

# **INF101202**

## **Algoritmos e Programação**

### **Modalidade Ead – Turma H**

**Material de apoio: capacitar a escrita de algoritmos iterativos**

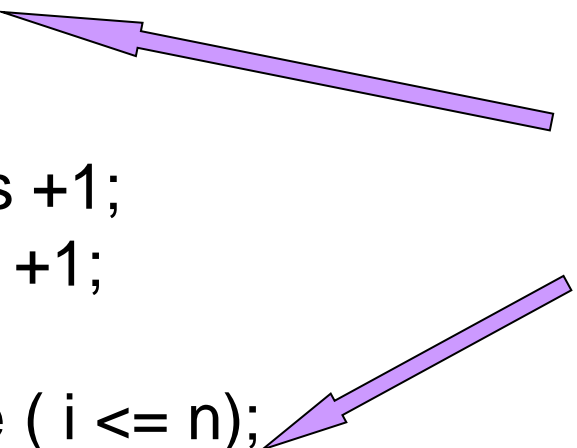
# Comando iterativo : do-while

é outra maneira de repetir comando(s).

Ex: dado um valor N, fazer  $s = 1 + 2 + 3 + \dots + N$ .

Veja a seguir um trecho de um programa em C:

```
.....;
s=0; // s variável p/ armazenar a soma dos inteiros gerados
i=0; // na variável i
do
{
    s= s +1;
    i= i +1;
}
while ( i <= n);
...
```



estrutura do – while: os comandos  $s=s+1$ ; e  $i=i+1$ ; são repetidos enquanto a condição  $(i \leq n)$  é verdade ou possuir valor diferente de zero

# Comando *do...while*

Pseudo-linguagem

Linguagem C

*faça*

**comando**

*enquanto* condição

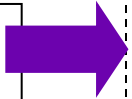
*do*

**comando**

*while* (condição)

