

Estruturas de Repetição

Algoritmos e Estruturas de Dados I

Prof. Lucas Astore

Prof. Cristiano Rodrigues

Problema

Como imprimir os 1000 primeiros números a partir do 1?

BEING RIGHT SUCKS
BEING RIGHT SUCKS
BEING RIGHT SUCKS
BEING RIGHT SUCKS
BEING RIGHT SUCKS
BEING RIGHT SUCKS
BEING RIGHT SUCKS
BEING RIGH

A yellow cartoon head of Bart Simpson, shown from the chest up, looking towards the left. He has his signature spiky hair and a small mischievous grin. The head is positioned in the bottom right corner of the image, partially overlapping the last row of text.

Problema

Como imprimir os 1000 primeiros números a partir do 1?

Uma solução seria:

```
escrever: 1;
```

```
...
```

```
escrever: 1000;
```

Problema

Como imprimir os 1000 primeiros números a partir do 1?

Uma solução seria:

```
escrever: 1;
```

...

```
escrever: 1000;
```

A solução acima não é prática!

Problema

Como imprimir os 1000 primeiros números a partir do 1?

Outra solução seria utilizar o comando enquanto (while).

while

Comando Enquanto

enquanto (expressão) **faça**

lista de comandos

fim enquanto

while: Comando Enquanto em C-like

```
while (expressão) {  
    lista de comandos  
}
```

• Como imprimir os 1000 primeiros números a partir do 1?

• Outra solução seria utilizar o comando enquanto (*while*)

```
int i = 1;  
  
while (i <= 1000){  
    escrever: i;  
    i++;  
}
```

Exercício

- Como imprimir os 1000 primeiros números a partir do 1?

- Outra solução seria utilizar o comando enquanto (*while*)

```
int i = 1;  
  
while (i <= 1000){  
    escrever: i;  
    i++;  
}
```

TELA

QUADRO DE MEMÓRIA

QUADRO DE MEMÓRIA	
i	

Exercício

- Como imprimir os 1000 primeiros números a partir do 1?

- Outra solução seria utilizar o comando enquanto (*while*)

```
int i = 1;  
  
while (i <= 1000){  
    escrever: i;  
    i++;  
}
```

TELA
1
2
3

QUADRO DE MEMÓRIA				
i	1	2	3	4

Exercício

- Como imprimir os 1000 primeiros números a partir do 1?

- Outra solução seria utilizar o comando enquanto (*while*)

```
int i = 1;  
  
while (i <= 1000){  
    escrever: i;  
    i++;  
}
```

TELA
1
2
3
...
999
1000

QUADRO DE MEMÓRIA						
i	1	2	3	4	...	1000

Exercício

- Como imprimir os 1000 primeiros números a partir do 1?

- Outra solução seria utilizar o comando enquanto (*while*)

```
int i = 1;  
  
while (i <= 1000){  
    escrever: i;  
    i++;  
}
```

TELA
1
2
3
...
999
1000

QUADRO DE MEMÓRIA						
i	1	2	3	4	...	1000

Exercício

- Como imprimir os 1000 primeiros números a partir do 1?

- Outra solução seria utilizar o comando enquanto (*while*)

```
int i = 1;  
  
while (i <= 1000){  
    escrever: i;  
    i++;  
}
```

TELA

1
2
3
...
999
1000

QUADRO DE MEMÓRIA

i	1	2	3	4...	1000	1001
---	---	---	---	------	------	------

Exercício

- Como imprimir os 1000 primeiros números a partir do 1?

- Outra solução seria utilizar o comando enquanto (*while*)

<code>int i = 1;</code>
<code>while (i <= 1000){</code>
<code> escrever: i;</code>
<code> i++;</code>
<code>}</code>

false

Término da execução do while

TELA

1

2

3

•••

999

1000

QUADRO DE MEMÓRIA

i	1	2	3	4	•••	1000	1001
---	---	---	---	---	-----	------	------

Exercício

- Faça o quadro de memória e mostre a saída na tela para o código abaixo

```
int num = 0;  
  
while (num < 4){  
    escrever: num++, " ";  
    num += 2;  
}
```

Exercício

- Faça o quadro de memória e mostre a saída na tela para o código abaixo

```
int num = 0;  
  
while (num < 4){  
    escrever: num++, " ";  
    num += 2;  
}
```

TELA
0
3


QUADRO DE MEMÓRIA	
num	0 1 3 4 6

Exercício

- Faça o quadro de memória e mostre a saída na tela para o código abaixo

```
int num = 0;  
  
while (num < 4){  
    escrever: num++, " ";  
    num += 2;  
}
```

false



TELA
0
3

QUADRO DE MEMÓRIA	
num	0 1 3 4 6

Algumas observações

- O corpo de um while pode ter zero, um ou n comandos
- A lista de comandos será executada zero ou mais vezes
- O { e o } são obrigatório apenas se tivermos mais de um commando. Uma boa prática de programação consiste em sempre usá-los.

Exercício

Faça um programa que mostre os 10 primeiros números inteiros positivos.

