

# Comandos de Repetição

Prof. Bruno Travençolo

# Estruturas de Repetição

---

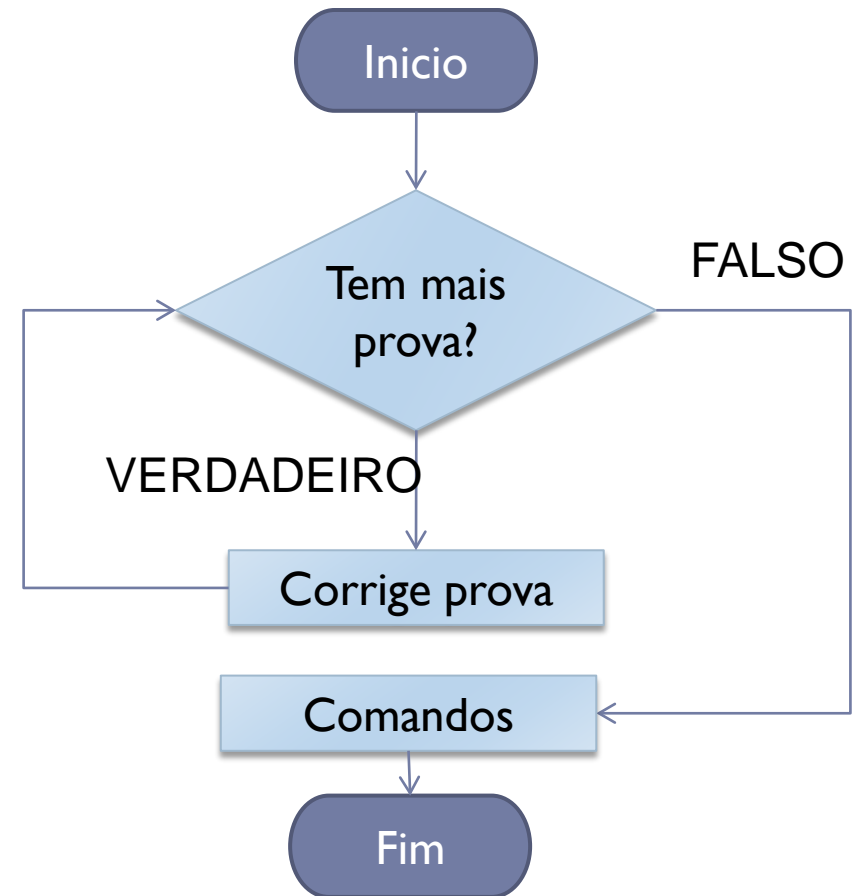
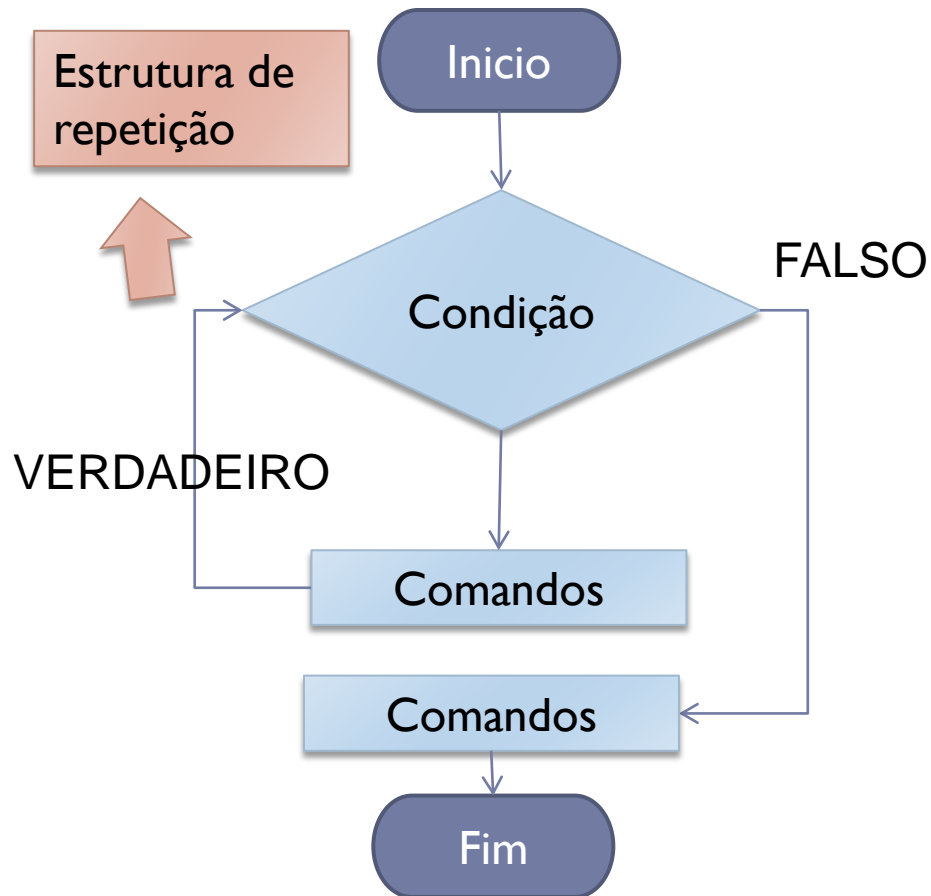
- ▶ **Repetição com Teste no Início**
- ▶ Repetição com Teste no Final
- ▶ Repetição Contada

Uma **estrutura de repetição** permite que uma sequência de comandos seja executada **repetidamente, enquanto** determinadas condições são satisfeitas. Essas condições são representadas por expressões lógicas (como, por exemplo,  $A > B$ ;  $C == 3$ ;  $\text{Letra} == \text{'a'}$ )



# Fluxograma

## Repetição com Teste no Início



# Repetição

---

O real poder dos computadores está na sua habilidade para repetir uma operação ou uma serie de operações muitas vezes.

Esta repetição chamada **laços** (*loop*) é um dos conceitos básicos da programação estruturada.



# Repetição por Condição

---

- ▶ Um conjunto de comandos de um **algoritmo** pode ser repetido quando subordinado a uma condição:

**enquanto** *condição* **faça**

comandos;

**fim enquanto**

- ▶ De acordo com a condição, os comandos serão repetidos zero (se falso) ou mais vezes (enquanto a condição for verdadeira).



# Repetição por Condição

---

## ► Condição

- qualquer expressão que resulte em um valor do tipo lógico e pode envolver operadores aritméticos, lógicos, relacionais e resultados de funções.

### ► Ex:

$x > 5$

$(N < 60) \ \&\& \ (N > 35)$



# Funcionamento

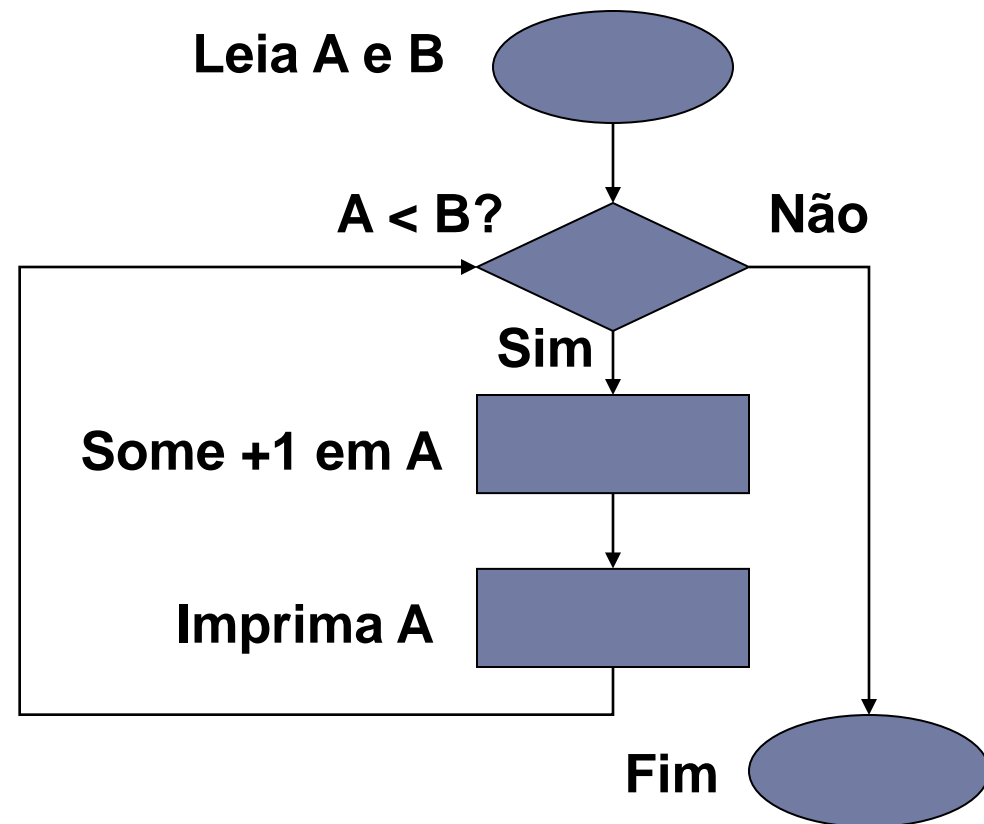
---

- ▶ A condição da cláusula **enquanto** é testada.
  - ▶ Se ela for verdadeira os comandos seguintes são executados em seqüência como em qualquer algoritmo, até a cláusula **fim enquanto**.
  - ▶ O fluxo nesse ponto é desviado de volta para a cláusula **enquanto** e o processo se repete.
  - ▶ Se a condição for falsa (ou quando finalmente for), o fluxo do algoritmo é desviado para o primeiro comando após a cláusula **fim enquanto**.



