PUC Minas

Estruturas Condicionais

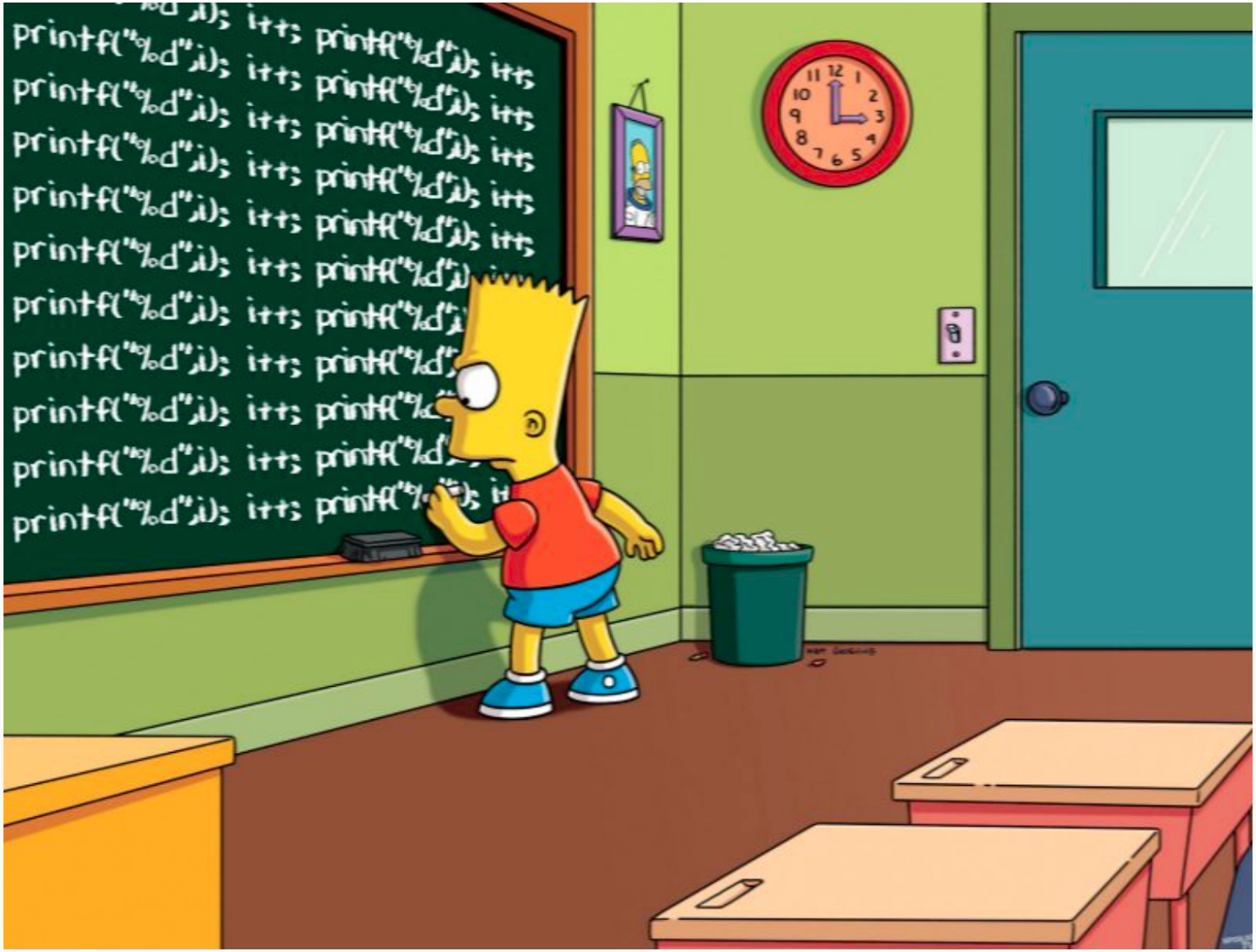
João Pedro Oliveira Batisteli

Fevereiro, 2025

Estruturas de Repetição

Como imprimir os 1000 primeiros números a partir do 1?

Estruturas de Repetição



Estruturas de Repetição

Como imprimir os 1000 primeiros números a partir do 1?

```
printf("1");
printf("2");
printf("3");
printf("4");
      ...
      ...
printf("1000");
```

Estruturas de Repetição

Como imprimir os 1000 primeiros números a partir do 1?

```
printf("1");
printf("2");
printf("3");
printf("4");
      ...
      ...
printf("1000");
```

NADA PRÁTICO

Estruturas de Repetição

Como imprimir os 1000 primeiros números a partir do 1?

