



# **Comando for**

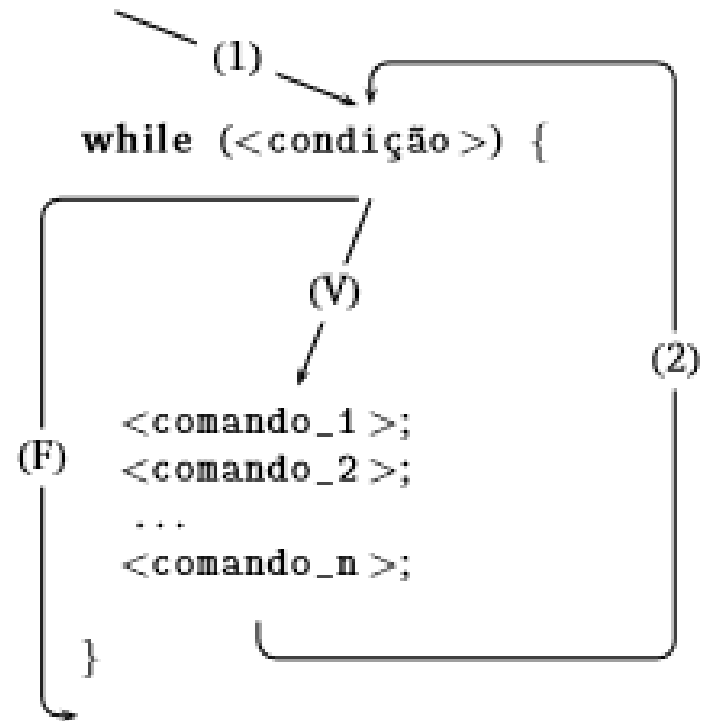
**Prof. Lilian Berton**

**São José dos Campos, 2019**

Baseado no material de Ronaldo F. Hashimoto e Carlos H. Morimoto – IME/USP

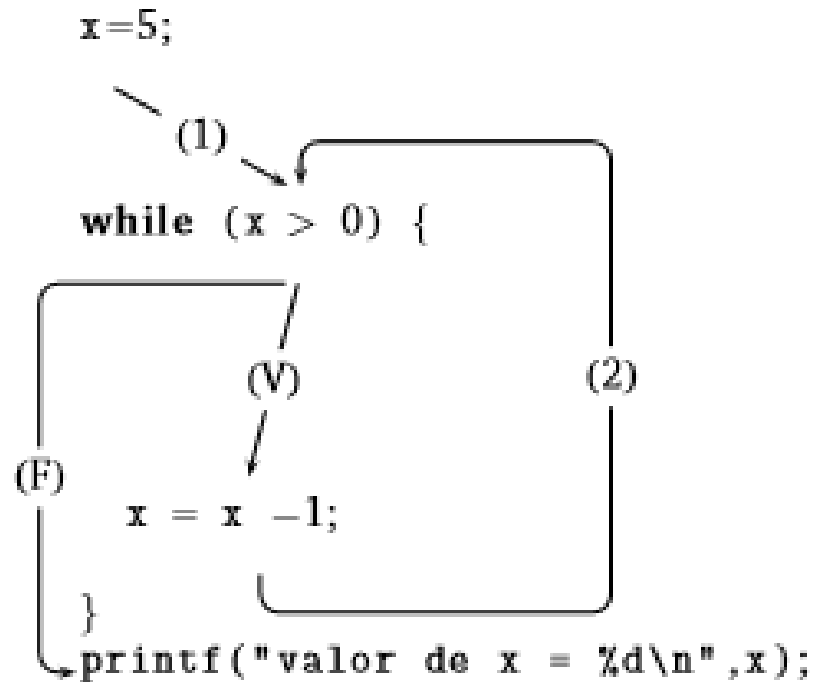
# Comando while

- A **<condição>** é uma expressão relacional que tem como resultado um valor verdadeiro ou falso.
- Se a **<condição>** é **verdadeira**, então o fluxo do programa segue a seta marcada com (V) repetindo a sequência de comandos dentro do while.
- Se **<condição>** é **falsa**, o fluxo do programa ignora a sequência de comandos e segue a seta marcada com (F).