# Repetição por Condição

- ▶ Relembrando em fluxogramas
  - ▶ Um processo pode ser repetido até atender ou não uma condição.





# Exemplo – Pseudo-Código

---

Leia A;

Leia B;

Enquanto  $A < B$

$A \leftarrow A + 1$ ;

    Imprima A;

Fim Enquanto



# Loop Infinito

---

- ▶ Um loop ou laço infinito ocorre quando cometemos algum erro
  - ▶ ao especificar a condição lógica que controla a repetição
  - ▶ ou por esquecer de algum comando dentro da iteração.



# Loop Infinito

---

## Exemplo: loop infinito (condição errônea)

```
01  X recebe 4;  
02  enquanto (X < 5) faca  
03      X recebe X - 1;  
04      Imprima X;  
05  fim enquanto
```

## Exemplo: loop infinito (não muda valor)

```
01  X recebe 4;  
02  enquanto (X < 5) faca  
03      Imprima X;  
04  fim enquanto
```



# Comando while

---

- ▶ Equivale ao comando “enquanto” utilizado nos pseudo-códigos.
  - ▶ Repete a seqüência de comandos enquanto a condição for verdadeira.
- ▶ Esse comando possui a seguinte forma geral:

```
while (expressão)  
    instruções
```

(em inglês)

```
while (expression)  
    statement
```



# Observação sobre a sintaxe

---

*while (expression)*  
*statement*

- ▶ O que um *statement* (instrução)?
  - ▶ É uma única instrução da linguagem
    - ▶ Um *statement* termina com um sinal de ponto e vírgula ;  
`printf("Hello World!");`
  - ▶ Ou é um conjunto de instruções delimitada por chaves, o que é chamado de Bloco de Instruções
    - ▶ Block Delimiter: `{ }`
    - ▶ Dentro de um bloco podemos colocar mais de uma instrução  
`{`  
`printf("Hello World!");`  
`printf("Hello World Again!");`  
`}`



# Exemplo

- ▶ Faça um programa que mostra na tela os número de 1 a 100
- ▶ Saída exemplo:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62
63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82
83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
Process returned 0 (0x0)   execution time : 0.343 s
Press any key to continue.
```

```
int main()
{
    // programa que mostra na tela números de 1 ate 100
    printf(" 1  2  3  4  .... ");
    return 0;
}
```

# Exemplo

- ▶ Faça um programa que mostra na tela os número de 1 a 100
- ▶ Saída exemplo:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62
63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82
83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
Process returned 0 (0x0)   execution time : 0.343 s
Press any key to continue.
```

```
int main()
```

```
{
```

```
// programa que mostra na tela números de 1 ate 100
```

```
printf(" 1 2 3 4 ..... ");
```

```
return 0;
```

```
}
```

INVIÁVEL

# Exemplo

---

- Faça um programa que mostra na tela os número de 1 a 100

```
int main()
{
    // programa que mostra na tela números de 1 ate 100
    int numero;

    numero = 1;
    while (numero <= 100){
        printf(" %d ", numero);
        numero = numero + 1;
    }

    return 0;
}
```





# Exemplo

---

- Faça um programa que mostra na tela os número de 1 a 100

*while (expression)  
statement*

```
int main()
{
    // programa que mostra na tela números de 1 ate 100
    int numero;

    numero = 1;
    while (numero <= 100){
        printf(" %d ", numero);
        numero = numero + 1;
    }

    return 0;
}
```



# Exemplo

---

- Faça um programa que mostra na tela os número de 1 a 100

*while (expression)  
statement*

```
int main()
{
    // programa que mostra na tela números de 1 ate 100
    int numero;

    numero = 1;
    while (numero <= 100){
        printf(" %d ", numero);
        numero = numero + 1;
    }

    return 0;
}
```



# Exemplo

---

- ▶ Faça um programa que mostra na tela os número de 1 a 100

*while (expression)  
statement*

```
int main()
{
    // programa que mostra na tela números de 1 ate 100
    int numero;

    numero = 1;
    while (numero <= 100){
        printf(" %d ", numero);
        numero = numero + 1;
    }

    return 0;
}
```



# Exemplo

---

- ▶ Faça um programa que mostra na tela os número de 1 a 100

*while (expression)  
statement*

```
int main()
{
    // programa que mostra na tela números de 1 ate 100
    int numero;

    numero = 1;
    while (numero <= 100){
        printf(" %d ", numero);
        numero = numero + 1;
    }

    return 0;
}
```



# Exemplo - CONTADOR

---

- Faça um programa que mostra na tela os número de 1 a 100

```
int main()
{
    // programa que mostra
    int numero;

    numero = 1;
    while (numero <= 100){
        printf(" %d ", numero);
        numero = numero + 1;
    }

    return 0;
}
```

Observe que a variável `numero` é usada como um contador, ou seja, vai contar quantas vezes o loop será executado



# Exemplo - CONTADOR

---

- Faça um programa que mostra na tela os número de 1 a 100

```
int main()
{
    // programa que mostra
    int numero;

    numero = 1;
    while (numero <= 100){
        printf(" %d ", numero);
        numero = numero + 1;
    }

    return 0;
}
```

Observe que a variável `numero` é usada como um contador, ou seja, vai contar quantas vezes o loop será executado

`numero = 1` inicializa o contador



# Exemplo - CONTADOR

---

- Faça um programa que mostra na tela os número de 1 a 100

```
int main()
{
    // programa que mostra
    int numero;

    numero = 1;
    while (numero <= 100){
        printf(" %d ", numero);
        numero = numero + 1;
    }

    return 0;
}
```

Observe que a variável `numero` é usada como um contador, ou seja, vai contar quantas vezes o loop será executado

`numero = 1` inicializa o contador

`numero = numero + 1`  
incrementa o contador



# Exemplo - CONTADOR

- Faça um programa que mostra na tela os número de 1 a 100

```
int main()
{
    // programa que mostra
    int numero;

    numero = 1;
    while (numero <= 100){
        printf(" %d ", numero);
        numero = numero + 1;
    }

    return 0;
}
```

Observe que a variável `numero` é usada como um contador, ou seja, vai contar quantas vezes o loop será executado

`numero = 1` inicializa o contador

`numero = numero + 1` incrementa o contador

```
1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22
23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62
63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82
83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
Process returned 0 (0x0)   execution time : 0.343 s
Press any key to continue.
```



# Exemplo

---

- Faça um programa para ler 5 números e mostrar o resultado da soma desses números

```
int main()
{
    double val1, val2, val3, val4, val5, soma;

    printf("\nDigite o 1o. numero: ");
    scanf("%lf", &val1);

    printf("\nDigite o 2o. numero: ");
    scanf("%lf", &val2);

    printf("\nDigite o 3o. numero: ");
    scanf("%lf", &val3);

    printf("\nDigite o 4o. numero: ");
    scanf("%lf", &val4);

    printf("\nDigite o 5o. numero: ");
    scanf("%lf", &val5);

    soma = val1 + val2 + val3 + val4 + val5;

    printf("\nO resultado da soma eh: %.2f", soma);
    return 0;
}
```

# Exemplo

- Faça um programa para ler 5 números e mostrar o resultado da soma desses números

```
int main()
{
    double val1, val2, val3, val4, val5, soma;

    printf("\nDigite o 1o. numero: ");
    scanf("%lf", &val1);

    printf("\nDigite o 2o. numero: ");
    scanf("%lf", &val2);

    printf("\nDigite o 3o. numero: ");
    scanf("%lf", &val3);

    printf("\nDigite o 4o. numero: ");
    scanf("%lf", &val4);

    printf("\nDigite o 5o. numero: ");
    scanf("%lf", &val5);

    soma = val1 + val2 + val3 + val4 + val5;

    printf("\nO resultado da soma eh: %.2f", soma);
    return 0;
}
```

É se for para somar  
100 números?  
Usar 100 variáveis?

## ► Melhorando um pouco – ACUMULADOR

```
int main()
{
    double val, soma;

    // inicializando o valor de soma
    soma = 0;

    printf("\nDigite o 1o. numero: ");
    scanf("%lf", &val);
    // somando o primeiro valor lido à variável soma
    soma = soma + val;

    printf("\nDigite o 2o. numero: ");
    scanf("%lf", &val);
    // somando o segundo valor lido à variável soma
    soma = soma + val;

    printf("\nDigite o 3o. numero: ");
    scanf("%lf", &val);
    soma = soma + val;

    printf("\nDigite o 4o. numero: ");
    scanf("%lf", &val);
    soma = soma + val;

    printf("\nDigite o 5o. numero: ");
    scanf("%lf", &val);
    soma = soma + val;

    printf("\nO resultado da soma eh: %.2f", soma);
    return 0;
}
```

Uma só variável **val** para leitura dos dados

Antes eram 5...

```
double val1, val2, val3,
val4, val5
```

## ► Melhorando um pouco – ACUMULADOR

```
int main()
{
```

```
    double val, soma;
```

Uma só variável **val** para leitura dos dados

```
    // inicializando o valor de soma
    soma = 0;
```

```
    printf("\nDigite o 1o. numero: ");
    scanf("%lf", &val);
    // somando o primeiro
    soma = soma + val;
```

