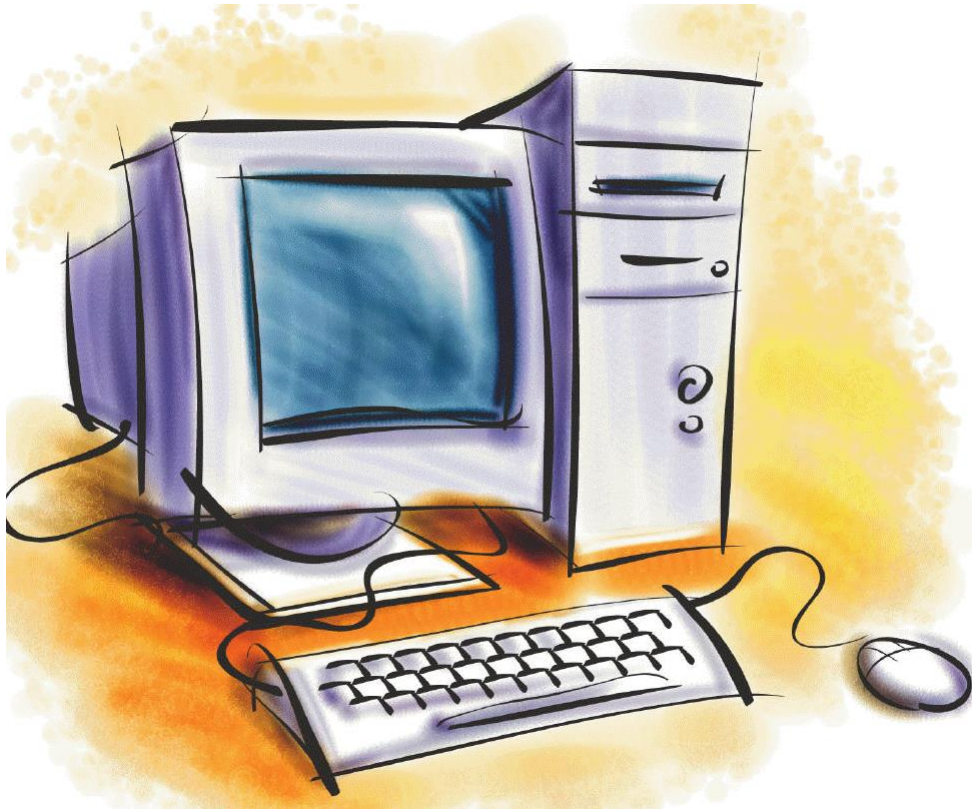


ALGORITMOS E ESTRUTURAS DE DADOS I



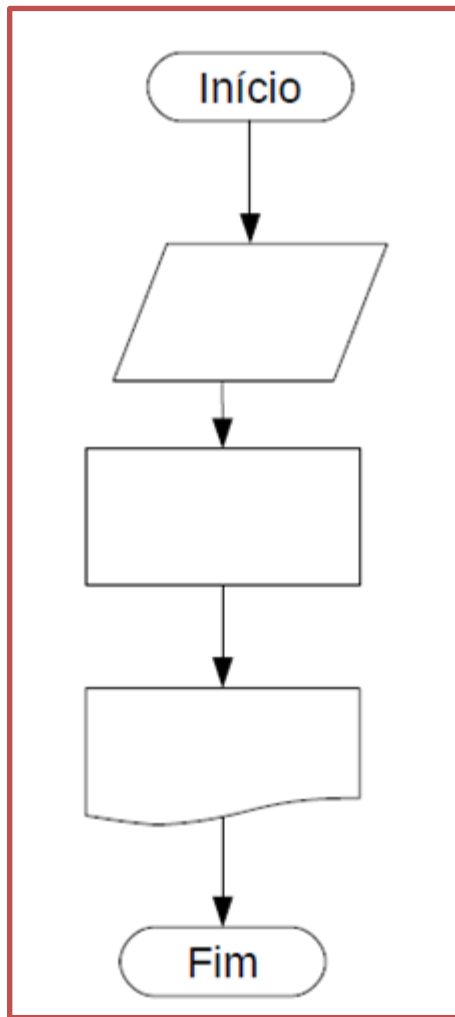
UNIDADE 2

ESTRUTURAS DE CONTROLE

PROF. NAÍSSÉS ZÓIA LIMA

Estrutura Sequencial

- Fluxograma



- Pseudocódigo

ALGORITMO

DECLARE <variáveis>

<Bloco de comandos>

FIM_ALGORITMO

- **Algoritmo – Calcular o dobro de um número**

- **Algoritmo – Calcular o dobro de um número**

Declaração de variáveis:

- Comando: DECLARE
- Tipos: NUMÉRICO, LITERAL e LÓGICO

ALGORITMO

DECLARE x, y NUMÉRICO

FIM_ALGORITMO

- **Algoritmo – Calcular o dobro de um número**

Entrada de dados:

– Comando: **LEIA**

ALGORITMO

DECLARE x, y **NUMÉRICO**

LEIA x

FIM_ALGORITMO

- **Algoritmo – Calcular o dobro de um número**

Atribuição a variáveis:

– Operador: \leftarrow

ALGORITMO

DECLARE x, y NUMÉRICO

LEIA x

y \leftarrow 2*x

FIM_ALGORITMO

- **Algoritmo – Calcular o dobro de um número**

Saída de dados:

– Comando: **ESCREVA**

ALGORITMO

DECLARE x, y **NUMÉRICO**

LEIA x

y \leftarrow 2*x

ESCREVA “O dobro é = ”, y

FIM_ALGORITMO

- **Algoritmo – Calcular o dobro de um número**

Pseudocódigo

Fluxograma

ALGORITMO

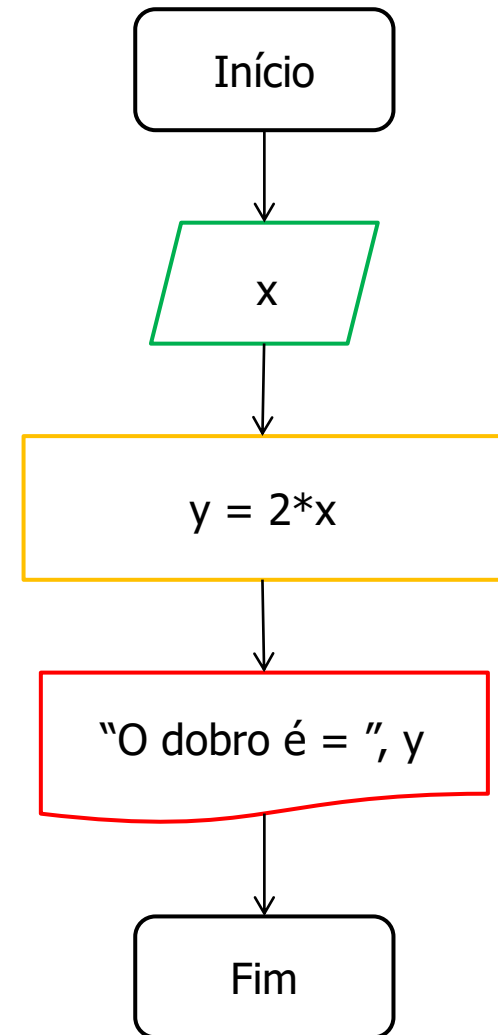
DECLARE x, y NUMÉRICO

LEIA x

$y \leftarrow 2 * x$

ESCREVA "O dobro é =", y

FIM_ALGORITMO



- **Exercício**

Faça um algoritmo em fluxograma e pseudocódigo, de acordo com os comandos mostrados anteriormente, para receber o valor de duas notas de provas, somar os valores e mostrar na tela a soma e a média das duas notas.

Estrutura Sequencial em C

- Estrutura sequencial em C

```
#include <nome da biblioteca>
```

```
int main()
```

```
{
```

```
    bloco de comandos;
```

```
}
```

Estrutura Sequencial em C

- Estrutura sequencial em C
 - Bibliotecas são arquivos contendo várias funções que podem ser incorporadas aos programas escritos em C.
 - A diretiva `#include` faz com que o texto da biblioteca especificada seja inserido no programa.
 - Exemplo:
 - A biblioteca `stdio.h` permite a utilização de diversos comandos de entrada e saída.

Estrutura Sequencial em C

- C é case sensitive: considera que letras maiúsculas são diferentes de minúsculas (por exemplo, *a* é diferente de *A*) .
 - *Ex.:*
 - *NOTA != nota != Nota....*
- **Comandos** em C são, na maioria das vezes, escritos com letras minúsculas.

Estrutura Sequencial em C

Declaração de Variáveis

- Declaração de variáveis em C
 - As variáveis são declaradas após a especificação de tipo das mesmas

```
int main()
{
    int Y;
    float X;
    bool W;
    char sexo, nome[40];
}
```

Estrutura Sequencial em C

Declaração de Variáveis

- Tipos básicos:

Tipo	Descrição
int	Utilizado para definir uma variável inteira que comporta valores pertencentes ao conjunto dos números inteiros
float	Utilizado para definir uma variável real que comporta valores pertencentes ao conjunto dos números reais
double	O tipo double é similar ao float , a diferença é que o este tipo comporta valores reais com um número de dígitos maior, assim, a precisão nos cálculos com casas decimais aumenta
char	Utilizado para armazenar um caractere. Comporta apenas um caractere

Estrutura Sequencial em C

Declaração de Variáveis

Data Type	Memory (bytes)	Range	Format Specifier
short int	2	-32,768 to 32,767	%hd
unsigned short int	2	0 to 65,535	%hu
unsigned int	4	0 to 4,294,967,295	%u
int	4	-2,147,483,648 to 2,147,483,647	%d
long int	4	-2,147,483,648 to 2,147,483,647	%ld
unsigned long int	4	0 to 4,294,967,295	%lu
long long int	8	$-(2^{63})$ to $(2^{63})-1$	%lld
unsigned long long int	8	0 to 18,446,744,073,709,551,615	%llu
signed char	1	-128 to 127	%c
unsigned char	1	0 to 255	%c
float	4	1.2E-38 to 3.4E+38	%f
double	8	1.7E-308 to 1.7E+308	%lf
long double	16	3.4E-4932 to 1.1E+4932	%Lf

Estrutura Sequencial em C

Comando de Atribuição

- Comando de atribuição em C
 - Utilizado para atribuir valores ou operação à variáveis.
 - Representado por = (sinal de igualdade)

Exemplos:

```
X = 4;
```

```
X = X + 2;
```

```
Sexo = 'F';
```

Estrutura Sequencial em C++

Comando de Saída

Comando de saída em C

- Comandos mais utilizados: `puts` e `printf`

Estrutura Sequencial em C++

Comando de Saída

Comando de saída em C

- **Função puts()**
 - Permite realizar a impressão de textos (string) na saída padrão (monitor), ou seja, é responsável pela saída de informações
 - Incluir uma nova linha (\n) ao final da impressão

- **Sintaxe:**

```
puts ("texto" );
```

- **Exemplos:**

```
puts ("Digite um numero" );
```

Estrutura Sequencial em C++

Comando de Saída

Comando de saída em C

- **Função printf()**

- Permite realizar a impressão de dados formatados na saída padrão (monitor), ou seja, é responsável pela saída de informações
- Possui um número variado de parâmetros, tantos quantos forem necessários

- **Sintaxe:**

```
printf("formato", argumentos);
```

- **Exemplos:**

```
int mat = 335642;  
float medF = 7;  
printf("Matricula: %d, Med Final: %2.2f ", mat,  
medF);
```