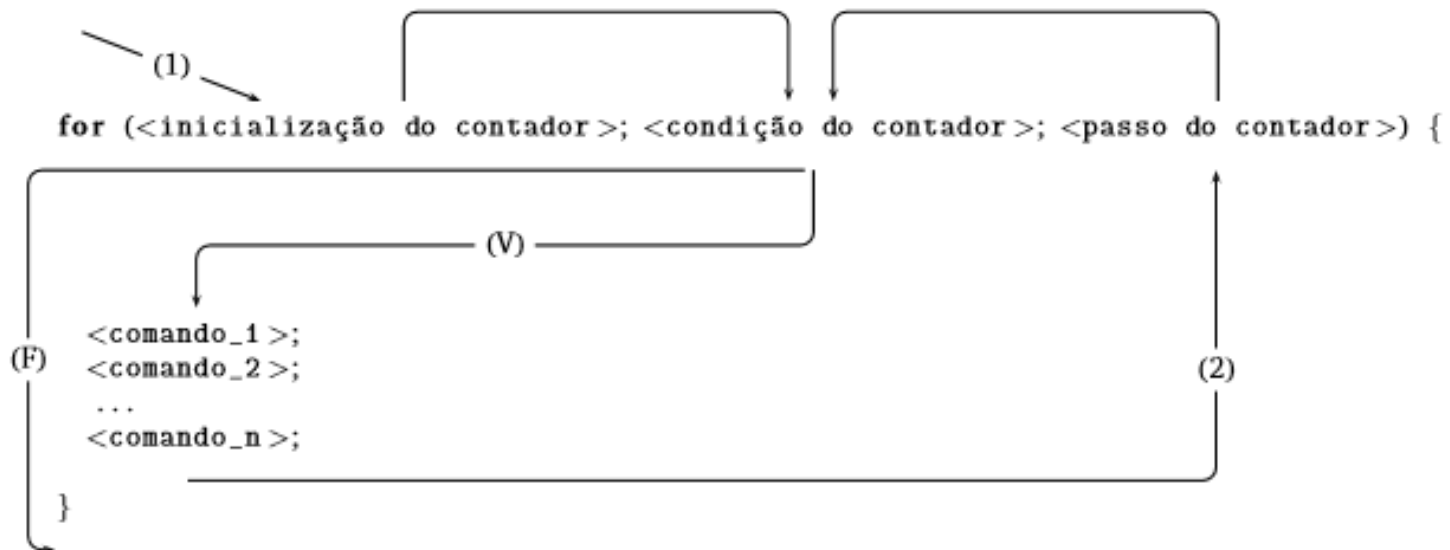
# Exemplo 1 comando while

- Por exemplo, seja x uma variável inteira. O segmento de programa abaixo simplesmente subtrai 1 de x, 5 vezes.



# Comando for

- Primeiramente, a execução do programa vem e faz a **inicialização do contador** (seta marcada com (1)).
- Depois a **<condição do contador> do for é testada**. Se “de cara” a <condição do contador> é falsa, o fluxo do programa ignora a sequência de comandos dentro do for e segue a seta marcada com (F).
- Agora, se a <condição do contador> é verdadeira, então o fluxo do programa segue a seta marcada com (V) e executa a sequência de comandos dentro do for; executado o último comando (<comando\_n>), o fluxo do programa segue a seta marcada com (2) e **executa <passo do contador> (aumenta/diminui o contador de passo) e volta a testar a <condição do contador>**.