Devo somar o valor de **val** à var. **soma** logo após a leitura, pois a var. **val** será reescrita no prox. scanf

```
    printf("\nDigite o 2o. numero: ");
    scanf("%lf", &val);
    // somando o segundo valor lido à variável soma
    soma = soma + val;
```

```
    printf("\nDigite o 3o. numero: ");
    scanf("%lf", &val);
    soma = soma + val;
```

```
    printf("\nDigite o 4o. numero: ");
    scanf("%lf", &val);
    soma = soma + val;
```

```
    printf("\nDigite o 5o. numero: ");
    scanf("%lf", &val);
    soma = soma + val;
```

```
    printf("\nO resultado da soma eh: %.2f", soma);
    return 0;
```

```
}
```

## ► Melhorando um pouco – ACUMULADOR

```
int main()
{
```

```
    double val, soma;
```

Uma só variável **val** para leitura dos dados

```
    // inicializando o valor de soma
    soma = 0;
```

```
    printf("\nDigite o 1o. numero: ");
    scanf("%lf", &val);
    // somando o primeiro
    soma = soma + val;
```

Devo somar o valor de **val** à var. **soma** logo após a leitura, pois a var. **val** será reescrita no prox. scanf

```
    printf("\nDigite o 2o. numero: ");
    scanf("%lf", &val);
    // somando o segundo valor lido à variável soma
    soma = soma + val;
```

```
    printf("\nDigite o 3o. numero: ");
    scanf("%lf", &val);
    soma = soma + val;
```

```
    printf("\nDigite o 4o. numero: ");
    scanf("%lf", &val);
    soma = soma + val;
```

```
    printf("\nDigite o 5o. numero: ");
    scanf("%lf", &val);
    soma = soma + val;
```

Observe que a variável **soma** é usada como um acumulador, ou seja, vai armazenar o valor da soma a cada passo

```
    printf("\nO resultado da soma eh: %.2f", soma);
    return 0;
```

```
}
```

## ► Versão correta, com loops

```
int main()
{
    double val, soma;
    int contagem;

    // inicializando o valor de soma
    soma = 0;

    // inicializando o contador
    contagem = 1;

    while (contagem <= 5){
        printf("\nDigite o %do. numero: ", contagem);
        scanf("%lf", &val);
        soma = soma + val;
        contagem = contagem + 1;
    }

    printf("\nO resultado da soma eh: %.2f", soma);
    return 0;
}
```

Mantenho a variável `val` para leitura dos dados

## ► Versão correta, com loops

```
int main()
{
    double val, soma;
    int contagem;

    // inicializando o valor de soma
    soma = 0;

    // inicializando o contador
    contagem = 1;

    while (contagem <= 5){
        printf("\nDigite o %do. numero: ", contagem);
        scanf("%lf", &val);
        soma = soma + val;
        contagem = contagem + 1;
    }

    printf("\nO resultado da soma eh: %.2f", soma);
    return 0;
}
```

## ► Versão correta, com loops

```
int main()
{
    double val, soma;
    int contagem;

    // inicializando o valor de soma
    soma = 0;

    // inicializando o contador
    contagem = 1;

    while (contagem <= 5){
        printf("\nDigite o %do. numero: ", contagem);
        scanf("%lf", &val);
        soma = soma + val;
        contagem = contagem + 1;
    }

    printf("\nO resultado da soma eh: %.2f", soma);
    return 0;
}
```

Mantenho a variável `val` para leitura dos dados

Crio a variável `contagem` para funcionar como contador

Mantenho a variável `soma` para atuar com acumulador



## ► Versão correta, com loops

```
int main()
{
    double val, soma;
    int contagem;

    // inicializando o valor de soma
    soma = 0;

    // inicializando o contador
    contagem = 1;

    while (contagem <= 5){
        printf("\nDigite o %do. numero: ", contagem);
        scanf("%lf", &val);
        soma = soma + val;
        contagem = contagem + 1;
    }

    printf("\nO resultado da soma eh: %.2f", soma);
    return 0;
}
```

Mantenho a variável **val** para leitura dos dados

Crio a variável **contagem** para funcionar como contador

Mantenho a variável **soma** para atuar com acumulador

**Acumulo soma** a cada passo do loop

## ► Versão correta, com loops

```
int main()
{
    double val, soma;
    int contagem;

    // inicializando o valor de soma
    soma = 0;

    // inicializando o contador
    contagem = 1;

    while (contagem <= 5){
        printf("\nDigite o %do. numero: ", contagem);
        scanf("%lf", &val);
        soma = soma + val;
        contagem = contagem + 1;
    }

    printf("\nO resultado da soma eh: %.2f", soma);
    return 0;
}
```

Mantenho a variável **val** para leitura dos dados

Crio a variável **contagem** para funcionar como contador

Mantenho a variável **soma** para atuar com acumulador

**Acumulo** **soma** a cada passo do loop

**Incremento** **contagem** a cada passo do loop

## ► Versão correta, com loops

### ► CONTADOR

```
int main()
{
    double val, soma;
    int contagem;

    // inicializando o valor de soma
    soma = 0;

    // inicializando o contador
    contagem = 1;

    while (contagem <= 5){
        printf("\nDigite o %do. numero: ", contagem);
        scanf("%lf", &val);
        soma = soma + val;
        contagem = contagem + 1;
    }

    printf("\nO resultado da soma eh: %.2f", soma);
    return 0;
}
```

## ► Versão correta, com loops

### ► ACUMULADOR

---

```
int main()
{
    double val, soma;
    int contagem;

    // inicializando o valor de soma
    soma = 0;

    // inicializando o contador
    contagem = 1;

    while (contagem <= 5){
        printf("\nDigite o %do. numero: ", contagem);
        scanf("%lf", &val);
        soma = soma + val;
        contagem = contagem + 1;
    }

    printf("\nO resultado da soma eh: %.2f", soma);
    return 0;
}
```

---



# Exemplo while

---

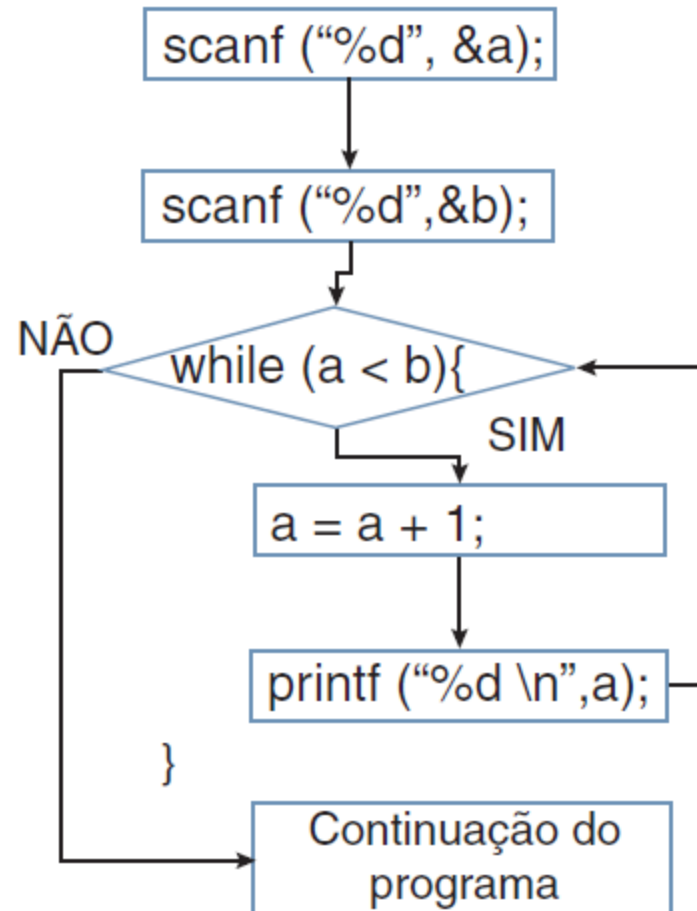
```
int main(){  
  
    int a,b;  
  
    printf("Digite o valor de a: ");  
    scanf("%d",&a);  
  
    printf("Digite o valor de b: ");  
    scanf("%d",&b);  
  
    while (a < b) {  
        a = a + 1;  
        printf("%d \n",a);  
    }  
  
}
```

Neste exemplo não há um número determinado de vezes que o loop é executado, como nos casos que usamos contadores explícitos. Quem define isso são os valores de **a** e **b**



# Exemplo while

---



# Exercício

---

Escreva um programa que dados  $n$  números inteiros, calcule a média destes números. O valor de  $n$  é dado pelo usuário. Imprima os números lidos e a média calculada:

$$m = \frac{\sum x}{N}$$



# Exercício

---

```
algoritmo "media n"  
var  
    n, contagem : inteiro  
    media, nota : real  
    acumulador : real  
inicio  
    escreva("Digite o número de notas: ")  
    leia(n)  
  
    acumulador <- 0  
    contagem <- 0  
  
    enquanto (contagem < n) faça  
        escreva("Digite a nota:" )  
        leia(nota)  
        acumulador <- acumulador + nota  
        contagem <- contagem + 1  
    fimenquanto  
  
    media <- acumulador/n  
  
    escreva("O valor da média é: ", media)  
fimalgoritmo
```





# Exercício – em C

---

```
int main()
{
    int n, contagem;
    float acumulador, nota, media;

    printf("Quantos números vc deseja digitar: ");
    scanf("%d", &n);

    acumulador = 0;
    contagem = 0;

    while (contagem < n) {
        printf("Digite a nota %d:", contagem+1);
        scanf("%f", &nota);
        acumulador = acumulador + nota;
        contagem = contagem + 1;
    }

    media = acumulador/n;

    printf("O valor da média é: %f", media);

    return 0;
}
```