Estruturas de Repetição

Como imprimir os 1000 primeiros números a partir do 1?

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

Estruturas de Repetição

Como imprimir os 1000 primeiros números a partir do 1?

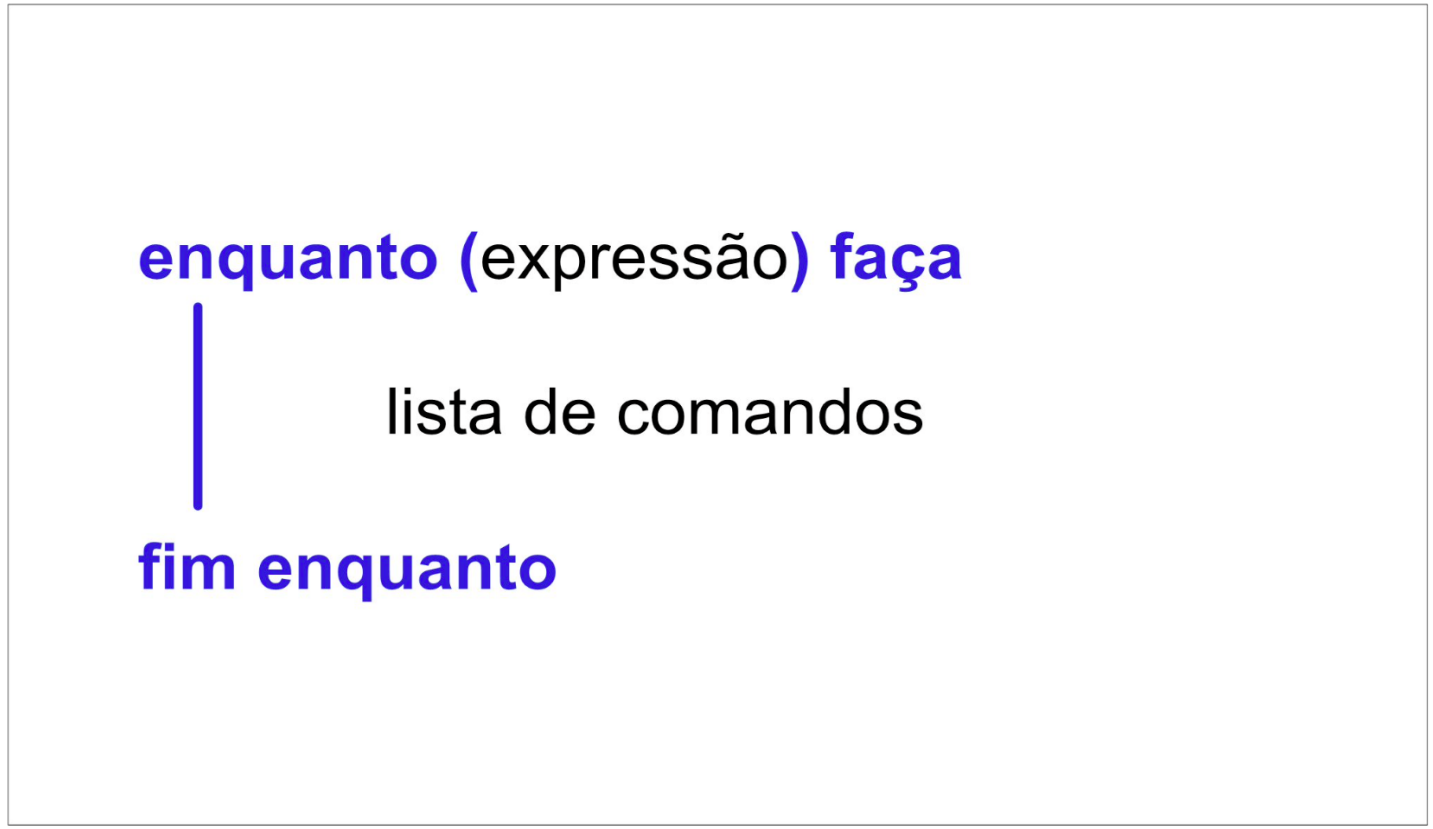
- Existem situações nas quais a lógica representada nos comandos torna-se repetitiva em sequência, mudando apenas o conteúdo de variáveis e valores.
- Uma solução para esses problemas é o uso de uma estrutura de repetição.

