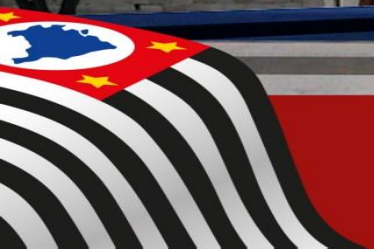


## Linguagem de Programação

### Estrutura de um programa

- Estruturas de repetição – iteração e comandos de desvio: break e continue, expressão e bloco.).
  - Material: LP\_Aula04



## Grupo de comandos Padrão ANSI

Seleção

Iteração

Desvio

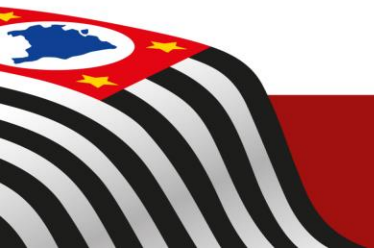
Rótulo – usado na instrução goto.

**Expressão (comandos de atribuição composta)**

**Bloco (bloco de comandos { } )**



# Estruturas de Comando: Repetição ou laço



## Comando for

### for

- O loop **for** é usado para repetir um comando, ou bloco de comandos, diversas vezes, de maneira que se possa ter um bom controle sobre o loop.

Sua forma geral é:

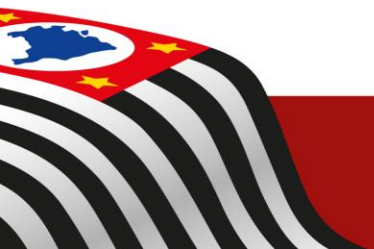
```
for (inicialização;condição;incremento) {  
  
}
```





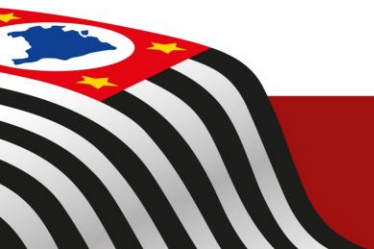
## Exemplo (for - 1)

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    int count;
    for (count=1;count<=100; count++) {
        printf ("%d  ",count);
        if ((count % 10) == 0) printf("\n");
    }
    return 0;
}
```



## Comentário do Exemplo

O incremento da variável **count** é feito usando o operador de incremento. Esta é a **forma usual** de se fazer o incremento (ou decremento) em um loop **for**.



## Cuidados: Loop infinito

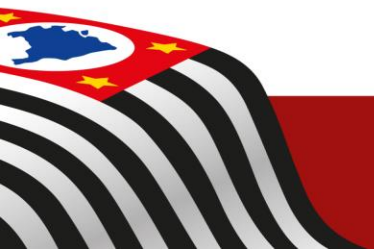
O loop infinito tem a forma:

- *for (inicialização;;incremento) comandos;*

O loop infinito porque será executado para sempre (não existindo a condição, ela será sempre considerada verdadeira), a não ser que ele seja interrompido.

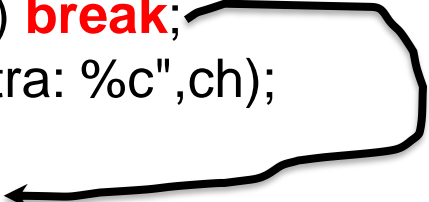
Para interromper um loop como este usamos o comando **break**.

O comando **break** vai quebrar o loop infinito e o programa continuará sua execução normalmente.



## Exemplo (for - 2) Uso do break

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    int Count;
    char ch;
    for (Count=1;;Count++) {
        ch = getch();
        if (ch == 'X') break;
        printf("\nLetra: %c",ch);
    }
    return 0;
}
```



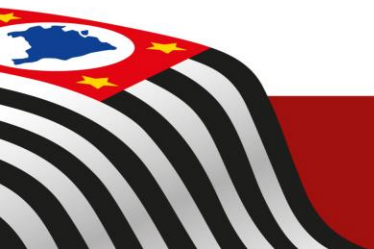


## Loop sem conteúdo

Loop sem conteúdo é aquele no qual se omite a declaração. Sua forma geral é (atenção ao ponto e vírgula!):

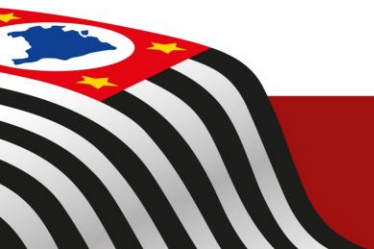
***for (inicialização;condição;incremento);***

Uma das aplicações desta estrutura é gerar tempos de espera.



