

# Fundamentos da Programação

Estruturas de repetição

# Conteúdo

- Estruturas de repetição:
  - `while`
  - `do... while`
  - `for`

## Estruturas de repetição

- É comum termos de **repetir** uma tarefa múltiplas vezes para atingir o objetivo pretendido.
  - Exemplo: ao confeccionar um bolo, bater claras até estas ficarem em castelo.
- A repetição é conseguida através de **estruturas de repetição** (ou ciclos) que executam um **conjunto de instruções**.
  - O **número de vezes** que o conjunto de instruções é executado é determinado pela **avaliação** de uma **condição** que devolve um valor lógico (Verdadeiro ou Falso).
  - Cada execução de um ciclo é chamada de **iteração**.

while

# while

- O **while** executa um bloco de instruções enquanto a condição for verdadeira.
  - A condição é avaliada antes da execução das instruções.

# while

```
while (condição) {  
    [instrução 1;  
    ...;  
    instrução n;]  
}  
instrução;
```

A condição é avaliada.

Se a condição for avaliada em verdadeiro, o bloco de instruções é executado (em sequência). No final, volta à avaliação da condição.

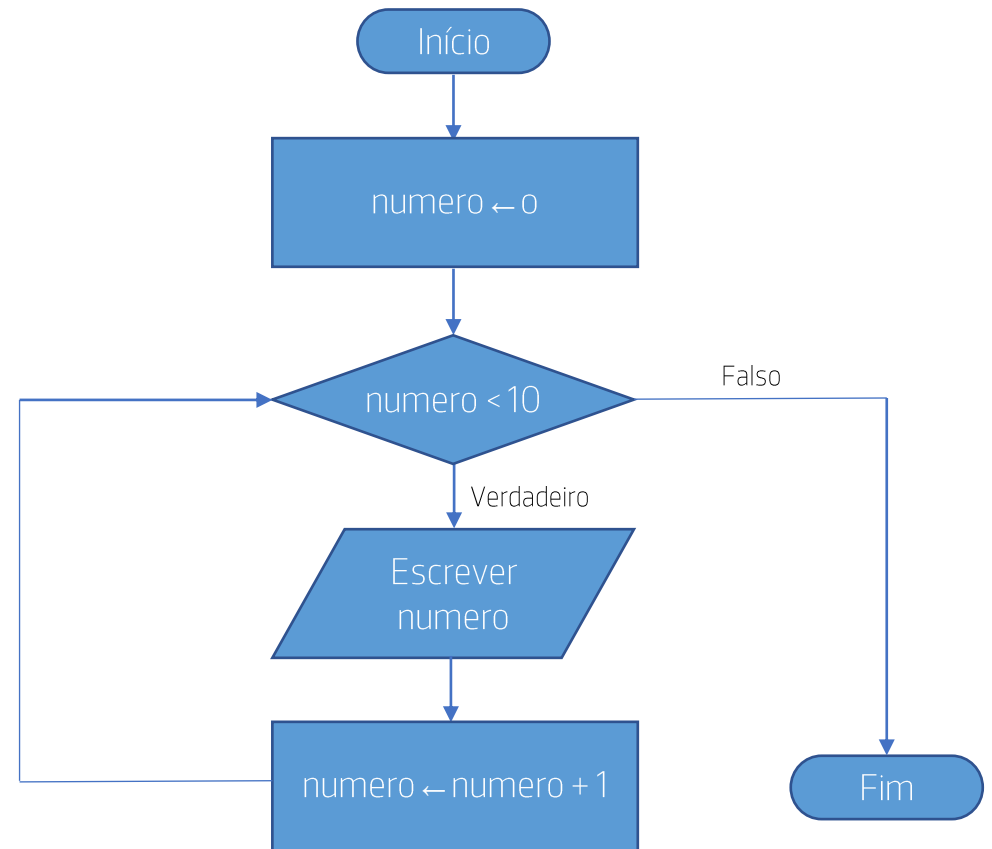
Se a condição for avaliada em falso, o ciclo termina e continua com a execução do programa.

```
...  
int limite = 1, contador = 0;  
while (limite < 1000) {  
    limite += limite;  
    printf("%d\n", limite);  
    contador++;  
}  
printf("Foram introduzidos %d numeros.", contador);  
...
```