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

```
int main() {
```

```
    int i = 1;
```

```
    printf("Os 10 primeiros num inteiros positivos são:\n");
```

```
    while (i <= 10) {
```

```
        printf("%d\n", i++);
```

```
    }
```

```
    return 0;
```

```
}
```

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

```
int main() {
```

```
    int i = 1;
```

```
    printf("Os 10 primeiros num inteiros positivos são:\n");
```

```
    while (i <= 10) {
```

```
        printf("%d\n", ++i);
```

```
    }
```

```
    return 0;
```

```
}
```

O código está correto?

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

```
int main() {
```

```
    int i = 1;
```

```
    printf("Os 10 primeiros num inteiros positivos são:\n");
```

```
    while (i <= 10) {
```

```
        printf("%d\n", i++);
```

```
    }
```

```
    return 0;
```

```
}
```

O código está correto?

Exercícios

Faça um programa que leia um número inteiro N e mostre na tela os N primeiros números inteiros ímpares.

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

```
int main() {
```

```
    int N, i = 1;
```

```
    printf("Digite um número inteiro: ");
```

```
    scanf("%d", &N);
```

```
    printf("Os %d primeiros números inteiros ímpares são:\n", N);
```

```
    while (N > 0) {
```

```
        if (i % 2 != 0) {
```

```
            printf("%d\n", i);
```

```
            N--;
```

```
        }
```

```
        i++;
```

```
    }
```

```
    return 0;
```

```
}
```

Exercícios

Faça um programa que leia um número inteiro N e mostre na tela os N primeiros números da sequência:

1, 5, 12, 16, 23, 27, 34...

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

```
int main() {
```

```
    int N, i, num = 1;
```

```
    printf("Digite um número inteiro N: ");
```

```
    scanf("%d", &N);
```

```
    printf("Os %d primeiros números da sequência são:\n", N);
```

```
    i = 0;
```

```
    while (i < N) {
```

```
        printf("%d ", num);
```

```
        if (i%2==0){
```

```
            num+=4;
```

```
        }
```

```
        else{
```

```
            num+=7;
```

```
        }
```

```
        i++;
```

```
    }
```

```
    return 0;
```

```
}
```

Exercício

Faça um programa que leia um número inteiro N indicando o valor de uma prova P . Em seguida, leia a nota de 20 alunos (entre 0 e N) e mostre na tela:

- a. A média da turma
- b. O número de alunos cuja nota foi menor que a média da Universidade (suponha 60%)
- c. A quantidade de alunos com conceito A (mais de 90%)

Exercício

Faça um programa que leia um número inteiro n e mostre na tela o n -ésimo termo da sequência de Fibonacci.

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

```
int main() {
```

```
    int n, i = 2, fib;
```

```
    int termo_anterior = 0;
```

```
    int termo_atual = 1;
```

```
    printf("Digite n: ");
```

```
    scanf("%d", &n);
```

```
    if (n <= 0) {
```

```
        printf("Erro!\n");
```

```
        return 0;
```

```
    }
```

```
    if (n == 1) {
```

```
        printf("O 1o termo da seq. Fibonacci é: 0\n");
```

```
        return 0;
```

```
    }
```

```
while (i <= n) {
```

```
    fib = termo_anterior + termo_atual;
```

```
    termo_anterior = termo_atual;
```

```
    termo_atual = fib;
```

```
    i++;
```

```
}
```

```
printf("O %d-ésimo termo da sequência de Fibonacci  
é: %d\n", n, termo_atual);
```

```
return 0;
```

```
}
```

do-while

Comando do-while (faça enquanto..)

- Similar ao comando while
- A diferença entre eles é o momento em que a expressão é avaliada
- No do-while (faça-enquanto), a lista de comandos é executada e, somente depois, a expressão é avaliada

Comando Faça – Enquanto

faça

|

lista de comandos

enquanto (expressão);

do-while: Comando Faça – Enquanto

do {

lista de comandos

} while (expressão);

Exercício

Qual o problema aqui?

Leia dois números reais e realize a divisão entre eles.

Exercício

Leia dois números reais e realize a divisão entre eles.

```
double num1, num2;

escrever: Digite o primeiro número: ";
ler num1;

do {
    escrever: Digite o segundo número: ";
    ler num2;
while (num == 0);

escrever: Divisao: ", (num1 / num2);
```

Exercício

Qual a restrição do problema?

Leia um número inteiro e garante que ele representa um mês válido.

Exercício

Leia um número inteiro e garante que ele representa um mês válido.

```
int mes;  
  
do {  
    ler mes;  
while ((mes >= 1 && mes <= 12) == false);
```

Exercício

Faça um programa que leia um número inteiro e garanta que ele é um ano bissexto.

Avaliando o problema

O primeiro ano bissexto da história foi o ano 8 d.C. no calendário juliano. Este foi o primeiro ano em que a prática de adicionar um dia extra ao calendário a cada quatro anos foi implementada.

