

# Linguagens de Programação 1

```
int aula = 1;  
printf("\aAula #%d\n", ++aula + 1);
```

## Message of the Day

*"UNIX is simple. It just takes a genius to understand its simplicity."*

*"C is quirky, flawed, and an enormous success."*

— Dennis Ritchie

## **Dennis Ritchie (1941–2011)**

Criador da linguagem de programação **C**.

Foi um dos criadores do sistema operativo **UNIX**

Fundou o **World Wide Web Consortium W3C**

## Conteúdo da Aula

- ✓ Entradas e Saídas Pré-formatadas
- ✓ Funções `printf()` e `scanf()`
- ✓ Operadores
- ✓ Operadores Aritméticos
- ✓ Operadores Primários
- ✓ Operadores Relacionais
- ✓ Operadores Lógicos

## Entradas e Saídas Pré-formatadas

 O C possui funções específicas para **entrada e saída de dados**, sendo as principais:

 `printf()` → Exibir dados formatados na saída padrão (ecrã).

 `scanf()` → Ler dados formatados da entrada padrão (teclado).

## Função `printf()`

- 📌 A função `printf()` escreve um **texto formatado** no terminal.
- 📌 Aceita um número variável de argumentos.
- 📌 Sintaxe:

```
printf("Texto com %formato", variáveis);
```

### 📌 Exemplo Simples:

```
int idade = 25;  
printf("Idade: %d anos\n", idade);
```

✅ **Saída:** Idade: 25 anos

## **AB** **CD** Formatadores de `printf()`

| Formato                            | Tipo de Dado              | Exemplo                                  | Saída  |
|------------------------------------|---------------------------|--|--------|
| <code>%d</code> ou <code>%i</code> | Inteiro Decimal           | <code>printf("%d", 42);</code>           | 42     |
| <code>%x</code> , <code>%X</code>  | Hexadecimal               | <code>printf("%x, %X", 255, 170);</code> | ff, AA |
| <code>%o</code>                    | Octal                     | <code>printf("%o", 255);</code>          | 377    |
| <code>%u</code>                    | Inteiro Decimal sem sinal | <code>printf("%u", 10);</code>           | 10     |

**AB**  
**CD** **Formatadores de `printf()` (cont.)**

| Formato         | Tipo de Dado | Exemplo                              | Saída |
|-----------------|--------------|--------------------------------------|-------|
| <code>%c</code> | Caractere    | <code>printf("%c", 'A');</code>      | A     |
| <code>%s</code> | String       | <code>printf("%s", "0la");</code>    | 0la   |
| <code>%f</code> | Numero real  | <code>printf("%.2f", 3.1415);</code> | 3.14  |

Supor que: `printf("%c", 97);`

## **AB** **CD** Formataores de `printf()` (cont.)

| Formato         | Tipo de Dado  | Exemplo                                 | Saída                  |
|-----------------|---|---|------------------------|
| <code>%e</code> | Numero real notação científica                          | <code>printf("%e",<br/>0.001);</code>   | <code>1e-<br/>3</code> |
| <code>%E</code> | Numero real em notação científica                       | <code>printf("%E",<br/>0.01);</code>    | <code>1E-<br/>2</code> |
| <code>%g</code> | escolhe automaticamente a forma que ocupar menos espaço | <code>printf("%g",<br/>0.00001);</code> | <code>1E-<br/>5</code> |



## Opções de Formatação no `printf()`

```
int a = 42
```

| Formato           | Exemplo                           | Saída                 | Explicação  |
|-------------------|-----------------------------------|-----------------------|---|
| <code>%5d</code>  | <code>printf("'%5d'", a);</code>  | <code>* 42*</code>    | imprime <code>a</code> ocupando 5 caracteres                              |
| <code>%-5d</code> | <code>printf("'%-5d'", a);</code> | <code>*42 *</code>    | imprime <code>a</code> ocupando 5 caracteres alinhado à esquerda          |
| <code>%06d</code> | <code>printf("'%06d'", a);</code> | <code>*000042*</code> | imprime <code>a</code> ocupando 6 caracteres, preenche os espaços com 0's |

## Opções de Formatação no `printf()` (cont.)

```
float pi = 3.14159
```

| Formato             | Exemplo                                  | Saída                | Explicação   |
|---------------------|--|----------------------|--|
| <code>%4.2f</code>  | <code>printf("*%4.2f*",<br/>pi);</code>  | <code>*3.14*</code>  | imprime <code>pi</code> ocupando 4 caracteres, 2 à direita da vírgula.                             |
| <code>%05.2f</code> | <code>printf("*%05.2f*",<br/>pi);</code> | <code>*03.14*</code> | imprime <code>pi</code> ocupando 5 caracteres, 2 à direita da vírgula. Preenche os espaços com 0's |

 A vírgula também é um caracter

## Função `scanf()`

 `scanf()` permite ler valores formatados do teclado.