enquanto... faz



```
início  
  numero ← 0  
  enquanto numero < 10  
    escrever numero  
    numero ← numero + 1  
  fimenquanto  
fim
```



do... while

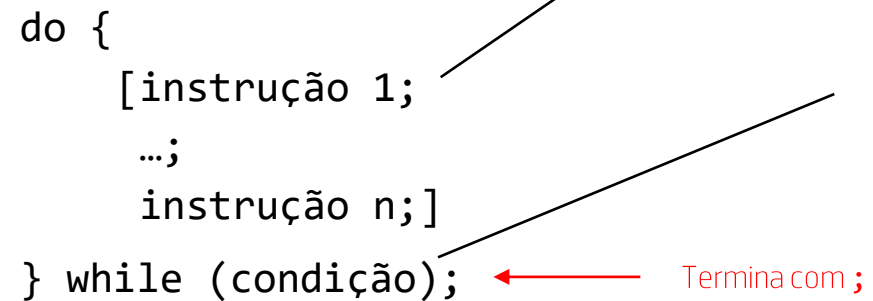


## do... while

- O **do... while** executa um bloco de instruções enquanto a condição for verdadeira.
  - A condição é avaliada depois da execução das instruções.

# do... while

```
do {  
    [instrução 1;  
    ...;  
    instrução n;]  
} while (condição);
```



As instruções são executadas.

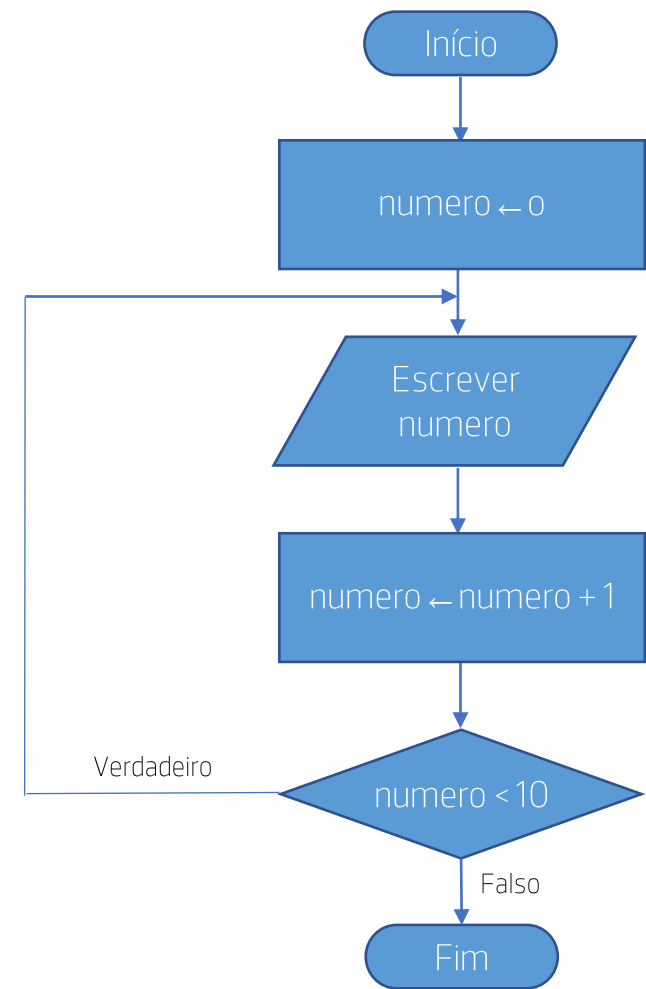
A condição lógica é avaliada. Se a condição for avaliada em verdadeiro, o bloco de instruções é executado novamente. Caso contrário, o cliço termina e continua com a execução do programa

```
...  
int numero = 0, soma = 0;  
do {  
    soma += numero;  
    scanf("%d", &numero);  
} while ( numero > 0 );  
...
```

faz... enquanto



```
início  
  numero ← 0  
  faz  
    escrever numero  
    numero ← numero + 1  
  enquanto numero < 10  
fim
```



for

# for

- O ciclo **for** deve ser utilizado quando o número de iterações a executar é conhecido.