No entanto, é importante notar que o conceito de anos bissextos foi refinado ao longo do tempo, e o calendário gregoriano, que é o calendário usado atualmente pela maioria das partes do mundo, foi introduzido em 1582 pelo Papa Gregório XIII para corrigir imprecisões no calendário juliano.

Avaliando o problema

Para determinar se o ano 200 foi um ano bissexto, devemos seguir as regras do calendário gregoriano:

- Se o ano for divisível por 4, é um ano bissexto, exceto:
- Se o ano for divisível por 100, não é um ano bissexto, exceto:
- Se o ano for divisível por 400, é um ano bissexto.

Exercício

Faça um programa que leia um número inteiro e garanta que ele é um ano bissexto.

Exercício

Faça um programa que leia um número inteiro e garanta que ele é um ano bissexto.

```
int ano;  
  
do {  
    ler ano;  
} while ((ano % 4 == 0 && ano % 100 != 0 || ano % 400 == 0) == false);
```

for

Comando Para

- Similar ao comando enquanto, contudo, ele permite: pré-comandos (início) e pós-comandos (incremento)

Comando Para

- Similar ao comando enquanto, contudo, ele permite: pré-comandos (início) e pós-comandos (incremento)

```
início;  
  
while (expressão) {  
    lista de comandos;  
    incremento;  
}
```

Comando Para

- Similar ao comando enquanto, contudo, ele permite: pré-comandos (início) e pós-comandos (incremento)

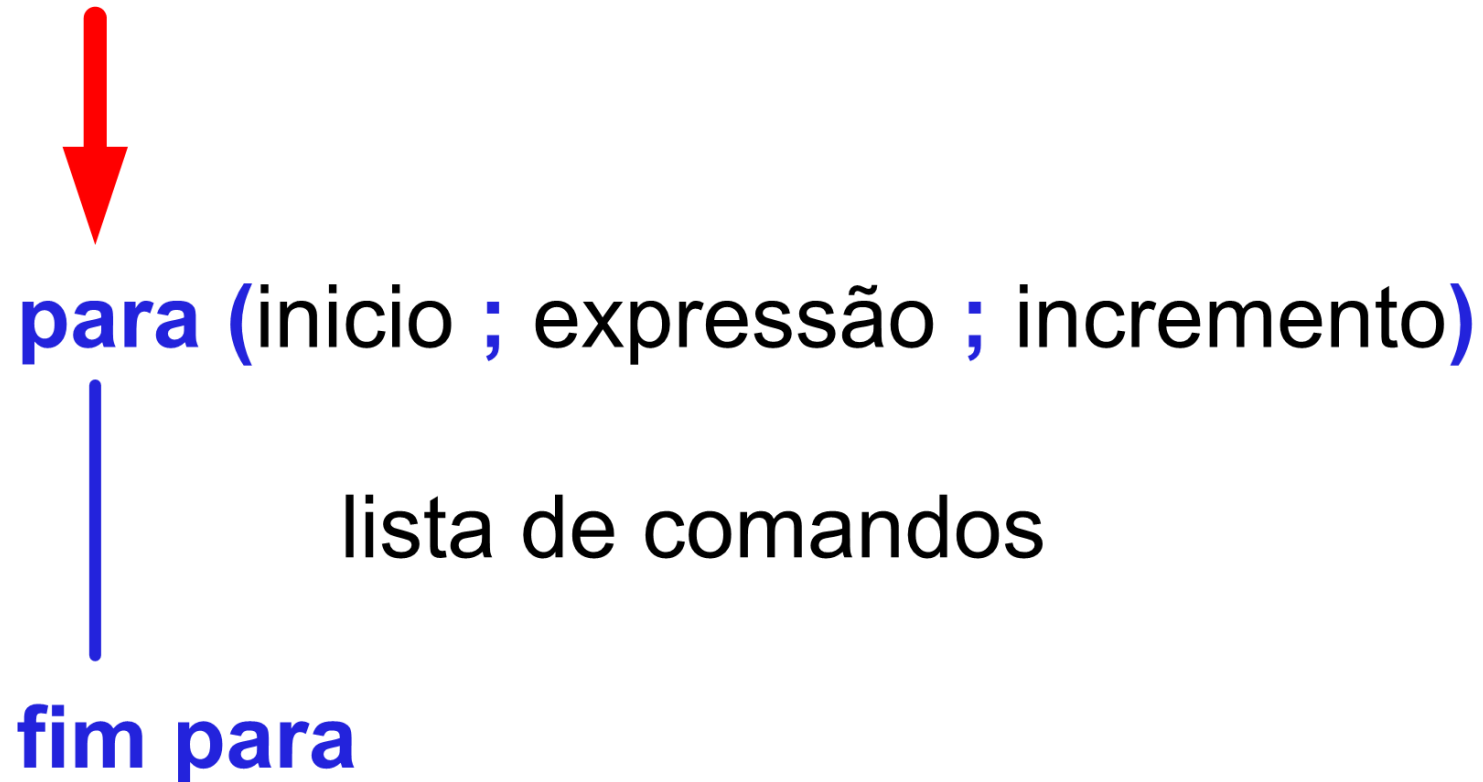
```
início;  
  
while (expressão) {  
    lista de comandos;  
  
    incremento;  
}
```

```
int i = 1;  
  
while (i <= 1000){  
    escrever: i;  
  
    i++;  
}
```



```
para (início ; expressão ; incremento)  
    |  
    lista de comandos  
fim para
```