 Sintaxe:

```
scanf("%formato", &variavel);
```


 Exemplo:




```
int num;  
scanf("%d", &num);
```

## **Formatadores de `scanf()`**

| Formato         | Tipo de Dado | Exemplo de Entrada |
|-----------------|--------------|--------------------|
| <code>%d</code> | Inteiro      | 42                 |
| <code>%f</code> | Float        | 3.14               |
| <code>%c</code> | Caractere    | A                  |
| <code>%s</code> | String       | Hello              |
| <code>%x</code> | Hexadecimal  | ff                 |

 **IMPORTANTE:**

 `scanf("%d", &var);` → Usa **&** para armazenar valores em variáveis!

 **Exceção:** Para strings (`%s`), não é necessário **&**  

## Erros Comuns no `scanf()`

✗ Falta do `&` (passagem por referência)

```
int x;  
scanf("%d", x); // ✗ ERRO
```

✓ Correcto:

```
int x;  
scanf("%d", &x); // ✓ Correto
```

✗ Leitura insegura de strings:

```
char nome[10];  
scanf("%s", nome);  
// ⚠ PERIGOSO! Pode causar `buffer overflow`
```

✓ Solução mais segura:

```
scanf("%9s", nome);  
// ✓ Limita a entrada a 9 caracteres
```

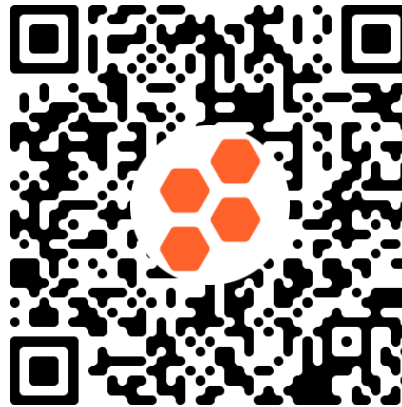
## Exemplo Completo

```
#include <stdio.h>
int main() {
    int idade;
    float altura;
    char nome[20];
    printf("Digite seu nome: ");
    scanf("%19s", nome); // Limita a 19 caracteres
    printf("Digite sua idade e altura: ");
    scanf("%d %f", &idade, &altura);
    printf("Nome: %s | Idade: %d | Altura: %.2f\n", nome, idade, altura);
    return 0;
}
```

✓ **Entrada:** Alice 25 1.65

✓ **Saída:** Nome: Alice | Idade: 25 | Altura: 1.65

## ? Quizz - Entradas e Saídas Pré-formatadas



No campo nome devem colocar o número de aluno 2XXXXXXX.

# Operadores



## O que são Operadores?

 Operadores são símbolos que realizam operações em literais ou variáveis.

 Exemplo: `+` é o operador de soma.

 Podem ser:

 Operadores **Aritméticos**

 Operadores **Relacionais**

 Operadores **Atribuição**

 Operadores **Lógicos**

## + Operadores Aritméticos

São Utilizados em **tipos inteiros e reais**.

 Podem ser:

✅ **Unários** → Requerem um único operando ( `x++` , `-a` ).

✅ **Binários** → Requerem dois operandos ( `x + y` , `a * b` ).

### Operadores Aritméticos Unários

| Operador       | Nome         | Exemplo             |
|----------------|--------------|---------------------|
| <code>+</code> | Unário Mais  | <code>i = +1</code> |
| <code>-</code> | Unário Menos | <code>j = -i</code> |

## **12** **34** Operadores Aritméticos Binários

| Operador | Nome          | Exemplo            |
|----------|---------------|--------------------|
| <b>+</b> | Adição        | <code>y + z</code> |
| <b>-</b> | Subtracção    | <code>x - y</code> |
| <b>*</b> | Multiplicação | <code>x * y</code> |
| <b>/</b> | Divisão       | <code>x / y</code> |
| <b>%</b> | Módulo        | <code>x % y</code> |

## **12** **34** Precedência e Associatividade

 Regras para resolver ambiguidades em expressões com múltiplos operadores.