1 comando

lógica

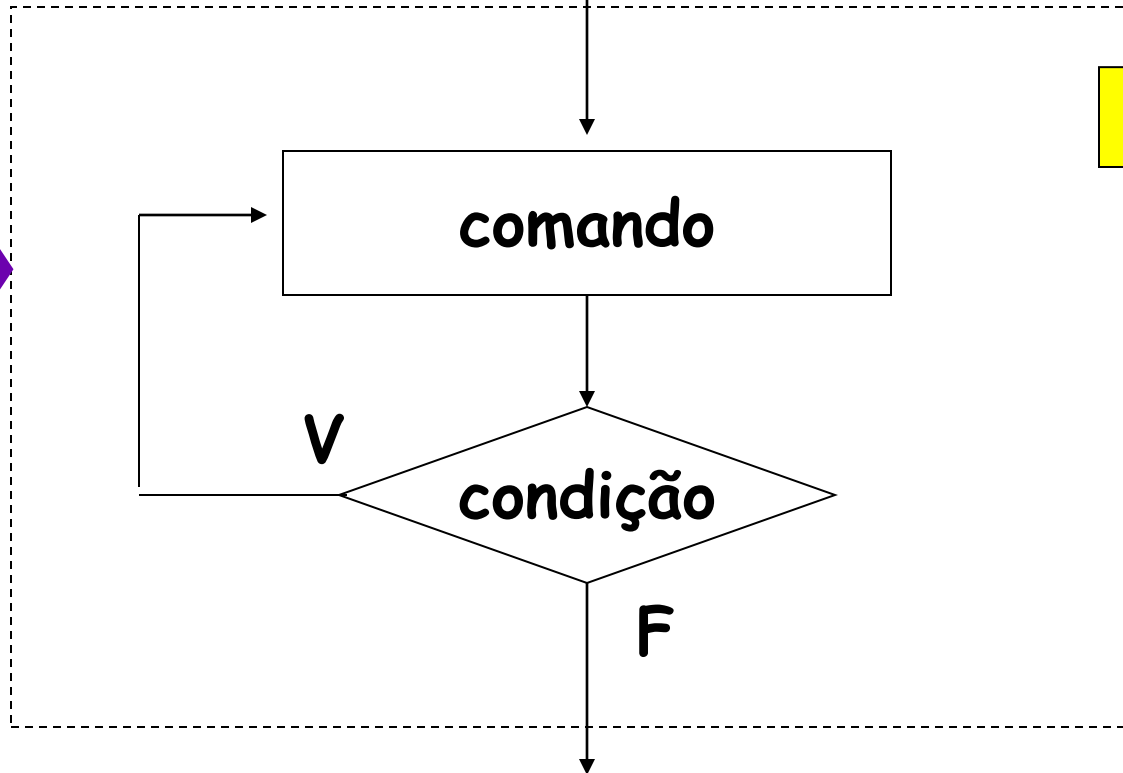


comando

V

condição

F

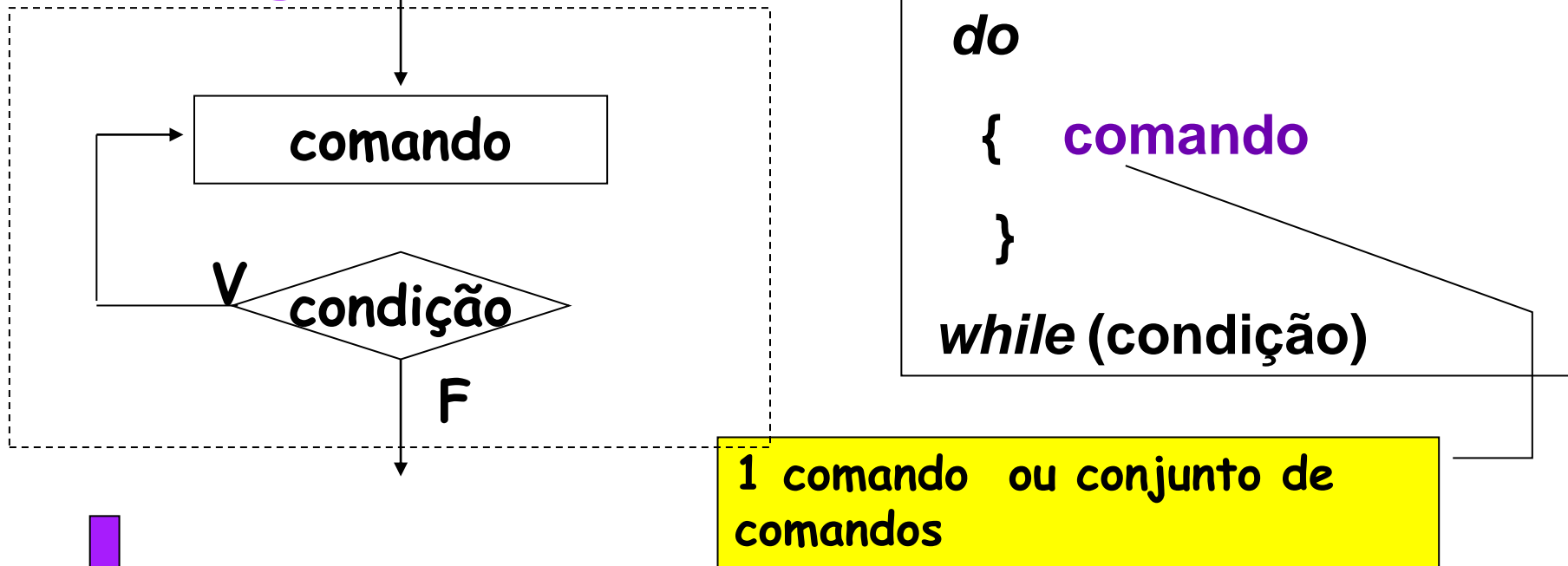


# Comando do-while:

*lógica*

*e*

*sintaxe*



Em palavras: o comando do-while permite que um conjunto de instruções seja executado tantas vezes quantas forem necessárias, enquanto o valor da condição associada for verdadeiro ( qualquer valor diferente de zero).