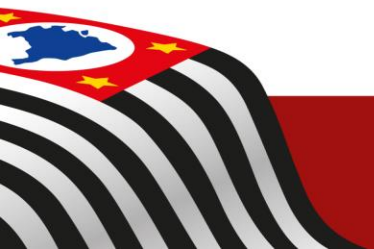
## Exemplo (for – 3) Gerar tempo de espera.

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    long int i;
    printf("Opa\n");
    for (i=0; i<10000000000; i++);
    printf("Acabou a espera\n");
    system("PAUSE");
    return(0);
}
```



## Nota sobre o comando for

A instrução for (laço for), **adapta-se particularmente em situações que o número de iterações é conhecido *a priori*.**



## Comando while

Vamos tentar como funciona o comando **while** fazendo uma analogia.

Então o **while** seria equivalente a:

```
if (condição) {  
    declaração;  
    "Volte para o comando if"  
}
```



## Comando while

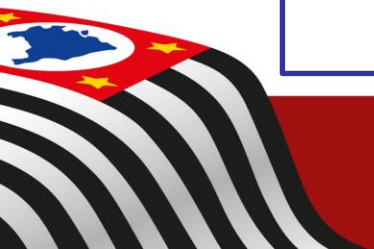
- A estrutura **while** testa uma condição.
- Se esta for verdadeira a declaração é executada e faz-se o teste novamente, e assim por diante.
- Assim como no caso do **for**, podemos fazer um loop infinito. Para tanto basta colocar uma expressão eternamente verdadeira na condição.
- Vamos ver um exemplo do uso do **while**.
- O próximo exemplo espera o usuário digitar a tecla 'q' e só depois finaliza:





## Exemplo (while - 1)

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    char Ch;
    Ch='\0';
    while (Ch!='q') {
        Ch = getch();
    }
    return 0;
}
```



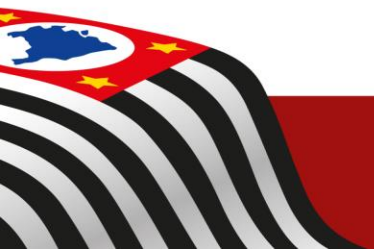
## O Comando do-while

A terceira estrutura de repetição conhecida é o do-while ,de forma geral:

*do {*

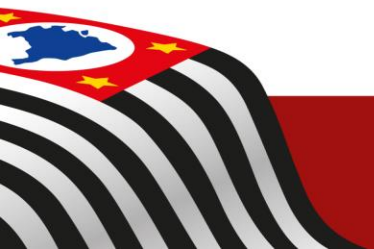
*comandos;*

*} while (condição);*



## Comando do while

- A estrutura **do-while** executa a declaração, testa a condição e, se esta for verdadeira, volta para a declaração.
- O comando **do-while**, ao contrário do **for** e do **while**, garante que a declaração será executada pelo menos uma vez, **pois testa a condição após execução, ou seja, condição pós testada.**



```

#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    int i;
    do {
        printf ("\n\nEscolha a fruta pelo numero:\n\n");
        printf ("\t(1)...Mamão\n");
        printf ("\t(2)...Abacaxi\n");
        printf ("\t(3)...Laranja\n\n");
        scanf("%d", &i);
    } while ((i<1)|| (i>3));

    switch (i)
    {
        case 1:
            printf ("\t\tVoce escolheu Mamão.\n");
            break;
        case 2:
            printf ("\t\tVoce escolheu Abacaxi.\n");
            break;
        case 3:
            printf ("\t\tVoce escolheu Laranja.\n");
            break;
    }
    return 0;
}

```

**Exemplo (do...while -1)**

## Resumo (Laços)

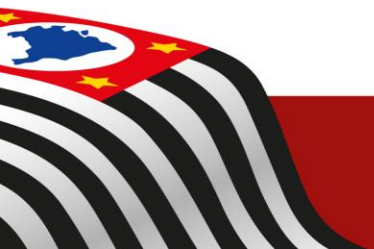
	<b>while</b>	<b>for</b>	<b>do ... while</b>
<b>Sintaxe</b>	while (cond) instrução	for (carga inic; cond ; pos-inst) instrução	do instrução while (condição)
<b>Executa a instrução</b>	zero ou mais vezes	zero ou mais vezes	1 ou mais vezes
<b>Testa a condição</b>	antes da instrução	antes da instrução	depois da instrução
<b>Utilização</b>	freqüente	freqüente	pouco freqüente





## Exercícios para prática

- 1) Escreva um algoritmo para gerar uma PA de **razão qualquer**, com uma série de 10 termos iniciando em 1.
- 2) Faça um programa que leia **X** números **N** onde ( $0 \leq N < 20$ ). Ao final, apresente a somatória dos mesmos. A condição de parada é a entrada de um valor 0.
- 3) Faça um programa que dados 10 números pelo usuário, verifique **quantos** são pares.
- 4) Escreva um algoritmo que gere uma tabela com a conversão de graus para Fahrenheit para Celsius, com valores variando de 1 em 1 grau, de 0 a 100 graus Celsius.  $[^{\circ}\text{C} = (^{\circ}\text{F} - 32) / 1,8]$
- 5) Construa um programa que leia 5 valores inteiros e:
  - Encontre o maior valor,
  - o menor valor e calcule
  - a média dos números lidos.