 Ordem de Precedência:

| Precedência           |   |   |   |           | Associatividade |
|-----------------------|---|---|---|-----------|-----------------|
| Mais alta <b>1</b> :  | + | − |   | (unário)  | dir → esq       |
| Média <b>2</b> :      | * | / | % |           | esq → dir       |
| Mais baixa <b>3</b> : | + | − |   | (binário) | esq → dir       |

 Operadores na mesma linha têm precedência igual

### ✓ Exemplos de precedências diferentes:

$i + j * k \rightarrow \text{equivale a } i + (j * k)$

$-i * -j \rightarrow \text{equivale a } (-i) * (-j)$

$+i + j / k \rightarrow \text{equivale a } (+i) + (j/k)$

### ✓ Exemplos de precedências iguais:

📌 **Associativdade à esquerda** (resolvem-se da esquerda  $\rightarrow$  direita):

$i - j - k \rightarrow \text{equivale a } (i - j) - k$

$i * j / k \rightarrow \text{equivale a } (i * j) / k$

📌 **Associativdade à direita** (resolvem-se da direita  $\rightarrow$  esquerda):

$- + i \rightarrow \text{equivale a } - (+i)$

$+ - i \rightarrow \text{equivale a } + (-i)$

## Modificadores de Precedência

 Parênteses `()` aumentam a precedência de uma operação.

 Exemplo:

```
int x = (1 + 2) * 3; // x = 9
```

 Uso correcto evita ambiguidades em expressões matemáticas.

# Operadores de Atribuição

## O que é um Operador de Atribuição?

 O operador de atribuição ( `=` ) é utilizado para armazenar valores em variáveis.

 Forma geral:

```
variável = expressão;
```

 Exemplos:

```
int i = 5;    // ● i recebe 5
```

```
float f = 3.14; // ● f recebe 3.14
```

 Atribuições podem envolver expressões:

```
int x = 10, y = 20;  
int z = x + y;    // ● z recebe 30
```



## Conversão Implícita na Atribuição

 A conversão ocorre automaticamente.

 Exemplos:

```
int i;  
float f;
```


```
i = 72.99f; // ⚠ i recebe 72 (parte decimal truncada)  
f = 136; // ● f recebe 136.0
```


 Atenção à perda de precisão!

## Efeito Colateral da Atribuição

 A atribuição **modifica** a variável do lado esquerdo do operador - efeito colateral.

 **Exemplo:**

```
int i = 0; //  Modificação direta
```


 No entanto, a atribuição continua a ser um operador, por isso a sua avaliação **retorna o valor da variável após a atribuição**.

```
int x;  
int y = (x = 10); //  y recebe 10
```

## Atribuições Encadeadas


 Podemos encadear várias atribuições.

```
int i, j, k;  
i = j = k = 0; //  Todas as variáveis recebem 0
```

 O operador de **atribuição** ( = ) é **associativo à direita**.

 Equivalente a:

```
i = (j = (k = 0));
```

 \*A execução ocorre da **direita** para a **esquerda**:

 k recebe 0

 j recebe o valor de k

 i recebe o valor de j

## Lvalue

- ✓ Operando do lado esquerdo do operador.
- ✓ Em C é **obrigatório** ser um **objeto armazenado na memória**, i.e. uma variável.

```
int a;  
a = 10; // ● 'a' é um lvalue válido
```

```
int a;  
10 = a; // ✗ 10 é um lvalue inválido
```

## Operadores de Atribuição Composta

 Atribuições compostas permitem simplificar expressões.