# Exemplos

```
...  
a = 0;  
do  
{  
    a = a + 1;  
    printf("%d", a);  
}  
while (a < 10);  
...
```

```
...  
achou = 0; // 0 = falso  
do  
{  
    scanf("%d",&numero);  
    printf("%d", numero);  
    if (numero == 250)  
        achou = 1;  
}  
while (achou != 1); // 1 = verdadeiro  
...
```

```
...  
do  
{  
    scanf("%d", &numero);  
    printf("%d", numero);  
    printf('Outro ? (1 = sim / 0 = nao) ');  
    scanf("%d", &resposta);  
}  
while (resposta == 1);  
...
```

# Exemplo

```
//escreve 20 vezes Bom dia
#include <stdlib.h> // para poder usar system("Pause")
#include <stdio.h>
int main ( )
{
    int cont;
    cont = 0;
    do
    {
        printf("Bom Dia !!!");
        cont = cont + 1;
    }
    while (cont < 20);
    system("PAUSE");
    return 0;
}
```

# Ex: soma dos números pares entre 10 e 100

## //soma pares versão 1

```
#include <stdio.h>
int main ( )
{
    int soma, par;
    soma = 0;
    par = 10;
    do
    {
        soma = soma + par;
        par = par + 2;
    }
    while (par <= 100);
    printf ( "Soma = %d" , soma);
    return 0;
}
```

## //soma pares versão 2

```
#include <stdio.h>
int main ( )
{
    int soma, par;
    soma = 0;
    par = 10;
    while (par <= 100)
    {
        soma = soma + par;
        par = par + 2;
    }
    printf ( "Soma = %d" , soma);
    return 0;
}
```

**Ex:** Média de números inteiros, lidos do teclado. Parar ao ler número 9999.

```
//Calcula media de um numero indeterminado de valores lidos
#include <stdlib.h>
#include <stdio.h>
int main ( )
{
    int numero, lidos = 0;
    long int  soma = 0;
    printf("Digite um numero inteiro de cada vez.");
    printf ("\nPara terminar, digite o valor 9999.");
    do
    {
        printf("\nNumero? ");
        scanf("%d",&numero);
        if (numero !=9999)
        {
            soma = soma + numero;
            lidos = lidos + 1;
        }
        while (numero != 9999);
    }
    printf ("\nMedia dos valores lidos: %5.2f\n",
(float)soma/lidos);
    system("PAUSE");  // segura a tela de execucao
    return 0;
}
```

```
...
printf("\nNumero? "); scanf("%d", &numero);
while (numero != 9999)
{
    soma = soma + numero;
    lidos = lidos + 1;
    printf("\nNumero? "); scanf("%d",
&numero);
}
```



**Ex:** Processar as notas dos alunos de uma turma.  
Para cada aluno, ler suas 4 notas, calcular e informar a média.  
**Encerrar quando forem feitas 40 leituras de 4 notas parciais.**

```
//Processa notas dos alunos de uma turma
int main ( )
{
    float  n1, n2, n3, n4;  // notas de um aluno
    float ma;
    int c;    // variável de contar alunos
    c = 0 ;  // inicializar contador c em zero
    do {
        scanf("%f %f %f%f", &n1, &n2, &n3,&n4);
        // obtém as 4 notas de um aluno
        ma = (n1 + n2 + n3 + n4)/4;
        c = c + 1;
        printf("Media do aluno %d : %6.2f", c, ma);

    }
    while ( c < 40 )
    // do processamento de um aluno
    system ("PAUSE");
    return 0;
}
```

Refazer o exemplo acertando-o para a execução, com 10 alunos e incluir o cálculo da média da turma.