---



# Estruturas de Repetição

---

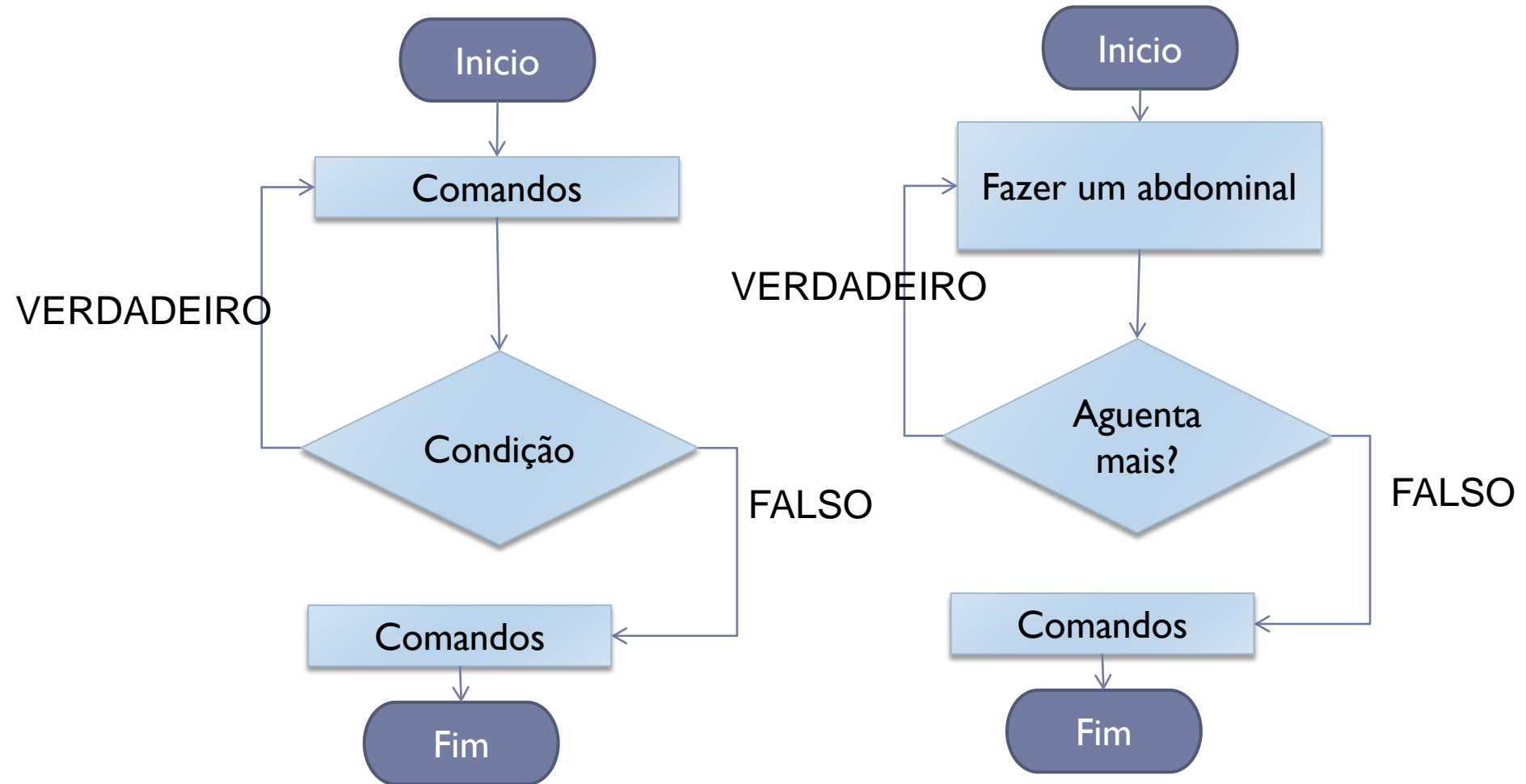
- ▶ Repetição com Teste no Início
- ▶ **Repetição com Teste no Final**
- ▶ Repetição Contada

Uma **estrutura de repetição** permite que uma sequência de comandos seja executada **repetidamente, enquanto** determinadas condições são satisfeitas. Essas condições são representadas por expressões lógicas (como, por exemplo,  $A > B$ ;  $C == 3$ ;  $\text{Letra} == \text{'a'}$ )



# Fluxograma

Repetição com Teste no Final



# Comando do-while

---

- ▶ Comando while: é utilizado para repetir um conjunto de comandos zero ou mais vezes.
- ▶ Comando do-while: é utilizado sempre que o bloco de comandos **deve ser executado ao menos uma vez.**



# Comando do-while

---

- ▶ executa comandos
- ▶ avalia condição:
  - ▶ se verdadeiro, re-executa bloco de comandos
  - ▶ caso contrário, termina o laço
- ▶ Sua forma geral é (**sempre termina com ponto e vírgula ;**)

```
do
    instruções
while (expressão);
```

```
do
    statement
while (expression);
```



# Observação sobre a sintaxe

---

*do*

*statement*

*while (expression);*

- ▶ O que um *statement* (instrução)?
  - ▶ É uma única instrução da linguagem
    - ▶ Um *statement* termina com um sinal de ponto e vírgula ;  
`printf("Hello World!");`
  - ▶ Ou é um conjunto de instruções delimitada por chaves, o que é chamado de Bloco de Instruções
    - ▶ Block Delimiter: `{ }`
    - ▶ Dentro de um bloco podemos colocar mais de uma instrução

```
{  
    printf("Hello World!");  
    printf("Hello World Again!");  
}
```



# Comando do-while

---

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      int i;
7      do{
8          printf("Escolha uma opcao:\n");
9          printf("(1) Opcao 1 \n");
10         printf("(2) Opcao 2 \n");
11         printf("(3) Opcao 3 \n");
12         scanf("%d", &i);
13     } while ((i<1) || (i>3));
14     printf("Opcao escolhida: %d \n", i);
15     return 0;
16 }
```



# Comando do-while

---

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      int i;
7      do{
8          printf("Escolha uma opcao:\n");
9          printf("(1) Opcao 1 \n");
10         printf("(2) Opcao 2 \n");
11         printf("(3) Opcao 3 \n");
12         scanf("%d", &i);
13     } while ((i<1)|| (i>3));
14     printf("Opcao escolhida: %d \n",i);
15     return 0;
16 }
```

**do**  
    *statement*  
**while** (*expression*);





# Comando do-while

---

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

int main( )
{
    int i;
    do{
        printf("Escolha uma opcao:\n");
        printf("(1) Opcao 1 \n");
        printf("(2) Opcao 2 \n");
        printf("(3) Opcao 3 \n");
        scanf("%d", &i);
    } while ((i<1)|| (i>3));
    printf("Opcao escolhida: %d \n",i);
    return 0;
}
```

*do*  
*statement*  
*while (expression);*



# Comando do-while

---

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

int main( )
{
    int i;
    do{
        printf("Escolha uma opcao:\n");
        printf("(1) Opcao 1 \n");
        printf("(2) Opcao 2 \n");
        printf("(3) Opcao 3 \n");
        scanf("%d", &i);
    } while ((i<1)|| (i>3));
    printf("Opcao escolhida: %d \n",i);
    return 0;
}
```

```
do
    statement
while (expression);
```



# Comando do-while

---

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

int main( )
{
    int i;
    do{
        printf("Escolha uma opcao:\n");
        printf("(1) Opcao 1 \n");
        printf("(2) Opcao 2 \n");
        printf("(3) Opcao 3 \n");
        scanf("%d", &i);
    } while ((i<1)||((i>3)));
    printf("Opcao escolhida: %d \n",i);
    return 0;
}
```

