in4mation HUB

Estruturas de Repetição

Estruturas do while e for

in4mation HUB

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.

in4mation HUB

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.

in4mation HUB

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:

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

in4mation HUB

EXEMPLOS

01 Contagem: (Conta de um valor até 1)

contador > 1

contador = x

senha digitada == senha correta

FIM

printf(contador)

contador = contador - 1

in4mation HUB

01

Contagem: (Conta de um valor até 1)

```c
#include <stdio.h>

int main() {
    int contador;

    printf("Digite o valor inicial da contagem regressiva: ");
    scanf("%d", &contador);

    do {
        printf("%d ", contador);
        contador--;
    } while (contador >= 1);

    printf("\nFim da contagem regressiva.\n");

    return 0;
}

```

in4mation HUB

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

palpite = x

tentativa = tentativa + 1

numero_se
creto = x

palpite = ?

tentativas = 0

if palpite › numero secreto
printf(“muito alto!”)

else if palpite ‹ numero secreto
printf(“muito baixo!”)

else FIM

palpite =! numero_secreto

in4mation HUB

02

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

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

int main() {
    int numero_secreto, palpite, tentativas = 0;

    // Inicializa o gerador de números aleatórios
    srand(time(NULL));

    // Gera um número aleatório entre 1 e 100
    numero_secreto = rand() % 100 + 1;

    do {
        printf("Digite seu palpite (1-100): ");
        scanf("%d", &palpite);
        tentativas++;

        if (palpite > numero_secreto) {
            printf("Muito alto! Tente novamente.\n");
        } else if (palpite < numero_secreto) {
            printf("Muito baixo! Tente novamente.\n");
        } else {
            printf("Parabens! Voce acertou o numero em %d tentativas.\n", tentativas);
        }
    } while (palpite != numero_secreto);

    return 0;
}
```

in4mation HUB

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.

in4mation HUB

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.

in4mation HUB

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  }
```

in4mation HUB

EXEMPLOS

01 Contagem: (Conta um intervalo passado pelo usuário)

inicio = x

fim = y

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


```
printf(i)
```

in4mation HUB

01

Contagem: (Conta um intervalo passado pelo usuário)

```c
#include <stdio.h>

int main() {
    int inicio, fim;

    printf("Digite o inicio do intervalo: ");
    scanf("%d", &inicio);

    printf("Digite o fim do intervalo: ");
    scanf("%d", &fim);

    printf("Numeros no intervalo [%d, %d]: ", inicio, fim);
    for (int i = inicio; i <= fim; i++) {
        printf("%d ", i);
    }

    return 0;
}
```

in4mation HUB

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
```

in4mation HUB

02

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

```c
#include <stdio.h>

int main() {
    int valorLimite;
    int soma = 0;

    printf("Digite um valor limite para a soma dos numeros pares: ");
    scanf("%d", &valorLimite);

    for (int i = 1; i <= valorLimite; i++) {
        if (i % 2 == 0) { // Verifica se o número é par
            soma += i; // Soma o número par à variável soma
        }
    }

    printf("A soma dos numeros pares de 1 a %d e: %d\n", valorLimite, soma);

    return 0;
}
```

in4mation HUB

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...