# for

```
for (inicializações; condição; pós-instruções) {  
    [instrução 1;  
    ...;  
    instrução n;]  
}
```

```
...  
int numero;  
for (numero = 1; numero <= 10; numero++) {  
    printf("%d\n", numero);  
}  
...
```

# for

```
for (inicializações; condição; pós-instruções) {  
    [instrução 1;  
    ...;  
    instrução n;]  
}
```

É executado o código presente em inicializações.  
É onde são geralmente **inicializadas** as variáveis do ciclo.  
Estas instruções são executadas **uma** única vez.

```
...  
int numero;  
for (numero = 1; numero <= 10; numero++) {  
    printf("%d\n", numero);  
}  
...
```

# for

```
for (inicializações; condição; pós-instruções) {  
    [instrução 1;  
    ...;  
    instrução n;]  
}
```

A condição é avaliada. Se o resultado da avaliação for **falso**, o ciclo termina.

```
...  
int numero;  
for (numero = 1; numero <= 10; numero++) {  
    printf("%d\n", numero);  
}  
...
```



# for

```
for (inicializações; condição; pós-instruções) {  
    [instrução 1;  
    ...;  
    instrução n;]  
}
```

A condição é avaliada. Se o resultado da avaliação for **verdadeiro**, são executadas as ações associadas ao ciclo.

```
...  
int numero;  
for (numero = 1; numero <= 10; numero++) {  
    printf("%d\n", numero);  
}  
...
```

# for

```
for (inicializações; condição; pós-instruções) {  
    [instrução 1;  
    ...;  
    instrução n;]  
}
```

São depois executadas as pós-instruções (geralmente de incremento ou decremento).

```
...  
int numero;  
for (numero = 1; numero <= 10; numero++) {  
    printf("%d\n", numero);  
}  
...
```

# for

```
for (inicializações; condição; pós-instruções) {  
    [instrução 1;  
    ...;  
    instrução n;]  
}
```

Volta à avaliação da condição lógica.

```
...  
int numero;  
for (numero = 1; numero <= 10; numero++) {  
    printf("%d\n", numero);  
}  
...
```