```
do
    statement
while (expression);
```



# Comando do-while

---

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

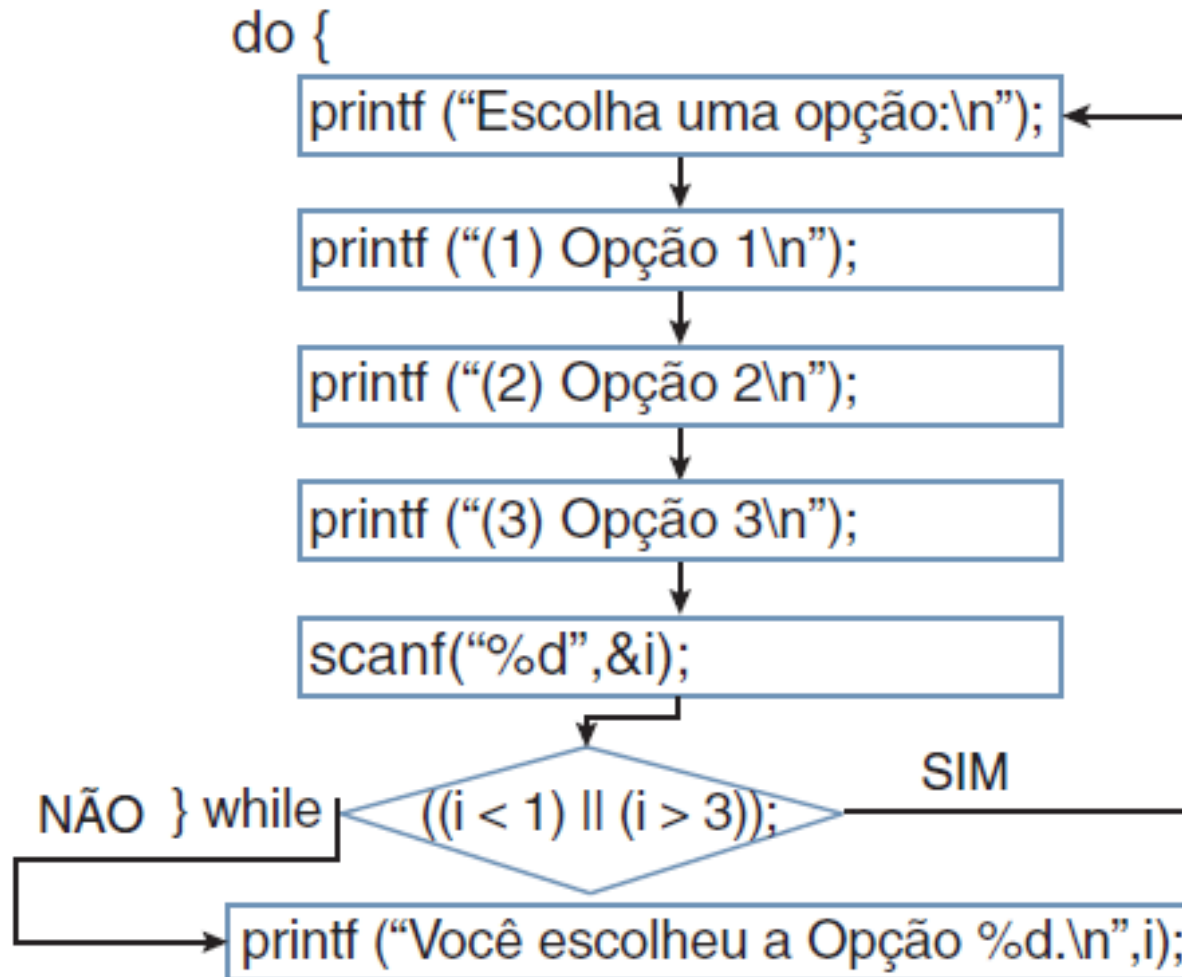
int main( )
{
    int i;
    do{
        printf("Escolha uma opcao:\n");
        printf("(1) Opcao 1 \n");
        printf("(2) Opcao 2 \n");
        printf("(3) Opcao 3 \n");
        scanf("%d", &i);
    } while ((i<1)|| (i>3));
    printf("Opcao escolhida: %d \n",i);
    return 0;
}
```

```
do
    statement
while (expression);
```



# Comando do-while

---



# Exercício

---

Escreva um programa chamado “ACUMULE”, que conta e mostra na tela o número de vezes que o número “7” é digitado pelo usuário. O programa lê vários números inteiros até que o número -1 seja digitado.



```
int main()  
{  
    int acumulador; // guarda a qte que o número 7 eh digitado  
    int x; // leitura do dado  
  
    // inicializando acumulador  
    acumulador = 0;  
  
    do{  
        printf("\n Digite um valor: ");  
        scanf("%d",&x);  
  
        if (x == 7)  
            acumulador = acumulador + 1;  
  
    } while (x != -1);  
  
    printf("vc digitou %d vezes o num. 7\n", acumulador);  
    return 0;  
}
```

---



# Estruturas de Repetição

---

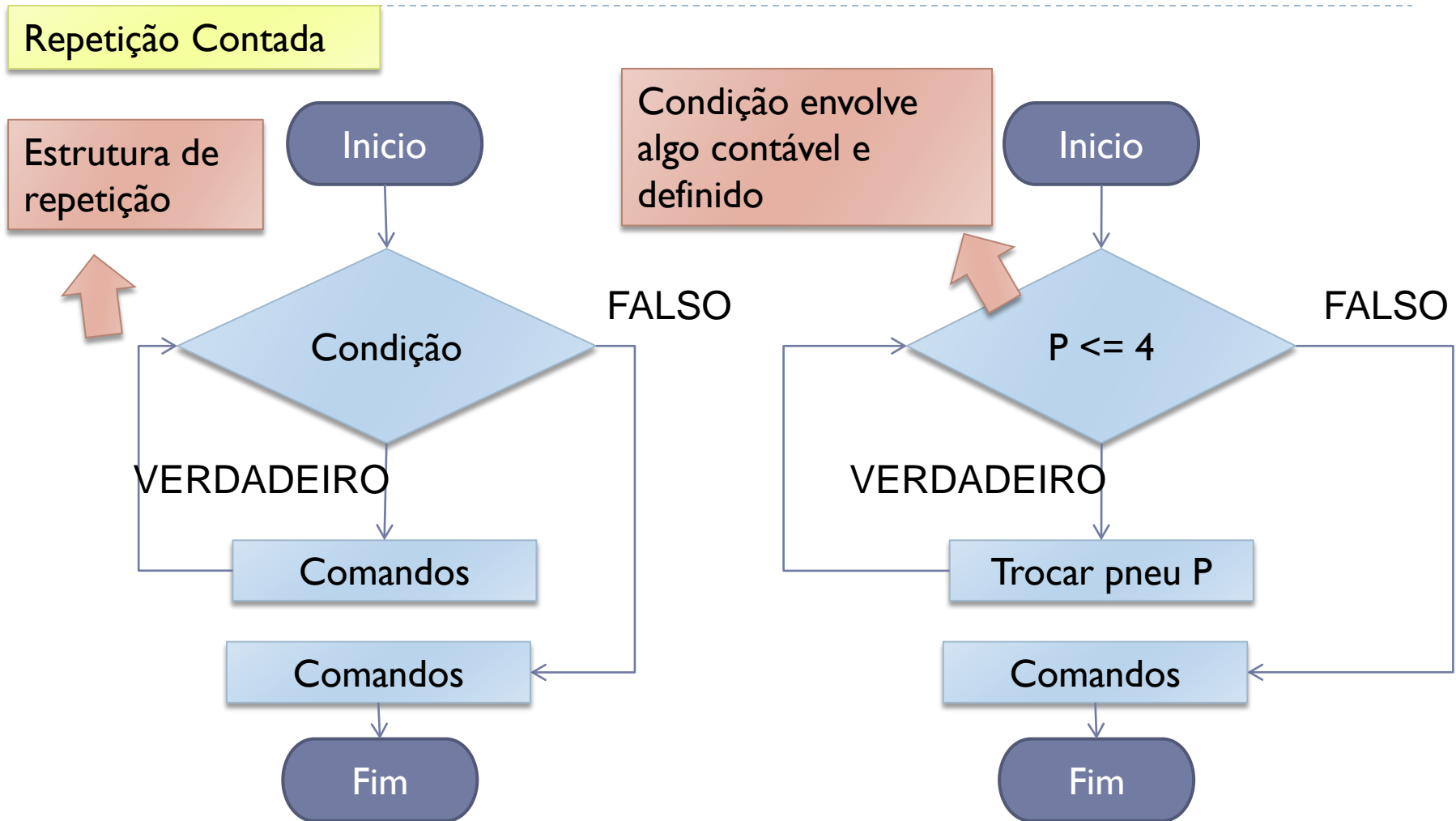
- ▶ Repetição com Teste no Início
- ▶ Repetição com Teste no Final
- ▶ **Repetição Contada**

Uma **estrutura de repetição** permite que uma sequência de comandos seja executada **repetidamente, enquanto** determinadas condições são satisfeitas. Essas condições são representadas por expressões lógicas (como, por exemplo,  $A > B$ ;  $C == 3$ ;  $\text{Letra} == \text{'a'}$ )





# Fluxograma



# Comando ‘para’ (for)

---

- ▶ Antes de aprender como funciona o comando for em C, veremos como ele funciona em pseudolinguagem (algoritmo)
- ▶ Para entendê-lo, vamos antes observar como funciona um loop while que envolve contagem



# Comparando com o **enquanto** (while)

```
algoritmo "media n"  
var  
    n, contagem ← inteiro  
    media, nota : real  
    acumulador : real  
inicio  
    escreva("Digite o número de notas: ")  
    leia(n)  
  
    acumulador ← 0  
    contagem ← 0  
  
    enquanto (contagem < n) faca  
        escreva("Digite a nota:")  
        leia(nota)  
        acumulador ← acumulador + nota  
        contagem ← contagem + 1  
    fimenquanto  
  
    media ← acumulador/n  
  
    escreva("O valor da média é: ", media)  
fimalgoritmo
```

Variável **contagem** foi declarada para controlar quantas vezes o *loop* será executado

Aqui o valor da variável **contagem** é inicializado em zero

Aqui **contagem** é usada como uma **condição** do comando **enquanto**

Aqui **contagem** é incrementada em uma unidade

# Comando **para** (for)

---

- ▶ Para utilizar o comando para é preciso ter
  - ▶ Uma variável para realizar a contagem. Exemplos:

**contagem** : inteiro

**i** : inteiro

**j** : inteiro

- ▶ Inicializar a variável de contagem com um valor.
- ▶ Especificar uma condição para continuar no loop
- ▶ Incrementar a variável usada para contagem

1

2

3

4

1

2

3

4

**para** <variável> **de** <valor-inicial> **ate** <valor-limite> [**passo** <incremento>]  
**faca**

<seqüência-de-comandos>

**fimpara**

---

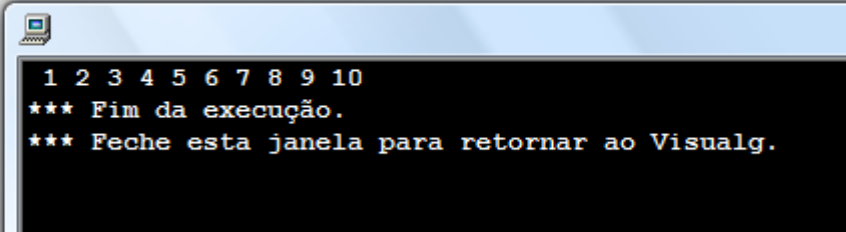


# Exemplo

- Fazer um algoritmo que mostre na tela uma contagem de 1 até 10

**para** <variável> **de** <valor-inicial> **ate** <valor-limite> [**passo** <incremento>] **faca**  
    <seqüência-de-comandos>  
**fimpara**

```
algoritmo "conta 1 a 10"  
  
var  
    i : inteiro // variável para contagem  
inicio  
  
    para i de 1 ate 10 passo 1 faca  
        escreva(i)  
    fimpara  
fimalgoritmo
```