## Interrupções com break e continue

O comando **break**, quando utilizado dentro de um laço, interrompe e termina a execução do mesmo. A execução prossegue com os comandos subsequentes ao bloco.

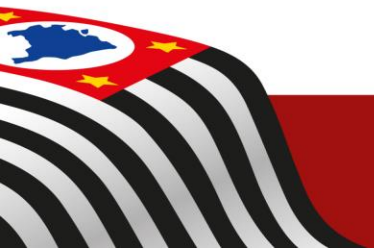
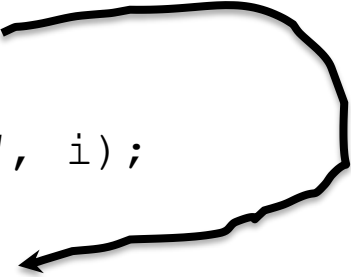
No próximo exemplo, se executado, a saída do programa será 0,1,2,3,4 pois, quando *i* tiver o valor 5, o laço é interrompido e finalizado.



## exemploBreak.c

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

int main(int argc, char *argv[]){
    int i;
    for(i=0; i<10; i++)
    {
        if(i==5) {
            break;
        }
        printf("%d\n", i);
    }
    return 0;
}
```



## Comando continue

Também interrompe a execução dos comandos de um laço.

A diferença básica em relação ao comando *break* é que o laço não é automaticamente finalizado.

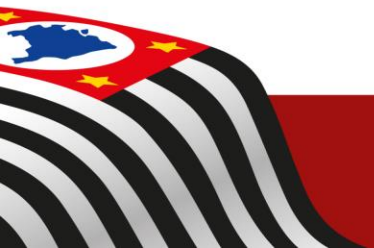
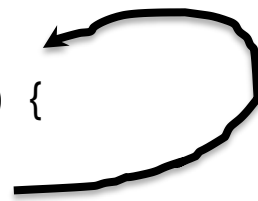
Observe o resultado da alteração do programa anterior, colocando o comando *continue*, ao invés do *break*.



## exemploContinue.c

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

int main(int argc, char *argv[])
{
    int i;
    for(i=0; i<10; i++) {
        if(i==5) {
            continue;
        }
        printf("%d\n", i);
    }
    return 0;
}
```





## Exercício com uso de interrupções

1) Faça um programa que imprima como resultado a tabuada de um número inteiro dado pelo usuário.

- Nota: os números resultantes múltiplos de 03 não deverão ser mostrados (interrompa e não liste). Entretanto a tabuada deve mostrar todos os outros elementos.

Exemplo:

- Tabuada do 5.
- $5 \times 1 = 5$ ,  $5 \times 2 = 10$ ,  $5 \times 4 = 20$  (note que  $5 \times 3 = 15$  que é múltiplo de 3, este não deverá ser listado).



## Operadores ++ e --

A linguagem C possui um conjunto de operadores particularmente úteis, que permitem a realização de **incremento** ou **decremento** de variáveis.

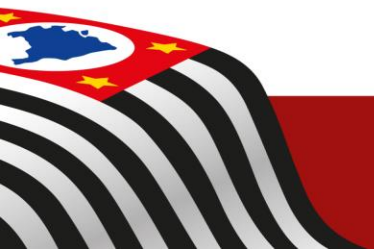
Veja os próximos exemplos.