**Ex:** Processar as notas dos alunos de uma turma de total desconhecido.  
Para cada aluno, ler suas 3 notas, calcular e informar a média.  
**Para encerrar, o sinal de parada é o valor -9 na primeira nota do aluno.**  
No final, informar a média da turma.

```
int main ( )           //Calcula media de alunos
{
    float n1, n2, n3; // notas de um aluno
    float media, soma;
    int c; // variável de contar alunos
    soma = 0;  c = 0;
    do
    {
        scanf ("%f %f %f", &n1, &n2, &n3); // obtém as notas de um aluno
        if ( n1 != -9)
        {
            media = (n1 + n2 + n3) / 3;
            soma = soma + media;
            c = c + 1;
            printf("Media do aluno %d: %7.2f ", c, media);
        }
    }
    while (n1 != -9) // fim do processamento de um aluno
    printf("\nMédia da turma: %7.2f\n", soma/c);
    system("PAUSE");
    return 0;
}
```

**Cuidado a  
tomar:**

se o 'sinal  
de parada' faz  
ou não parte  
dos dados.

# Diferença entre o while e o do-while !!!!! (1)

Há uma diferença entre as 2 estruturas iterativas.

No comando ***while***, o valor da condição é verificado antes da execução do conjunto de comandos associados. Nesse caso, se o valor da condição for falso de início, o comando ou bloco associado **não é executado, nenhuma vez !!!**

Logo o comando while é mais adequado para situações em que é possível nunca ser necessário executá-lo.

## Diferença entre o while e o do-while !!!!! (2)

No comando **do-while**, o conjunto de instruções será executado pelo menos **uma vez**, **sempre**, pois a condição é analisada depois da execução dos comandos ou bloco associado.

Logo o comando **do-while** é mais adequado para situações em que se sabe que pelo menos uma vez, com certeza, será necessário executá-lo.

Ex: fazer  $s=1 + 2 = \dots + N$  , com  $N \geq 0$

### //soma versão 1

```
#include <stdio.h>
#include <stdlib.h>
int main ( )
{
    int s, x, n;
    printf("Entrar com N >0:");
    scanf("%d", n);
    s = 0; x = 1;
    do
    {
        s = s + x;
        x = x + 1;
    }
    while (x <= n);
    printf ( "Soma = %d", s);
    system("PAUSE");
    return 0;
}
```

### //soma versão 2

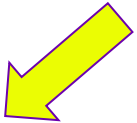
```
#include <stdio.h>
#include <stdlib.h>
int main ( )
{
    int s, x, n;
    printf("Entrar com N >0:");
    scanf("%d", n);
    s = 0; x = 1;
    while (x <= n)
    {
        s = s + x;
        x = x + 1;
    }
    printf ( "Soma = %d", s);
    system("PAUSE"); return 0;
}
```

**Ex: fazer  $s=1 + 2 = \dots + N$ ,  
com  $N \geq 0$**

**Observar!!! Executar versões com  $n=5$  e  
depois com  $n = 0$**

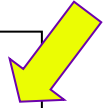
**//soma versão 1**

```
#include <stdio.h>
#include <stdlib.h>
int main ( )
{
    int s, x,n;
    printf("Entrar com N >0:");
    scanf("%d",n); s= 0;
    x=1;
    do
    {
        s = s+ x;
        x = x+ 1;
    }
    while (x<= n);
    printf ( "Soma = %d",s);
    system("PAUSE");
    return 0;
}
```



**//soma versão 2**

```
#include <stdio.h>
#include <stdlib.h>
int main ( )
{
    int s, x,n;
    printf("Entrar com N >0:");
    scanf("%d",n);
    s = 0; x=1;
    while (x <= n)
    {
        s = s + x;
        x= x + 1;
    }
    printf ( "Soma = %d",s);
    system("PAUSE");return 0;
}
```



**Para  $n=5$ , resultado das versões 1 e 2:  $s= 15$**

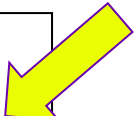
**Para  $n=0$ , resultado versão 1:  $s=1$  (incorreto!Deve ser 0)**

**resultado versão 2:  $s=0$  (correto!)**

Ex: fazer  $s=1 + 2 = \dots + N$ , Observar!!! Executar versões com  $N \geq 0$  com  $n=5$  e depois com  $n=0$