Estruturas de Repetição

Como imprimir os 1000 primeiros números a partir do 1?

- Existem situações nas quais a lógica representada nos comandos torna-se repetitiva em sequência, mudando apenas o conteúdo de variáveis e valores.
- Uma solução para esses problemas é o uso de uma estrutura de repetição.
- Um exemplo é o comando *Enquanto* - *while*.

Comando WHILE (enquanto)



Comando WHILE (enquanto)

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

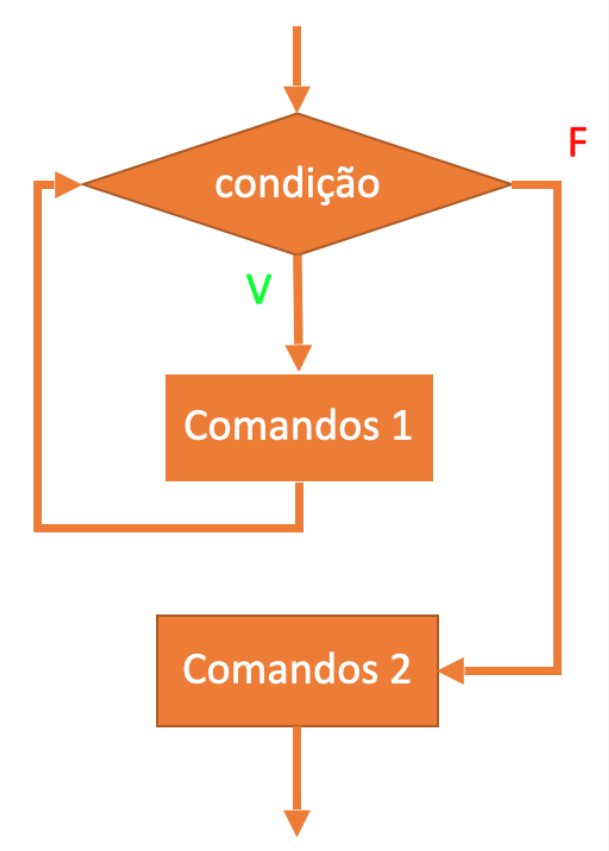
Comando WHILE (enquanto)

- 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.

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

Comando WHILE (enquanto)

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



Comando WHILE (enquanto)

Como imprimir os 1000 primeiros números a partir do 1?

Comando WHILE (enquanto)

Como imprimir os 1000 primeiros números a partir do 1?

```
int i=1;
while(i<=1000){
    printf("%d",i);
    i++;
}
```

Execução do comando while

```
int i = 1;

while (i <= 1000){
    printf("%d",i);
    i++;
}
```

Memória	
i	
1 2 3 4 5 ... ... 999 1000 1001	

Tela
1 2 3 4 5 ... ... 999 1000

Exercício

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

```
int i= 2;

while(i<=10){
    printf("\n%d",(i-1));
    i+=3;
}
```

Memória	
i	

Tela

Comando DO-WHILE

- Similar ao comando *while*.

Comando DO-WHILE

- Similar ao comando *while*.
- A diferença entre eles é o **momento em que a expressão é avaliada**.

