

Estruturas de Repetição

Estruturas do while e for

ESTRUTURA DE REPETIÇÃO DO-WHILE



01

Garantia de Execução mínima: O laço **do-while** garante a execução do bloco de código pelo menos uma vez, mesmo que a condição de controle seja falsa desde o início.

02

Maior Clareza na Lógica de Execução: Possibilita ao código ser estruturado de forma clara e organizada, trazendo uma maior compreensão do seu fluxo de execução.

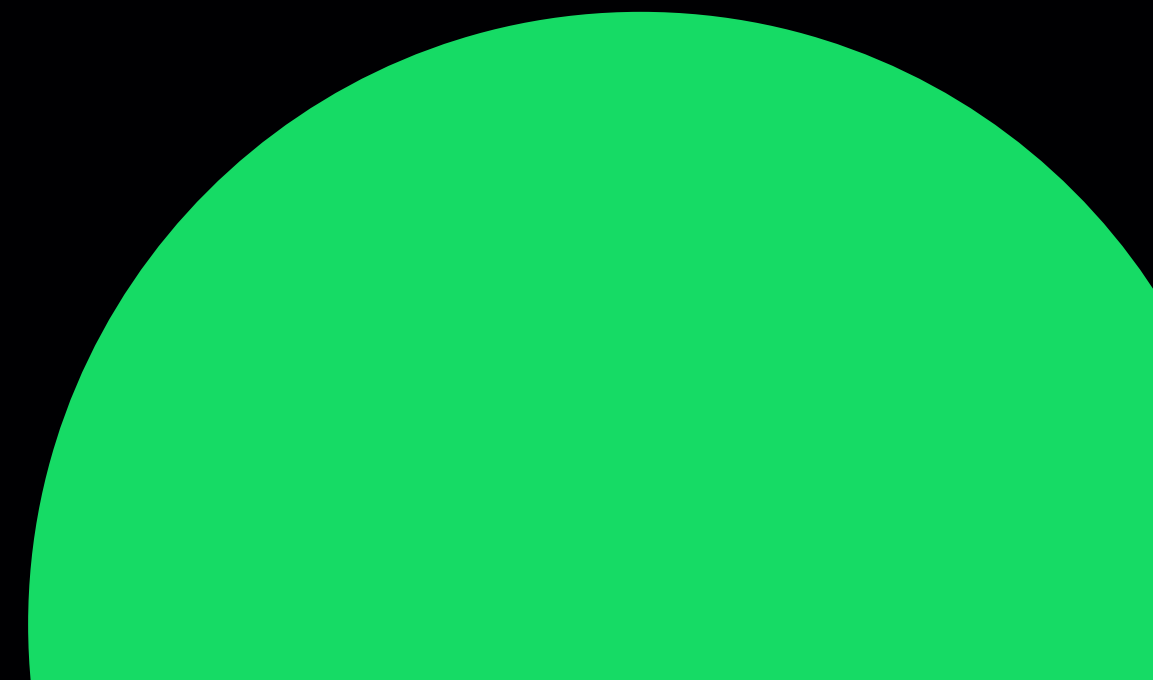
03

Facilidade de Uso em Menus e Interfaces de Usuário: É comum utilizar o **do-while** em menus e interfaces de usuário, onde é necessário solicitar uma entrada do usuário pelo menos uma vez antes de verificar a condição de saída.

04

Evita Repetição de Código: O do-while pode ajudar a evitar a repetição de código, pois o bloco de código é executado antes da verificação da condição. Isso pode simplificar o código e reduzir a redundância.

O loop do-while é uma ferramenta poderosa em programação, oferecendo garantia de execução mínima e clareza na lógica de execução. Ele é especialmente útil em situações onde é necessário realizar uma ação pelo menos uma vez antes de verificar uma condição de saída.