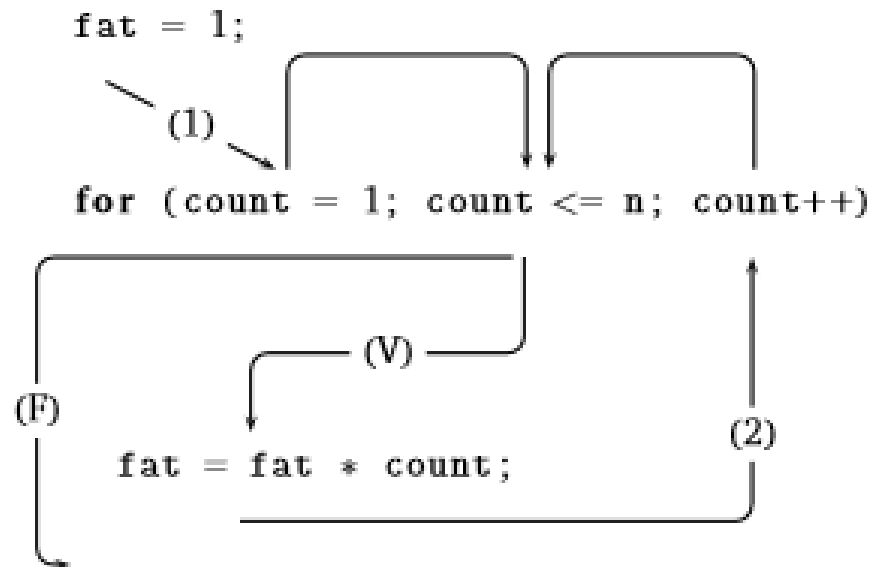


# Exemplo 1 comando for

- Dado um número inteiro  $n \geq 0$ , calcular  $n!$ . Precisamos gerar a sequência de números  $1, 2, \dots, n$ . E acumular a multiplicação para cada número gerado.

$$4! = 1 * 2 * 3 * 4 = 24$$

count	fat
1	$1 * 1 = 1$
2	$1 * 2 = 2$
3	$2 * 3 = 6$
4	$6 * 4 = 24$



# Exemplo 2 comando for

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

```
int main(void) {  
    int contador; //variável de controle do loop  
    for(contador = 1; contador <= 5; contador++) {  
        printf("%d ", contador);  
    }  
  
    return(0);  
}
```

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

```
int main(void) {  
    int contador; //variável de controle do loop  
    for(contador = 5; contador >= 1; contador--) {  
        printf("%d ", contador);  
    }  
  
    return(0);  
}
```

# Exemplo 2 comando for

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

```
int main(void) {  
    int contador; //variável de controle do loop  
    for(contador = 1; contador <= 5; contador++) {  
        printf("%d ", contador);  
    }  
  
    return(0);  
}
```



1 2 3 4 5

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

```
int main(void) {  
    int contador; //variável de controle do loop  
    for(contador = 5; contador >= 1; contador--) {  
        printf("%d ", contador);  
    }  
  
    return(0);  
}
```



5 4 3 2 1

# Repetições aninhadas

- Repetições aninhadas são **repetições com outras repetições dentro**.
- A execução **inicia pelo laço de fora** (mais externo), depois **desvia para o laço de dentro** e só volta para o laço de fora quando terminar toda execução do laço de dentro (quando o contador chegar no final).
- **Um laço fica “parado” enquanto o outro laço é executado**, ou seja, enquanto seu contador varia até chegar no valor final.

```
1 for (<inicialização do contador>; <condição do contador>; <passo do contador>) {  
2     for (<inicialização do contador>; <condição do contador>; <passo do contador>) {  
3         <comando1>;  
4         <comando2>;  
5         ...  
6     }  
7 }
```

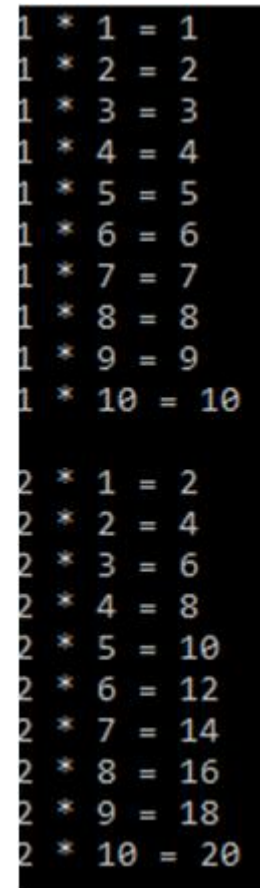
```
1 while (<condição 1>) {  
2     while (<condição 2>) {  
3         while (<condição 3>) {  
4             }  
5         }  
6     }  
7 }
```