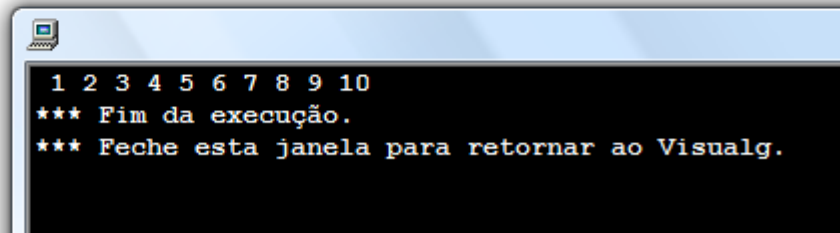


```
1 2 3 4 5 6 7 8 9 10  
*** Fim da execução.  
*** Feche esta janela para retornar ao Visualg.
```

# Exemplo

---

```
algoritmo "conta 1 a 10"  
  
var  
  i : inteiro // variável para contagem  
inicio  
  
  para i de 1 ate 10 passo 1 faca  
    escreva(i)  
  fimpara  
fimalgoritmo
```

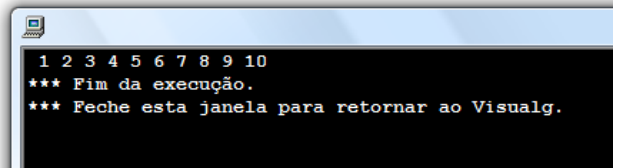


- ▶ Quantos vezes o comando `escreva(i)` é executado?
  - ▶ 10 vezes

# Exemplo

- ▶ Quantos vezes o comando **escreva(i)** é executado?
  - ▶ 10 vezes
- ▶ Na primeira vez, o valor de i é igual ao da inicialização
  - ▶  $i \leftarrow 1$ ;
  - ▶ escreve("1")
- ▶ Ainda no primeiro passo, o valor de i é incrementado em 1 (pois o valor do **passo é 1**). Esse incremento ocorre após a execução de todos os comandos dentro do **para**
  - ▶  $i \leftarrow i + 1$
  - ▶ Valor final de i: 2
- ▶ No segundo passo, o valor de i é 2 e também é incrementado em 1 (**passo é 1**)
  - ▶ escreve("2")
  - ▶  $i \leftarrow i + 1$
  - ▶ Valor final de i: 3

```
algoritmo "conta 1 a 10"
var
    i: inteiro // variável para contagem
início
    para i de 1 ate 10 passo 1 faça
        escreva(i)
    fimpara
fimalgoritmo
```



```
1 2 3 4 5 6 7 8 9 10
*** Fim da execução.
*** Feche esta janela para retornar ao Visualg.
```

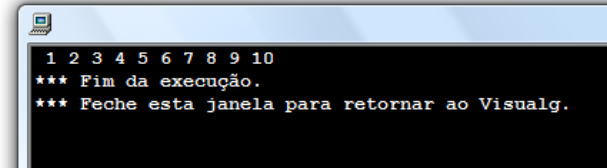
# Exemplo

- ▶ No terceiro passo, o valor de  $i$  é 3 também é incrementado em 1 (**passo é 1**)
  - ▶ escreve("3")
  - ▶  $i \leftarrow i + 1$
  - ▶ Valor final de  $i$ : 4
- ▶ No décimo passo o valor de  $i$  é 10 e é incrementado em 1 (**passo é 1**)
  - ▶ escreve("10")
  - ▶  $i \leftarrow i + 1$
  - ▶ Valor final de  $i$ : 11
- ▶ O décimo primeiro passo não existe, visto que o loop vai até o valor de  $i == 10$  ( **$i$  de 1 até 10**) e o valor de  $i$  é 11. Assim, o loop é finalizado.

```
escreva "conta 1 a 10"

var
  i : inteiro // variável para contagem
inicio

  para i de 1 ate 10 passo 1 faca
    escreva(i)
  fimpara
finalgoritmo
```



```
1 2 3 4 5 6 7 8 9 10
*** Fim da execução.
*** Feche esta janela para retornar ao Visualg.
```

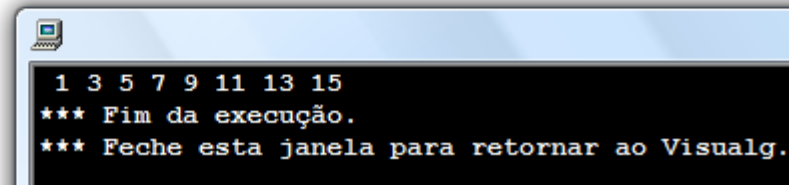


## Exemplo 2

- Fazer um algoritmo que mostre na tela uma contagem de 1 até 16, mostrando somente os números ímpares

**para** <variável> **de** <valor-inicial> **ate** <valor-limite> [**passo** <incremento>] **faca**  
    <seqüência-de-comandos>  
**fimpara**

```
algoritmo "contagem"  
  
var  
    i : inteiro // variável para contagem  
inicio  
  
    para i de 1 ate 16 passo 2 faca  
        escreva(i)  
    fimpara  
fimalgoritmo
```



1 3 5 7 9 11 13 15  
\*\*\* Fim da execução.  
\*\*\* Feche esta janela para retornar ao Visualg.

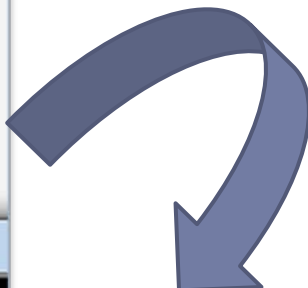
# Observação

- ▶ Quando o incremento for de uma unidade, não é necessário especificar o **passo**. **Obs: isso não vale para códigos em C**

```
algoritmo "conta 1 a 10"

var
  i : inteiro // variável para contagem
inicio

  para i de 1 ate 10 passo 1 faca
    escreva(i)
  fimpara
finalgoritmo
```



```
1 2 3 4 5 6 7 8 9 10
*** Fim da execução
*** Feche esta janela para retornar ao Visualg.
```

```
algoritmo "contagem"

var
  i : inteiro // variável para contagem
inicio

  para i de 1 ate 10 faca
    escreva(i)
  fimpara
finalgoritmo
```

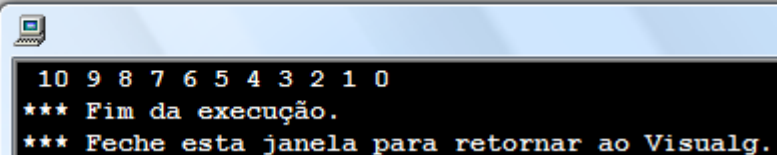
```
1 2 3 4 5 6 7 8 9 10
*** Fim da execução.
*** Feche esta janela para retornar ao Visualg.
```

# Exemplo 3

---

- Fazer um algoritmo que mostre na tela uma contagem regressiva de 10 até 0

```
algoritmo "contagem"  
  
var  
  i : inteiro // variável para contagem  
inicio  
  
  para i de 10 ate 0 passo -1 faca  
    escreva(i)  
  fimpara  
fimalgoritmo
```



```
10 9 8 7 6 5 4 3 2 1 0  
*** Fim da execução.  
*** Feche esta janela para retornar ao Visualg.
```

## Exemplo 4 – loops aninhados

---

- ▶ É possível inserir um loop dentro de outro loop (quantas vezes for necessário)

```
algoritmo "tabuada"  
  
var  
    i,j : inteiro // variáveis para contagem  
inicio  
    para i de 1 ate 10 faca  
        para j de 1 ate 10 faca  
            escreval(i, "x" , j, "=", i*j)  
        fimpara  
    fimpara  
fimalgoritmo
```



# Comando *for* (para) em C

---

- ▶ O loop ou laço *for* é usado para repetir um comando, ou bloco de comandos, diversas vezes
  - ▶ Maior controle sobre o loop.
- ▶ Sua forma geral em C é um pouco distinta do pseudocódigo, mas possui os mesmo elementos

```
for (inicialização; condição; incremento)  
    instruções
```

```
for (expression1; expression2; expression3)  
    statement
```



# Observação sobre a sintaxe

---

*for (expr1, expr2, expr3)*  
*statement*

- ▶ O que um *statement* (instrução)?
  - ▶ É uma única instrução da linguagem
    - ▶ Um *statement* termina com um sinal de ponto e vírgula ;  
`printf("Hello World!");`
  - ▶ Ou é um conjunto de instruções delimitada por chaves, o que é chamado de Bloco de Instruções
    - ▶ Block Delimiter: `{ }`
    - ▶ Dentro de um bloco podemos colocar mais de uma instrução  
`{`  
`printf("Hello World!");`  
`printf("Hello World Again!");`  
`}`



# Comando for

---

```
for (expression1; expression2; expression3)  
    statement
```

1. **expression1**: inicialização: iniciar variáveis (contador).
2. **expression2**: condição: avalia a condição. Se verdadeiro, executa comandos do bloco, senão encerra laço.
3. **expression3**: incremento: ao término do bloco de comandos, incrementa o valor do contador
4. repete o processo até que a condição (**expression2**) seja falsa.