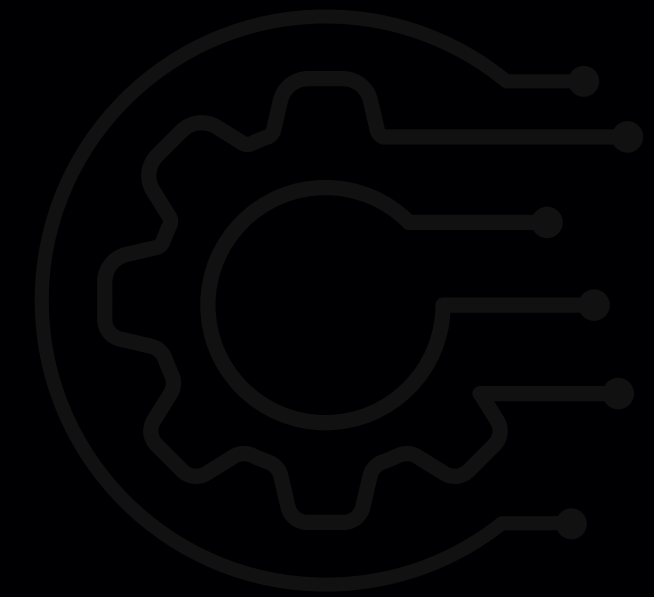
SINTAXE

A palavra-chave **do** inicia o bloco de código do loop e após a execução do bloco de código, a condição especificada após o **while** é verificada.

Como pode ser visto no código abaixo:

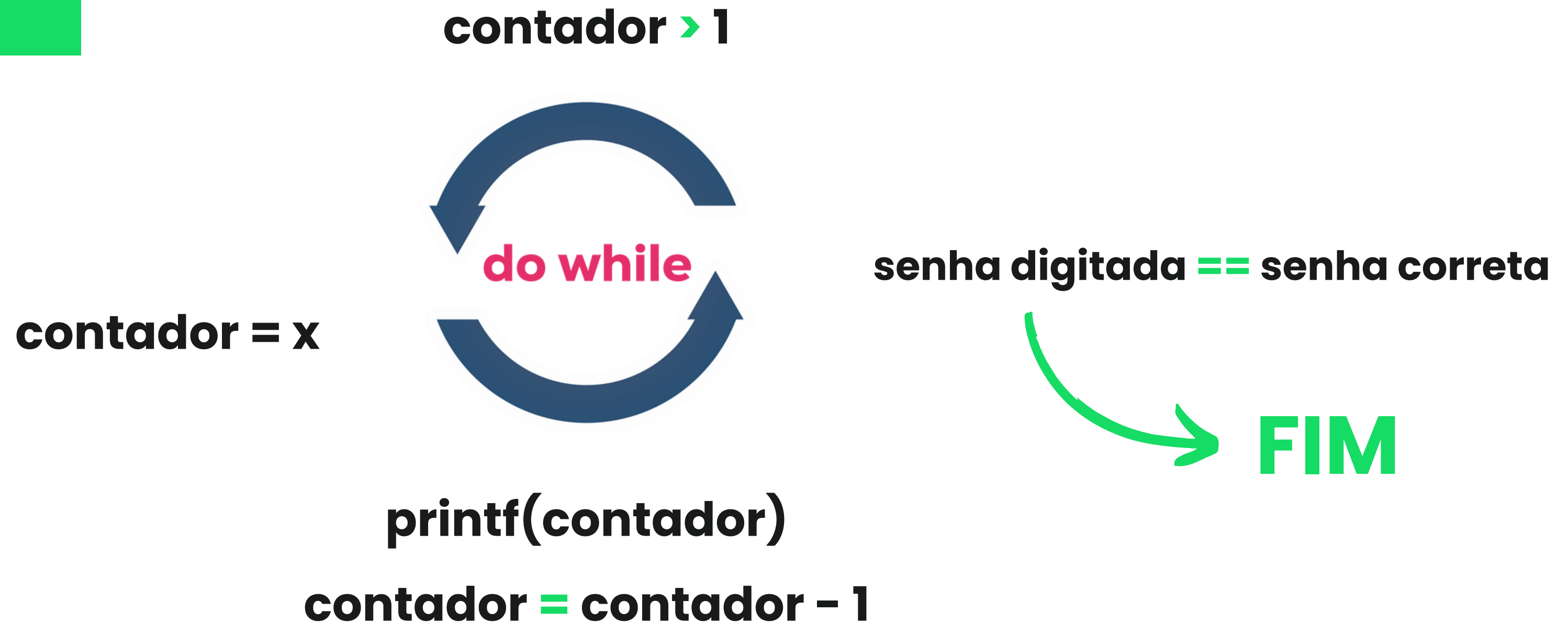
```
1  do {  
2      // Corpo do loop  
3      // Código a ser repetido pelo menos uma vez  
4  } while (condição);
```