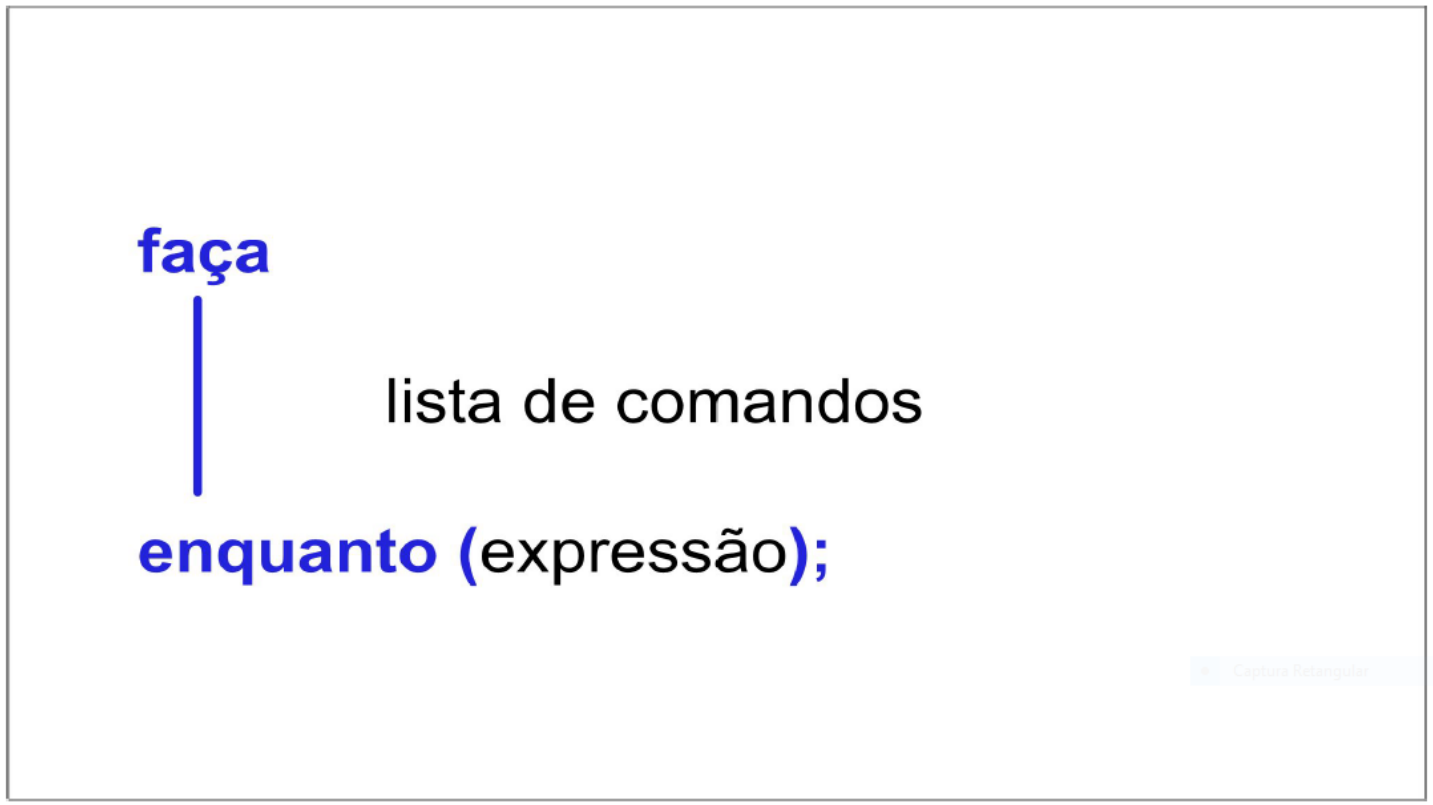
Comando DO-WHILE

- Similar ao comando *while*.
- A diferença entre eles é o **momento em que a expressão é avaliada**.
- No *while*, a expressão é avaliada antes da execução dos comandos.

Comando DO-WHILE

- Similar ao comando *while*.
- A diferença entre eles é o **momento em que a expressão é avaliada**.
- No *while*, a expressão é avaliada antes da execução dos comandos.
- No *do-while*, a lista de comandos é executada e, depois, a expressão é avaliada.

Comando DO-WHILE



Comando DO-WHILE

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

Comando DO-WHILE

Qual será a saída dos seguintes códigos?

```
int i= 100;
while(i<=10){
    printf("\n%d",i);
    i+=10;
}
```