# for

```
for (inicializações; condição; pós-instruções) {  
    [instrução 1;  
    ...;  
    instrução n;]  
}
```

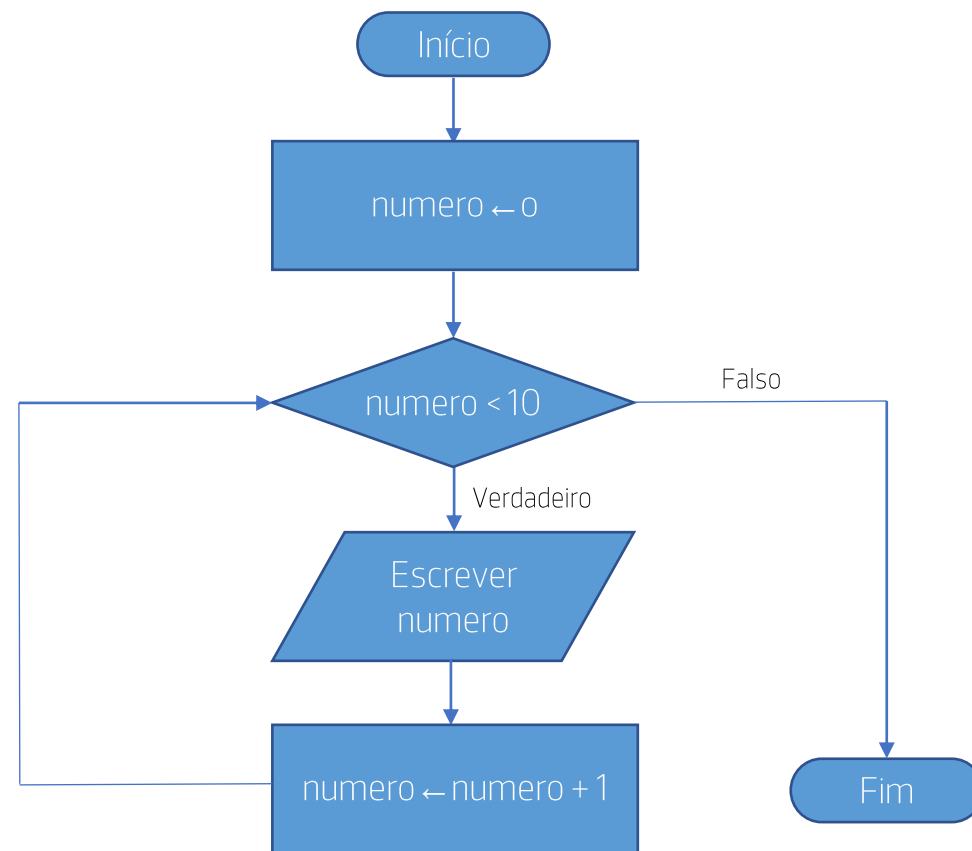
As inicializações e pós-instruções são separadas por vírgulas.

```
...  
int i, j;  
for (i = 20, j = 0; i > j; i--, j++) {  
    printf("\n%d %d", i, j);  
}  
...
```

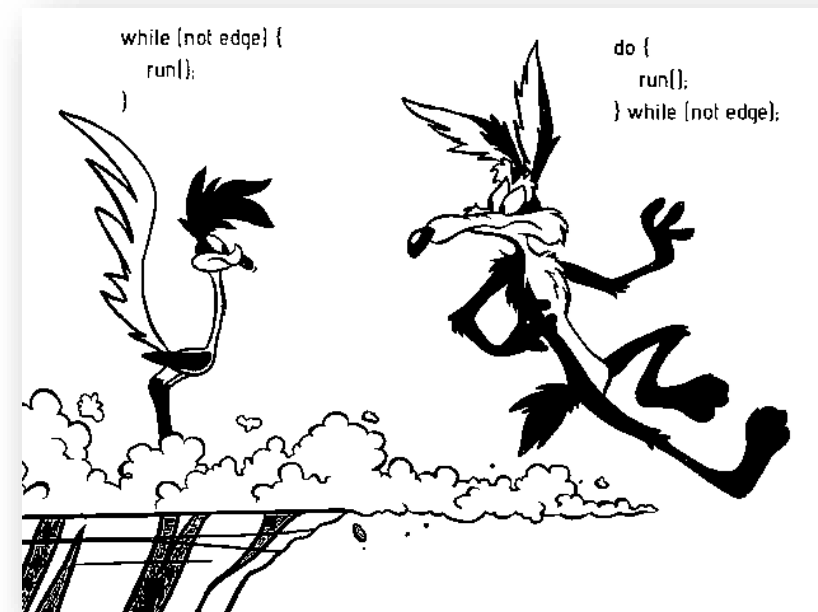
Para



início  
para numero de 0 até 9 passo 1  
escrever numero  
próximo  
fim



## Estruturas de repetição



## Ciclos resumo

	<b>while</b>	<b>do... while</b>	<b>for</b>
Executa a instrução	0 ou + vezes	1 ou + vezes	0 ou + vezes
Testa a condição	Antes da execução	Depois da execução	Antes da execução
Utilização	Frequente	Pouco frequente	Frequente