EXEMPLOS



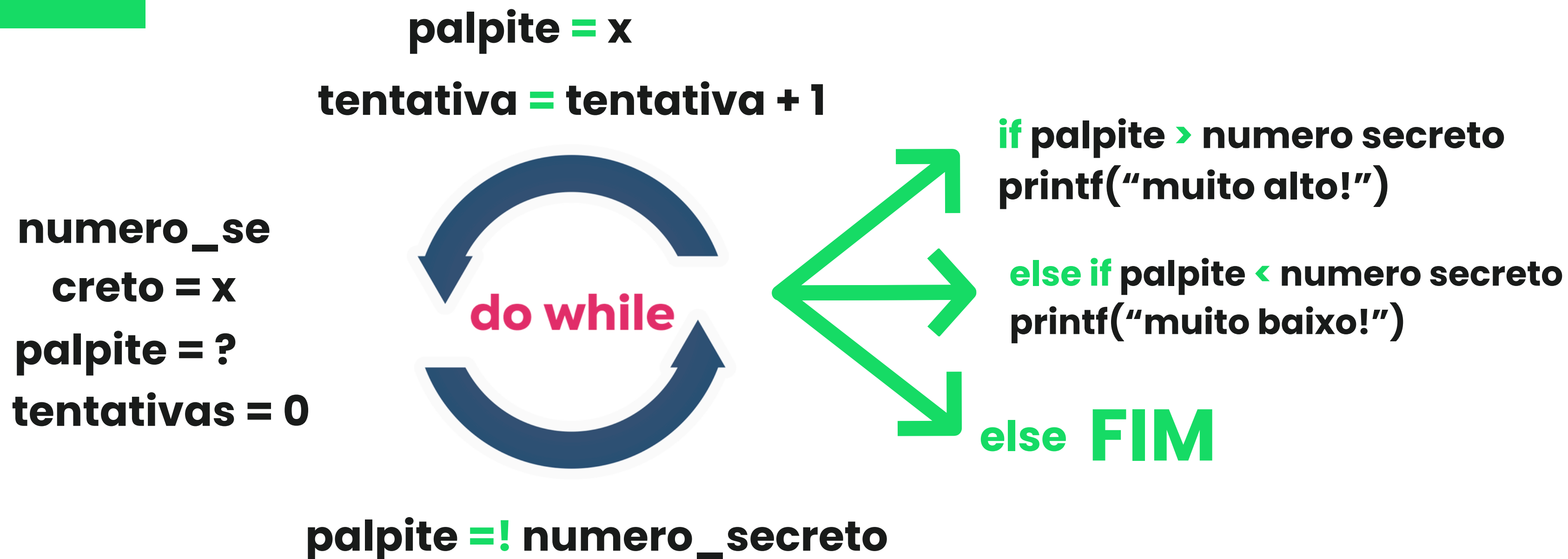
01

Contagem: (Conta de um valor até 1)



Contagem: (Conta de um valor até 1)

```
1  #include <stdio.h>
2
3  int main() {
4      int contador;
5
6      printf("Digite o valor inicial da contagem regressiva: ");
7      scanf("%d", &contador);
8
9      do {
10         printf("%d ", contador);
11         contador--;
12     } while (contador >= 1);
13
14     printf("\nFim da contagem regressiva.\n");
15
16     return 0;
17 }
18
```

02**Adivinhação: (Ajuda ao usuário adivinhar um numero de 1 a 100)**

Adivinhação: (Ajuda ao usuário adivinhar um numero de 1 a 100)