# Repetições aninhadas – exemplo 1

- Faça um programa que imprima as tabuadas do 1 ao 10:

```
#include <stdio.h>
main(void){
    int n, i, j;
    for(i = 1; i <= 10; i++) {
        for(j = 1; j <= 10; j++) {
            if (j == 10){
                printf("%d * %d = %d\n\n",i, j, i*j)
            } else{
                printf("%d * %d = %d\n",i, j, i*j);
            }
        }
    }
}
```



```
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
1 * 6 = 6
1 * 7 = 7
1 * 8 = 8
1 * 9 = 9
1 * 10 = 10

2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
2 * 10 = 20

...
```

# Repetições aninhadas – exemplo 2

- Dados dois naturais  $n > 0$  e  $m > 0$ , determinar entre todos os pares de números inteiros  $(x, y)$  tais que  $0 \leq x \leq n$  e  $0 \leq y \leq m$ , um par para o qual **o valor da expressão  $x*y - x^2 + y$  seja máximo** e calcular também este máximo.
- Em caso de empate, imprima somente um valor. Exemplo: para  $n=2$  e  $m=3$ , a saída do programa deve ser  $(1, 3)$  com valor máximo igual a 5.

# Repetição aninhada – exemplo 2

- Tomando o exemplo  $n=2$  e  $m=3$ . Para cada  $x$  fixo, temos que gerar os valores para  $y$  de 0 a  $m=3$ . Assim, temos:
- $x=0$   
**(0,0) (0,1) (0,2) (0,3)**  $\Rightarrow$  Para  $x=0$ , um laço para gerar os valores de  $y$  de 0 a 3.
- $x=1$   
**(1,0) (1,1) (1,2) (1,3)**  $\Rightarrow$  Para  $x=1$ , um laço para gerar os valores de  $y$  de 0 a 3.
- $x=2$   
**(2,0) (2,1) (2,2) (2,3)**  $\Rightarrow$  Para  $x=2$ , um laço para gerar os valores de  $y$  de 0 a 3.
- Assim, para cada  $x$  fixo, temos uma repetição para gerar os valores para  $y$  de 0 a 3.

# Repetição aninhada – exemplo 2

```
1      x = 0;
2      while (x <= n) {
3          /* gera um valor para x */
4          /* para um valor fixo de x */
5          y = 0;
6          while (y <= m) {
7              printf ("%d, %d\n", x, y);
8              y = y + 1;
9          }
10         x = x + 1;
11     }
```

- Para cada par (x,y) gerado pelo código anterior, calcular o valor da expressão  $x*y - x^2 + y$ .
- Assim, basta ter uma variável max que, para cada par gerado, guarda o valor máximo até o presente momento, de forma que:
- x=0
- (0,0)⇒ 0, max=0    (0,1)⇒ 1, max=1    (0,2)⇒ 2, max=2    (0,3)⇒ 3, max=3
- x=1
- (1,0)⇒ -1, max=3    (1,1)⇒ 1, max=3    (1,2)⇒ 3, max=3    (1,3)⇒ 5, max=5
- x=2
- (2,0)⇒ -4, max=5    (2,1)⇒ -1, max=5    (2,2)⇒ 2, max=5    (2,3)⇒ 5, max=5

# Repetição aninhada – exemplo 2

```
1  # include <stdio.h>
2
3  int main () {
4      int x, y, n, m, v, x_max, y_max;
5
6      printf ("Entre com n>0: ");
7      scanf ("%d", &n);
8
9      printf ("Entre com m>0: ");
10     scanf ("%d", &m);
11
12     x = 0; max = x_max = y_max = 0;
13     while (x <= n) {
14         /* gera um valor para x */
15         y = 0;
16         while (y <= m) {
17             v = x*y - x*x + y;
18             if (v > max) {
19                 max = v;
20                 x_max = x;
21                 y_max = y;
22             }
23             y = y + 1;
24         }
25         x = x + 1;
26     }
27     printf ("O valor maximo = %d em x = %d e y = %d\n", max, x_max, y_max);
28     return 0;
29 }
```

n	m	x	y	v	y_max	x_max	max
2	3	0	0	0	0	0	0
			1	1	0	1	1
			2	2	0	2	2
			3	3	0	3	3
		1	0	-1	0	3	3
			1	1	0	3	3
			2	3	0	3	3
			3	5	1	3	5
		2	0	-4	1	3	5
			1	-1	1	3	5
			2	2	1	3	5
			3	5	1	3	5

# Comando break

- O comando **break** é um modo conveniente de **terminar imediatamente a execução** de um bloco controlado por uma estrutura de repetição, sem necessidade de esperar a próxima avaliação da condição.
- Assim, o programa pode economizar algum tempo de execução.
- No próximo exemplo, que verifica se um número é primo, o comando break será usado para interromper as repetições assim que mais de dois divisores forem encontrados.