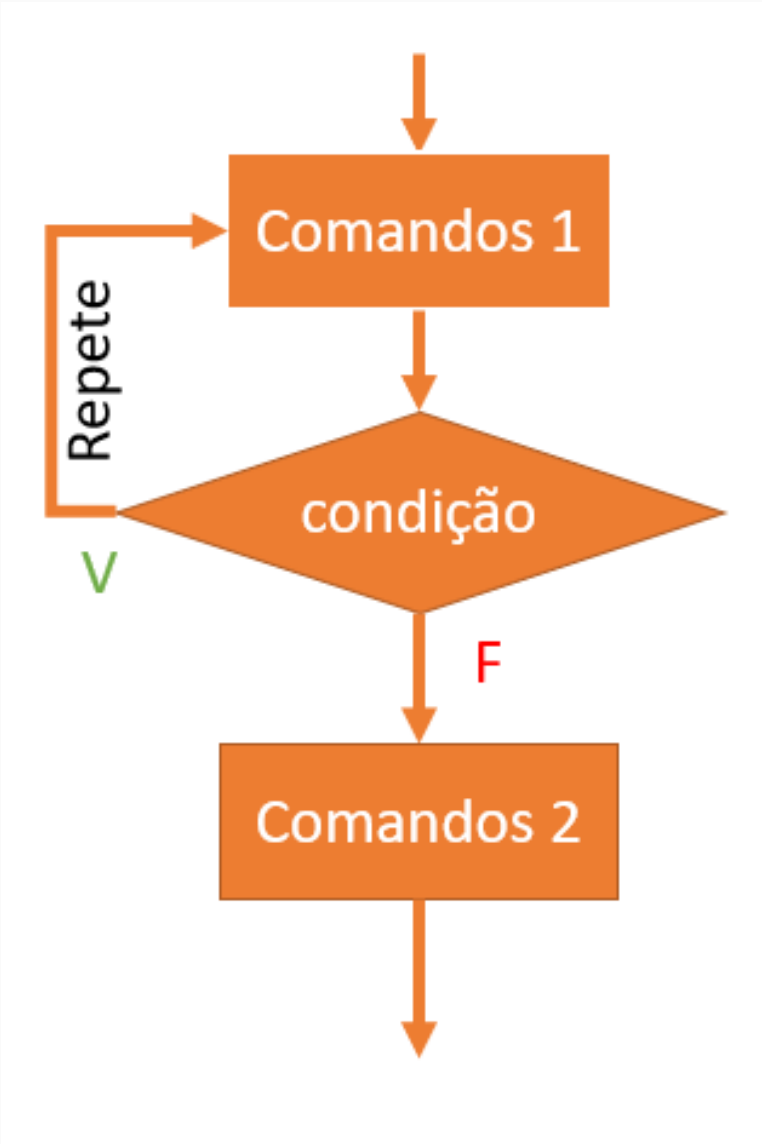
```
int i= 100;
do{
    printf("\n%d",i);
    i+=10;
}while(i<=10);
```

Comando DO-WHILE

- 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 o ";" após o término da condição.
- Se a condição for verdadeira, o bloco de comandos 1 é executado novamente e após o seu término, avalia-se novamente a condição.
- O bloco de comandos 1 será executado pelo menos uma vez, independente do resultado da condição.

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

Comando DO-WHILE




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

Exercício

- Após a execução do seguinte código, qual o valor de i?

```
int i=0;
int x=20;
do{
  i++;
  x--;
} while(x>10);
```

Exercício

Leia um número inteiro e garanta que ele é um mês válido.

Exercício

Leia um número inteiro e garanta que ele é um mês válido.

```
int main(){
  int mes;
  do{
    printf("Insira um mês válido:");
    scanf("%d",&mes);
  }while(mes < 1 || mes > 12);
  printf("O mes digitado foi %d", mes);
  return 0;
}
```

Comando - For

Como seria o código usando *while* e *do-while* caso quiséssemos executar algum comando por um número fixo de vezes (por exemplo 10 vezes)?

Comando - For

Como seria o código usando *while* e *do-while* caso quiséssemos executar algum comando por um número fixo de vezes (por exemplo 10 vezes)?

do-while:

```
int main(){
  int i=0;
  do{
    //faz algo
    i++;
  }while(i<10);
  return 0;
}
```

while:

```
int main(){
  int i=0;
  while(i<10){
    //faz algo
    i++;
  }
  return 0;
}
```

Comando - FOR

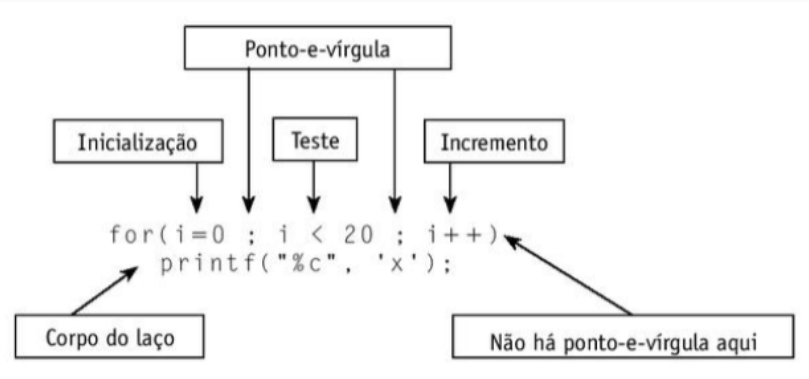
- O *for* é geralmente usado quando queremos repetir algo por um número fixo de vezes.