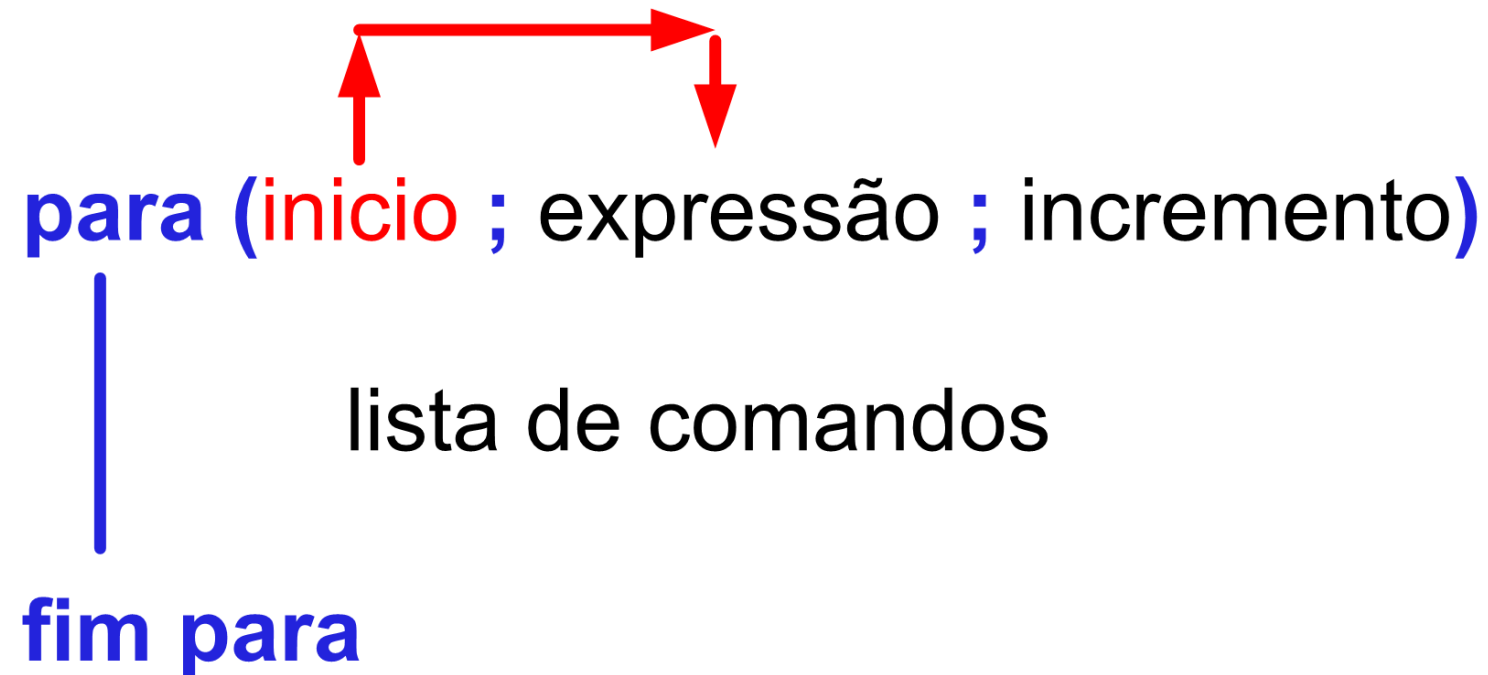
Comando Para



para (início ; expressão ; incremento)
|
lista de comandos
fim para

```
para (início ; expressão ; incremento)  
    |  
    lista de comandos  
fim para
```

Comando Para



The diagram illustrates the syntax of a 'para' (for) loop. The word 'para' is in blue. The opening parenthesis contains 'início' in red, followed by a semicolon, 'expressão' in blue, another semicolon, and 'incremento' in blue, all enclosed in a closing parenthesis. A red arrow points up to 'início', a red arrow points right to 'expressão', and a red arrow points down from 'expressão'. A vertical blue line descends from 'para' to the word 'fim para' at the bottom. The text 'lista de comandos' is positioned to the right of the vertical line.

```
para (início ; expressão ; incremento)  
    lista de comandos  
fim para
```

```
para (início ; expressão ; incremento)
    |
    lista de comandos
fim para
```

Comando Para



para (inicio ; expressão ; incremento)

lista de comandos

fim para

Comando Para

```
para (início ; expressão ; incremento)
    ↓ true
    lista de comandos
fim para
```