# Exemplo

```
#include <stdio.h>
int main( ) {
    int numero;
    int divisor;
    int resto;
    int numero_divisores = 0;
    printf("Digite o numero: ");
    scanf("%d", &numero);
    for (divisor = 1; divisor <= numero; divisor++) {
        resto = numero % divisor;
        if (resto == 0) {
            numero_divisores = numero_divisores + 1;
            if (numero_divisores >= 3) {
                break;
            }
        }
    }
    if (numero_divisores == 2) {
        printf("O numero %d eh primo!\n", numero);
    } else {
        printf("O numero %d NAO eh primo!\n", numero);
    }
    return 0;
}
```

# Comando continue

- O comando **continue** **reinicia imediatamente a execução de um bloco de uma estrutura de repetição**. O continue não espera o término da execução do restante do bloco.
- No caso do while, a execução retorna imediatamente para avaliar a expressão, antes de executar novamente o bloco, se for o caso. Se a expressão avaliar como falso, então o while é finalizado, caso contrário ele realiza uma nova iteração.
- Para o for, o continue interrompe a execução normal do bloco, realiza imediatamente a atualização das variáveis de controle para, em seguida, realizar novamente o teste. Se o teste resultar em falso, então o for é finalizado, caso contrário ele realiza uma nova iteração



# Exemplo

- Um programa que imprime uma tabela com a imagem da função tangente, em intervalos de 10 em 10 graus.
- O for executa o bloco para diferentes valores de ângulos, em passos de 10 em 10 graus. No entanto, ao chegar ao valor de 90 graus, a função tangente não está definida! Por este motivo, este valor precisa ser ignorado. Para tal, utiliza-se o comando continue para reiniciar o for com o próximo valor (ou seja, 100 graus).

```
#include <stdio.h>
#include <math.h>
int main() {
    double angulo;
    double pi = 3.14159265358979;
    for (angulo = 0; angulo <= 180; angulo += 10.0) {
        if (angulo == 90.0) {
            continue;
        }
        printf("tan(%f) = %f\n", angulo, tan(angulo/180*pi));
    }
    return 0;
}
```

# Exercícios

Assista a **Aula 11 (estruturas de repetição – parte III)**:

<https://www.cursoemvideo.com/course/curso-de-algoritmos/>



HOME / CURSO / ALGORITMO / CURSO DE ALGORITMO

## Curso de Algoritmo

120022 ALUNOS

Hoje em dia, algoritmos computacionais estão presentes em quase tudo na nossa vida. Além dos tradicionais computadores e notebooks, muitos estão totalmente acostumados com o uso de aplicativos para smartphones e tablets, TVs inteligentes podem executar programas personalizados e até mesmo outros aparelhos que usamos no nosso dia-a-dia.

O Curso de Algoritmo é a base necessária para quem quer aprender em linguagens famosas do mercado, como C, Java, PHP e muitas outras. Inscreva-se no curso agora mesmo e aprenda as técnicas básicas para a construção de programas para dispositivos eletrônicos.

### EMENTA DO CURSO



Curso de Algoritmos – 01 –  
Introdução a Algoritmos

GRÁTIS



00:14:00

# Exercícios

1. Dados números inteiros  $n$  e  $k$ , com  $k \geq 0$ , determinar  $n^k$  ( $n$  elevado a  $k$ ). Por exemplo, dados os números 3 e 4 o seu programa deve escrever o número 81.

**Obs: não pode usar a função pow e deve ser feito com o comando for. Mostre o teste de mesa!**

2. Faça um programa que calcula o  $n$ -ésimo termo da sequência de Fibonacci, sendo  $n$  informado pelo usuário. **Use o comando for e mostre o teste de mesa!**

n	0	1	2	3	4	5	6	7	8	9	10	11	...
F(n)	0	1	1	2	3	5	8	13	21	34	55	89	...



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



```
for(i=0;i<n;i++)
```