## Comparando

```
1: #include <stdio.h>
2:
3: main()
4: {
5:     int i;
6:     i=1;
7:     while (i <= 10)
8:     {
9:         printf("%d\n",i);
10:        i = i+1;
11:    }
12: }
```

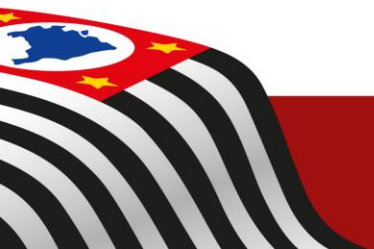
```
1: #include <stdio.h>
2:
3: main()
4: {
5:     int i;
6:     i=1;
7:     while (i <= 10)
8:     {
9:         printf("%d\n",i);
10:        i++;
11:    }
12: }
```



## Operadores

Operador	Significado	Exemplos
++	Incremento de 1	i++ , ++k
--	Decremento de 1	j-- , --alfa

Operador	Exemplo	Equivalente
++	x++ ou ++x	x = x + 1
--	x-- ou --x	x = x - 1



## Diferença entre ++x e x++

Quando se executa

---

**y = x++;**

---

Acontecem duas coisas, nessa ordem:

1. O valor de x é atribuído a y
  2. O valor de x é incrementado
- 

---

**y = ++x;**

---

Acontecem duas coisas, nessa ordem:

1. O valor de x é incrementado
  2. O valor de x é atribuído a y
- 

*Nota:*

**Quando o operador de incremento ou decremento está antes da variável, esta é operada antes de ser usada. Quando o operador está depois da variável, esta é usada e só depois é incrementada ou decrementada.**





## Exemplos

```
x=5;  
y=x++;
```

*Coloca o valor 5 na variável y.  
Em seguida incrementa a variável x.*

Valores finais:  $x \rightarrow 6$  e  $y \rightarrow 5$

```
x=5;  
y=++x;
```

*Incrementa o valor de x.  
Em seguida coloca o valor x na variável y.*

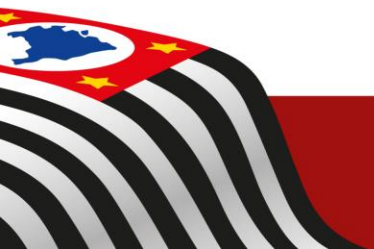
Valores finais:  $x \rightarrow 6$  e  $y \rightarrow 6$

**Como se pode observar, o valor final das variáveis não é o mesmo.**

Dessa forma, verificam-se as seguintes equivalências:

$y = x++;$	é equivalente a	$y = x;$ $x++;$
------------	-----------------	--------------------

$y = ++x;$	é equivalente a	$x++;$ $y = x;$
------------	-----------------	--------------------



## Atribuição composta

A linguagem C permite-nos reduzir a quantidade de código escrita sempre que se pretende que uma variável receba um valor que depende do valor que ela já tem.

### Exemplo:

```
x = x + 1;  
y = y * (a+5+b);  
z = z % 3;
```

Nessas situações, é desnecessário repetir o nome da variável no lado direito da atribuição. Vamos supor que queríamos adicionar 3 à variável x. Normalmente escreveríamos:

```
x = x + 3;
```

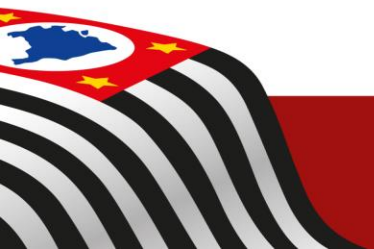
No entanto, usando uma atribuição composta bastaria escrever:

```
x += 3;
```



## Atribuição composta - Exemplos

Exemplo	Significado
$x += 1$	$x = x + 1$
$y *= 2+3$	$y = y * (2+3)$
$a -= b+1$	$a = a - (b+1)$
$k /= 12$	$k = k / 12$
$r \% = 2$	$r = r \% 2$



## Exercício Resolvido

Escreva um programa em C que escreva na tela toda a tabela ASCII (0..255 **chars**) , escrevendo a cada linha de código ASCII o caracter correspondente.

### Exemplo:

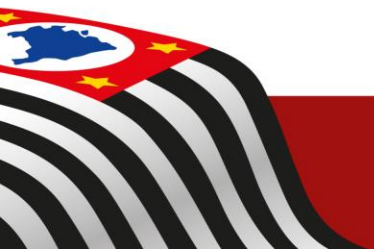
```
...  
65 --> A  
66 --> B  
67 --> C  
...
```



## Tabela ASCII

```
1: #include <stdio.h>
2:
3: main()
4: {
5:     int i;
6:     for (i=0 ; i<=255 ; i++)
7:         printf("%3d -> %c\n",i, (char) i);
8: }
```

**Agora, modifique o programa de forma que a cada 10 números mostrado, aconteça uma pausa automática na execução.**



# CENTRO PAULA SOUZA

---

## Fatec

Mogi Mirim  
Arthur de Azevedo

Prof. Me. Marcos Roberto de Moraes, o Maromo

# FIM



## Referências Bibliográficas

DAMAS, L. M. D. **Linguagem C**. LTC, 2007.

HERBERT, S. **C completo e total**. 3a. ed. Pearson, 1997.