Estrutura Sequencial em C++

Comando de Saída

Comando de saída em C

Formato	Tipo da Variável	Conversão Realizada
%c	Caracteres	char, short int, int, long int
%d	Inteiros	int, short int, long int
%e	Ponto flutuante, notação científica	float, double
%f	Ponto flutuante, notação decimal	float, double
%o	Saída em octal	int, short int, long int, char
%s	String	char *, char[]
%u	Inteiro sem sinal	unsigned int, unsigned short int, unsigned long int
%x	Saída em hexadecimal (0 a f)	int, short int, long int, char
%X	Saída em hexadecimal (0 a F)	int, short int, long int, char

Estrutura Sequencial em C

Comando de Entrada

- Comando de entrada em C
 - Comandos mais utilizados: `scanf` e `fgets`

Estrutura Sequencial em C

Comando de Entrada

- **Função scanf()**

- Similar à função `printf()`, também suporta uma quantidade "n" de argumentos e permite que os dados digitados pelo usuário sejam armazenados nas variáveis do programa

- **Sintaxe:**

```
scanf("formato", enderecosArgumentos);
```

- **Exemplos:**

```
int mat;  
scanf("%d", &mat);  
  
float nota1, nota2;  
scanf("%f %f", &nota1, &nota2);
```

• O caractere **&** é de uso **obrigatório** e responsável por indicar que o retorno será o **endereço de memória** de uma determinada variável

- **Observação:** a função `scanf()` não é a mais adequada para leitura de textos. Neste caso deve ser substituída pela função `fgets()`

Estrutura Sequencial em C

Comando de Entrada

- **Função fgets()**

- A função **scanf ()** é ótima para entrada de números mas não consegue lidar com entrada de textos que contém mais de uma palavra
- Nestes casos deve ser utilizada a função **fgets ()** que é capaz de armazenar os textos corretamente, desde que o tamanho limite especificado na variável seja respeitado

• **Primeiro argumento:** nome da variável (não é necessário &)

• **Segundo argumento:** tamanho especificado para a variável produto

• **Terceiro argumento:** instrução **stdin** para indicar que a leitura do dado será feita à partir do teclado (entrada padrão)

```
#include <stdio.h>
#include <stdlib.h>
void main() {
    char produto[30];
    printf("Informe o nome do produto: \n");
    fgets(produto, sizeof(produto), stdin);
    printf("Produto: %s \n", produto);
}
```


Estrutura Sequencial em C

Comentários

- Comentários são textos que podem ser inseridos em um programa com o objetivo de documentá-lo e para facilitar o entendimento do mesmo.
 - Lembre-se: o código não é feito somente para você! Futuramente outros poderão necessitar do seu código.
- Os comentários não são analisados pelo compilador.
- Os comentários podem ocupar uma ou várias linhas, devendo ser inseridos nos programas utilizando:
 - **`/* */` : comentário de várias linhas**
 - A região de comentários é aberta com os símbolos **`/*`** e é encerrada com os símbolos **`*/`**
 - **`//` : comentário de uma linha**
 - A região de comentários é aberta com os símbolos **`//`** e é encerrada automaticamente ao final da linha

Estrutura Sequencial em C

Comentários

- Comentários em C

Exemplo:

`/*`

Linhas de comentários...

Linhas de comentários...

`*/`

Ou

`//` Linha de comentário

Estrutura Sequencial em C

Exemplo

Pseudocódigo

ALGORITMO

DECLARE

x, y NUMÉRICO

ESCREVA "Digite um no."

LEIA x

$y \leftarrow 2 * x$

ESCREVA "O dobro é = ", y

FIM_ALGORITMO

C

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

```
int main() {
```

```
    int x, y;
```

```
    printf("Digite um no.\n");
```

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

```
    y = 2 * x;
```

```
    printf("O dobro e = %d", y);
```

```
}
```

Linguagem C

Operadores e Funções Predefinidas

- A linguagem C possui operadores e funções predefinidas destinadas a cálculos matemáticos e à manipulação de caracteres.

Operador de Atribuição

Operador	Exemplo	Comentário
=	X = Y	O conteúdo da variável Y é atribuído a variável X.

Linguagem C

Operadores Matemáticos

Operador	Exemplo	Comentário
+	$X + Y$	Soma o conteúdo de X e de Y.
-	$X - Y$	Subtrai o conteúdo de Y do conteúdo de X
*	$X * Y$	Multiplica o conteúdo de X pelo conteúdo de Y
/	X / Y	Obtém o quociente da divisão de X por Y
%	$X \% Y$	Obtém o resto da divisão de X por Y
++	$X ++$	Aumenta o conteúdo de X em uma unidade
--	$X --$	Diminui o conteúdo de X em uma unidade

*O operador % só pode ser utilizado com operandos do tipo **inteiro***

Linguagem C

Operadores Relacionais

Operador	Exemplo	Comentário
<code>==</code>	<code>X == Y</code>	O conteúdo de X é igual ao conteúdo de Y
<code>!=</code>	<code>X != Y</code>	O conteúdo de X é diferente do conteúdo de Y
<code><=</code>	<code>X <= Y</code>	O conteúdo de X é menor ou igual ao conteúdo de Y
<code>>=</code>	<code>X >= Y</code>	O conteúdo de X é maior ou igual ao conteúdo de Y
<code><</code>	<code>X < Y</code>	O conteúdo de X é menor que o conteúdo de Y
<code>></code>	<code>X > Y</code>	O conteúdo de X é maior que o conteúdo de Y

Linguagem C

Operadores de Atribuição (Matemáticos)

Operador	Exemplo	Comentário
<code>+=</code>	<code>X += Y</code>	Equivale a <code>X = X + Y</code> .
<code>-=</code>	<code>X -= Y</code>	Equivale a <code>X = X - Y</code> .
<code>*=</code>	<code>X *= Y</code>	Equivale a <code>X = X * Y</code> .
<code>/=</code>	<code>X /= Y</code>	Equivale a <code>X = X / Y</code> .
<code>%=</code>	<code>X %= Y</code>	Equivale a <code>X = X % Y</code> .

Linguagem C

Funções Matemáticas

- Algumas das funções disponíveis da biblioteca `math.h` são:

Função	Finalidade
<code>abs(i)</code>	Retorna o valor absoluto de i
<code>ceil(d)</code>	Arredonda para cima, para o próximo valor inteiro maior que d
<code>cos(d)</code>	Retorna o cosseno de d
<code>floor(d)</code>	Arredonda para baixo, para o próximo valor inteiro menor que d
<code>log(d)</code>	Calcula o logaritmo neperiano <code>log(d)</code>
<code>pow(d1, d2)</code>	Retorna d1 elevado a d2
<code>rand()</code>	Retorna um inteiro positivo aleatório
<code>sin(d)</code>	Retorna o seno de d
<code>sqrt(d)</code>	Retorna a raiz quadrada de d
<code>tan(d)</code>	Retorna a tangente de d

- ▶ As funções que possuem **retorno**, necessitam de uma variável para receber o valor retornado. Exemplo: `potencia = pow(b, 2)`

Exercícios

- ❑ Faça um programa em C para calcular e imprimir a área de um triângulo. Os valores da base e altura são digitados pelo usuário.
- ❑ Faça programa em C que receba o nome e as 3 notas de uma aluno. O programa deve exibir o nome e a nota final do aluno, que é calculada como a média das três notas.

Estrutura Condicional

- **Proposição**

- É um enunciado verbal (expressão), ao qual deve ser atribuído, sem ambiguidade, um valor lógico verdadeiro (V) ou falso (F).

- **Exemplos:**

- A copa do mundo 2014 será no Brasil.
 - Todo ser humano é imortal.
 - $2 + 5 < 5$
 - $1 + 1 \geq 2$

- **Expressões lógicas**

- Expressões compostas de relações que sempre retornam um valor lógico (verdadeiro ou falso).

- Exemplos:

- $5 + 3 < 10 \Rightarrow V$

- $1 + 10 < 11 \Rightarrow F$

- $5 + 0 = 5 \Rightarrow V$

- Expressões lógicas podem ser unidas usando operadores lógicos.

- **Operadores lógicos**

- Atuam sobre expressões lógicas retornando sempre valores lógicos (verdadeiro ou falso).

Operador	Tipo	Resultado
E	Binário	Retorna verdadeiro se ambas expressões forem verdadeiras
OU	Binário	Retorna verdadeiro se pelo menos uma das expressões for verdadeira
NÃO	Unário	Inverte o estado, retorna verdade caso a expressão seja falsa e vice-versa

- **Operadores lógicos**

- Operador **E (AND)**: operador de conjunção

- Terá valor **V** quando as ambas proposições forem **V**. Basta uma proposição ser **F** para o resultado ser **F**.

- Exemplos:

- $(2 + 5 > 4) \text{ E } (1 + 3 \leq 2) \Rightarrow \text{F}$

- $(2 + 5 > 4) \text{ E } (1 + 3 \geq 2) \Rightarrow \text{V}$

- $(2 + 5 < 4) \text{ E } (1 + 3 \leq 2) \Rightarrow \text{F}$

- **Operadores lógicos**

- Operador **OU (OR)**: operador de disjunção

- Terá valor **V** quando as uma das proposições forem **V**.

- Exemplos:

- $(2 + 5 > 4)$ **OU** $(1 + 3 \leq 2)$ \Rightarrow **V**

- $(2 + 5 > 4)$ **OU** $(1 + 3 \geq 2)$ \Rightarrow **V**

- $(2 + 5 < 4)$ **OU** $(1 + 3 \leq 2)$ \Rightarrow **F**

- **Operadores lógicos**

- Operador **NÃO (NOT)**: operador de negação

- Inverte o valor lógico

- Exemplos:

- Não $(2 + 5 > 4)$ \Rightarrow **F**

- Não $(1 + 3 \leq 2)$ \Rightarrow **V**

- **Tabela Verdade**

A	B	A <u>E</u> B	A <u>OU</u> B	<u>NÃO</u> (A)
F	F			
F	V			
V	F			
V	V			

- Tabela Verdade

A	B	A <u>E</u> B	A <u>OU</u> B	<u>NÃO</u> (A)
F	F	F	F	V
F	V	F	V	V
V	F	F	V	F
V	V	V	V	F

- **Operadores lógicos em C**

Operador	C	Precedência
E	&&	2
OU		3
NÃO	!	1

- Note que o operador OU (||) é o que possui a menor precedência.

