

# Curso Linguagem C++ – Nível 1: Bitwalker

 "Aquele que dá os primeiros passos entre zeros e uns."

## 1. Bitwalker – Aula 1

Tema: Primeiros passos em C++ (Dev-C++)

Download **Dev-C++**: <https://www.bloodshed.net>

---

### Objetivo da Aula

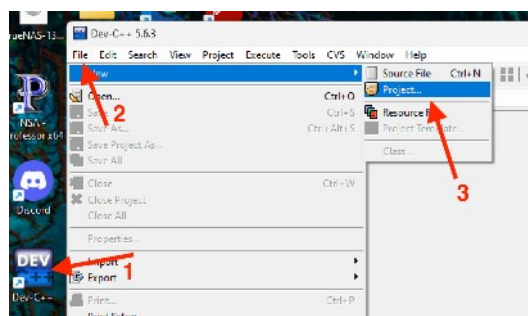
- Compreender o que é um programa em C++ e como ele é executado no Dev-C++.
  - Declarar variáveis e utilizar entrada/saída de dados (`cin` e `cout`).
  - Escrever programas simples de lógica sequencial.
  - Reconhecer e corrigir erros comuns de iniciantes.
- 

### Conteúdo da Aula

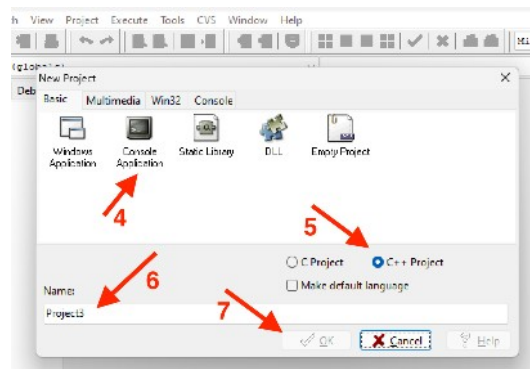
## 1) Criar o projeto e compilar

1. Iniciando o Dev-C++: Duplo click no ícone
2. Criando um projeto:

Click em: (2) File → (3) New → project



3. Click: (4) Console Application → (5) C++ Project

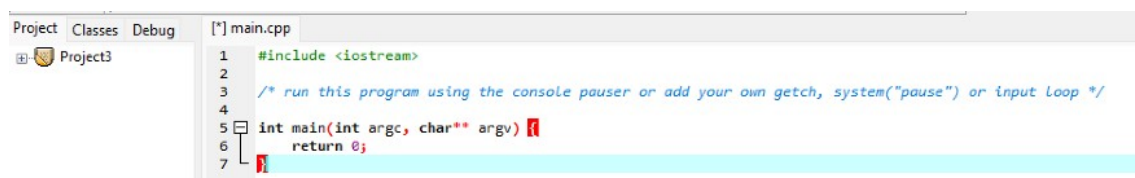


4. (6) Coloque o nome do Projeto → (7) Click OK

5. Escolha onde será gravado o projeto. Sugestão: para evitar problemas de sobrepor arquivos, crie um PASTA para cada Projeto.

6. Note que parte do que é necessário para o programa funcionar já está escrito. Você deve digitar seu programa entre as linha 5 e 6. Para criar novas linhas, click no final da linha 5 e tecle <ente>. Cada vez que pressionar <enter> uma nova linha é criada.

1. Estrutura mínima de um programa C++



Digite o Programa abaixo:

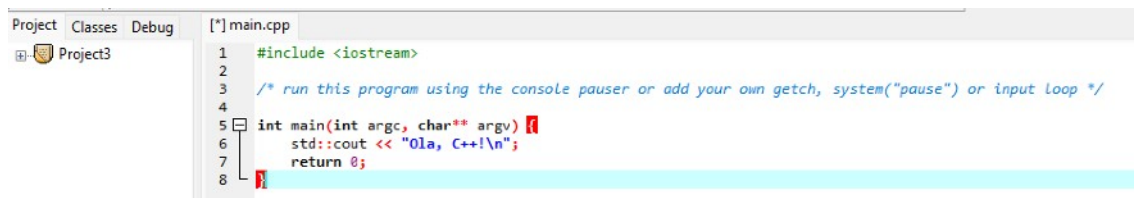
Observe que a única linha que falta é:

**std::cout << "Ola, C++!\n";**

```
#include <iostream>
```

```
/* run this program using the console pauser or add your own getch, system("pause") or input loop */
```

```
int main(int argc, char** argv) {  
    std::cout << "Ola, C++!\n";  
    return 0;  
}
```



```
1 #include <iostream>
2
3 /* run this program using the console pauser or add your own getch, system("pause") or input loop */
4
5 int main(int argc, char** argv) {
6     std::cout << "Ola, C++!\n";
7     return 0;
8 }
```

- `#include <iostream>` → biblioteca de entrada/saída.
- `int main()` → ponto inicial do programa.
- `std::cout` → saída de dados.
- `return 0;` → finalização correta do programa.

**Salve (Grave) como ola.cpp.** Click: File → Save

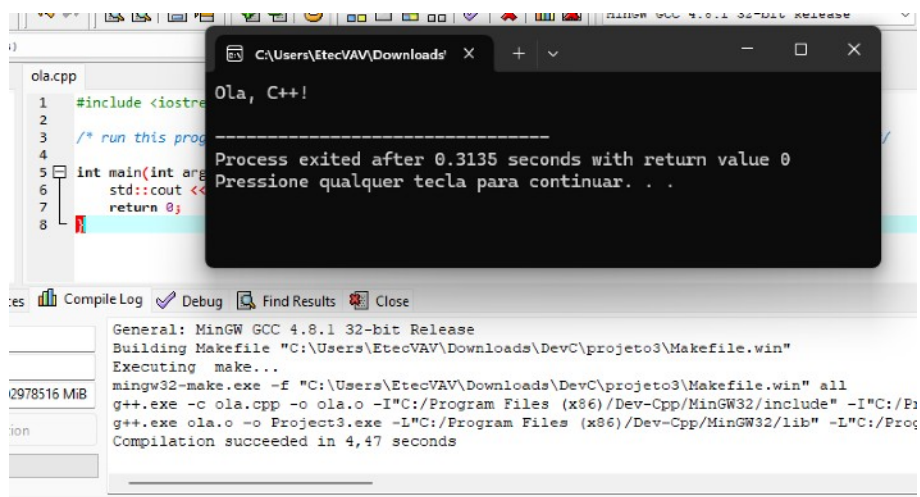
Escreva o nome: ola.cpp

Click no botão **Salvar**

**Executando o programa:** Tecle **F11** ou click: **Execute** → **Compile & Run**

Se pedir, salve o projeto.

O prompt de comandos (**CMD**) do Windows irá aparecer o programa irá rodar (funcionar) dentro dela. Pressione qualquer tecla para fechar o CMD.



## 2. Variáveis e Tipos Básicos

- `int` → números inteiros
- `double` → números decimais
- `char` → um único caractere

- bool → valores lógicos (true/false)
- string → texto (palavras, frases)

Exemplo:

As linhas que faltam estão em vermelho.

```
#include <iostream>
#include <string>

/* run this program using the console pauser or add your own getch, system("pause") or input loop */

int main(int argc, char** argv) {
    std::string nome;
    int idade;
    std::cout << "Digite seu nome: ";
    std::cin >> nome;
    std::cout << "Digite sua idade: ";
    std::cin >> idade;
    std::cout << "Ola, " << nome << "! Voce tem " << idade << " anos.\n";
    return 0;
}
```

---

### 3. Lógica Sequencial

Um programa executa instruções na ordem em que são escritas.