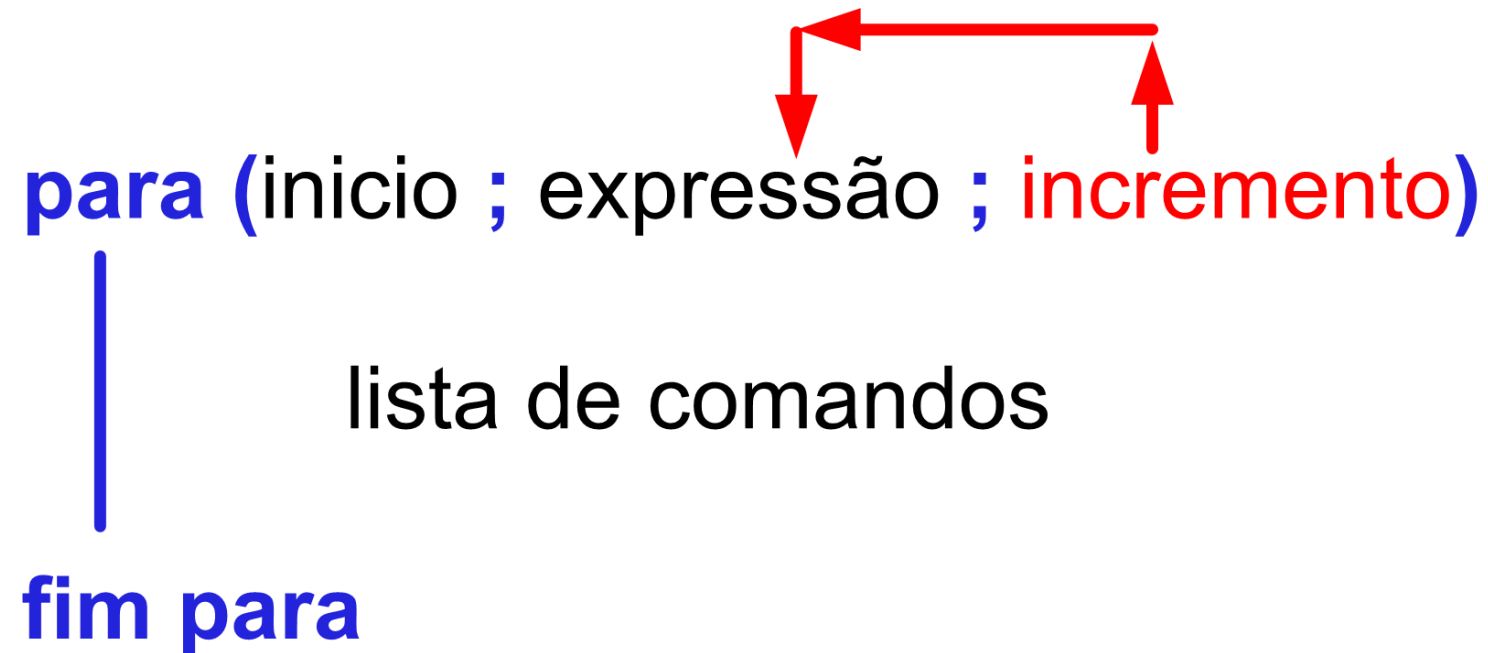
```
para (início ; expressão ; incremento)  
    |  
    lista de comandos  
fim para
```


Comando Para

para (início ; expressão ; incremento)
|
lista de comandos → ↑
fim para

```
para (início ; expressão ; incremento)  
    |  
    lista de comandos  
fim para
```

Comando Para



The diagram illustrates the syntax of a 'para' loop. The word 'para' is in blue. The opening parenthesis follows. 'início' is in black, followed by a semicolon. 'expressão' is in black, followed by a semicolon. 'incremento' is in red. A red arrow points from the 'incremento' to the 'expressão', and another red arrow points from the 'expressão' down to the 'lista de comandos'. A blue vertical line connects 'para' to 'fim para'.

```
para (início ; expressão ; incremento)  
    lista de comandos  
fim para
```

Comando Para



para (inicio ; expressão ; incremento)