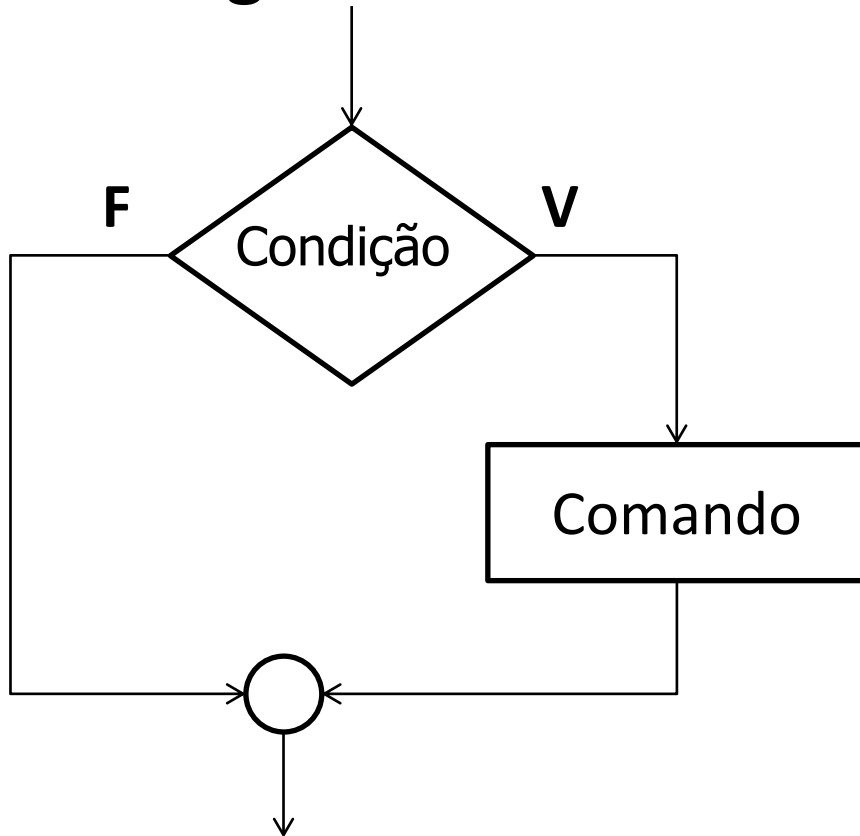
- **Tipo lógico em C**

- Em C, inteiros são usados para tipos lógicos. Nesse caso, 0 é considerado FALSO e 1 VERDADEIRO.
- Podemos usar o tipo **bool** importando “stdbool.h”. Nesse caso, o valor será convertido em 0 ou 1.

```
bool x = true;  
bool y = false;  
printf("%d", x);    //imprime 1  
printf("%d", y);    //imprime 0
```

Estrutura Condicional Simples

- Fluxograma

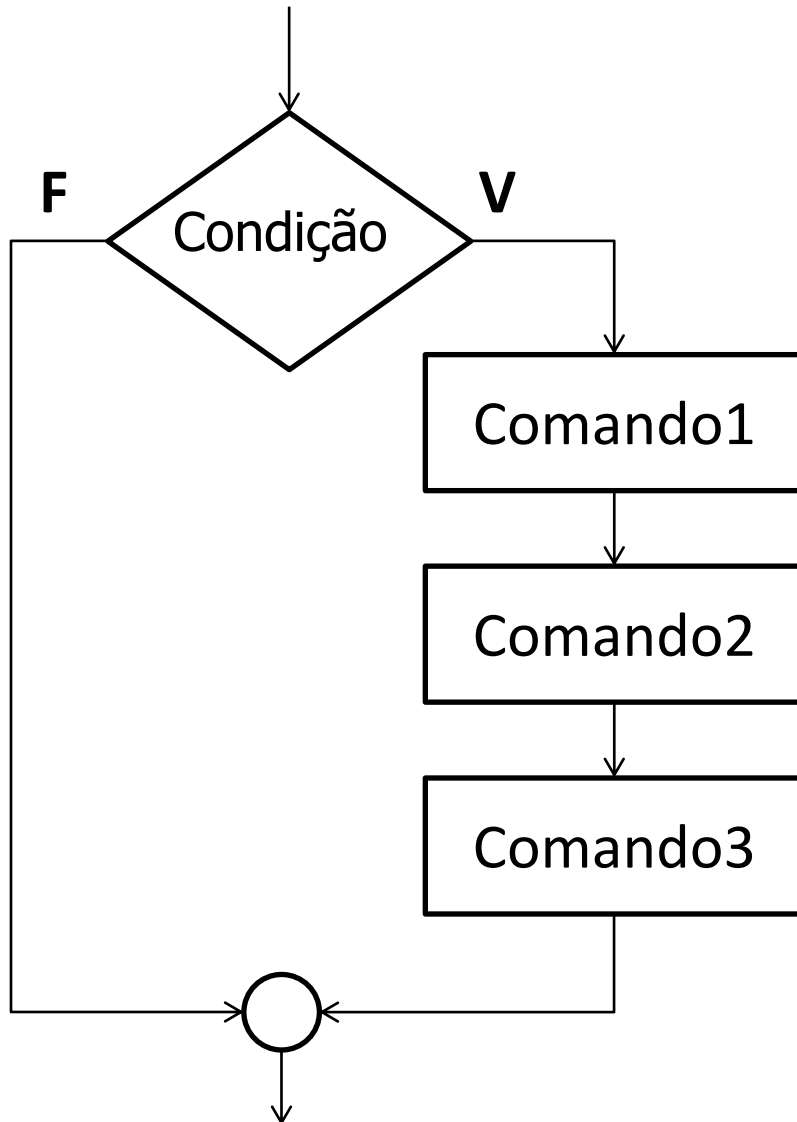


- Pseudocódigo

SE *Condição* **ENTÃO**
Comando

Estrutura Condicional Simples

- Fluxograma



- Pseudocódigo

SE *Condição* **ENTÃO**
INÍCIO
 Comando1
 Comando2
 Comando3
FIM

Estrutura Condicional Simples em C

```
if (Condição)  
    Comando1
```

```
if (Condição) {  
    Comando1  
    Comando2  
    Comando3  
}
```

- **Exemplo – Faça um algoritmo que, após um número ser sorteado de 1 e 10, informa se você ganhou ou perdeu. Você ganha se o número for 5.**

ALGORITMO

DECLARE x NUMÉRICO

LEIA x

SE $x = 5$ ENTÃO

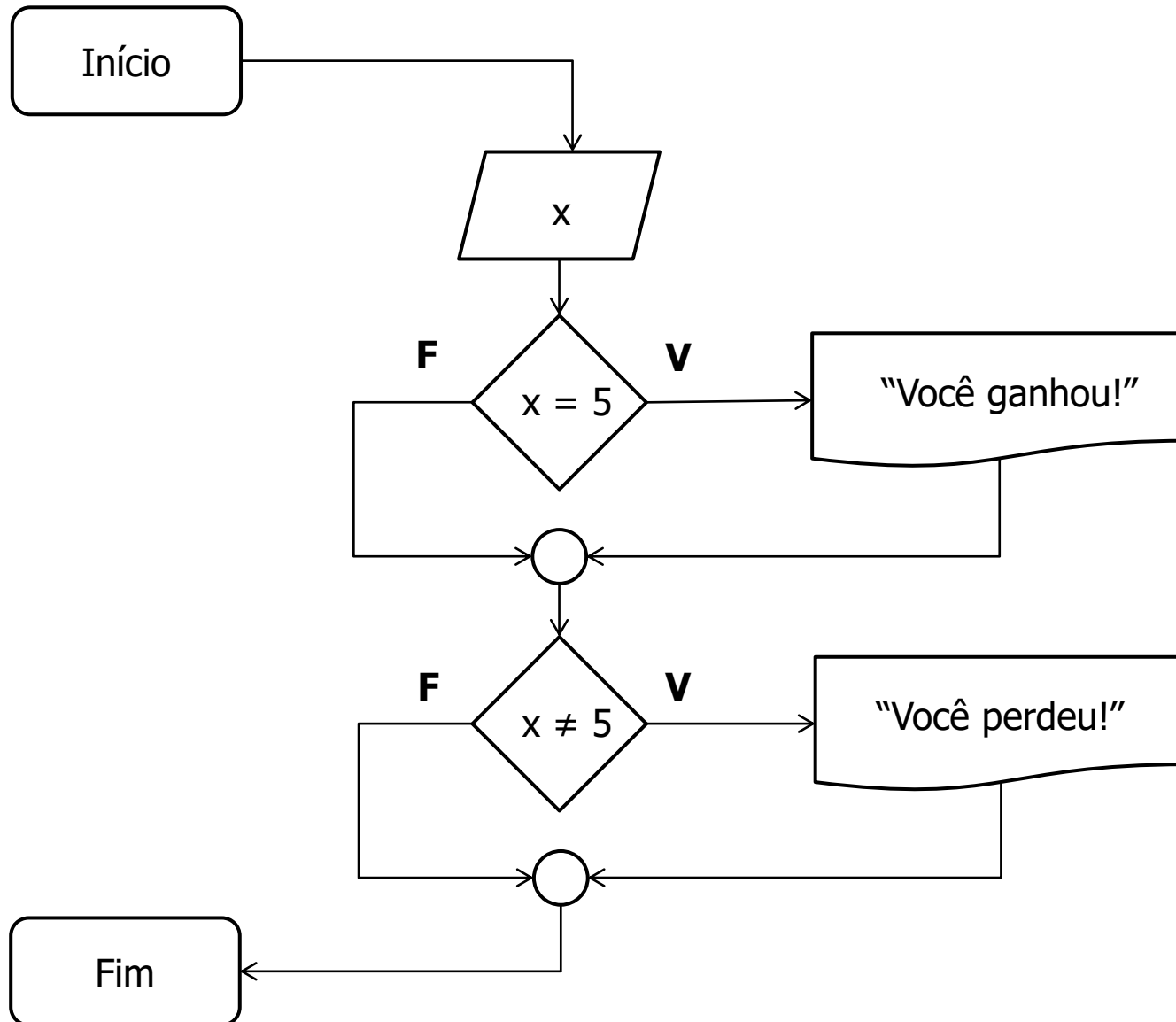
 ESCREVA “Você ganhou!”

SE $x \neq 5$ ENTÃO

 ESCREVA “Você perdeu!”

FIM_ALGORITMO

- Exemplo – Faça um algoritmo que, após um número ser sorteado de 1 e 10, informa se você ganhou ou perdeu. Você ganha se o número for 5.



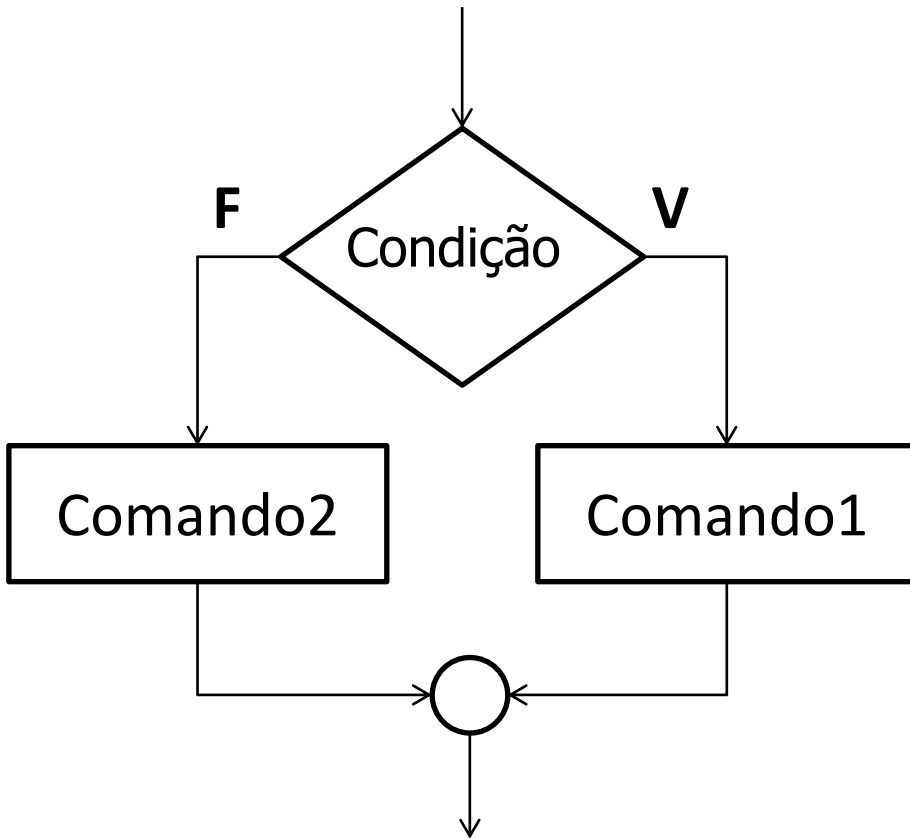
- **Exemplo – Faça um algoritmo que, após um número ser sorteado de 1 e 10, informa se você ganhou ou perdeu. Você ganha se o número for 5.**

```
#include <stdio.h>

int main()
{
    int x;
    puts("Digite o numero sorteado:");
    scanf("%d", &x);
    if(x == 5) {
        puts("Voce ganhou!");
    }
    if(x != 5) {
        puts("Voce perdeu!");
    }
    return 0;
}
```

