

# Linguagem C

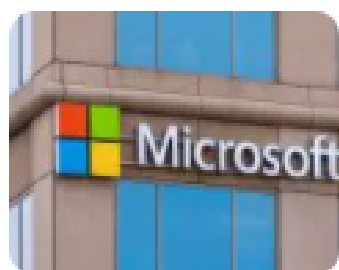
# Por que estudar linguagem C?



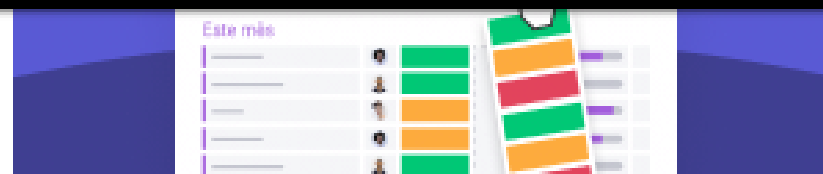
COMPARTILHAR



LEIA A SEGUIR



Microsoft supera



## Que linguagem de programação é usada para criar o Windows 10?

Axel Rietschin, engenheiro da Microsoft, explicou que a maior parte do *kernel* do sistema é escrito em C: “Você pode encontrar cópias filtradas do Kernel de Pesquisa do Windows, mesmo no [Github](#) e conferir”.

Essa linguagem de programação, no entanto, não é a única responsável por modelar o sistema operacional de Redmond. Em sua criação, além de C, as linguagens C #, JavaScript, TypeScript, VB.NET e C ++ também intervêm. Na verdade, o engenheiro explica que à medida que nos aproximamos do modo de usuário e de desenvolvimentos mais recentes, encontraremos menos C e mais C ++.

**As linguagens C e C++: qual a  
diferença entre elas?**



Quando mergulhamos no mundo da programação somos apresentados a diversas linguagens. É muito comum termos contato com o C ou o C++ e junto pode surgir o questionamento: **Qual a diferença entre elas?**

A linguagem de programação C é descrita como a **linguagem mãe**. Isso porque diversas outras linguagens utilizadas hoje surgiram utilizando C como base, PHP, Java, C# e o **C++**

Ela foi criada em 1972, **Dennis Ritchie**, com um dos objetivos de criar sistemas operacionais. É por isso que os sistemas baseados no Unix utilizam essa linguagem como a principal.



O C++ foi desenvolvido cerca de 8 anos depois, em 1980. Nessa época ainda era chamado de "*C with Classes*". Em 1982 ele passou a ser o C++ como conhecemos. Um “Olá mundo” ficaria da seguinte forma:

## “Olá mundo” em C:

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    printf("Olá Mundo!\n");
    return 0;
}
```

## “Olá mundo” em C++:

```
#include <iostream>

int main()
{
    std::cout << "Olá, Mundo!" << std::endl;
}
```

Basicamente, a diferença entre linguagem C e C++ é a POO (programação orientada a objetos).

A Programação orientada a objetos é um paradigma de desenvolvimento bastante utilizado atualmente.

Diversas linguagens, como Java e Python, suportam essa aplicação.

**Sugestão de leitura:** [POO: o que é programação orientada a objetos? | Alura Cursos Online](#)

# O esqueleto de um programa em Linguagem C

Para estruturar um programa em linguagem C, é necessário primeiramente incluir as seguintes bibliotecas:

`#include <stdio.h>`



Escrita e Leitura

`#include <stdlib.h>`



Biblioteca Auxiliar

## stdlib.h

comando	descrição
cd PASTA	abre uma pasta
chdir PASTA	abre uma pasta
cls	limpa a tela
color XX	mudar a cor da tela. XX é um hexadecimal onde o primeiro número é a cor do fundo e a segunda é a cor da letra.
copy ORIGEM DESTINO	copia o arquivo de origem para o seu destino.
date/t	mostra a data do sistema, sem alterá-la.
dir	exibe uma lista de arquivos e subpastas em
diskcopy UNIDADE1 UNIDADE2	copia o conteúdo de um disquete para o outro.
md PASTA	cria uma pasta
mem	mostra a memória utilizada e livre do sistema.
mkdir PASTA	cria uma pasta.
pause	pausa o programa e solicita o pressionamento de uma tecla para continuar
time/t	mostra a hora do sistema, sem alterá-la.
title NOME	define um nome para a janela do prompt.
tree	mostra a estrutura de pastas de uma unidade de forma gráfica
ver	mostra a versão do sistema operacional
vol	mostra o nome e o número de série do volume, caso haja um.

Após incluir as bibliotecas, iniciamos o programação com o comando que é o corpo do programa:

```
int main () {  
  
}
```

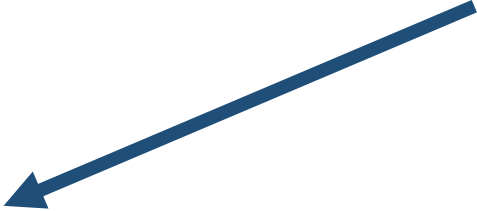
Este comando é do tipo **int**, ele precisa retornar no final da execução, então, inserimos o comando **return 0;** na ultima linha do código:

```
int main () {  
    return 0;  
}
```




Alguns comandos:

```
int main () {  
    printf("Hello World !\n");  
    return 0;  
}
```



O comando **printf** é utilizado para escrever mensagens na tela.



O comando **\n** pula uma linha após a escrita da frase

Para que o sistema, dê uma pausa no fim da execução, utiliza-se o comando `system("pause");` da biblioteca auxiliar `#include <stdlib.h>`:

```
int main () {  
    printf("Hello World !\n");  
    system("pause");  
    return 0;  
}
```

Esta é a estrutura básica para a construção de aplicações:

```
int main () {  
    system("pause");  
    return 0;  
}
```

# Declaração de Variáveis

Para executarmos uma variável, ela necessita estar na seguinte sequência:

**tipo nome;**

**Tipo** - Temos alguns tipos de variáveis:

**char:** Caracteres

**int:** Valores Inteiros

**float:** Valores Reais

**double:** Valores reais com uma maior precisão

**bool:** Valor lógico (true / false).

Para declararmos uma variável, ela necessita estar na seguinte sequência:

**tipo** nome;

**nome** - Para definir o nome das variáveis podemos utilizar:

a, b, c, d, ...

A, B, C, D, ...

0, 1, 2, 3, ...

–

Usos Proibidos:

Á, é, ô, ç, ....

!, \*, &, \$, ...

**Palavras reservadas:** Palavras que já existem dentro da linguagem.

Exemplos de nome de variáveis:

x  
soma  
\_soma  
soma1

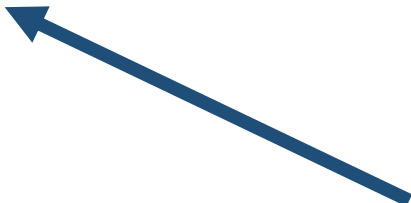
**Observação:** O nome das variáveis não devem começar com números.  
Ex.: **1soma**

A linguagem é *Case Sensitive*, ela diferencia letras maiúsculas de minúsculas.

A variável pode ser declarada linha a linha, ou na mesma linha quando for do mesmo tipo:

```
char letra = 'a';  
char numero = '0';  
int numero1 = 10;  
float numero2 = 5.25;
```

```
int nro1 = 10, nro2, nro3;
```



Observação: Utiliza-se pontos como separador decimal.



```
#include <stdio.h>  
#include <stdlib.h>
```

```
int main () {  
    char letra = 'a';  
    int numero1 = 10;  
    float numero2 = 5.25;  
  
    printf("Hello World !\n");  
    system("pause");  
    return 0;  
}
```

Resumo Geral

# Comando printf (saída)

O comando **printf** é utilizado para apresentar mensagens na tela (constantes e variáveis)

```
char letra = 'a';  
float n = 5.25;  
int nro = 10;
```

```
printf("%c \n", letra);  
printf("%d \n", letra);  
printf("%d \n", nro);  
printf("%d \n", n);
```