# Exemplo for

---

```
int main()
{
    // mostra os valores de 1 até 10
    int i;
    for (i = 1; i <= 10; i = i+1){
        printf("%d ",i);
    }
}
```





# Exemplo for

---

## Inicialização

```
int main()
{
    // mostra os valores de 1 até 10
    int i;
    for (i = 1; i <= 10; i = i+1){
        printf("%d ",i);
    }
}
```



# Exemplo for

---

Condição

```
int main()
{
    // mostra os valores de 1 até 10
    int i;
    for (i = 1; i <= 10; i = i+1){
        printf("%d ",i);
    }
}
```



# Exemplo for

---

## Incremento

```
int main()
{
    // mostra os valores de 1 até 10
    int i;
    for (i = 1; i <= 10; i = i+1){
        printf("%d ",i);
    }
}
```



# Comando while / for

---

- ▶ Comando while: repete uma seqüência de comandos enquanto uma condição for verdadeira.
- ▶ Comando for: repete uma seqüência de comandos “N vezes”.



# Comparação while / For

---

```
int main()
{
    // mostra os valores de 1 até 10
    int i;
    for (i = 1; i <= 10; i = i+1){
        printf("%d ",i);
    }
}
```

```
int main()
{
    // mostra os valores de 1 até 10
    int i = 1;
    while (i <= 10){
        printf("%d ",i);
        i = i + 1;
    }
}
```



# Exemplo for

---

```
int main()
{
    // mostra os valores de 1 até 10
    int i;
    for (i = 1; i <= 10; i = i+1){
        printf("%d ",i);
    }
}
```

## Inicialização

```
int main()
{
    // mostra os valores de 1 até 10
    int i = 1;
    while (i <= 10){
        printf("%d ",i);
        i = i + 1;
    }
}
```



# Exemplo for

---

```
int main()
{
    // mostra os valores de 1 até 10
    int i;
    for (i = 1; i <= 10; i = i+1){
        printf("%d ",i);
    }
}
```

## Condição

```
int main()
{
    // mostra os valores de 1 até 10
    int i = 1;
    while (i <= 10){
        printf("%d ",i);
        i = i + 1;
    }
}
```



# Exemplo for

---

```
int main()
{
    // mostra os valores de 1 até 10
    int i;
    for (i = 1; i <= 10; i = i+1){
        printf("%d ",i);
    }
}
```

## Incremento

```
int main()
{
    // mostra os valores de 1 até 10
    int i = 1;
    while (i <= 10 ){
        printf("%d ",i);
        i = i + 1;
    }
}
```





# Exemplo for – Ordem de execução dos comandos

---

Valor de i: lixo

```
int main()
{
    // mostra os valores de 1 até 10
    int i;
    for (i = 1; i <= 10; i = i+1){
        printf("%d ",i);
    }
}
```



# Exemplo for – Ordem de execução dos comandos

---

Valor de i: 1

```
int main()
{
    // mostra os valores de 1 até 10
    int i;
    for (i = 1; i <= 10; i = i+1){
        printf("%d ",i);
    }
}
```



# Exemplo for – Ordem de execução dos comandos

---

Valor de i: 1

```
int main()
{
    // mostra os valores de 1 até 10
    int i;
    for (i = 1; i <= 10; i = i+1){
        printf("%d ",i);
    }
}
```

Observe que este é um comando de inicialização do contador. Ele só é executado uma vez no loop



# Exemplo for – Ordem de execução dos comandos

---

```
int main()
{
    // mostra os valores de 1 até 10
    int i;
    for (i = 1; i <= 10; i = i+1){
        printf("%d ",i);
    }
}
```

Valor de i: 1  
Testa se i é menor  
que 10 (verdade)



# Exemplo for – Ordem de execução dos comandos

---

```
int main()
{
    // mostra os valores de 1 até 10
    int i;
    for (i = 1; i <= 10; i = i+1){
        printf("%d ",i);
    }
}
```

Valor de i: 1  
Mostra "1" na tela



# Exemplo for – Ordem de execução dos comandos

---

```
int main()
{
    // mostra os valores de 1 até 10
    int i;
    for (i = 1; i <= 10; i = i+1){
        printf("%d ",i);
    }
}
```

Valor de i: 2  
Incrementa o valor  
de i



# Exemplo for – Ordem de execução dos comandos

---

```
int main()
{
    // mostra os valores de 1 até 10
    int i;
    for (i = 1; i <= 10; i = i+1){
        printf("%d ",i);
    }
}
```

Valor de i: 2  
Incrementa o valor  
de i

Observe que o `i = i + 1` só ocorreu após o fim dos comandos do *statement* do comando *for*. Ou seja, olhando a sintaxe, *expr3* só é executado após o *statement*

*for (expr1, expr2, expr3)*  
*statement*

---



# Exemplo for – Ordem de execução dos comandos

---

```
int main()
{
    // mostra os valores de 1 até 10
    int i;
    for (i = 1; i <= 10; i = i+1){
        printf("%d ",i);
    }
}
```

Valor de i: 2  
Testa se i é menor  
ou igual a 10  
(verdade)





# Exemplo for – Ordem de execução dos comandos

---

```
int main()
{
    // mostra os valores de 1 até 10
    int i;
    for (i = 1; i <= 10; i = i+1){
        printf("%d ",i);
    }
}
```

Valor de i: 2  
Mostra "2" na tela



# Exemplo for – Ordem de execução dos comandos

---

```
int main()
{
    // mostra os valores de 1 até 10
    int i;
    for (i = 1; i <= 10; i = i+1){
        printf("%d ",i);
    }
}
```

Valor de i: 3  
Incrementa o valor  
de i



# Exemplo for – Ordem de execução dos comandos

---

```
int main()
{
    // mostra os valores de 1 até 10
    int i;
    for (i = 1; i <= 10; i = i+1){
        printf("%d ",i);
    }
}
```

Valor de i: 3  
Testa se i é menor  
ou igual a 10  
(verdade)



# Exemplo for – Ordem de execução dos comandos

---

(pulando algumas etapas...)

Valor de i: 10  
Mostra “10” na tela

```
int main()
{
    // mostra os valores de 1 até 10
    int i;
    for (i = 1; i <= 10; i = i+1){
        printf("%d ",i);
    }
}
```



# Exemplo for – Ordem de execução dos comandos

---

```
int main()
{
    // mostra os valores de 1 até 10
    int i;
    for (i = 1; i <= 10; i = i+1){
        printf("%d ",i);
    }
}
```

Valor de i: 11  
Incrementa o valor  
de i



# Exemplo for – Ordem de execução dos comandos

---

```
int main()
{
    // mostra os valores de 1 até 10
    int i;
    for (i = 1; i <= 10; i = i+1){
        printf("%d ",i);
    }
}
```

Valor de i: 11  
Testa se i é menor  
ou igual a 10 (falso)



# Exemplo for – Ordem de execução dos comandos

---

```
int main()
{
    // mostra os valores de 1 até 10
    int i;
    for (i = 1; i <= 10; i = i+1){
        printf("%d ",i);
    }

    // continua o programa! Valor de i é 11
}
```

Valor de i: 11

Fim do loop. O valor de i fica 11



# Operador Incremento

---

- ▶ Em C, existe um operador de incremento cujo símbolo é **++**
- ▶ Ele serve para incrementar em uma unidade o valor de uma variável.
  - ▶ Exemplo **i++** tem o mesmo efeito que **i = i + 1**
- ▶ Esse operador é muito comum em loops *for*

```
int main()
{
    // mostra os valores de 1 até 10
    int i;
    for (i = 1; i <= 10; i++){
        printf("%d ",i);
    }
}
```