Estrutura Condicional Composta

- Fluxograma



- Pseudocódigo

SE *Condição* **ENTÃO**

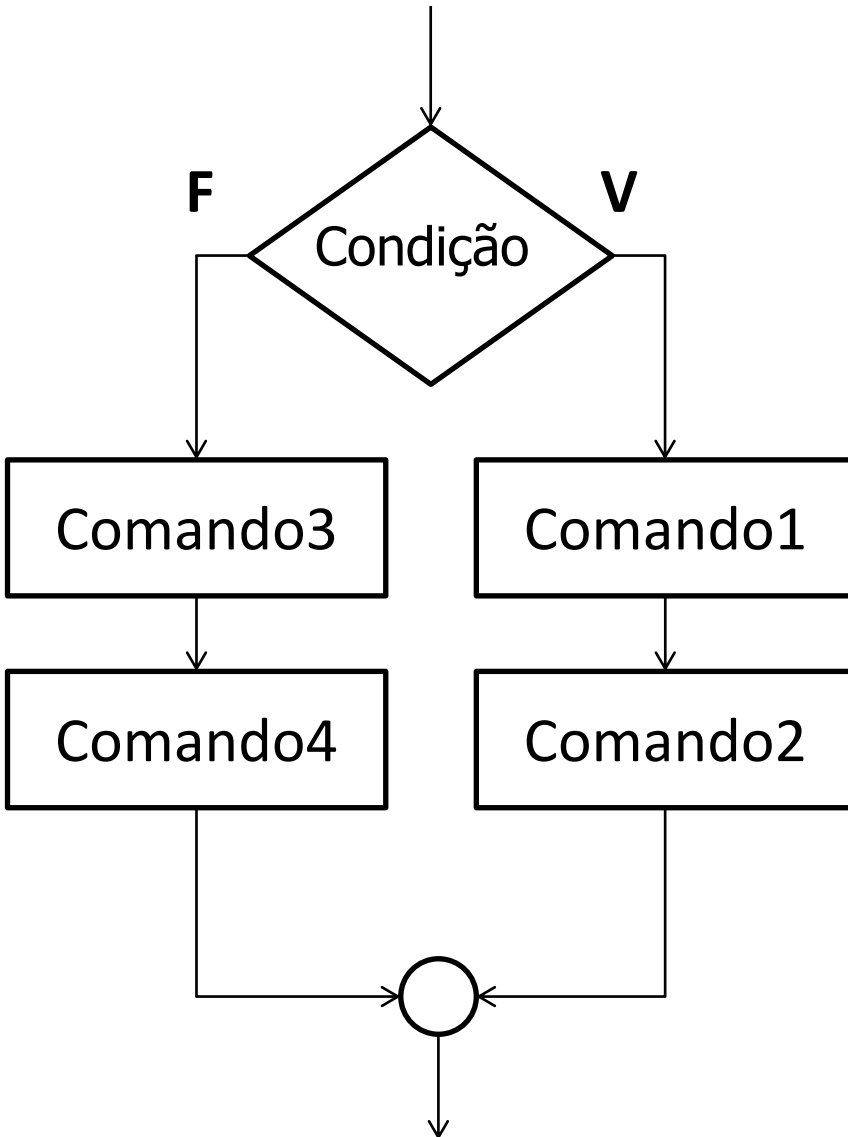
Comando1

SENÃO

Comando2

Estrutura Condicional Composta

- Fluxograma



- Pseudocódigo

SE *Condição* **ENTÃO**

INÍCIO

Comando1

Comando2

FIM

SENÃO

INÍCIO

Comando3

Comando4

FIM

Estrutura Condicional Composta em C

```
if (Condição)  
    Comando1  
else  
    Comando2
```

```
if (Condição) {  
    Comando1  
    Comando2  
} else {  
    Comando3  
    Comando4  
}
```

- **Exemplo – Faça um algoritmo que, após um número ser sorteado de 1 e 10, informa se você ganhou ou perdeu. Você ganha se o número for 5.**

ALGORITMO

DECLARE x NUMÉRICO

LEIA x

SE x = 5 ENTÃO

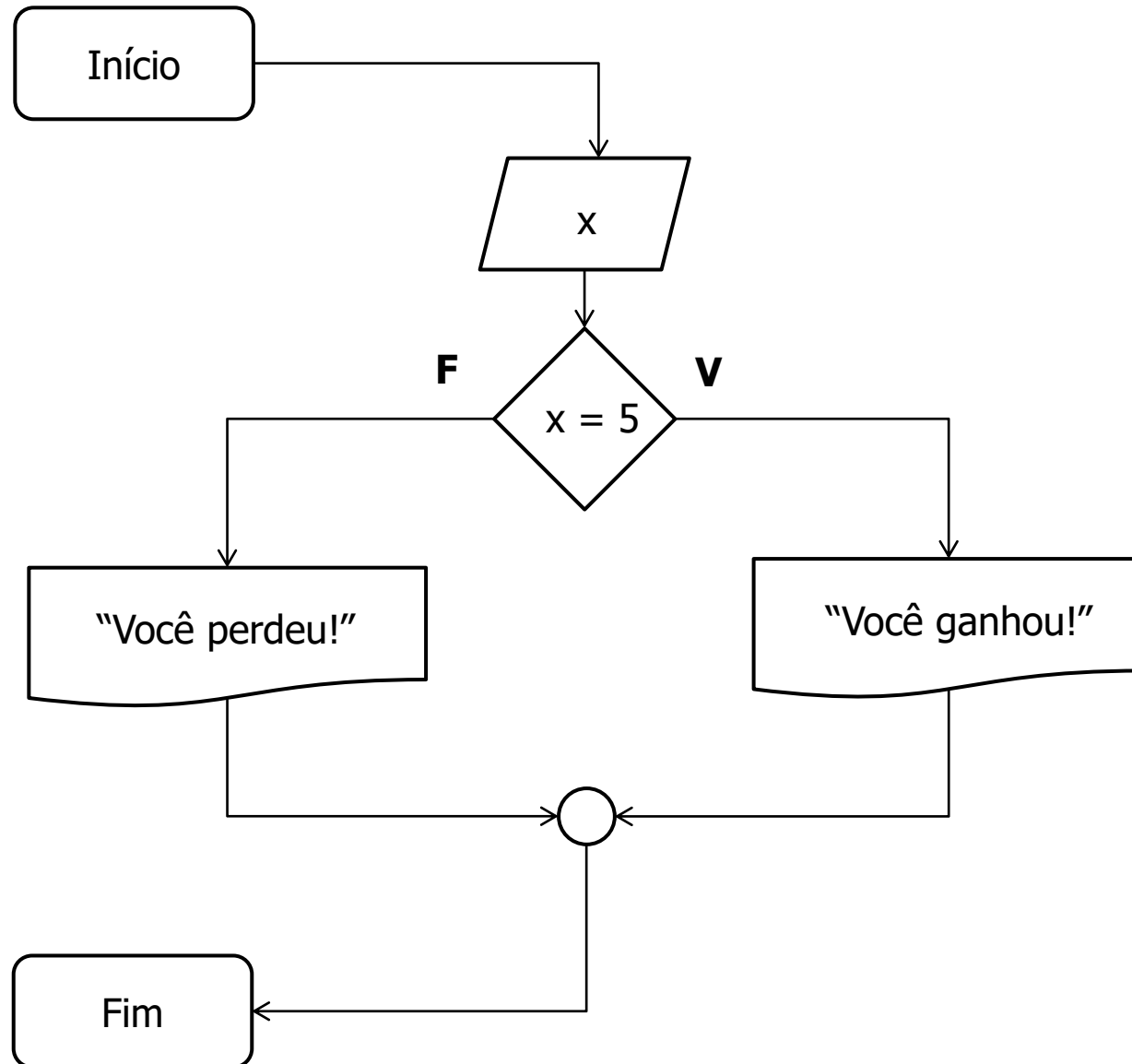
 ESCREVA “Você ganhou!”

SENÃO

 ESCREVA “Você perdeu!”

FIM_ALGORITMO

- Exemplo – Faça um algoritmo que, após um número ser sorteado de 1 e 10, informa se você ganhou ou perdeu. Você ganha se o número for 5.



- **Exemplo – Faça um algoritmo que, após um número ser sorteado de 1 e 10, informa se você ganhou ou perdeu. Você ganha se o número for 5.**

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

```
int main()
```

```
{
```

```
    int x;
```

```
    puts("Digite o numero sorteado:");
```

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

```
    if(x == 5) {
```

```
        puts("Voce ganhou!");
```

```
    } else {
```

```
        puts("Voce perdeu!");
```

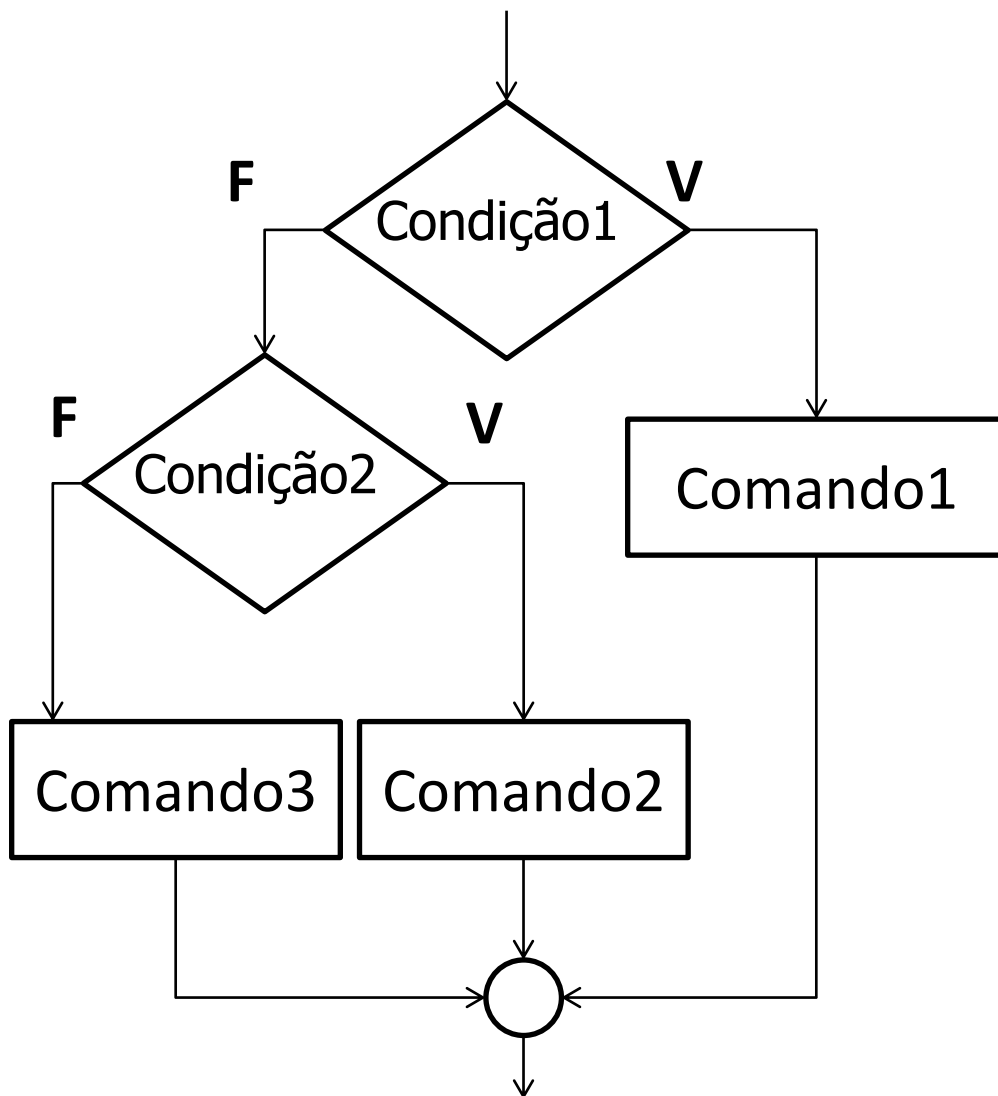
```
    }
```

```
    return 0;
```

```
}
```