```
printf("%c \n", letra);
```

%c = caracter

```
printf("%d \n", nro);
```

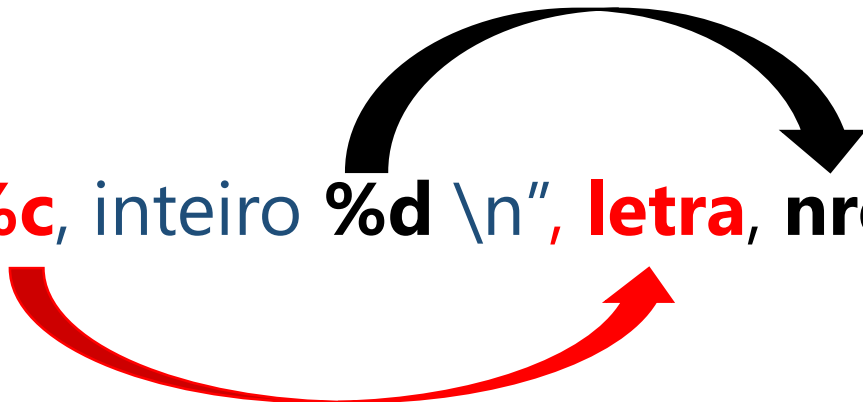
%d ou %i= número  
inteiro e caracter

```
printf("%f \n", n);
```

%f = número real

Podemos escrever mais de uma variável em uma linha:

```
printf("Letra %c, inteiro %d \n", letra, nro);
```

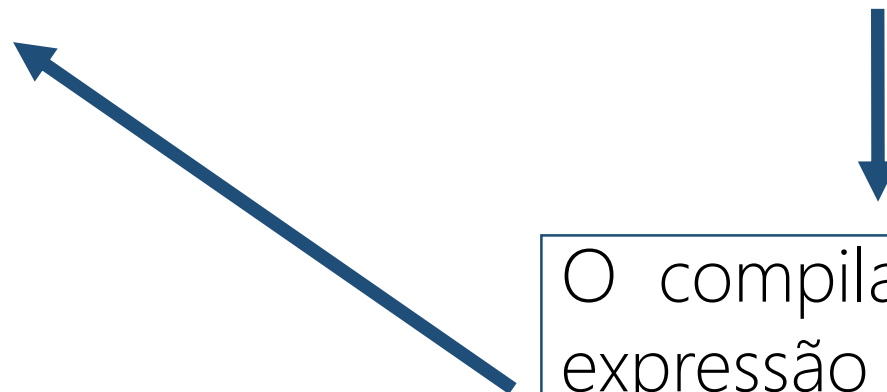


A mensagem apareceria na tela:

**Letra a, inteiro 10**

Podemos escrever expressões:

```
printf("Letra %c, inteiro %d \n", letra, nro+1);
```



O compilador resolve primeiro a expressão para depois apresentar o valor.

# Comando scanf (entrada)

O comando **scanf** é utilizado para fazer a leitura de dados inseridos pelo teclado para ser atribuído às variáveis:

**scanf**("tipo de entrada", variável);

Exemplos:

**scanf**("%c", &letra);

**scanf**("%d", &nro);

**scanf**("%f", &n);

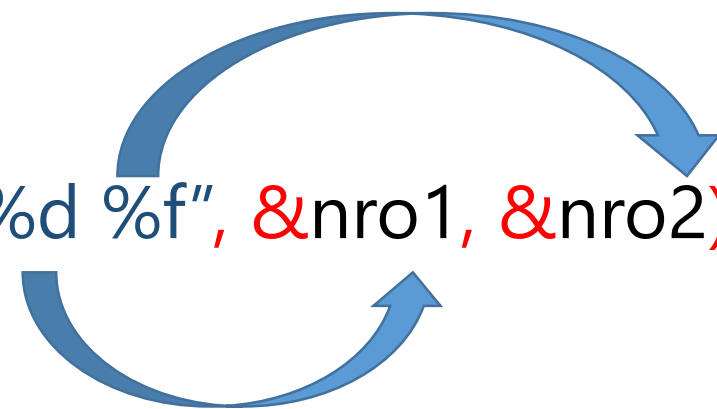
**scanf**("%s", &nome);

Definimos o tipo de entrada e atribuímos à variável.

Obs. É necessário utilizar o **&** antecedendo a variável



Podemos receber mais de um valor:



```
scanf("%d %f", &nro1, &nro2);
```

```
#include <stdio.h>  
#include <stdlib.h>
```

```
int main () {  
    int nro;
```

```
    printf("Digite um número: ");  
    scanf("%d", &nro);  
    printf("Número digitado: %d \n", nro);  
    system ("pause");  
    return 0;  
}
```

Exemplo

Comando gets (entrada de *strings*)

```
#include <stdio.h>
#include <stdlib.h>
```

```
int main () {
    char nome[50];
```

```
    printf("Digite o nome: ");
    gets(nome);
    system ("pause");
    return 0;
}
```

O comando gets permite a entrada de dados (apenas do tipo string)

# Operador de Atribuição

O operador da atribuição, funciona da seguinte forma:

**tipo** variável = expressão;

Pode receber:

int x = 5, y;



Um valor

x = y;



Outra Variável

x = x + 10;



Uma Expressão

Fique atento!

**tipo** variável = expressão;

int x = 5, y;

x = y;

x = x + 10;



Forma errada:

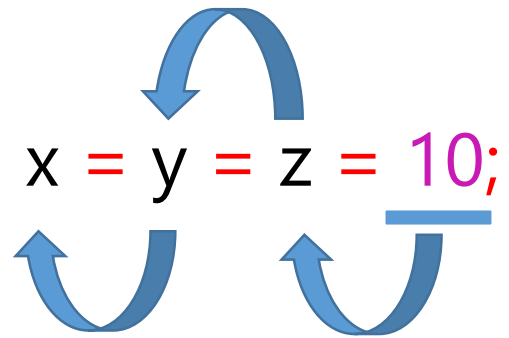
5 = x;

x + 10 = x;

Atribuições múltiplas:

```
int x , y , z;
```

```
x = y = z = 10;
```





# Constantes

A declaração de constantes, é a declaração de variáveis, que não poderão sofrer alterações:

```
const int nro = 5;
```



**Obs.:** O identificador deve sempre ser iniciada.

Outra forma, é definindo o valor, juntamente com as bibliotecas:

```
#define NRO 5
```

# Operações Aritméticas

Existem 4 tipos de operações:

Soma: +

Subtração: -

Multiplicação: \*

Divisão: /

Resto: %

```
int x1 , x2 = 10, x3 = 12;
```

```
char ch1, ch2 = '0', ch3 = 'A';
```

```
float f1, f2 = 5.25, f3 = 10.5;
```

Exemplos:

Soma:

```
x1 = x2 + x3;
```

```
ch1 = ch2 + ch3;
```

```
f1 = f2 + f3;
```

Subtração:

$x1 = x3 - x2;$

int e float

$f1 = -f2;$

Multiplicação:

$x1 = x3 * x2;$

int e float

$f1 = 3 * f2;$

Divisão:

$x1 = x2 / 2;$

int e float

$f1 = x2 / 2;$

$f2 = x2 / 2.0;$

Resto:

$x1 = x2 \% 2;$

int

$x1 = x2 \% 5;$

Obs.: Cuidado com a ordem de precedência:

$$f1 = f2 + 10 / 2.0; \quad \neq \quad f1 = (f2 + 10) / 2.0;$$

# Comentários



O comentário é uma forma de deixar um código oculto para o compilador ou, para inserir algum tipo de nota na programação.