lista de comandos

fim para

Comando Para

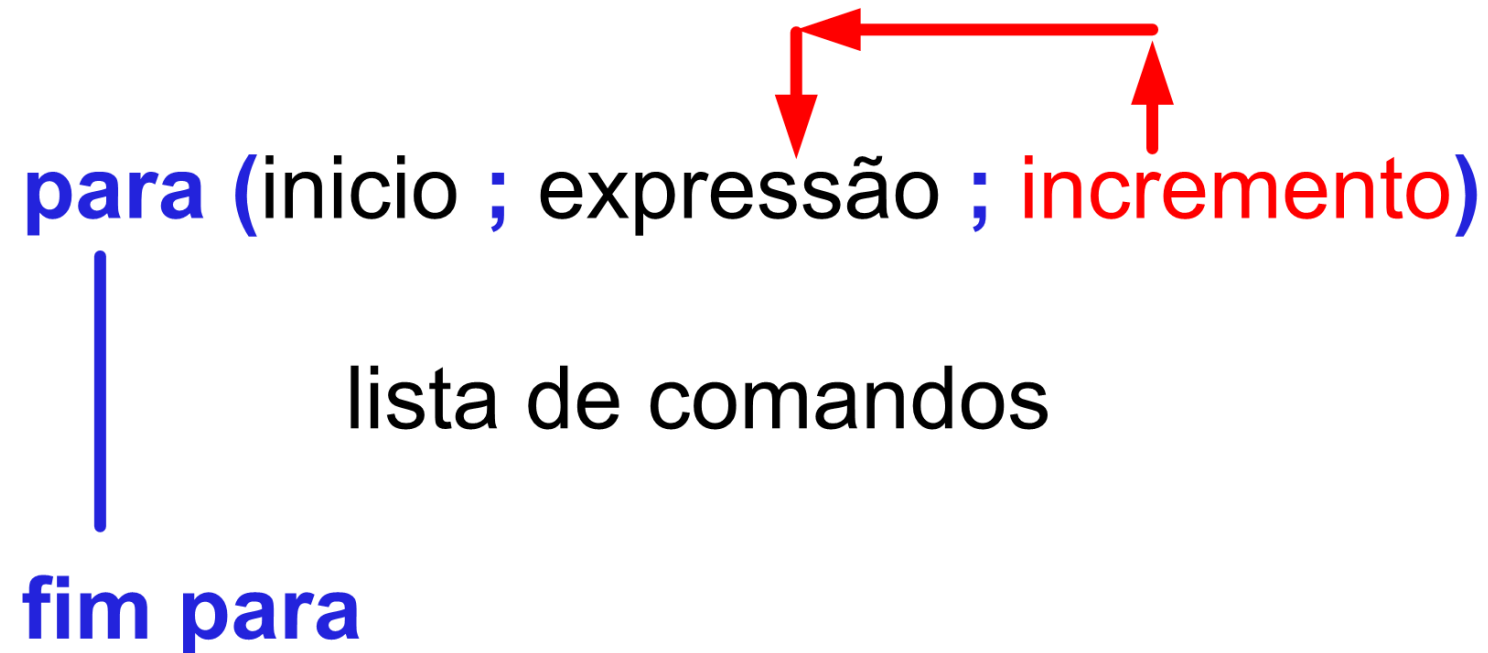
```
para (início ; expressão ; incremento)
    ↓ true
    lista de comandos
fim para
```

```
para (início ; expressão ; incremento)  
    |  
    lista de comandos  
fim para
```

para (início ; expressão ; incremento)
|
lista de comandos → ↑
fim para

```
para (início ; expressão ; incremento)  
    |  
    lista de comandos  
fim para
```


Comando Para

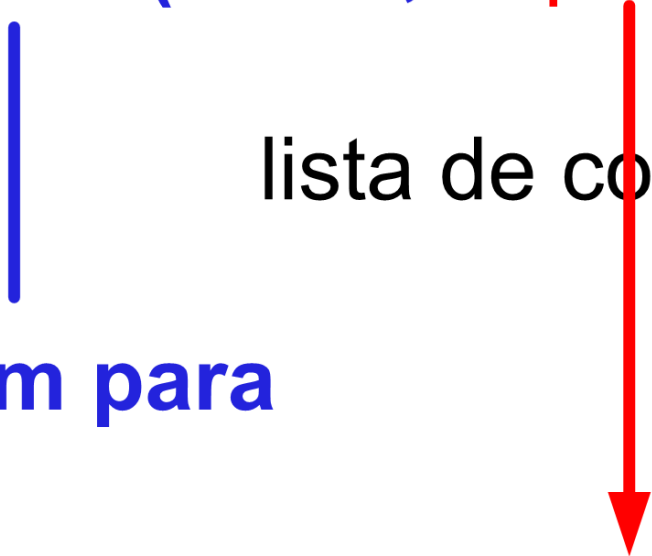


The diagram illustrates the syntax of a 'para' loop. The word 'para' is in blue. The opening parenthesis follows. 'início' is in black, followed by a semicolon. 'expressão' is in black, followed by a semicolon. 'incremento' is in red. A red arrow points from the 'incremento' to the 'expressão', and another red arrow points from the 'expressão' down to the 'lista de comandos'. A blue vertical line connects 'para' to 'fim para'.

```
para (início ; expressão ; incremento)  
    lista de comandos  
fim para
```

para (início ; **expressão** ; incremento)
|
fim para

false
lista de comandos



for: Comando Para em C-like

```
for (início ; expressão ; incremento) {  
    lista de comandos  
}
```

- Como imprimir os 3 primeiros números a partir do 1?

```
for ( int i = 1 ; i <= 3 ; i++ ){  
    escrever: i;  
}
```

- Como imprimir os 3 primeiros números a partir do 1?

```
for ( int i = 1 ; i <= 3 ; i++ ){  
    escrever: i;  
}
```

TELA
1
2
3

QUADRO DE MEMÓRIA				
i	1	2	3	4

- Como imprimir os 3 primeiros números a partir do 1?

```
for ( int i = 1 ; i <= 3 ; i++ ){  
    escrever: i;  
}
```

TELA
1
2
3

QUADRO DE MEMÓRIA				
i	1	2	3	4

Exercício

- Faça o quadro de memória e mostre a saída na tela para o código abaixo :

```
for ( int i = 0 ; i < 10 ; i += 2 ){  
    escrever: i++, " ";  
}
```