Estrutura Condicional Aninhada

- Fluxograma



- Pseudocódigo

SE *Condição1* **ENTÃO**

Comando1

SENÃO

SE *Condição2* **ENTÃO**

Comando2

SENÃO

Comando3

Estrutura Condicional Aninhada

SE *Condição1* **ENTÃO**

Comando1

SENÃO

SE *Condição2* **ENTÃO**

Comando2

SENÃO

Comando3

SE *Condição1* **ENTÃO**

Comando1

SENÃO SE *Condição2* **ENTÃO**

Comando2

SENÃO

Comando3

Estrutura Condicional Aninhada em C

```
if (Condição1) {  
    Comando1  
} else {  
    if (Condição2) {  
        Comando2  
    } else {  
        Comando3  
    }  
}
```

```
if (Condição1) {  
    Comando1  
} else if (Condição2) {  
    Comando2  
} else {  
    Comando3  
}
```

- **Exemplo – Faça um algoritmo que, após um número ser sorteado de 1 e 10, informa se você ganhou ou perdeu. Você ganha se o número for 5 ou 7.**

ALGORITMO

DECLARE x NUMÉRICO

LEIA x

SE x = 5 ENTÃO

 ESCREVA “Você ganhou!”

SENÃO

 SE x = 7 ENTÃO

 ESCREVA “Você ganhou!”

 SENÃO

 ESCREVA “Você perdeu!”

FIM_ALGORITMO

- **Exemplo – Faça um algoritmo que, após um número ser sorteado de 1 e 10, informa se você ganhou ou perdeu. Você ganha se o número for 5 ou 7.**

ALGORITMO

DECLARE x NUMÉRICO

LEIA x

SE x = 5 ENTÃO

 ESCREVA “Você ganhou!”

SENÃO SE x = 7 ENTÃO

 ESCREVA “Você ganhou!”

SENÃO

 ESCREVA “Você perdeu!”

FIM_ALGORITMO

- **Exemplo – Faça um algoritmo que, após um número ser sorteado de 1 e 10, informa se você ganhou ou perdeu. Você ganha se o número for 5 ou 7.**

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

```
int main()
```

```
{
```

```
    int x;
```

```
    puts("Digite o numero sorteado:");
```

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

```
    if(x == 5) {
```

```
        puts("Voce ganhou!");
```

```
    } else if(x == 7) {
```

```
        puts("Voce ganhou!");
```

```
    } else {
```

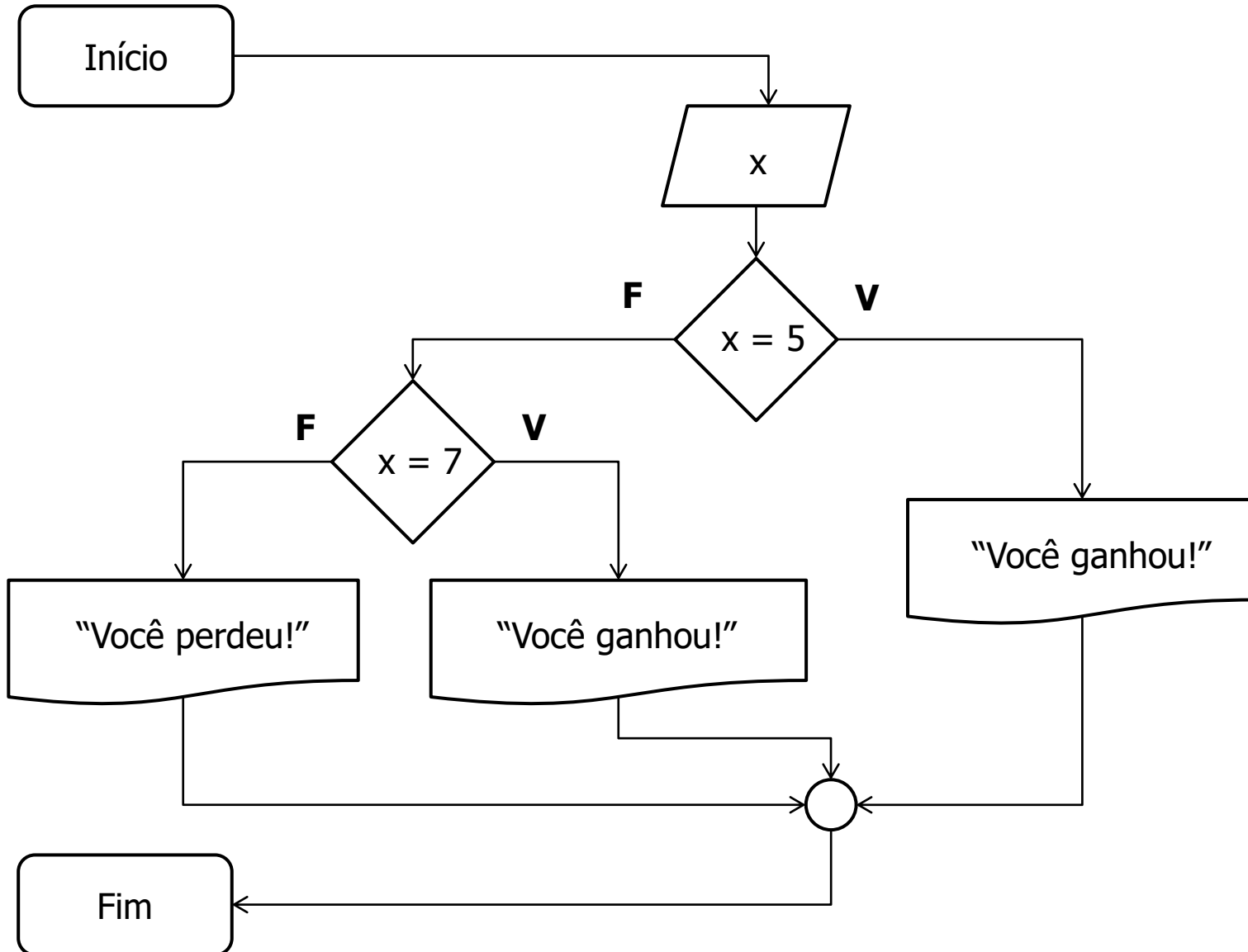
```
        puts("Voce perdeu!");
```

```
    }
```

```
    return 0;
```

```
}
```

- Exemplo – Faça um algoritmo que, após um número ser sorteado de 1 e 10, informa se você ganhou ou perdeu. Você ganha se o número for 5 ou 7.



- **Exemplo – Faça um algoritmo que, após um número ser sorteado de 1 e 10, informa se você ganhou ou perdeu. Você ganha se o número for 5 ou 7.**

ALGORITMO

DECLARE x NUMÉRICO

LEIA x

SE $x = 5$ OU $x = 7$ ENTÃO

 ESCREVA “Você ganhou!”

SENÃO

 ESCREVA “Você perdeu!”

FIM_ALGORITMO

- **Exemplo – Faça um algoritmo que, após um número ser sorteado de 1 e 10, informa se você ganhou ou perdeu. Você ganha se o número for 5 ou 7.**

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

```
int main()
```

```
{
```

```
    int x;
```

```
    puts("Digite o numero sorteado:");
```

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

```
    if(x == 5 || x == 7) {
```

```
        puts("Voce ganhou!");
```

```
    } else {
```

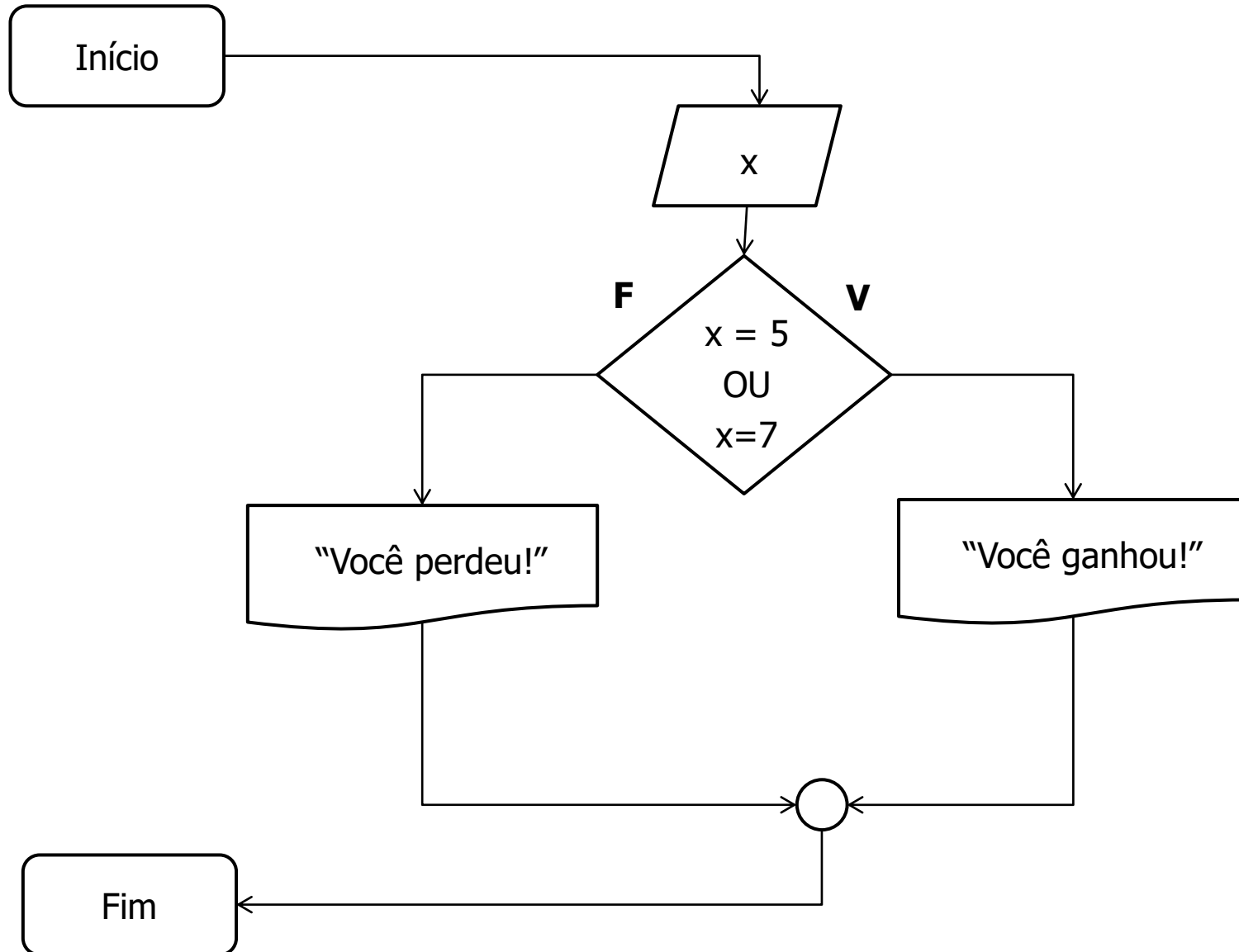
```
        puts("Voce perdeu!");
```

```
    }
```

```
    return 0;
```

```
}
```

- Exemplo – Faça um algoritmo que, após um número ser sorteado de 1 e 10, informa se você ganhou ou perdeu. Você ganha se o número for 5 ou 7.



