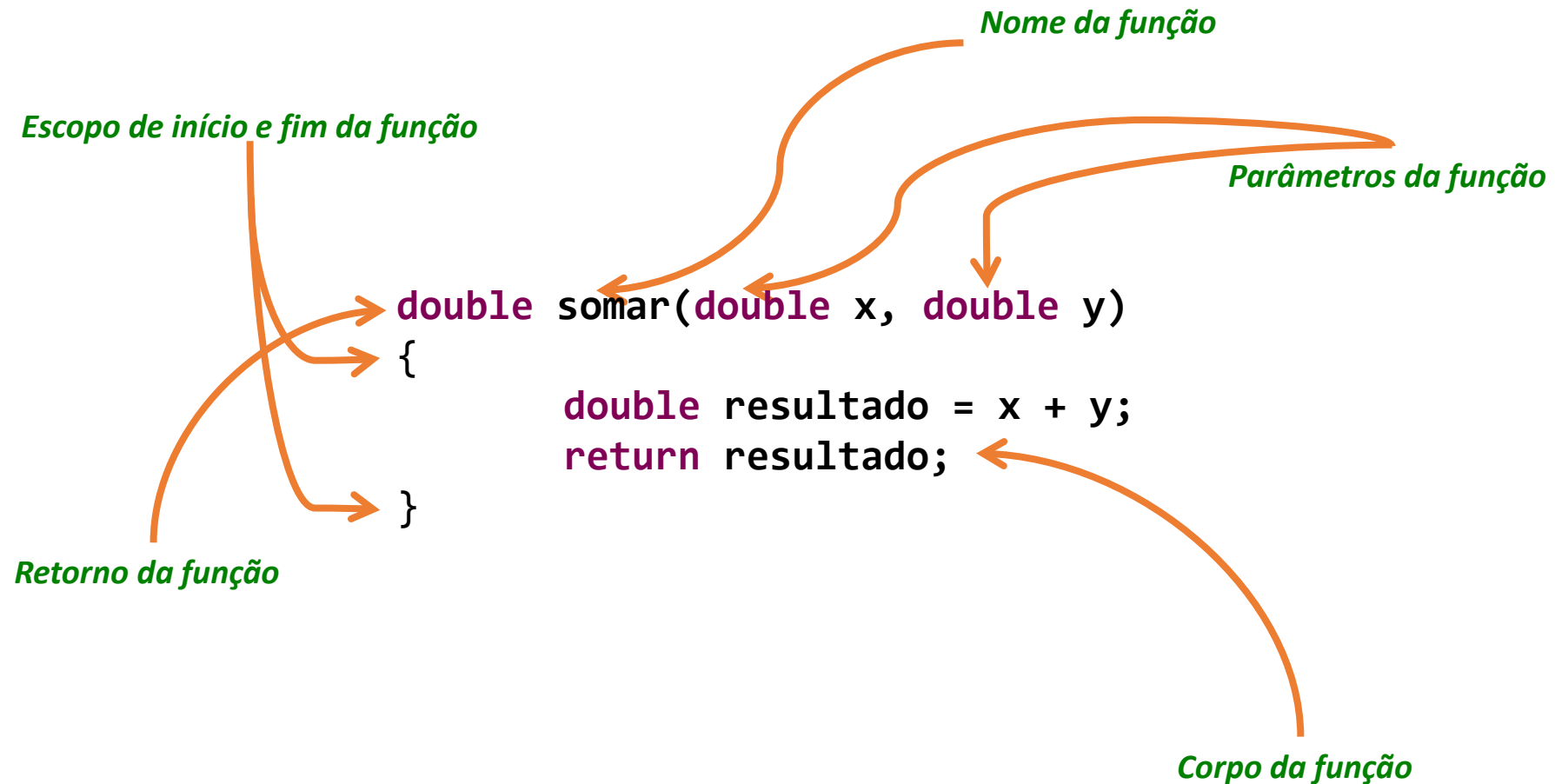
O que é:

- 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.



# Funções

## Estrutura de uma função:



# 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);

int main()
{
    double valor;
    valor = somar(10,15);
}
```

*Protótipo da função*




*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"*

# 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;
}
```

*Função não retorna um valor*

*Retorno da função do tipo void*

# Funções

## Usando return em uma função void:

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

```
#include <iostream>

using namespace std;

void dividir(int x, int y);

int main()
{
    int x, y;
    cout << "Digite os valores para dividir:\n";
    cin >> x >> y;
    dividir(x, y);
}

void dividir(int x, int y)
{
    if (y == 0)
    {
        cout << "Impossível dividir por zero";
        return;
    }
    cout << "Valor da Divisão:" << x / y;
}
```

*Terminando a execução da função com return*

*Não existe divisão por zero*

*Este código não vai ser executado*

# Parâmetros por Valor

O que é:

É quando uma variável é passada como parâmetro de uma função, e **seu valor original não é alterado**. Somente o **valor da sua cópia** pode ser alterado dentro da função.



# 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*

# Parâmetros por Referência

**O que é:**

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



# 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*