Exemplo:

As linhas que faltam estão em vermelho.

```
#include <iostream>

/* run this program using the console pauser or add your own getch, system("pause") or input loop */

int main(int argc, char** argv) {
    double preco, desconto;
    std::cout << "Preco do produto: ";
    std::cin >> preco;
    std::cout << "Desconto em %: ";
    std::cin >> desconto;

    double valorFinal = preco - (preco * desconto / 100.0);
    std::cout << "Preco final: " << valorFinal << "\n";
    return 0;
}
```

---

### 4. Erros Comuns

- Esquecer ; no final da linha.
  - Trocar = (atribuição) por == (comparação).
  - Usar cin para texto com espaços → só lê a primeira palavra (veremos solução na Aula 2).
-



## Exercícios Propostos

### 1. Eco

Leia um número inteiro e mostre: `Você digitou: X`.

### 2. Área de Retângulo

Leia base e altura e calcule a área.

### 3. Conversão de Minutos

Leia um número de minutos e mostre em horas:minutos.

### 4. Média de Três Valores

Leia três números e mostre a média aritmética.

### 5. Troca de Valores

Leia `a` e `b` e troque seus valores usando uma variável auxiliar.

---



## Conclusão da Aula

- Você aprendeu que todo programa C++ começa em `main()`.
- Viu como declarar variáveis e interagir com o usuário usando `cin` e `cout`.
- Testou programas simples de lógica sequencial.
- Identificou erros comuns que ocorrem nos primeiros programas.



Próximo passo (Aula 2): Operadores, ordem de execução e leitura de texto com `getline`.

# 1. Bitwalker – Aula 2

Tema: Operadores, ordem de execução e leitura de textos

---

## Objetivo da Aula

- Compreender os operadores aritméticos, relacionais e lógicos.
  - Aprender sobre precedência (ordem de execução) das operações.
  - Usar bibliotecas para formatar saídas (`iomanip`).
  - Ler frases inteiras usando `getline`.
  - Identificar erros comuns relacionados à entrada de dados.
- 

## Conteúdo da Aula

A partir deste ponto, iremos simplificar a escrita, mas você deve observar as linhas que estão faltando no novo projeto. Lembre-se que as linhas abaixo são criadas automaticamente.

---

```
#include <iostream>
```

```
/* run this program using the console pauser or add your own getch, system("pause") or input loop */
```

```
int main(int argc, char** argv) {
```

```
    return 0;
```

```
}
```

---

### 1. Operadores Aritméticos

- + soma
- - subtração
- \* multiplicação
- / divisão
- % resto (inteiros)

Exemplo:

```
#include <iostream>
int main() {
    int a = 10, b = 3;
    std::cout << "Soma: " << a + b << "\n";
    std::cout << "Divisao inteira: " << a / b << "\n";
    std::cout << "Resto: " << a % b << "\n";
    return 0;
}
```

---

## 2. Precedência (ordem)

- Parênteses → primeiro
- Multiplicação, divisão, resto → depois
- Soma e subtração → por último

Exemplo:

```
#include <iostream>
int main() {
    int r1 = 2 + 3 * 4;    // = 14
    int r2 = (2 + 3) * 4;  // = 20
    std::cout << "r1=" << r1 << " r2=" << r2 << "\n";
    return 0;
}
```

---

## 3. Formatando a saída (iomanip)

```
#include <iostream>
#include <iomanip> // std::fixed, std::setprecision

int main() {
    double pi = 3.14159265;
    std::cout << "Padrao: " << pi << "\n";
    std::cout << std::fixed << std::setprecision(2);
    std::cout << "Com 2 casas: " << pi << "\n";
    return 0;
}
```

---

## 4. Leitura de texto completo (getline)

```
#include <iostream>
#include <string>
int main() {
    std::string nomeCompleto;
    std::cout << "Digite seu nome completo: ";
    std::getline(std::cin, nomeCompleto);
    std::cout << "Ola, " << nomeCompleto << "!\n";
    return 0;
}
```

⚠️ Atenção: quando usamos `cin >>` antes de `getline`, sobra um `\n` no buffer. Para evitar problemas:

```
std::cin.ignore(); // limpa o \n antes de getline
```

---



## Exercícios Propostos

### 1. Operações Básicas

Leia dois inteiros e mostre a soma, subtração, multiplicação, divisão inteira e resto.