- **Exemplo – Faça um algoritmo que, após um número ser sorteado de 1 e 10, informa se você ganhou ou perdeu. Você ganha se o número for 5 ou 7.**

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

```
int main()
```

```
{
```

```
    int x;
```

```
    puts("Digite o numero sorteado:");
```

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

```
    if(x != 5 && x != 7) {
```

```
        puts("Voce perdeu!");
```

```
    } else {
```

```
        puts("Voce ganhou!");
```

```
    }
```

```
    return 0;
```

```
}
```

- **Exemplo – Faça um algoritmo que receba dois números inteiros e imprima o maior deles.**

- **Exemplo – Faça um algoritmo que receba dois números inteiros e imprima o maior deles.**

ALGORITMO

DECLARE A, B NUMÉRICO

LEIA A, B

SE $A > B$ ENTÃO

 ESCREVA A, “é maior que ”, B

SENÃO

 SE $B > A$ ENTÃO

 ESCREVA B, “é maior que ”, A

 SENÃO

 ESCREVA A, “é igual a ”, B

FIM_ALGORITMO

- **Exemplo – Faça um algoritmo que receba dois números inteiros e imprima o maior deles.**

ALGORITMO

DECLARE A, B NUMÉRICO

LEIA A, B

SE $A > B$ ENTÃO

 ESCREVA A, “é maior que ”, B

SENÃO SE $B > A$ ENTÃO

 ESCREVA B, “é maior que ”, A

SENÃO

 ESCREVA A, “é igual a ”, B

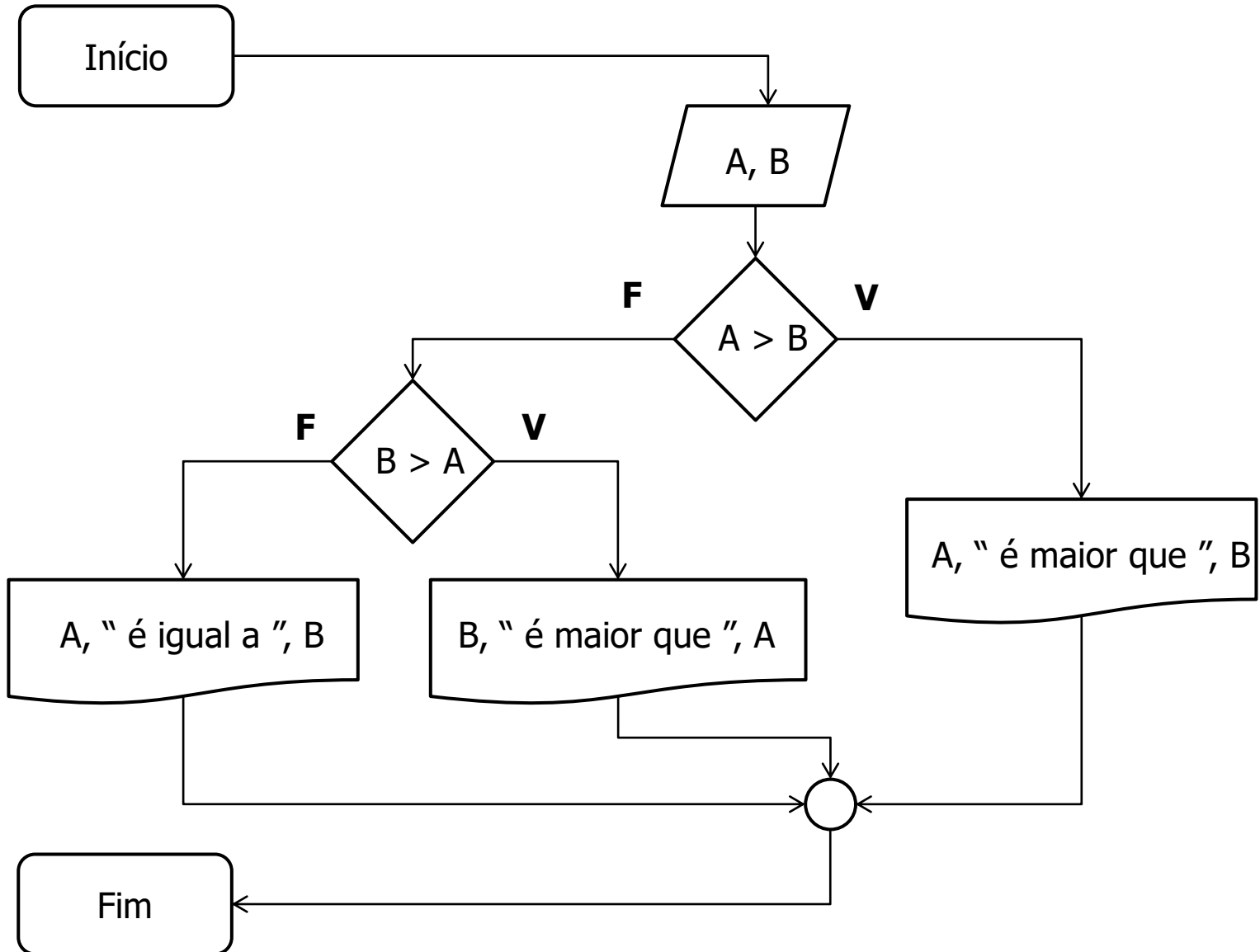
FIM_ALGORITMO

- **Exemplo – Faça um algoritmo que receba dois números inteiros e imprima o maior deles.**

```
#include <stdio.h>

int main()
{
    int A, B;
    puts("Digite o primeiro numero:");
    scanf("%d", &A);
    puts("Digite o segundo numero:");
    scanf("%d", &B);
    if(A > B) {
        printf("%d e maior que %d", A, B);
    } else if(B > A) {
        printf("%d e maior que %d", B, A);
    } else {
        printf("%d e igual a %d", A, B);
    }
    return 0;
}
```

- Exemplo – Faça um algoritmo que receba dois números inteiros e imprima o maior deles.



- **Exemplo – Faça um programa que receba um número inteiro e verifique se é par ou ímpar.**

- **Exemplo – Faça um programa que receba um número inteiro e verifique se é par ou ímpar.**

```
#include <stdio.h>

int main()
{
    int num;
    puts("Digite o numero:");
    scanf("%d", &num);
    if(num % 2 == 0) {
        printf("%d e par", num);
    } else {
        printf("%d e impar", num);
    }
    return 0;
}
```

Estrutura Condicional Switch-Case

- Ao invés de usar múltiplos if-else, temos a alternativa do switch-case.
- A cláusula switch seleciona um dos múltiplos blocos a ser executado.

```
switch(expressao) {  
    case x:  
        // code block  
        break;  
    case y:  
        // code block  
        break;  
    default:  
        // code block  
}
```

- A cláusula switch é avaliada uma única vez
- O valor da expressão é comparado com os valores de cada case
- Se a comparação resultar verdadeiro, o bloco é executado
- A cláusula break interrompe a execução do switch naquele momento
- A cláusula default é opcional e especifica código a ser executado caso nenhuma comparação tenha correspondido
- Switch só pode ser usado com expressões representadas por inteiros

Estrutura Condicional Switch-Case

- Exemplo. Qual o valor impresso a seguir?

```
int day = 4;
```

```
switch (day) {  
    case 1:  
        printf("Monday");  
        break;  
    case 2:  
        printf("Tuesday");  
        break;  
    case 3:  
        printf("Wednesday");  
        break;  
    case 4:  
        printf("Thursday");  
        break;  
    case 5:  
        printf("Friday");  
        break;  
    case 6:  
        printf("Saturday");  
        break;  
    case 7:  
        printf("Sunday");  
        break;  
}
```

Estrutura Condicional Switch-Case

- Exemplo. Qual o valor impresso a seguir?

```
int day = 4;
```

```
switch (day) {  
    case 1:  
        printf("Monday");  
        break;  
    case 2:  
        printf("Tuesday");  
        break;  
    case 3:  
        printf("Wednesday");  
        break;  
    case 4:  
        printf("Thursday");  
        break;  
    case 5:  
        printf("Friday");  
        break;  
    case 6:  
        printf("Saturday");  
        break;  
    case 7:  
        printf("Sunday");  
        break;  
}
```



Thursday

Operador Ternário Condicional (?:)

- É uma alternativa mais compacta ao if-else.

```
variable = condition ? expressionTrue : expressionFalse;
```

- No código acima, “variable” recebe “expressionTrue” se “condition” for verdadeira.
- Caso contrário, “variable” recebe “expressionFalse”.

Operador Ternário Condicional (?:)

- Exemplo:

```
int x, b = 10;  
if (b < 20)  
    x = 100;  
else  
    x = 200;
```

- É equivalente a:

```
int x, b = 10;  
x = b < 20 ? 100 : 200;
```

Operador Ternário Condicional (?:)

- Exemplo:

```
int time = 20;
if (time < 18) {
    printf("Good day.");
} else {
    printf("Good evening.");
}
```

- É equivalente a:

```
int time = 20;
(time < 18) ? printf("Good day.") : printf("Good evening.");
```


Estrutura de Repetição

- Imagine um algoritmo que tenha que imprimir os primeiros 1000 números começando pelo 1

ALGORITMO “Imprimir”

ESCREVA 1

ESCREVA 2

ESCREVA 3

ESCREVA 4

....

ESCREVA 1000

FIM_ALGORITMO

**Solução não
prática!**

- Solução: estrutura de repetição!
 - Certos tipos de problemas podem ser resolvidos com sequências de instruções executadas apenas uma vez;
 - Porém, outros algoritmos requerem a execução de determinados trechos de código várias vezes, por isso o surgimento das estruturas de repetição;
 - Uma estrutura de repetição permite que uma sequência de instruções seja executada várias vezes até que uma condição seja satisfeita;
 - Quando utilizar? Analisar se uma mesma sequência de instruções necessita ser executada várias vezes.