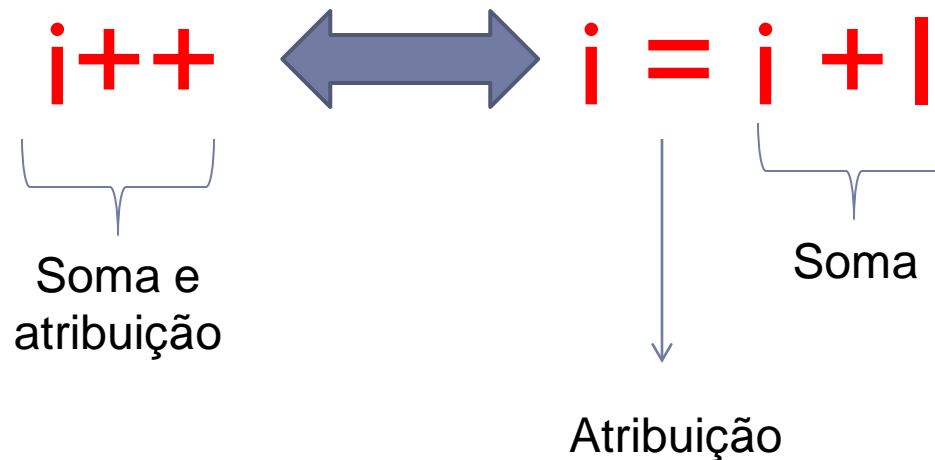
# Operador Incremento

---

- ▶ Note que duas operações são realizadas pelo operador

**++**

- ▶ Soma
- ▶ Atribuição



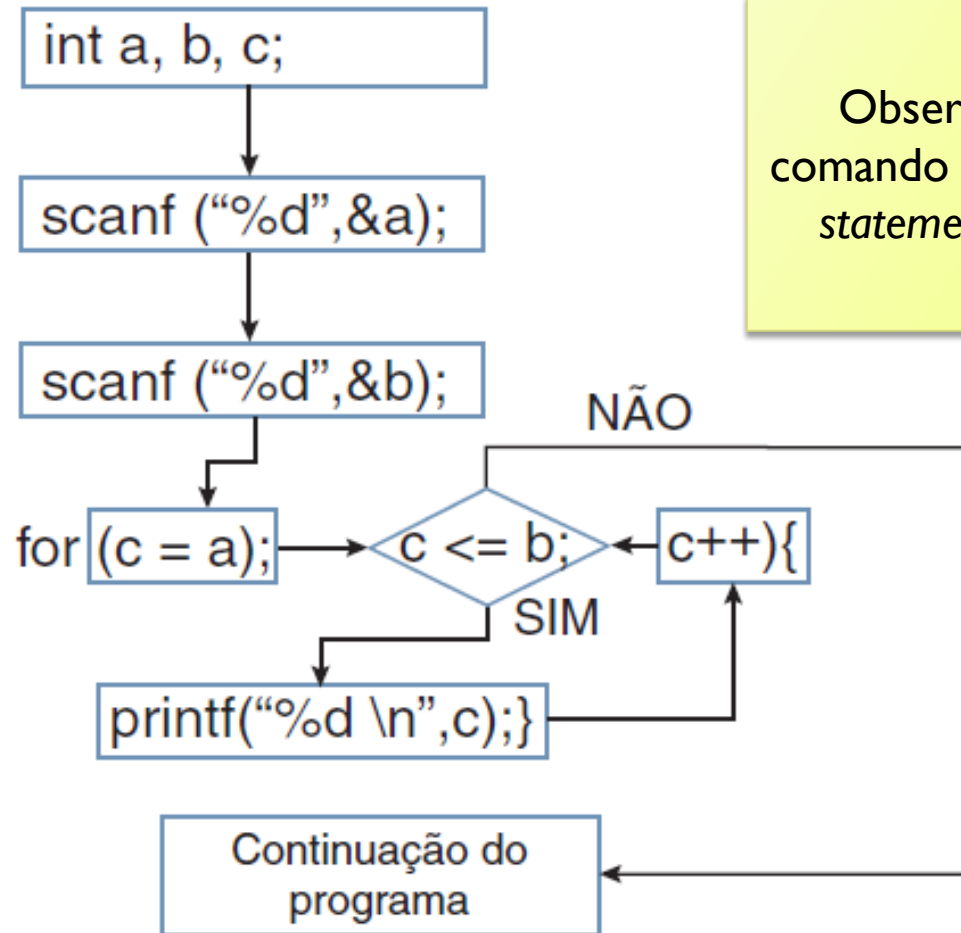
# Exemplo for

---

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04      int a,b,c;
05      printf("Digite o valor de a: ");
06      scanf("%d",&a);
07      printf("Digite o valor de b: ");
08      scanf("%d",&b);
09      for (c = a; c <= b; c++){
10          printf("%d \n",c);
11      }
12      system("pause");
13      return 0;
14  }
```



# Exemplo for



Observe neste código que o comando `c++` só é realizado após o *statement* do `for` for executado

# Exercício

---

- ▶ Escreva, usando for, um algoritmo para calcular a soma dos elementos de 1 a 10.



# Exercício

---

```
int n;  
int soma = 0;  
for (n = 1; n <= 10; n++){  
    soma = soma + n;  
}  
printf("%d", soma);
```



# for *versus* while

---

	for	while
01	<b>#include</b> <stdio.h>	<b>#include</b> <stdio.h>
02	<b>#include</b> <stdlib.h>	<b>#include</b> <stdlib.h>
03	<b>int</b> main(){	<b>int</b> main(){
04	<b>int</b> i, s = 0;	<b>int</b> i, s = 0;
05	<b>for</b> (i = 1; i <= 10; i++){	i = 1
06	s = s + i;	<b>while</b> (i <= 10){
07	}	s = s + i;
08	printf("Soma = %d \n",s);	i++;
09	system("pause");	}
10	<b>return</b> 0;	printf("Soma = %d \n",s);
11	}	system("pause");
12		<b>return</b> 0;
13		}



# Operado Incremento / Decremento

---

- ▶ Vimos que  $i = i + 1$  pode ser escrito como  $i++$
- ▶ O  $++$  é o operador de incremento
- ▶ Existe também o operador de decremento  $--$ 
  - ▶  $i = i - 1$  equivale à  $i--$
- ▶ Esses operadores  $++$  e  $--$  são operadores **unários**
- ▶ Eles podem ser pré ou pós incrementado/decrementado
  - ▶  $++i$  : pré-incremento
  - ▶  $i++$  : pós-incremento
  - ▶  $--i$  : pré-decremento
  - ▶  $i--$  : pós-decremento



# Operadores

---

- ▶ Diferença entre pré e pós incremento/decremento
  - ▶  $y = x++$ : incrementa **depois** de atribuir
  - ▶  $y = ++x$ : incrementa **antes** de atribuir





# Operadores

---

## ► Ex:

```
int x,y;  
x = 10;  
y = x++; // pós incremento (atribui e depois  
          //incrementa)  
printf("%d \n",x); // 11  
printf("%d \n",y); // 10  
y = ++x; // pré incremento (incrementa e  
          // depois atribui)  
printf("%d \n",x); // 12  
printf("%d \n",y); // 12
```

## ► Resultado

```
11  
10  
12  
12
```



# Operadores

---

## ► Ex:

```
int k=10;  
printf("Valor de k: %d \n\n\n", k);
```

Saída

Valor de k: 10

```
k=10;  
printf("Valor de k: %d \n", k);  
printf("Valor de k++: %d \n", k++);  
printf("Valor de k: %d \n\n\n", k);
```

Valor de k: 10

Valor de k++: 10

Valor de k: 11

```
k=10;  
printf("Valor de k: %d \n", k);  
printf("Valor de ++k: %d \n", ++k);  
printf("Valor de k: %d \n", k);
```

Valor de k: 10

Valor de ++k: 11

Valor de k: 11

---



# Precedência dos Operadores

Maior Precedência



Menor Precedência

## Operators (grouped by precedence)

structure member operator	<i>name.member</i>
structure pointer	<i>pointer-&gt;member</i>
increment, decrement	<b>++</b> , <b>--</b>
plus, minus, logical not, bitwise not	<b>+</b> , <b>-</b> , <b>!</b> , <b>~</b>
indirection via pointer, address of object	<b>*pointer</b> , <b>&amp;name</b>
cast expression to type	<b>(type) expr</b>
size of an object	<b>sizeof</b>
multiply, divide, modulus (remainder)	<b>*</b> , <b>/</b> , <b>%</b>
add, subtract	<b>+</b> , <b>-</b>
left, right shift [bit ops]	<b>&lt;&lt;</b> , <b>&gt;&gt;</b>
comparisons	<b>&gt;</b> , <b>&gt;=</b> , <b>&lt;</b> , <b>&lt;=</b>
comparisons	<b>==</b> , <b>!=</b>
bitwise and	<b>&amp;</b>
bitwise exclusive or	<b>^</b>
bitwise or (incl)	<b> </b>
logical and	<b>&amp;&amp;</b>
logical or	<b>  </b>
conditional expression	<i>expr<sub>1</sub> ? expr<sub>2</sub> : expr<sub>3</sub></i>
assignment operators	<b>+=</b> , <b>-=</b> , <b>*=</b> , <b>...</b>
expression evaluation separator	<b>,</b>

Unary operators, conditional expression and assignment operators group right to left; all others group left to right.



# Material Complementar

---

## ▶ Vídeo Aulas

- ▶ Aula 18: Comando While
- ▶ Aula 19: Comando For
- ▶ Aula 20: Comando Do-While
- ▶ Aula 21: Aninhamento de Repetições
- ▶ Aula 22: Comando Break
- ▶ Aula 23: Comando Continue
- ▶ Aula 24: Comando Goto



# Comando for

---

- ▶ Podemos omitir qualquer um de seus elementos
  - ▶ inicialização, condição ou incremento.
- ▶ **Ex.: for sem inicialização**

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04      int a,b,c;
05      printf("Digite o valor de a: ");
06      scanf("%d",&a);
07      printf("Digite o valor de b: ");
08      scanf("%d",&b);
09      for (; a <= b; a++){
10          printf("%d \n",a);
11      }
12      system("pause");
13      return 0;
14  }
```



# Comando for

---

- ▶ **Cuidado: for sem condição**
  - ▶ omitir a condição cria um laço infinito;
  - ▶ condição será sempre verdadeira.

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04      int a,b,c;
05      printf("Digite o valor de a: ");
06      scanf("%d",&a);
07      printf("Digite o valor de b: ");
08      scanf("%d",&b);
09      //o comando for abaixo e um laco infinito
10      for (c = a; ; c++){
11          printf("%d \n",c);
12      }
13      system("pause");
14      return 0;
15  }
```

# Comando for

---

- ▶ Cuidado: for sem incremento
  - ▶ omitir o incremento cria um laço infinito;
  - ▶ Incremento pode ser feito nos comandos.

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04      int a,b,c;
05      printf("Digite o valor de a: ");
06      scanf("%d",&a);
07      printf("Digite o valor de b: ");
08      scanf("%d",&b);
09      for (c = a; c <= b; ){
10          printf("%d \n",c);
11          c++;
12      }
13      system("pause");
14      return 0;
15  }
```



# Referências

---

- ▶ Baseado em slides do Prof. André Backes