### 2. Média com precisão

Leia duas notas (`double`) e mostre a média com duas casas decimais.

### 3. Expressão matemática

Calcule o valor de  $(a+b)*c$  e compare com  $a+b*c$ . Mostre a diferença.

### 4. Nome completo

Peça ao usuário o nome completo e a idade, depois exiba:

"Olá, NOME, você tem X anos."

### 5. Conversor simples

Leia a temperatura em Celsius e converta para Fahrenheit. Fórmula:

$$F = C * 9/5 + 32$$

---



## Conclusão da Aula

- Aprendemos a usar operadores aritméticos, entender a ordem de execução e aplicar parênteses quando necessário.
- Descobrimos como formatar saídas numéricas com `io manip`.
- Vimos como ler frases inteiras com `getline`, resolvendo a limitação do `cin`.
- Exercícios reforçam a prática da matemática básica, entrada/saída e manipulação de strings.



Próxima aula (Aula 3): Estruturas de decisão (`if/else`) e primeiros testes lógicos.



# 1. Bitwalker — Aula 3

Tema: Estruturas de decisão (if/else), operadores relacionais e lógicos

---

## Objetivo da Aula

- Entender e aplicar if, else e else if.
  - Usar operadores relacionais (== != < <= > >=) e lógicos (&& || !).
  - Escrever validações simples de entrada e regras com múltiplas condições.
  - Evitar armadilhas comuns (comparação com double, = vs ==, alcance de blocos).
- 

## Conteúdo da Aula

### 1) If / Else básicos

```
#include <iostream>
int main() {
    int idade;
    std::cout << "Idade: ";
    std::cin >> idade;

    if (idade >= 18) {
        std::cout << "Maior de idade.\n";
    } else {
        std::cout << "Menor de idade.\n";
    }
    return 0;
}
```

### 2) Cadeia de decisões (else if)

```
#include <iostream>
int main() {
    int nota;
    std::cout << "Nota (0 a 100): ";
    std::cin >> nota;

    if (nota < 0 || nota > 100) {
        std::cout << "Nota invalida.\n";
    } else if (nota >= 90) {
        std::cout << "Conceito A\n";
    } else if (nota >= 70) {
        std::cout << "Conceito B\n";
    } else if (nota >= 50) {
        std::cout << "Conceito C\n";
    } else {
        std::cout << "Reprovado\n";
    }
    return 0;
}
```

### 3) Operadores lógicos (combinações)

```
#include <iostream>
int main() {
    double renda;
    int idade;
    std::cout << "Renda mensal e idade: ";
    std::cin >> renda >> idade;

    bool elegivel = (renda <= 2500.0 && idade >= 18) || (idade >= 60);
    if (elegivel) {
        std::cout << "Pode solicitar o beneficio.\n";
    } else {
        std::cout << "Nao atende aos requisitos.\n";
    }
    return 0;
}
```

### 4) Verificação de intervalo

```
#include <iostream>
int main() {
    int x;
    std::cout << "Valor inteiro: ";
    std::cin >> x;

    if (x >= 10 && x <= 20) {
        std::cout << "Dentro do intervalo [10,20].\n";
    } else {
        std::cout << "Fora do intervalo.\n";
    }
    return 0;
}
```

### 5) Ano bissexto (regras clássicas)

- É bissexto se:
  - divisível por 400 ou (divisível por 4 e não por 100).

```
#include <iostream>
int main() {
    int ano;
    std::cout << "Ano: ";
    std::cin >> ano;

    bool bissexto = (ano % 400 == 0) || (ano % 4 == 0 && ano % 100 != 0);
    if (bissexto) std::cout << "Bissexto\n";
    else          std::cout << "Nao bissexto\n";
    return 0;
}
```

---

### ⚠ Armadilhas comuns

- Atribuição vs comparação: `if (x = 5)` (ERRO) vs `if (x == 5)` (CERTO).
- Comparar `double`: evite `if (a == b)` com ponto flutuante; prefira tolerância:

```
const double EPS = 1e-9;
if (std::abs(a - b) < EPS) { /* considera "igual" */ }
```

- Alcance de blocos: sempre use chaves `{ }` ao escrever `if/else` com mais de uma instrução.
- 



### Exercícios Propostos (Aula 3)

#### 1. Maior de 2 números

Leia dois inteiros e imprima qual é o maior (ou “iguais”, se forem).