TELA

QUADRO DE MEMÓRIA	
i	

Exercício

- Faça um programa para ler um número inteiro n e mostrar na tela os n primeiros números inteiros e positivos

```
int n;  
  
ler n;  
  
for (int i = 1; i < n; i += 1){  
    escrever: i;  
}
```


- Faça um programa para imprimir os quatro primeiros múltiplos de cinco

```
for ( int i = 0, valor = 5 ; i < 4 ; i += 1, valor += 5 ){  
    escrever: valor, " ";  
}
```

Exercício

- Faça um programa para ler dois números inteiros e multiplicá-los sem utilizar a operação de multiplicação

Algoritmo

```
int n1, n2, i, soma;  
escrever: "Digite o multiplicando: ";  
ler n1;  
escrever: "Digite o multiplicador: ";  
ler n2;  
for(int i = 1, soma = 0; i <= n2; i++, soma = soma + n1);  
escrever: "Produto: ", soma;
```

Fim Algoritmo

- Faça um programa para ler um número inteiro n e mostrar na tela os n primeiros termos da sequência 1, 5, 12, 16, 23, 27, 34, ... que sejam divisíveis por dois e não divisíveis por três

• Faça um programa para ler um número n do teclado e calcular o n -ésimo termo da sequência de Fibonacci (1, 1, 2, 3, 5, 8, 13, 21, ...) utilizando:

- (a) o comando **for**;
- (b) o comando **do-while**.

🔵 **Desafio:** Sabendo que os alunos de uma turma com N_A fizeram N_P provas, faça um programa para ler cada uma das N_P provas feitas por cada um dos N_A alunos e mostrar na tela (i) a média de cada aluno, (ii) a média da turma e o (iii) percentual dos alunos cuja média foi maior ou igual a 80%

Resumo

Estruturas de Repetição

Existem situações nas quais a lógica representada nos comandos torna-se repetitiva em sequência, mudando apenas o conteúdo de variáveis e valores.

Pode-se então utilizar **estruturas de repetição** com **iterações (loops)** vinculadas ao resultado de determinada **condição**.

As repetições continuam enquanto a condição for avaliada como **verdadeira**.

A partir do momento em que a condição for avaliada como **falsa**, os comandos não serão executados e o fluxo de execução continua para o primeiro comando após a estrutura de repetição. Se isso não ocorrer, tem-se um **loop infinito**.

Tipos: **while**, **do-while** (indeterminadas ou não fixas)

for (determinada segundo variável de controle)

while (enquanto)

A avaliação da expressão condicional ocorre **antes** da execução do bloco de comandos, por isso se diz que o **teste é feito no início** da repetição. Proibido uso de ; após o término da condição, caso queira que o bloco seja executado.

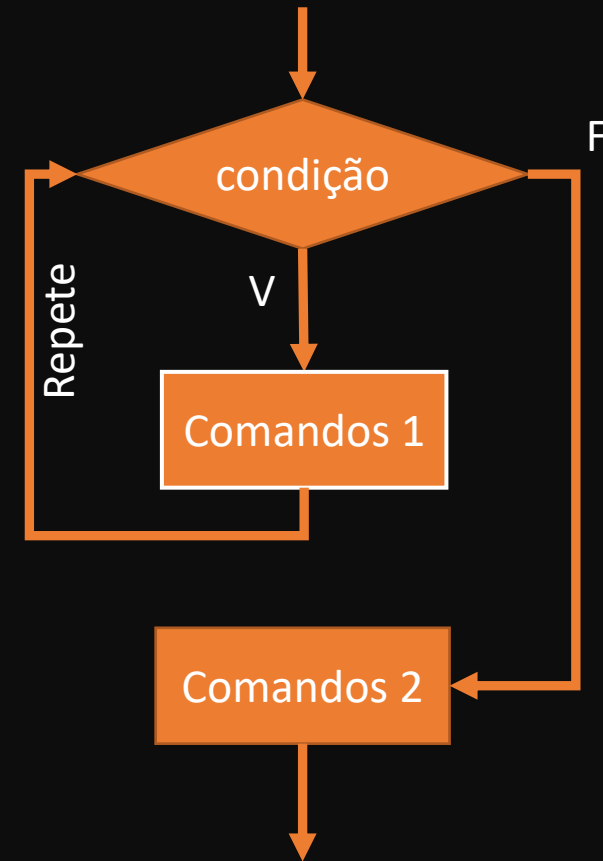
Se ela for **verdadeira**, o bloco de comandos é executado novamente e após o seu término, avalia-se novamente a condição. **Caso contrário**, ele é ignorado.

Formatos

```
while (condição)
{
    ... comandos 1
}
comandos 2
```

```
i = 0;
while (x > 10)
{
    i++;
    x--;
}
```

Nesse ponto, quanto vale 'i' se 'x' for 20 antes do while?



do-while (faça-enquanto)

A avaliação da expressão condicional ocorre **após** a execução do bloco de comandos, por isso se diz que o **teste é feito no final** da repetição. Obrigatório ; após o término da condição.