Comando - FOR

- O *for* é geralmente usado quando queremos repetir algo por um número fixo de vezes.
- Utilizaremos o *for* quando sabemos de antemão o número de vezes a repetir.

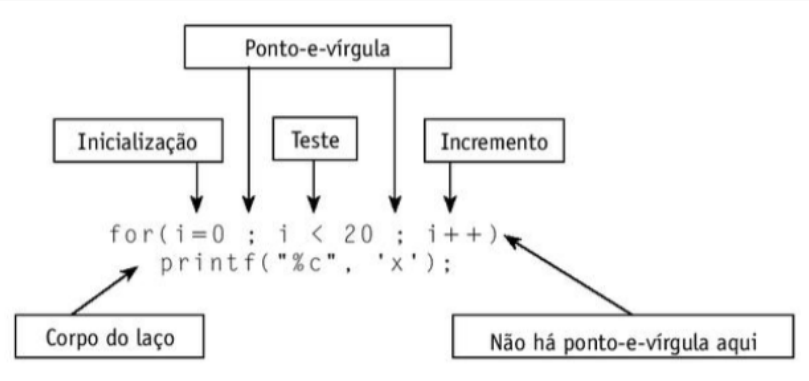
Comando - FOR

- A sintaxe consiste na palavra-chave **for** seguida de parênteses que contêm três expressões separadas por ";".
- A primeira expressão é chamada de *inicialização*, a segunda é chamada de *teste* e a terceira de *incremento*.



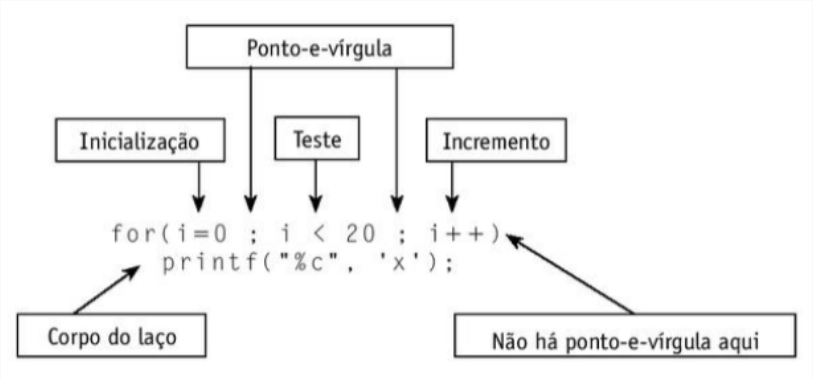
Comando - FOR

- A inicialização é uma instrução de atribuição (**i=0**) e é executado apenas uma vez antes que a repetição comece.
- O teste é a uma condição avaliada como *verdadeira* ou *falsa*, e controla o laço (**i<20**). Essa expressão é avaliada toda vez que o laço é iniciado ou reiniciado.



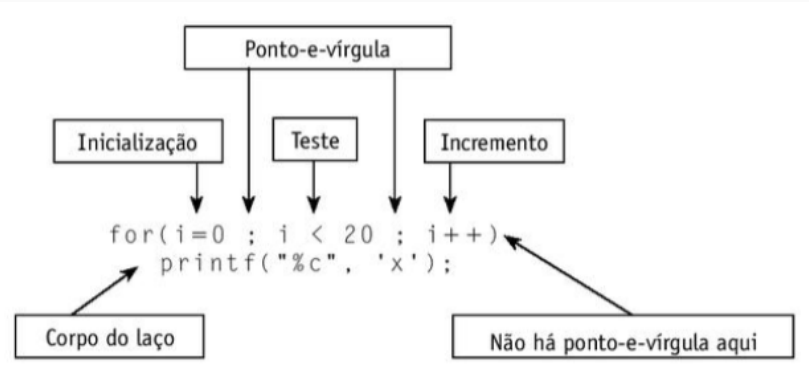
Comando - FOR

- Caso a condição do teste seja *true*, as instruções dentro do bloco do *for* são executadas **(printf("%c",'*');**).
- Quando o teste se torna *false*, o laço é encerrado e o controle passa para as instruções seguintes.