#### 2. Imposto simples

Leia o preço de um produto (double).

- Se `preco < 100`, imposto = 5%.
  - Se `100 <= preco < 500`, imposto = 10%.
  - Caso contrário, imposto = 15%.
- Mostre imposto e preço final.

#### 3. Classificação de triângulo

Leia três lados (double).

- Se não formam triângulo (violação da desigualdade triangular), informe.
- Caso formem: identifique equilátero, isósceles ou escaleno.

#### 4. Login simplificado

Leia `usuario` e `senha` (strings sem espaço).

Se `usuario=="admin"` e `senha=="1234"`, imprima “Acesso permitido”; senão, “Acesso negado”.

#### 5. Dentro de ao menos um intervalo

Leia um inteiro `x` e verifique se ele está em `[10,20]` ou em `[30,40]` (inclusive). Informe o resultado.

#### 6. Bandeira de preço

Leia `preco` (double) e `pagamento` (string: “avista” ou “prazo”).

- À vista: 8% de desconto.
  - A prazo: acréscimo de 5%.
- Mostre o valor final e uma mensagem coerente.
- 



### Conclusão da Aula

- Você aprendeu a tomar decisões no código com `if/else` e `else if`.
- Conseguiu combinar condições com `&&`, `||` e `!`.
- Viu padrões práticos: faixa/intervalo, classificação por faixas, regras combinadas.
- Reconheceu erros típicos e como evitá-los (especialmente `=` vs `==` e comparações com double).

➡ Próximo passo (Aula 4): laços de repetição (`while/for`), contadores e acumuladores.

# 1. Bitwalker — Aula 4

Tema: Laços de repetição — `while`, `for`, `do...while`, contadores e acumuladores

---

## Objetivo da Aula

- Entender e aplicar os laços `while`, `for` e `do...while`.
  - Usar contadores e acumuladores para somatórios, médias e contagens condicionais.
  - Controlar o fluxo com `break` e `continue` de forma consciente.
  - Evitar armadilhas comuns (loops infinitos e “off-by-one”).
- 

## Comentários no programa

**comentário de bloco:**

```
/*    */
```

**comentário de linha:**

```
//
```

---

## Exemplos:

```
/* isto é um comentário */
```

```
/*
```

```
isto também é um comentário.
```

```
*/
```

```
// comentário de linha
```

```
i = 2; // comentário
```

/\*

**O código a seguir é ignorado pois está dentro de um bloco de comentário.**

```
std::cout << "Valor inteiro: ";  
std::cin >> x;
```

\*/

---



## Conteúdo da Aula

1) **while** — repete enquanto a condição for verdadeira

```
#include <iostream>  
int main() {  
    int i = 1; // contador  
    while (i <= 5) { // condição de continuação  
        std::cout << i << " ";  
        i = i + 1; // atualização do contador  
    }  
    std::cout << "\n";  
    return 0;  
}
```

2) **for** — inicialização, condição, atualização no cabeçalho

```
#include <iostream>  
int main() {  
    for (int i = 1; i <= 5; i++) { // i++ é i = i + 1  
        std::cout << i << " ";  
    }  
    std::cout << "\n";  
    return 0;  
}
```

3) **do...while** — executa ao menos uma vez

```
#include <iostream>  
int main() {  
    int x;  
    do {  
        std::cout << "Digite um numero positivo: ";  
        std::cin >> x;  
    } while (x <= 0);  
    std::cout << "Valido: " << x << "\n";  
    return 0;  
}
```