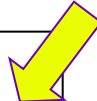
### //soma versão 1

```
#include <stdio.h>
#include <stdlib.h>
int main ( )
{
    int s, x,n;
    printf("Entrar com N >0:");
    scanf("%d",n); s= 0;
    x=1;
    do
    {
        s = s+ x;
        x = x+ 1;
    }
    while (x<= n);
    printf ( "Soma = %d", s);
    system("PAUSE");
    return 0;
}
```



### //soma versão 2

```
#include <stdio.h>
#include <stdlib.h>
int main ( )
{
    int s, x,n;
    printf("Entrar com N >0:");
    scanf("%d",n);
    s = 0; x=1;
    while (x <= n)
    {
        s = s + x;
        x= x + 1;
    }
    printf ( "Soma = %d", s);
    system("PAUSE");
    return 0;
}
```



Isto acontece porque na **versão 1 (Do-While)**, a condição ( $x \leq n$ ) ocorre **após** uma execução que realiza a soma de  $s$  com  $x$ .

A **versão 1** deve ser corrigida atualizando o comando de atribuição inicial de  $x$ , para  $x = 0$  ,



**Versão 1** foi alterada e corrigida. Os programas agora estão funcionalmente equivalentes e obtêm os mesmos resultados para  $n = 0$ .

```
//soma versão 1-Do-While
#include <stdio.h>
#include <stdlib.h>
int main ( )
{
    int s, x,n;
    printf("Entrar com N >0:");
    scanf("%d",n);
    s= 0;
    x = 0;
    do
    {
        s = s + x;
        x = x + 1;
    }
    while (x <= n);
    printf ( "Soma = %d" , s);
    system("PAUSE");
    return 0;
}
```

```
//soma versão 2 -While
#include <stdio.h>
#include <stdlib.h>
int main ( )
{
    int s, x,n;
    printf("Entrar com N >0:");
    scanf("%d",n);
    s = 0; x=1;
    while (x <= n)
    {
        s = s + x;
        x= x + 1;
    }
    printf ( "Soma = %d" , s);
    system("PAUSE");return 0;
}
```



# Problemas com laços:

- Laço (ou *loop*) infinito;
- Aninhamento incorreto de laços.

# Qual o problema com esse código?

**Laço (ou loop) infinito!**

```
//soma pares
#include <stdio.h>
int main ( )
{
    int soma, par;
    soma = 0;
    par = 10;
    while (par <= 100)
    {
        soma = soma + par;
        par = par - 2;
    }
    printf ( "Soma = %d\n" , soma);
    return 0;
}
```

**Não há como par  
ser maior do que  
100!**

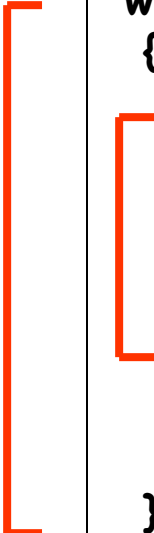
**Assim o *while*  
nunca encerra!**

**Correção:**

**par = par + 2;**

# Observe o seguinte aninhamento de laços: um laço *do...while* dentro de um laço *while*

```
#include <stdio.h>
#include <stdlib.h>
int main ( )
{
    int acum, x, n;
    acum = 0;
    x = 0;
    while (x <= 5)
    {
        do
        {
            printf ("Forneca um numero:");
            scanf ("%d", &n);
        }
        while (n < 0);
        acum = acum + n;
        x++;
    }
    printf("Somatorio: %d\n", acum);
    system("pause");
    return 0;
}
```

A diagram consisting of two large red brackets on the left side of the code. The outer bracket spans from the start of the 'while (x <= 5)' loop to the end of the 'return 0;' statement. The inner bracket spans from the start of the 'do' loop to the end of the 'while (n < 0);' loop, illustrating the nesting of the loops.

Quando um laço for inserido dentro de outro, cuidar **que o laço interno esteja totalmente contido no laço externo.**

Observe as linhas à esquerda!

Veja como elas **não** se cruzam:

A diagram consisting of two red brackets on the right side. The outer bracket is larger and the inner bracket is smaller, positioned such that they do not cross, visually representing the correct nesting of loops where the inner loop is completely contained within the outer loop.