- Algoritmo comer um cacho de uva

ALGORITMO *“Comer cacho de uva”*

Pegar o cacho de uva

Lavar o cacho de uva

ENQUANTO (houver_uva && não_satisfeito) **FAÇA**

 Início

 Comer uva

 Fim

SE (cacho_vazio) **ENTÃO**

 Início

 Jogar cacho no lixo

 Fim

SENÃO

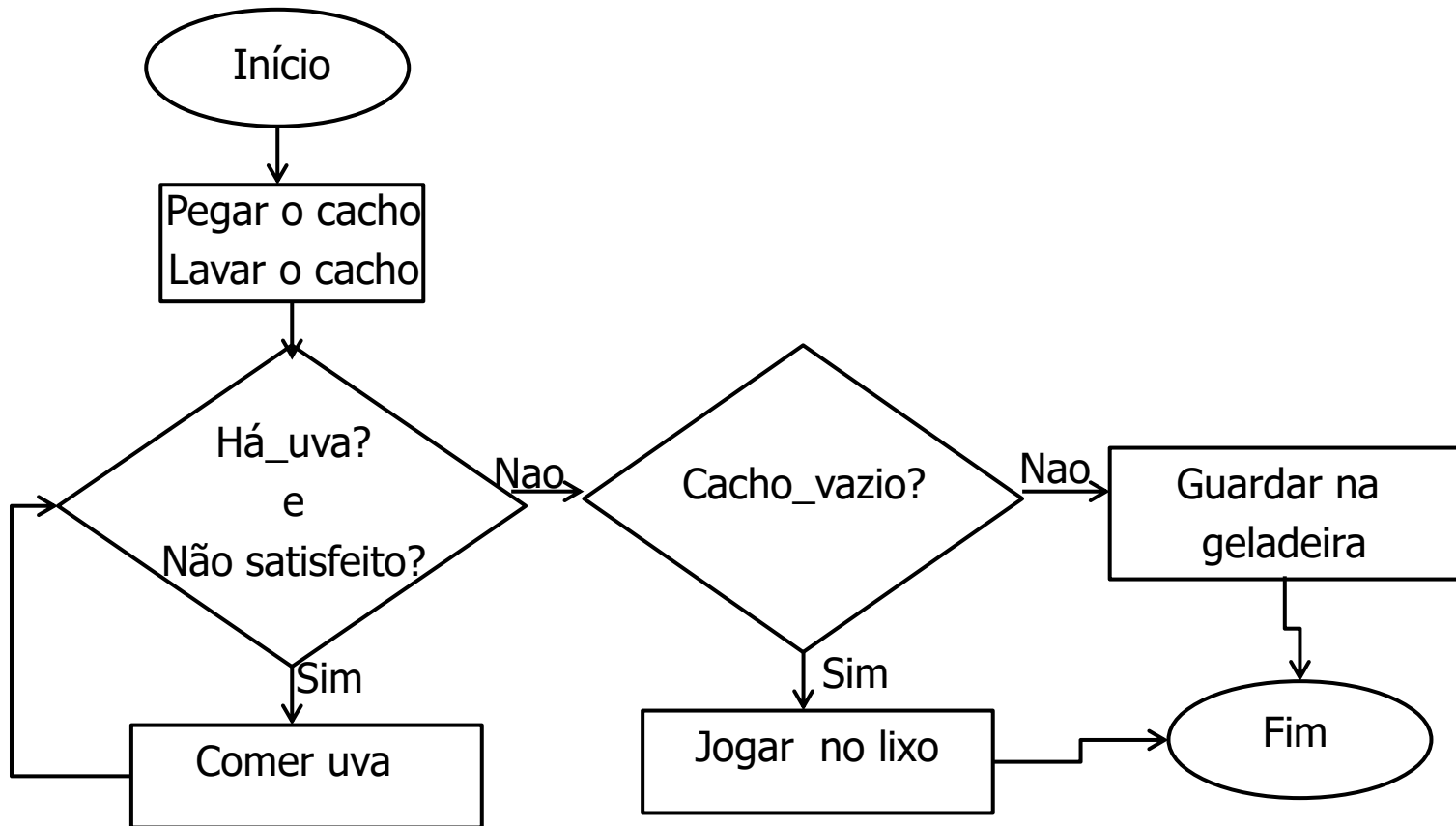
 Início

 Guardar cacho na geladeira

 Fim

FIM_ALGORITMO

- Fluxograma do Algoritmo



Tipos de Estruturas de Repetição

- Existem três tipos básicos de estruturas de repetição:
 - Com número definido de repetições
 - Estrutura **PARA**
 - Com número indefinido de repetições e teste no início
 - Estrutura **ENQUANTO**
 - Com número indefinido de repetições e teste no final
 - Estrutura **REPITA**

Tipos de Estruturas de Repetição

PARA

- Número definido de repetições: **PARA**
 - Utilizada quando se sabe o número de repetições que o trecho do algoritmo deve ser repetido (teste controlado por contador)
 - Sintaxe:

PARA **i** ← valor inicial até valor final **FAÇA**

INÍCIO

comando1

comando2

....

FIM

comando1 e **comando2** serão executados utilizando a variável **i** como controle, cujo valor variará de **valor inicial** até **valor final**

- Número definido de repetições: **PARA**

- Utilizada quando se sabe o número de repetições que o trecho do algoritmo deve ser repetido (teste controlado por contador)

PARA $i \leftarrow$ valor inicial **ATÉ** valor final **FAÇA**

comando1

comando1 será executado utilizando a variável **i** como controle, cujo valor variará de **valor inicial** até **valor final**

PARA $i \leftarrow$ valor inicial **ATÉ** valor final **FAÇA**

INÍCIO

comando1

comando2

comando1 e **comando2** serão executados utilizando a variável **i** como controle, cujo valor variará de **valor inicial** até **valor final**

FIM

PARA $i \leftarrow$ valor inicial **ATÉ** valor final **FAÇA PASSO** n

comando1

comando1 será executado utilizando a variável **i** como controle, cujo valor variará de **valor inicial** até **valor final**, com passo **n**

Tipos de Estruturas de Repetição PARA

- Exemplo: Pseudocódigo

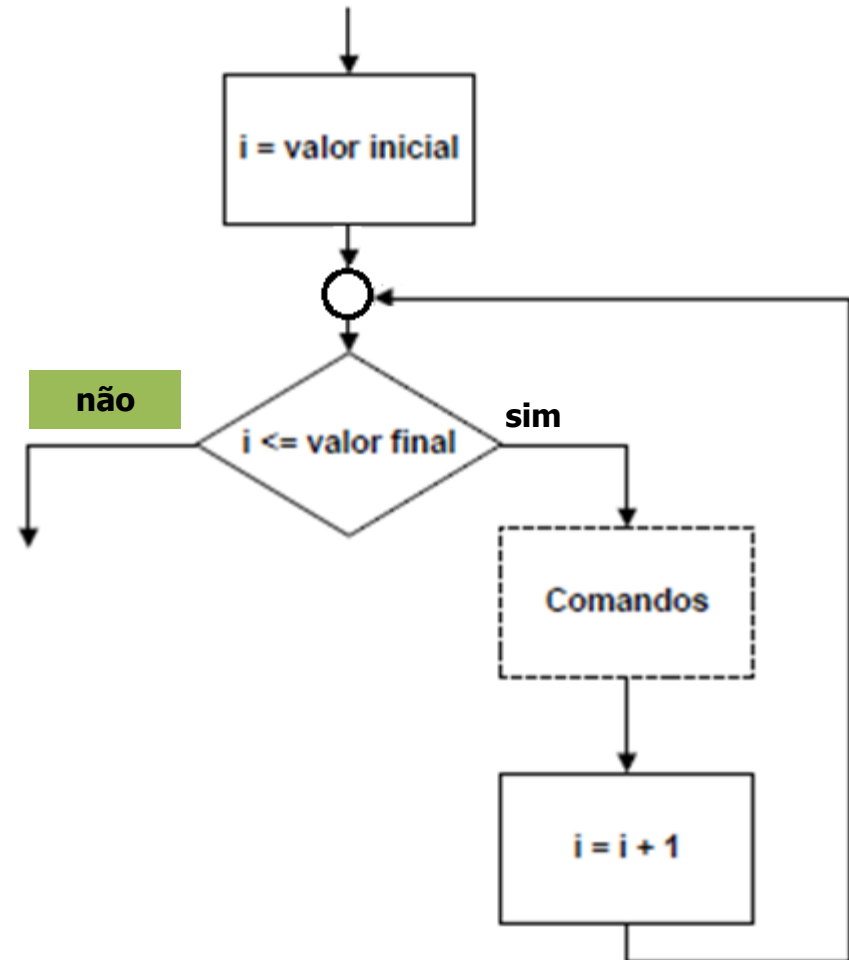
```
ALGORITMO  
  DECLARE a NUMÉRICO  
  PARA a ← 1 ATÉ 10 FAÇA  
    INÍCIO  
      ESCREVA “o valor de a é ”, a  
    FIM  
  FIM_ALGORITMO
```

- A estrutura de repetição **PARA**, no exemplo acima, repetirá o comando **ESCREVA** dez vezes (1 à 10)

Tipos de Estruturas de Repetição PARA

- Estrutura: **PARA**

Essa estrutura utiliza um **contador** que é inicializado com o **valor inicial** e, por meio **incrementos** (de 1 em 1) alcançará o **valor final** predefinido.



Tipos de Estruturas de Repetição PARA

ALGORITMO

DECLARE a NUMÉRICO

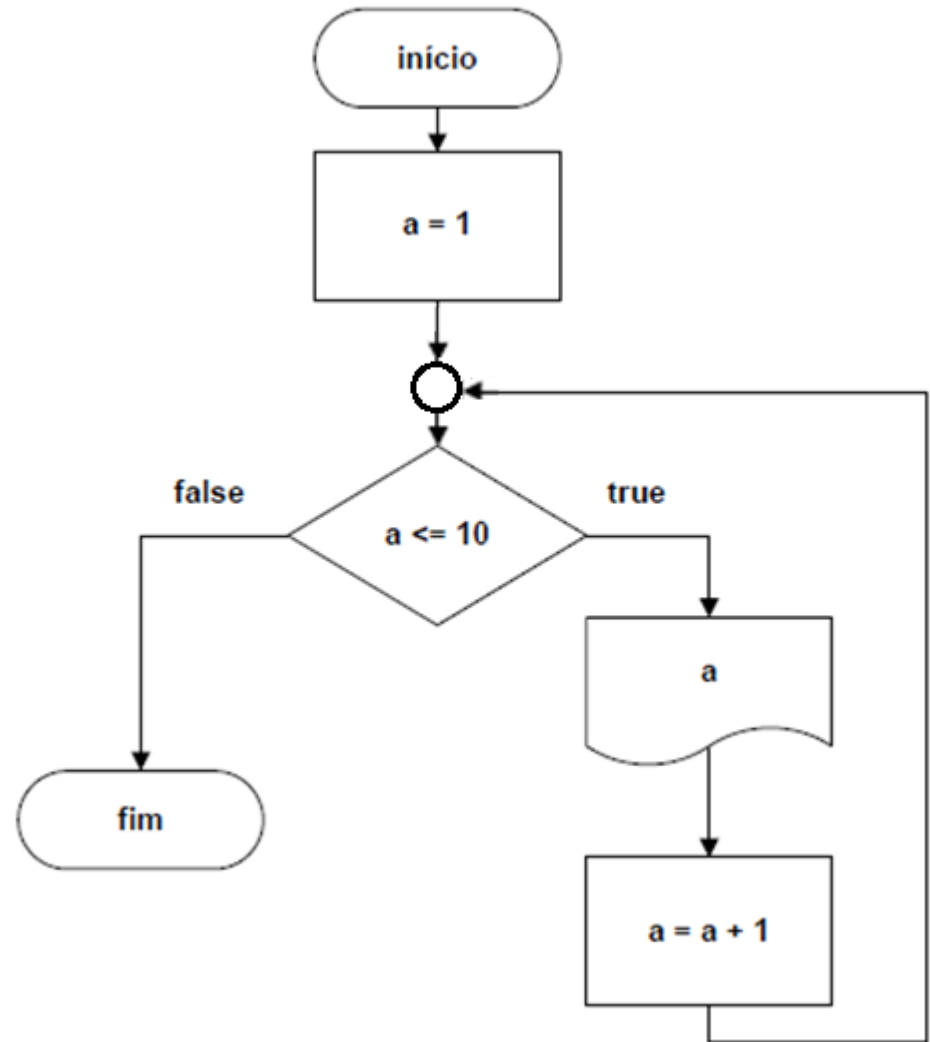
PARA a \leftarrow 1 ATÉ 10 FAÇA

INÍCIO

ESCREVA "o valor de a é ", a

FIM

FIM_ALGORITMO



Tipos de Estruturas de Repetição

PARA – for em C

- Estrutura de repetição – **for**

for (i = valor inicial; condição; incremento ou decremento de i)