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  int main() {
6      int numero_secreto, palpite, tentativas = 0;
7
8      // Inicializa o gerador de números aleatórios
9      srand(time(NULL));
10
11     // Gera um número aleatório entre 1 e 100
12     numero_secreto = rand() % 100 + 1;
13
14     do {
15         printf("Digite seu palpite (1-100): ");
16         scanf("%d", &palpite);
17         tentativas++;
18
19         if (palpite > numero_secreto) {
20             printf("Muito alto! Tente novamente.\n");
21         } else if (palpite < numero_secreto) {
22             printf("Muito baixo! Tente novamente.\n");
23         } else {
24             printf("Parabens! Voce acertou o numero em %d tentativas.\n", tentativas);
25         }
26     } while (palpite != numero_secreto);
27
28     return 0;
29 }
```




ESTRUTURA DE REPETIÇÃO FOR

01

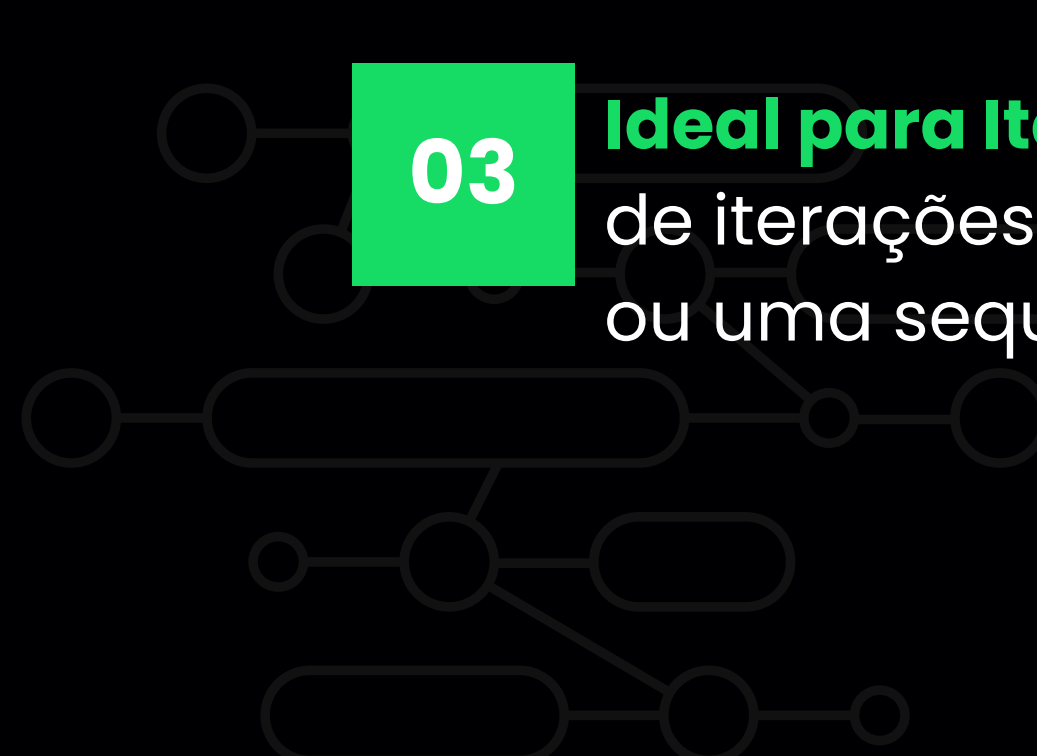
Estrutura Simplificada: O laço **for** oferece uma estrutura mais concisa e simplificada em comparação com o **while** e o **do-while**.

02

Controle Integrado de Iteração: O **for** permite especificar a inicialização, a condição de término e a atualização do contador de forma integrada na própria declaração do laço.

03

Ideal para Iterações Conhecidas: É especialmente útil quando o número de iterações é conhecido antecipadamente, como percorrer uma matriz ou uma sequência de elementos.



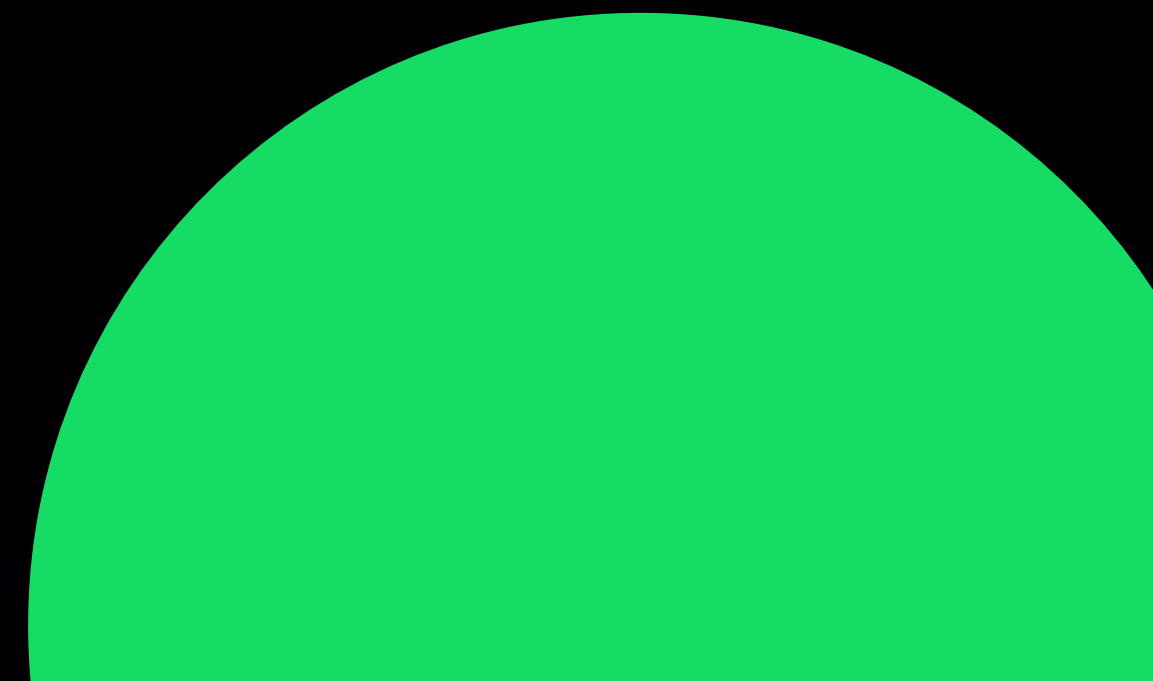
04

Legibilidade Aprimorada: O uso do **for** pode resultar em um código mais legível e de fácil compreensão, especialmente em iterações simples.

05

Flexibilidade na Atualização do Contador: O **for** permite uma atualização flexível do contador dentro do loop, o que pode simplificar a lógica de iteração em certos casos.

O laço **for oferece uma maneira eficiente e elegante de realizar iterações em situações onde o número de iterações é conhecido antecipadamente e quando uma estrutura de controle de loop integrada é desejada.**



SINTAXE

A palavra-chave **do** inicia o bloco de código do loop e após a execução do bloco de código, a condição especificada após o **while** é verificada.

Como pode ser visto no código abaixo:

```
1  for (inicialização; condição; atualização) {  
2      // Corpo do loop  
3      // Código a ser repetido  
4  }
```

EXEMPLOS



01

Contagem: (Conta um intervalo passado pelo usuário)

```
for (i = inicio; i <= fim; i++)
```

inicio = x

fim = y



```
printf(i)
```

Contagem: (Conta um intervalo passado pelo usuário)

```
1  #include <stdio.h>
2
3  int main() {
4      int inicio, fim;
5
6      printf("Digite o inicio do intervalo: ");
7      scanf("%d", &inicio);
8
9      printf("Digite o fim do intervalo: ");
10     scanf("%d", &fim);
11
12     printf("Numeros no intervalo [%d, %d]: ", inicio, fim);
13     for (int i = inicio; i <= fim; i++) {
14         printf("%d ", i);
15     }
16
17     return 0;
18 }
```


02

Soma de números pares: (No intervalo dado pelo usuário)

```
for (i = 1; i <= valor_limite; i++)
```

```
valor_limite = x
```

```
soma = 0
```



```
printf(soma)
```

```
if i % 2 == 0
```

```
soma = soma + i
```

Soma de números pares: (No intervalo dado pelo usuário)

```
1  #include <stdio.h>
2
3  int main() {
4      int valorLimite;
5      int soma = 0;
6
7      printf("Digite um valor limite para a soma dos numeros pares: ");
8      scanf("%d", &valorLimite);
9
10     for (int i = 1; i <= valorLimite; i++) {
11         if (i % 2 == 0) { // Verifica se o número é par
12             soma += i; // Soma o número par à variável soma
13         }
14     }
15
16     printf("A soma dos numeros pares de 1 a %d e: %d\n", valorLimite, soma);
17
18     return 0;
19 }
```

DESAFIO

Escreva um programa em C que solicite ao usuário que insira um número inteiro positivo e exiba os primeiros **n** termos da sequência de **Fibonacci**, onde **n** é o número inserido pelo usuário.

Dica: A série de Fibonacci consiste em que cada número é a soma de seus dois anteriores.

Exemplo: 1,1,2,3,5,8,13...