4) Contadores e acumuladores (somatório e média)

```
#include <iostream>  
#include <iomanip>  
int main() {  
    int n;  
    std::cout << "Quantos valores? ";  
    std::cin >> n;
```

```

double soma = 0.0;
for (int i = 1; i <= n; i++) {
    double v;
    std::cout << "Valor " << i << ": ";
    std::cin >> v;
    soma += v; // acumulador
}
double media = (n > 0) ? (soma / n) : 0.0;
std::cout << std::fixed << std::setprecision(2);
std::cout << "Soma = " << soma << ", Media = " << media << "\n";
return 0;
}

```

## 5) Tabuada (laço simples)

```

#include <iostream>
int main() {
    int n;
    std::cout << "Tabuada de: ";
    std::cin >> n;

    for (int i = 0; i <= 10; i++) {
        std::cout << n << " x " << i << " = " << (n * i) << "\n";
    }
    return 0;
}

```

## 6) break e continue

```

#include <iostream>
int main() {
    for (int i = 1; i <= 10; i++) {
        if (i == 7) break; // encerra o laço
        if (i % 2 == 0) continue; // pula pares
        std::cout << i << " "; // imprime apenas ímpares < 7
    }
    std::cout << "\n";
    return 0;
}

```

## 7) Fatorial (atenção a overflow em int)

```

#include <iostream>
int main() {
    int n;
    std::cout << "n (0..12 recomendado): ";
    std::cin >> n;

    if (n < 0) { std::cout << "Invalido\n"; return 0; }

    unsigned long long fat = 1;
    for (int i = 2; i <= n; i++) fat *= i;
    std::cout << n << "! = " << fat << "\n";
    return 0;
}

```

### 8) Fibonacci (n termos)

```
#include <iostream>
int main() {
    int n;
    std::cout << "Quantidade de termos: ";
    std::cin >> n;

    long long a = 0, b = 1; // primeiros termos
    for (int i = 0; i < n; i++) {
        std::cout << a << " ";
        long long prox = a + b;
        a = b;
        b = prox;
    }
    std::cout << "\n";
    return 0;
}
```

### 9) Validação em loop (sentinela)

Ler números até digitar -1 (sentinela). Somar somente os positivos.

```
#include <iostream>
int main() {
    int x;
    long long soma = 0;
    std::cout << "Digite inteiros (fim: -1)\n";
    while (true) {
        std::cin >> x;
        if (x == -1) break; // sentinela
        if (x > 0) soma += x; // acumula só positivos
    }
    std::cout << "Soma dos positivos = " << soma << "\n";
    return 0;
}
```

---

### ⚠️ Armadilhas comuns

- Loop infinito: esquecer de atualizar o contador dentro do `while`.
- Off-by-one: confundir limites (ex.: `i < 10` vs `i <= 10`).
- Divisão inteira sem querer: `3/2 == 1`. Para média, use `double`.
- Overflow: fatorial cresce muito rápido; evite `int`.
- Variáveis não inicializadas: sempre inicialize acumuladores (ex.: `soma = 0`).



### Exercícios Propostos (Aula 4)

#### 1. Contagem regressiva

Leia um inteiro `n` (`n >= 0`) e exiba `n`, `n-1`, ..., `0`.

Use `while` ou `for`.



## 2. Somatório com filtro

Leia  $n$  e depois  $n$  inteiros. Some apenas os pares e exiba a soma e a quantidade de pares.

## 3. Média até sentinela

Leia números `double` até o usuário digitar `-1`. Calcule e exiba a média dos valores  $\geq 0$  (ignore negativos, exceto o sentinela). Trate o caso de nenhum valor válido.

## 4. Tabuada em intervalo

Leia  $a$  e  $b$  (com  $a \leq b$ ). Para cada  $k$  em  $[a, b]$ , imprima a tabuada de  $k$  de 0 a 10.

Exige laço aninhado.

## 5. Contagem de dígitos

Leia um inteiro não negativo e conte quantos dígitos ele possui (ex.:  $0 \rightarrow 1$  dígito;  $12345 \rightarrow 5$ ).

Dica: use divisão inteira repetida por 10.

## 6. Maior e menor

Leia  $n$  e depois  $n$  inteiros. Determine o maior e o menor valores lidos.