Se ela for **verdadeira**, o bloco de comandos é executado novamente e após o seu término, avalia-se novamente a condição. **Caso contrário**, ele não será repetido.

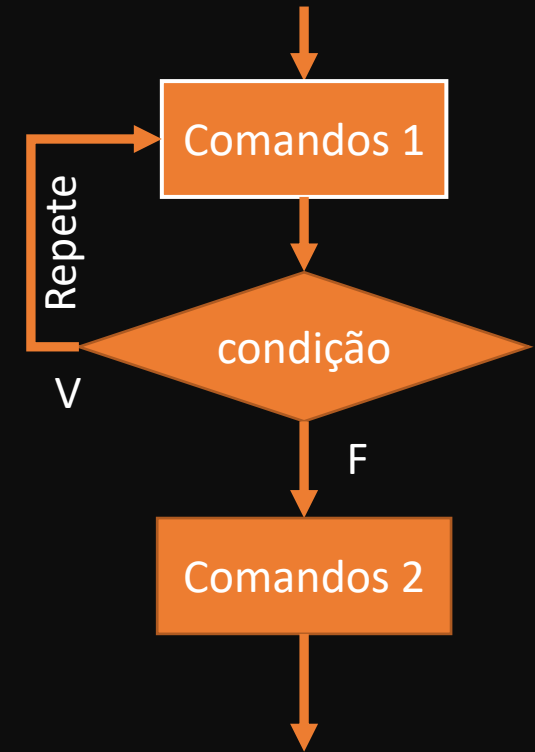
Pelo menos uma vez ocorre a execução do bloco, independente do resultado da condição.

Formatos

```
do
{
    ... comandos 1
} while (condição);
comandos 2
```

```
i = 0;
do
{
    i++;
    x--;
} while (x > 10);
```

Nesse ponto, quanto vale 'i' se 'x' for 20 antes do while?



for (para)

Estrutura composta de 3 expressões entre os parênteses: **inicialização** da variável de controle (outras variáveis, caso necessário), **condição**, **atualização** da variável de controle (outras variáveis, caso necessário). Proibido uso de ; após o término da condição, caso queira que o bloco seja executado.

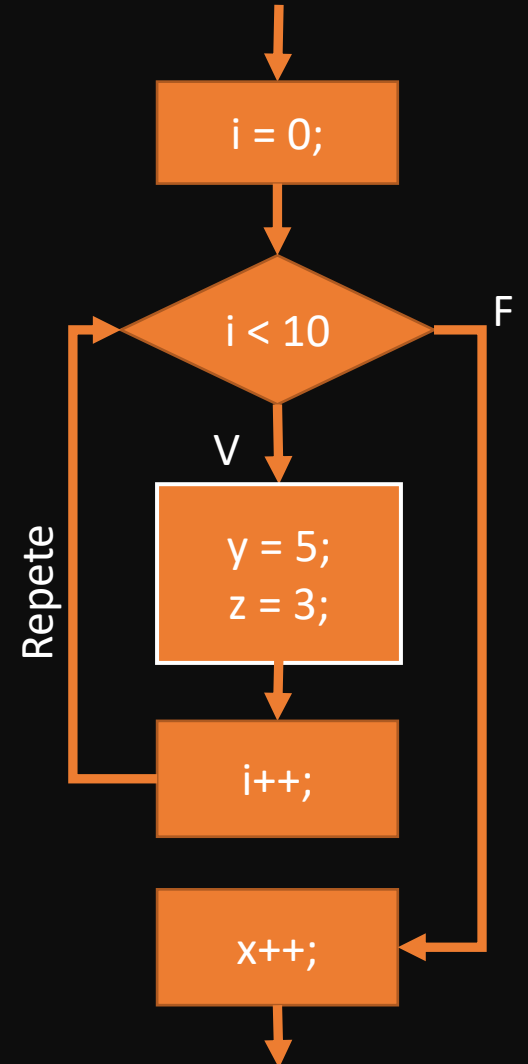
A **inicialização** é feita *uma única vez*, no *início* da execução da estrutura. Para a execução do bloco de comandos, avalia-se a **condição**. Se ela for verdadeira, o bloco é executado. Caso contrário, ele é ignorado. Após cada término de execução do bloco, a **atualização** é realizada.

for (inicialização; condição; atualização)

```
{  
    ... comandos 1  
}  
comandos 2
```

for (int i = 0; i < 10; i++)

```
{  
    y = 5;  
    z = 3;  
}  
x++;
```



Estruturas Aninhadas

Como dentro de qualquer bloco podem ser inseridos comandos, a estrutura de repetição e a condicional também são comandos.

Dessa forma podem existir estruturas condicionais e de repetição dentro de outras estruturas de repetição, quantos níveis forem necessários.

```
for (int i = 0; i < 3; i++)  
{  
    for (int j = 0; j < 5; j++)  
    {  
        printf("%d e %d", i, j);  
    }  
}
```