 Forma geral:

```
variável operador= expressão;
```

 Equivalente a:

```
variável = variável operador expressão;
```

 Exemplo:

```
int x = 10;  
x += 5; //  Equivalente a x = x + 5;
```

## Principais Operadores de Atribuição Composta

### Lista dos operadores mais comuns:

✓ `+=` (Soma e atribuição)

✓ `-=` (Subtração e atribuição)

✓ `*=` (Multiplicação e atribuição)

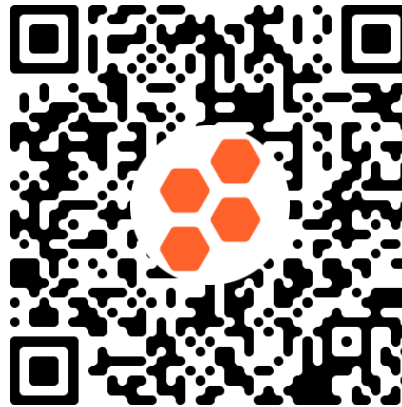
✓ `/=` (Divisão e atribuição)

✓ `%=` (Módulo e atribuição)

### Exemplos:

```
int x = 10;  
x *= 2; // ● x agora é 20  
x /= 4; // ● x agora é 5
```

## ? Quizz Operadores Aritméticos e de Atribuição



No campo nome devem colocar o número de aluno 2XXXXXXX.

# Operadores de Incremento e Decremento



## O que são os operadores `++` e `--`?

 Operadores unários utilizados para aumentar ou diminuir o valor de uma variável.

 Forma geral:

```
variável++; // Pós-incremento (incrementa depois)  
variável--; // Pós-decremento (decrementa depois)
```

```
++variável; // Pré-incremento (incrementa antes)  
--variável; // Pré-decremento (decrementa antes)
```

## Diferença entre Pré e Pós

### Pré-Incremento ( `++x` ) e Pré-Decremento ( `--x` )

✓ A variável é modificada **antes** de ser usada na expressão.

```
int x = 5;  
int y = ++x; // ● x agora é 6, y também recebe 6
```

### Pós-Incremento ( `x++` ) e Pós-Decremento ( `x--` )

✓ A variável é usada na expressão e só **depois** é modificada.




```
int x = 5;  
int y = x++; // ● y recebe 5, x agora é 6
```

## Exemplo de Uso

```
int i = 1;  
printf("i = %d\n", ++i); // ● Imprime "i = 2"  
printf("i = %d\n", i++); // ● Imprime "i = 2", mas i agora é 3
```

- 📌 O pré-incremento altera **i** **antes** da impressão.
- 📌 O pós-incremento altera **i** **depois** da impressão.

## Precedência e Associatividade

-  O `++` e `--` têm precedência maior que operadores aritméticos.
-  O pré-incremento tem associatividade da direita para a esquerda.
-  O pós-incremento tem associatividade da esquerda para a direita.

```
int a = 2, b = 3, c;  
c = ++a + b++; //  a = 3, b = 4, c = 6
```

## Exercício: Qual será a saída?

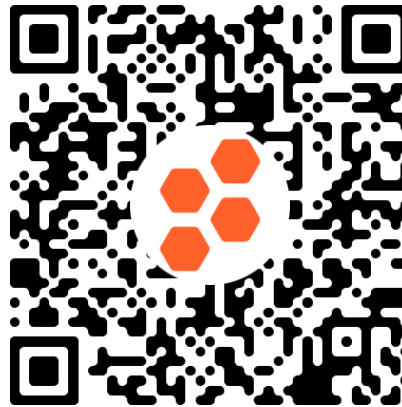
```
int i = 1, j = 2, k;  
k = ++i + j++;  
printf("i = %d, j = %d, k = %d\n", i, j, k);
```

### O que será impresso?

- (A) `i = 1, j = 3, k = 3`
- (B) `i = 2, j = 3, k = 4`
- (C) `i = 2, j = 2, k = 4`
- (D) `i = 2, j = 3, k = 5`

 Tenta resolver e depois experimenta o código!

## ? Quizz Operadores Unários (++ --)



No campo nome devem colocar o número de aluno 2XXXXXXX.

## Operadores Relacionais

 Usados para comparar valores e determinar relações entre eles.

 Retornam **1** (verdadeiro) ou **0** (falso).

| Operador | Significado    |
|----------|----------------|
| <        | Menor que      |
| >        | Maior que      |
| <=       | Menor ou igual |
| >=       | Maior ou igual |

```
int a = 5, b = 10;
if (a < b) {
    printf("a é menor que b\n");
}
```


## Operadores de Igualdade

 Usados para verificar igualdade ou diferença entre valores.