Comando - FOR

- O *incremento* define a maneira pela qual a variável de controle será alterada cada vez que o laço for repetido (i++).
- O incremento sempre é executado logo após a execução das instruções dentro do bloco do *for*.



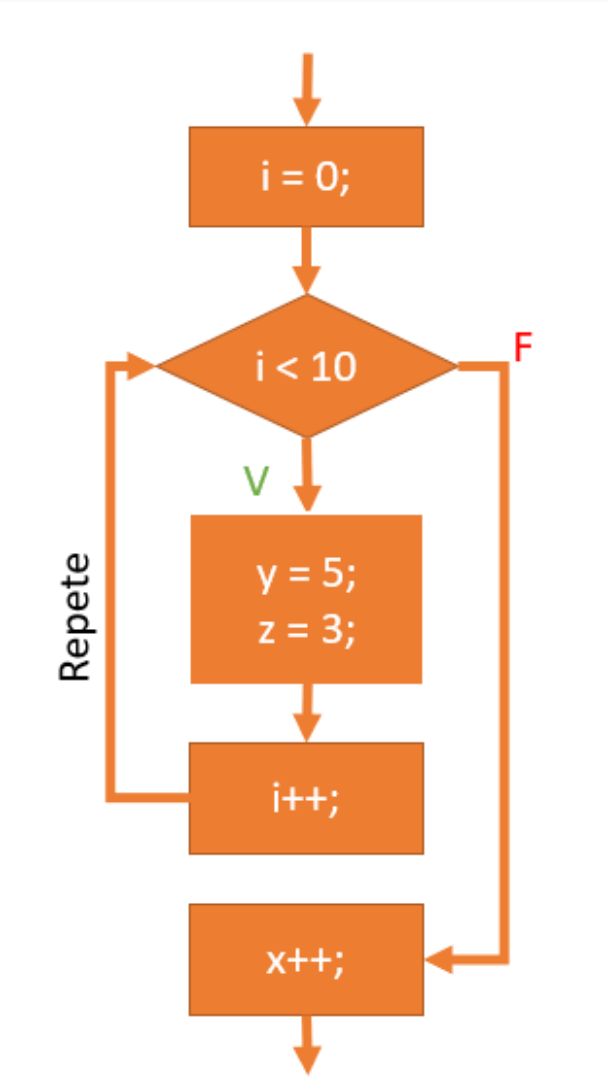
Comando - FOR

Qual a saída do código a seguir?

```
for(int i=0; i<5; i++){
    printf("%c \n", x);
}
```

FOR - Exemplo

```
for (int i=0; i<10; i++){
  y = 5;
  z = 3;
}
x++;
```



Comando - FOR

Ao invés de incrementar a variável de controle do *for*, podemos decrementá-la caso faça mais sentido no problema que estamos solucionando.

Comando - FOR

Ao invés de incrementar a variável de controle do *for*, podemos decrementá-la caso faça mais sentido no problema que estamos solucionando.

```
for(int i=0; i<20; i++){
   printf("%c", x);
}
```

```
for(int i=20; i>0; i--){
   printf("%c", x);
}
```

Exercício

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

Exercício

Desenvolva um programa que leia um número inteiro N. Em seguida, o programa deve ler N números inteiros e exibir a média deles na tela.

Flexibilidade do for

- Nos exemplos e exercícios anteriores, foram utilizados o *for* na sua forma mais simples.

Flexibilidade do for

- Nos exemplos e exercícios anteriores, foram utilizados o *for* na sua forma mais simples.
- A forma mais simples consiste em:

Flexibilidade do for

- Nos exemplos e exercícios anteriores, foram utilizados o *for* na sua forma mais simples.
- A forma mais simples consiste em:
  - A primeira expressão para *inicializar* a variável;
  - A segunda para expressar um *limite/condição*;
  - A terceira para *incrementar* ou *decrementar* a variável.

Flexibilidade do for

- Nos exemplos e exercícios anteriores, foram utilizados o *for* na sua forma mais simples.
- A forma mais simples consiste em:
  - A primeira expressão para *inicializar* a variável;
  - A segunda para expressar um *limite/condição*;
  - A terceira para *incrementar* ou *decrementar* a variável.
- Entretanto, as expressões não são restritas a essa forma.

Comando FOR - operador vírgula

- Qualquer uma das expressões do *for* pode conter várias instruções separadas por vírgulas.