## Estruturas aninhadas

- As instruções dentro de um ciclo podem ser outras estruturas.

```
...  
int num1 = 1, num2;  
while (num1 <= 5) {  
    for (num2 = 1; num2 <= num1; num2++) {  
        printf("%d", num2);  
    }  
    printf("\n");  
    ++num1;  
}  
...
```



## Ciclo infinito

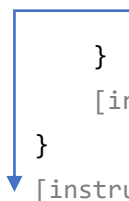
- Ciclo que nunca termina.
  - Usualmente, resultado de erros de programação.

```
...  
int numero = 1;  
while (numero <= 10) {  
    printf("%d", numero);  
    numero--;  
}  
...
```

# break

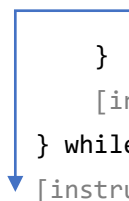
- O **break** termina o ciclo imediatamente quando é encontrado.

```
while (condição) {  
    [instruções]  
    if (condição) {  
        break;  
    }  
    [instruções]  
}
```



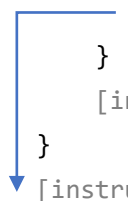
[instruções]

```
do {  
    [instruções]  
    if (condição) {  
        break;  
    }  
    [instruções]  
} while (condição);
```



[instruções]

```
for (inicializações; condição; pós-instruções) {  
    [instruções]  
    if (condição) {  
        break;  
    }  
    [instruções]  
}
```

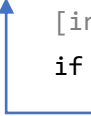


[instruções]

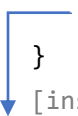
# continue

- O **continue** ignora o resto da iteração corrente e continua com a próxima iteração.

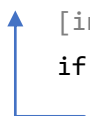
```
while (condição) {  
    [instruções]  
    if (condição) {  
        continue;  
    }  
    [instruções]  
}  
[instruções]
```



```
do {  
    [instruções]  
    if (condição) {  
        continue;  
    }  
    [instruções]  
} while (condição);  
[instruções]
```



```
for (inicializações; condição; pós-instruções) {  
    [instruções]  
    if (condição) {  
        continue;  
    }  
    [instruções]  
}  
[instruções]
```



# for

- No **for** as expressões (inicializações, condição e pós-instruções) são opcionais.

```
...  
i = 0;  
for (; i < 4; i++) {  
    printf("%d", i);  
}  
...
```

```
...  
for (i = 0;; i++) {  
    if (i > 3) {  
        break;  
    }  
    printf("%d", i);  
}  
...
```

```
...  
for (i = 0; i < 4;) {  
    printf("%d", i++);  
}  
...
```

```
...  
i = 0;  
for (;;) {  
    if (i > 3) {  
        break;  
    }  
    printf("%d", i++);  
}  
...
```

# Traçagem

- Consiste em analisar um programa (algoritmo, ...) com determinados valores de entrada, observando o seu comportamento.

# Traçagem

		input	valor	contador	Output
1	int valor, contador = 0;			0	
2	scanf("%d", &valor);	1	1		
3	while (valor < 30)				
4	valor += valor;		2		
5	++contador;			1	
6	while (valor < 30)				
7	valor += valor;		4		
8	++contador;			2	
9	while (valor < 30)				
10	valor += valor;		8		
11	++contador;			3	
12	while (valor < 30)				
13	valor += valor;		16		
14	++contador;			4	
15	while (valor < 30)				
16	valor += valor;		32		
17	++contador;			5	
18	while (valor < 30)				
19	printf("Núm. introduzidos: %d", contador);				Núm. introduzidos: 5

```
...
int valor, contador = 0;
scanf("%d", &valor);
while (valor < 30) {
    valor += valor;
    ++contador;
}
printf("Núm. introduzidos: %d", contador);
...
```

## Leitura recomendada

- (Capítulo 4) Damas, L. Linguagem C; FCA – Editora de Informática, Lda, 1999; ISBN 9789727221561.