Dica: inicialize com o primeiro valor lido.

## 7. Aproximação de $\pi$ (Leibniz)

Leia  $n$  ( $n$  termos) e calcule  $\pi \approx 4 * \sum_{k=0}^{n-1} [ (-1)^k / (2k+1) ]$ . Mostre o resultado com 6 casas decimais (<iomanip>).

## 8. Validação forte de entrada

Leia um inteiro entre 1 e 100. Enquanto estiver fora da faixa, mostre mensagem e leia novamente.

## 9. Fibonacci — posição

Leia  $n$  ( $n \geq 1$ ) e imprima o  $n$ -ésimo termo de Fibonacci (começando  $F_1=0$ ,  $F_2=1$ ).

## 10. Retângulo de asteriscos

Leia `largura` e `altura` e desenhe um retângulo usando `*` (texto).

Use laço aninhado.

---

## Conclusão da Aula

- Você aprendeu a escolher o laço adequado: `while` (condição antes), `for` (contagem clássica), `do...while` (executa ao menos 1 vez).
- Usou contadores e acumuladores para somatórios e médias.
- Viu sentinela, laços aninhados, `break/continue` e cuidados com limites.
- Está pronto(a) para resolver problemas que exigem repetição e controle de fluxo.

➡ Próxima aula (Aula 5): Funções — criar, chamar, passar parâmetros, retornar valores e organizar o código.

# 1. Bitwalker — Aula 5

Tema: Funções, parâmetros, retorno e organização do código

---

## Objetivo da Aula

- Entender o que é uma função e por que utilizá-la (reuso, clareza, testes).
  - Declarar, definir e chamar funções com e sem retorno.
  - Usar parâmetros (por valor e por referência) e `const` para segurança.
  - Conhecer prototipagem (declaração antes do `main`) e organização em arquivos.
- 

## Conteúdo da Aula

### 1) Anatomia de uma função

```
// retorno   nome      (parâmetros)
int soma(int a, int b) {           // definição
    return a + b;
}

#include <iostream>
int main() {
    int r = soma(3, 4);           // chamada
    std::cout << r << "\n";      // 7
    return 0;
}
```

- Tipo de retorno: `void` (sem retorno) ou um tipo (`int`, `double`, `std::string`, ...).
  - Parâmetros: lista de entradas; podem ser zero, um ou vários.
- 

### 2) Funções `void` (efeito colateral/saída)

```
#include <iostream>
void linha(int n) {                // sem retorno
    for (int i = 0; i < n; ++i) std::cout << '-';
    std::cout << "\n";
}
int main() {
    linha(10);
    std::cout << "Titulo\n";
    linha(10);
}
```

### 3) Prototipagem (declara antes, define depois)

```
#include <iostream>

// protótipo (assinatura)
```

```
double media(double a, double b);

int main() {
    std::cout << media(7.5, 8.0) << "\n";
    return 0;
}

// definição (pode vir depois do main)
double media(double a, double b) {
    return (a + b) / 2.0;
}
```

Útil quando deseja organizar seu arquivo: `main` primeiro, funções depois, ou quando dividir em múltiplos arquivos.

---

#### 4) Escopo e variáveis locais

```
int x = 10; // global (evite em projetos grandes)

int duplica(int x) { // este x é local (sombreia o global)
    return x * 2;
}
```

- Variáveis locais existem apenas dentro do bloco `{}`.
  - Evite depender de globais — dificulta testes e manutenção.
- 

#### 5) Parâmetros — por valor vs por referência

- Por valor (padrão): a função recebe cópias.
- Por referência (&): a função recebe acesso direto ao argumento (pode modificar).

```
#include <iostream>
void troca(int& a, int& b) { // referencia
    int tmp = a; a = b; b = tmp;
}
int main() {
    int x = 5, y = 9;
    troca(x, y);
    std::cout << x << " " << y << "\n"; // 9 5
}
```

Se não quer modificar, mas quer evitar cópias grandes, use referência constante: `const Tipo&`.