Comando FOR - operador vírgula

- Qualquer uma das expressões do *for* pode conter várias instruções separadas por vírgulas.
- A vírgula é um operador que significa *"faça isso e depois isso"*.

Comando FOR - operador vírgula

- Qualquer uma das expressões do *for* pode conter várias instruções separadas por vírgulas.
- A vírgula é um operador que significa *"faça isso e depois isso"*.
- Um par de expressões separadas por vírgula é avaliado da esquerda para direita.

Operador Vírgula - Exemplos

Imprimir os números de 0 a 10, de 2 em 2.

Operador Vírgula - Exemplos

Imprimir os números de 0 a 10, de 2 em 2.

```
int main(){
  int i,j;
  for(i=0, j=0; (i+j)<=10; i++, j++){
    printf("%d ", i+j);
  }
  return 0;
}
```

Outros operadores

- A variável do *for* pode ser do tipo char.

```
int main(){
  char ch;
  for(ch='a'; ch<='z'; ch++){
    printf("\nO valor ASCII de %c é %d", ch, ch);
  }
   return 0;
}
```

Outros operadores

- A variável do *for* pode ser do tipo char.
- É possível chamar funções dentro de expressões do *for*.

Outros operadores

- A variável do *for* pode ser do tipo char.
- É possível chamar funções dentro de expressões do *for*.
- O corpo do laço pode ser vazio, entretanto, o ";" deve permanecer.

Outros operadores

- A variável do *for* pode ser do tipo char.
- É possível chamar funções dentro de expressões do *for*.
- O corpo do laço pode ser vazio, entretanto, o ";" deve permanecer.
- Como no *if*, caso o bloco de instruções do *for* contiver apenas uma instrução, o { e o } não são necessários.

Outros operadores

- A variável do *for* pode ser do tipo char.
- É possível chamar funções dentro de expressões do *for*.
- O corpo do laço pode ser vazio, entretanto, o ";" deve permanecer.
- Como no *if*, caso o bloco de instruções do *for* contiver apenas uma instrução, o { e o } não são necessários.
- O incremento da variável de controle (e outras eventuais variáveis) não precisa ser unitário.

Exercício

Qual será a saída do seguinte código?

```
...
int valor=5;
for(int i=0, valor=5, i<4, i+=1, valor+=5){
   printf("%d\n"", valor);
}
printf("Valor = %d", valor);
```

Exercício

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

Exercício

Qual será a saída do seguinte código?

```
...
int valor=5;
for(int i=0, valor=5; i<4, i+=1; valor+=5){
   printf("%d\n", valor);
}
printf("Valor = %d", valor);
printf("I = %d", i);
```

Exercício

Qual será a saída do seguinte código?

```
Message
=== Build: Debug in ex (compiler: GNU GCC Compiler) ===
In function 'main':
error: 'i' undeclared (first use in this function)
note: each undeclared identifier is reported only once for each function it appears in
=== Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===
```

Visibilidade de variáveis de bloco

- Um aspecto importante dos blocos de código é o de que uma variável declarada dentro de um bloco não é visível fora dele (variável *i* do exemplo anterior).

Visibilidade de variáveis de bloco

```
int main(){
  float soma=0;
  for(int i=0; i<10; i++){
    float nota;
    printf("Digite uma nota \n");
    scanf("%f", &nota);
    soma += nota;
  }
  printf("Última nota recebida = %f\n", nota);
  return 0;
}
```

Laços for aninhados

- Quando um laço *for* faz parte do corpo de outro laço *for*, dizemos que o laço interno está **aninhado**.
- 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);
  }
}
```

Laços for aninhados

- O nome das variáveis de controle de cada um dos *for* deve ser diferente.
- Podemos usar o valor da variável de controle do nível acima na condição de parada do *for* mais interno.

```
for (int i = 0; i < 4; i++){
  for (int j = 0; j < i; j++){
    printf(" %d e %d ", i, j);
  }
}
```

Exercício

Escreva um programa que exiba a tabuada de multiplicação do 1 ao 10. Para isso, utilize dois laços *for* aninhados: o primeiro para representar os números de 1 a 10 e o segundo para calcular e exibir a multiplicação desses números de 1 a 10.

Saída esperada:

```
Tabuada do 1
1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
...
1 x 10 = 10
Tabuada do 2
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
...
2 x 10 = 20
...
```

Dúvidas?