```
//programa de soma
```

```
int x = 10;
```

```
// x = 2 + x;
```

```
printf("x = %d \n", x);
```

```
/*int z = 150;
```

```
z = 50 + x;*/
```

As `//` tornam uma linha de código "comentada".

Já `/* */` atribui comentário a um bloco do programa.

# Pré e Pós-incremento

Sempre que necessário incrementar + 1 em uma variável podemos utilizar:

`y++;`  
`++y;`

ao invés de `y = y + 1;`

E para decrementar -1 na variável, podemos utilizar:

`y--;`  
`--y;`

ao invés de `y = y - 1;`

Pré-incremento:

```
x = 10;  
y = ++x;  
// x++;  
// y = x;
```



Resultado:

```
y = 11;  
x = 11;
```



Neste exemplo, o incremento é feito **antes** do valor ser atribuído a variável **y**.

Pós-incremento:

```
x = 10;  
y = x++;  
// y = x;  
// x++;
```



Resultado:

```
y = 10;  
x = 11;
```



Neste exemplo, o incremento é feito **depois** do valor ser atribuído a variável **y**.

Atribuição simplificada  
(veremos posteriormente)

A atribuição simplificada é uma maneira de reescrever certas operações:

Método tradicional:

variável = variável "operação" expressão;

Atribuição simplificada:

variável "operação=" expressão;

Exemplo:

Método tradicional:

$x = x + y;$

Atribuição simplificada:

$x += y;$



Para outras operações é só substituir o operador.

# Operadores Relacionais



Utilizamos os operadores relacionais para fazer comparações entre valores. Ela funciona da seguinte forma:

**valor1 "operador\_relacional" valor2**

Esta comparação sempre resultará:

**0:** Comparação Falsa

**1:** Comparação Verdadeira

## Operadores Relacionais:

- > "Maior que"
- >= "Maior ou igual"
- < "Menor que"
- <= "Menor ou igual"
- == "Igual a"
- != "Diferente de"

Exemplos de operadores relacionais:

```
int x = 5;
```

```
printf("Resultado = %d \n", x == 5);
```

```
printf("Resultado = %d \n", x > 3);
```

```
printf("Resultado = %d \n", x <= 2);
```

# Operadores Lógicos

Utilizamos o operador lógico para comparar duas expressões relacionais. Ela funciona da seguinte forma:

expressão **“operador lógico”** expressão

Operadores:

**&&** - Operador “e”

**||** - Operador “ou”

Estas comparações sempre resultarão:

**0**: resultado Falso

**1**: resultado Verdadeiro

Exemplos:

```
int r, x = 5, y = 3;  
r = (x > 2) && (y > x);  
printf("Resultado = %d \n", r);
```

Resultado = 0

```
int r, x = 5, y = 3;  
r = (x > 2) || (y < x);  
printf("Resultado = %d \n", r);
```

Resultado = 1

Mais um operador:

**!(expressão)** - Operador de Negação:

Estas comparações sempre resultarão:

**0:** Se a expressão valer 1

**1:** Se a expressão valer 0



Negação do  
resultado

**Obs.** o símbolo **!** significa negação

Exemplos:

```
int r, x = 5, y = 3;
```

```
r = ! (x > 2);
```

```
printf("Resultado = %d \n", r);
```



Resultado = 0



Tabela Verdade:

A	B	!A	!B	A&&B	A  B
0	0	1	1	0	0
0	1	1	0	0	1
1	0	0	1	0	1
1	1	0	0	1	1

## Comando `scanf`

**#1** É aconselhável utilizar o comando `fflush(stdin)` para limpar o *buffer* do teclado logo após utilizarmos o comando de entrada `scanf`. Este comando evitará que o próximo comando de entrada deixe de ser executado por haver o caractere 13 (ENTER) no *buffer* referente a entrada anterior.

```
scanf("%c", &sexo);  
fflush(stdin);
```

## Comando **scanf**

**#2** Ao ser solicitado **string**, a formatação **"%s"** receberá os dados mas armazenará na variável apenas a informação até o espaço.

Exemplo: Digitado o nome **"Alexandre Domingues Moreno"** será armazenado o conteúdo **"Alexandre"**, o restante será ignorado.

Para resolver isso, utilize a formatação **"%[a-z A-Z]s"** ou substitua o comando **scanf** por **gets**.

```
scanf("%[a-z A-Z]s", &nome);  
fflush(stdin);
```

ou

```
gets(nome);
```

## Acentuação

Esta é a estrutura permite que o compilador interprete corretamente os caracteres de acentuação brasileira.

```
#include <locale.h>
int main () {
    setlocale(LC_ALL, "portuguese");
}
```

Linguagem C