---

#### 6) `const` para segurança

```
#include <string>
int tamanho(const std::string& s) { // não copia e não permite alterar s
    return (int)s.size();
}
```

---

## 7) Sobrecarga simples (mesmo nome, assinaturas diferentes)

```
int soma(int a, int b)          { return a + b; }  
double soma(double a, double b) { return a + b; } // duas versões
```

O compilador escolhe pela combinação de tipos ao chamar a função.

---

## 8) Organização mínima em arquivos (conceito)

- Em projetos maiores:

- arquivo de cabeçalho (.h): declarações (protótipos).

- arquivo de código (.cpp): definições (implementações).

- No Dev-C++, para começar, pode manter tudo em um .cpp. Quando crescer, migre para .h + .cpp.

---

## Exemplos práticos

### Exemplo A — Calculadora com funções

```
#include <iostream>  
double soma(double a, double b) { return a + b; }  
double sub (double a, double b) { return a - b; }  
double mul (double a, double b) { return a * b; }  
double divs(double a, double b) { return (b==0)?0.0:a/b; }  
  
int main() {  
    double a, b; char op;  
    std::cin >> a >> op >> b;  
    if (op == '+') std::cout << soma(a,b) << "\n";  
    else if (op == '-') std::cout << sub(a,b) << "\n";  
    else if (op == '*') std::cout << mul(a,b) << "\n";  
    else if (op == '/') std::cout << divs(a,b) << "\n";  
    else std::cout << "Operador invalido\n";  
}
```

### Exemplo B — Limpeza de string (passagem por referência)

```
#include <iostream>  
#include <string>  
void trimFim(std::string& s) {          // modifica o argumento  
    while (!s.empty() && (s.back()==' ' || s.back()=='\t'))  
        s.pop_back();  
}  
int main() {  
    std::string t = "Kael  \t";  
    trimFim(t);  
    std::cout << "[" << t << "]\n"; // [Kael]  
}
```

---



## Exercícios Propostos (Aula 5)

### 1. Máximo de três

Implemente `int max3(int a, int b, int c)` que retorna o maior.

### 2. Normalização de nome

Escreva `void capitaliza(std::string& s)` que deixa a primeira letra maiúscula e o restante minúsculo (somente letras ASCII).

### 3. Média ponderada

Crie `double mediaP(double n1, double p1, double n2, double p2)` que retorna  $(n1 * p1 + n2 * p2) / (p1 + p2)$ . Trate  $p1 + p2 == 0$ .

### 4. Inversão in-place

Escreva `void inverte(std::string& s)` que inverte a string no próprio parâmetro (sem criar string nova grande).

### 5. Contagem de vogais

`int contaVogais(const std::string& s)` que conte a, e, i, o, u (maiúsculas/minúsculas).

### 6. Fatorial via função

`unsigned long long fat(int n)` que calcule  $n!$  (trate  $n < 0$ ). Teste no `main`.

### 7. Fibonacci via função

`unsigned long long fib(int n)` que devolva o  $n$ -ésimo termo (assuma  $fib(0)=0$ ,  $fib(1)=1$ ).

### 8. Ordenação de 3 valores (swap)

Faça `void ordena3(int& a, int& b, int& c)` que deixa  $a \leq b \leq c$  usando trocas.

### 9. Distância euclidiana

Crie `double dist2d(double x1, double y1, double x2, double y2)` que calcula  $\sqrt{(x2-x1)^2 + (y2-y1)^2}$ .

## 10. Menu com funções

Monte um pequeno menu no `main` que chama funções:

1) `soma`, 2) `sub`, 3) `mul`, 4) `div` — até o usuário digitar 0 para sair.

---

## Conclusão da Aula

- Você aprendeu a declarar, definir e chamar funções.
- Diferenciou parâmetro por valor (cópia) de referência (modifica o argumento) e aplicou `const` com segurança.
- Viu prototipagem e uma noção de organização em `.h/.cpp`.
- Com funções, seu código fica modular, testável e reutilizável — pronto para projetos maiores.

---

### **Assinado por:**

Zahroniel Syrran & Kael'Aran