 Diferentes dos operadores de atribuição!

| Operador        | Significado  |
|-----------------|--------------|
| <code>==</code> | Igual a      |
| <code>!=</code> | Diferente de |

```
int x = 5, y = 10;
if (x != y) {
    printf("x é diferente de y\n");
}
```

 **Erro comum:** Usar `=` no lugar de `==` !

```
if (x = y) { // ❌ ERRO: Atribuição em vez de comparação
    printf("Isso sempre será verdadeiro!");
}
```



## Operadores Lógicos

 Usados para combinar expressões booleanas.

 Retornam **1** (verdadeiro) ou **0** (falso).

 Tabela de Operadores:

| Operador | Significado    |
|----------|----------------|
| !        | Negação lógica |
| &&       | E lógico       |
|          | Ou lógico      |

```
int idade = 20;
if (idade > 18 && idade < 65) {
    printf("Idade está na faixa adulta\n");
}
```

## Operadores Lógicos

### AND

| <b>&amp;&amp;</b> | 0 | 1 |
|-------------------|---|---|
| 0                 | 0 | 0 |
| 1                 | 0 | 1 |

### OR

| <b>  </b> | 0 | 1 |
|-----------|---|---|
| 0         | 0 | 1 |
| 1         | 1 | 1 |

### NOT

| <b>!</b> | Saida |
|----------|-------|
| !0       | 1     |
| !1       | 0     |

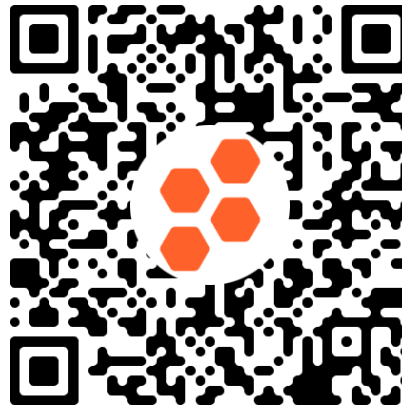
## Operadores Lógicos Exemplos

$!(-10) = 0$

$!10 = 0$

$!(-a + a) = 1$

## ? Quizz Operadores Relacionais



No campo nome devem colocar o número de aluno 2XXXXXXX.

# Precedência dos operadores

1 - mais prioritário = avaliado primeiro

15 - menos prioritário = avaliado em último

| Precedence | Operator | Description                                       | Associativity |
|------------|----------|---|---------------|
| 1          | ++ --    | Suffix/postfix increment and decrement            | Left-to-right |
|            | ()       | Function call                                     |               |
|            | []       | Array subscripting                                |               |
|            | .        | Structure and union member access                 |               |
|            | ->       | Structure and union member access through pointer |               |
| 2          | ++ --    | Prefix increment and decrement                    | Right-to-left |
|            | + -      | Unary plus and minus                              |               |
|            | ! ~      | Logical NOT and bitwise NOT                       |               |
|            | (type)   | Type cast   |               |
|            | *        | Indirection (dereference)                         |               |
|            | &        | Address-of  |               |
|            | sizeof   | Size-of   |               |
| 3          | * / %    | Multiplication, division, and remainder           | Left-to-right |
| 4          | + -      | Addition and subtraction                          | Left-to-right |
| 5          | << >>    | Bitwise left shift and right shift                | Left-to-right |
| 6          | < <=     | For relational operators < and ≤ respectively     | Left-to-right |
|            | > >=     | For relational operators > and ≥ respectively     |               |
| 7          | == !=    | For relational = and ≠ respectively               | Left-to-right |
| 8          | &        | Bitwise AND                                       | Left-to-right |
| 9          | ^        | Bitwise XOR (exclusive or)                        | Left-to-right |
| 10         |          | Bitwise OR (inclusive or)                         | Left-to-right |
| 11         | &&       | Logical AND                                       | Left-to-right |
| 12         |          | Logical OR  | Left-to-right |
| 13         | ?:       | Ternary conditional                               | Right-to-Left |
| 14         | =        | Simple assignment                                 | Right-to-Left |
|            | += -=    | Assignment by sum and difference                  |               |
|            | *= /= %= | Assignment by product, quotient, and remainder    |               |
|            | <<= >>=  | Assignment by bitwise left shift and right shift  |               |
|            | &= ^=  = | Assignment by bitwise AND, XOR, and OR            |               |
| 15         | ,        | Comma   | Left-to-right |

 **Q&A**

 **Dúvidas?**