{

comando1;

}

- Estrutura utilizada quando **se sabe** o número de vezes que o trecho do programa deve ser repetido
- Estrutura composta por 3 partes
 - 1ª: atribuição do **valor inicial** à variável de controle i.
 - 2ª: expressão relacional (**condição**) que, quando assume valor **falso**, determinará o fim das repetição (exemplo: **i <= valor final**)
 - 3ª: **incremento** ou **decremento** de i. Objetiva tornar a **condição** falsa em algum momento (exemplo: **i = i + 1** ou **i++**; **i = i – 1** ou **i--**)

Tipos de Estruturas de Repetição

PARA – for em C

- Exemplo:

```
#include <stdio.h>
int main ()
{
    for (int a =1; a<=10; a++)
    {
        printf("O valor de a e: %d \n", a);
    }
}
```

Tipos de Estruturas de Repetição

PARA – for em C++

- Exemplo: Faça o teste de mesa do código abaixo:

```
#include <stdio.h>
int main ()
{
    int val, a;
    val = 0;
    for (a =1; a<=20; a++)
    {
        printf("\n O valor de a e: %d", a);
        val = a + 1;
        printf("\n O valor de val e: %d", val);
    }
}
```



Resultado - tela

... continuação

```
0 valor de a e: 1
0 valor de val e: 2
0 valor de a e: 2
0 valor de val e: 3
0 valor de a e: 3
0 valor de val e: 4
0 valor de a e: 4
0 valor de val e: 5
0 valor de a e: 5
0 valor de val e: 6
0 valor de a e: 6
0 valor de val e: 7
0 valor de a e: 7
0 valor de val e: 8
0 valor de a e: 8
0 valor de val e: 9
0 valor de a e: 9
0 valor de val e: 10
0 valor de a e: 10
0 valor de val e: 11
0 valor de a e: 11
```

```
0 valor de val e: 12
0 valor de a e: 12
0 valor de val e: 13
0 valor de a e: 13
0 valor de val e: 14
0 valor de a e: 14
0 valor de val e: 15
0 valor de a e: 15
0 valor de val e: 16
0 valor de a e: 16
0 valor de val e: 17
0 valor de a e: 17
0 valor de val e: 18
0 valor de a e: 18
0 valor de val e: 19
0 valor de a e: 19
0 valor de val e: 20
0 valor de a e: 20
0 valor de val e: 21_
```

Tipos de Estruturas de Repetição

- Existem três tipos básicos de estruturas de repetição:
 - Com número definido de repetições
 - Estrutura **PARA**
 - Com número indefinido de repetições e teste no início
 - Estrutura **ENQUANTO**
 - Com número indefinido de repetições e teste no final
 - Estrutura **REPITA**

Tipos de Estruturas de Repetição

ENQUANTO

- Nº indefinido de repetições e teste no início: ENQUANTO
 - Vantagem: não necessita conhecer o nº de repetições
 - Sintaxe:

ENQUANTO condição FAÇA

INÍCIO

comando1

FIM

O comando1 será executado enquanto a condição for verdadeira

- Número indefinido de repetições: **ENQUANTO**

- Repete um trecho de algoritmo enquanto uma condição for verdadeira, fazendo teste sempre no início. Não necessita conhecer o número de repetições

ENQUANTO condição **FAÇA**

comando1

comando1 será executado enquanto a **condição** for verdadeira

ENQUANTO condição **FAÇA**

INÍCIO

comando1

comando2

FIM

comando1 e **comando2** serão executados enquanto a **condição** for verdadeira

Tipos de Estruturas de Repetição

ENQUANTO

- Exemplo: Pseudocódigo

```
ALGORITMO
  DECLARE X, Y NUMÉRICO
  X ← 1
  Y ← 5
  ENQUANTO X < Y FAÇA
    INÍCIO
      X ← X + 2
      Y ← Y + 1
    FIM
  FIM_ALGORITMO
```

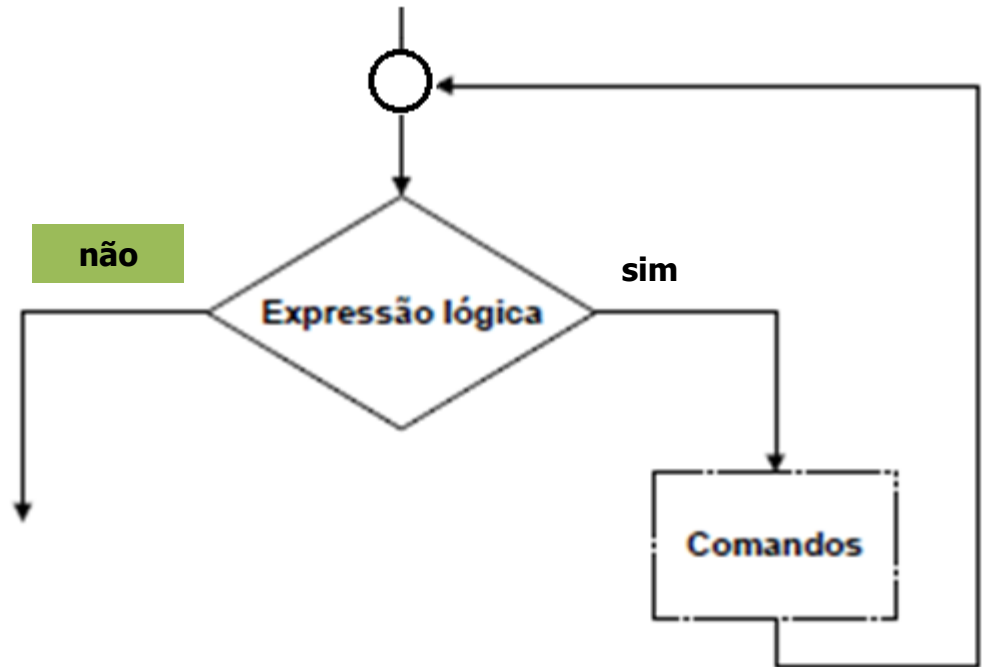
- A estrutura de repetição **ENQUANTO**, no exemplo acima, repetirá os comandos **enquanto** X for menor que Y

Tipos de Estruturas de Repetição

ENQUANTO

- Estrutura: **Enquanto**

Na figura ao lado, enquanto a expressão lógica for **verdadeira** os comandos serão executados; já quando for **falsa** o fluxo segue para outros comandos fora do laço de repetição



Tipos de Estruturas de Repetição

ENQUANTO

ALGORITMO

DECLARE X, Y NUMÉRICO

$X \leftarrow 1$

$Y \leftarrow 5$

ENQUANTO $X < Y$ FAÇA

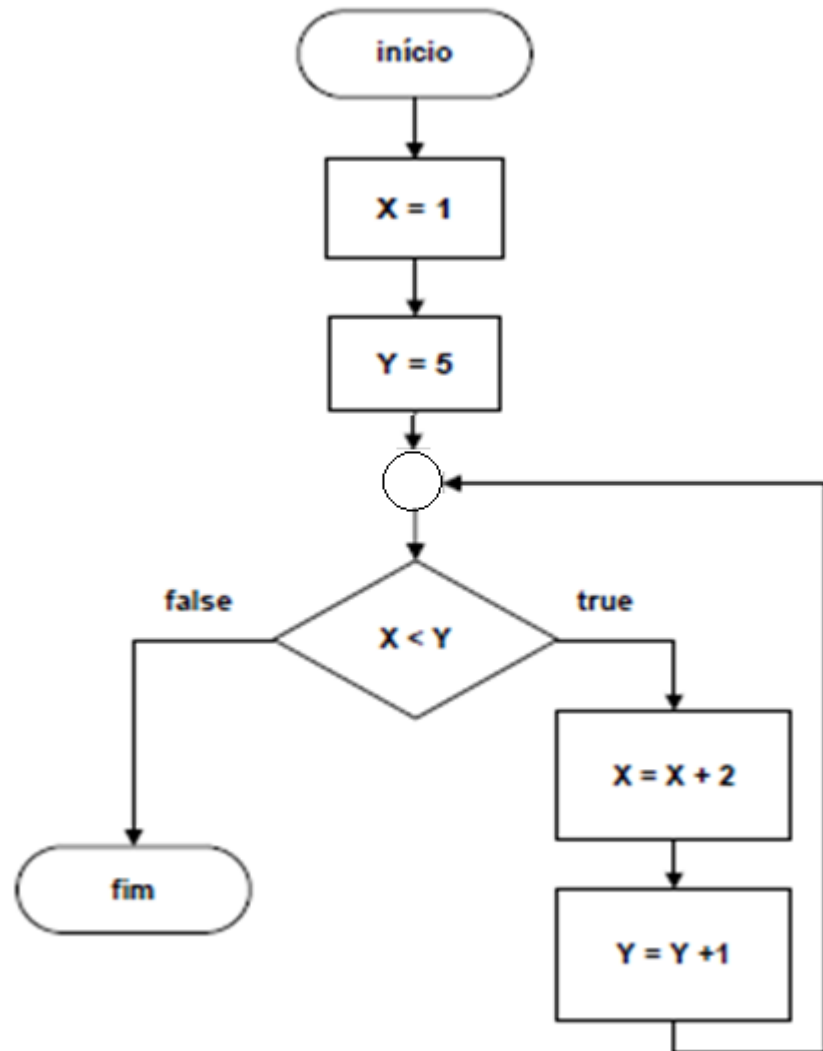
INÍCIO

$X \leftarrow X + 2$

$Y \leftarrow Y + 1$

FIM

FIM_ALGORITMO



Tipos de Estruturas de Repetição

ENQUANTO – while em C

- Estrutura de repetição – **while**

```
while (condição)
```

```
{
```

```
    comando1;
```

```
    comando2;
```

```
}
```

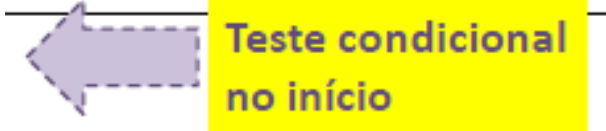
- **comando1** e **comando2** serão executados enquanto a **condição** for verdadeira

Tipos de Estruturas de Repetição

ENQUANTO – while em C

- Estrutura de repetição – **while**

```
while (condição)
{
    comando1;
    comando2;
}
```



Exemplo:

```
i = 1;
j = 3
while (i > j) {
    i = i + 3;
    j = j - 2;
}
```

- Existem situações em que o teste condicional (condição) da estrutura de repetição, que fica no **início**, resulta em **falso** logo na primeira comparação
- Nesses casos, os comandos dentro do laço de repetição **não serão executados**

Tipos de Estruturas de Repetição

ENQUANTO – while em C

- Exemplo:

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

```
int main ()
```

```
{
```

```
    int x, y;
```

```
    x = 1;
```

```
    y = 5;
```

```
    while (x < y)
```

```
    {
```

```
        x = x + 2;
```

```
        y = y + 1;
```

```
        printf("\\n O valor de x %d", x);
```

```
        printf("\\n O valor de y %d", y);
```

```
    }
```

```
}
```


Resultado exemplo anterior



Resultado - tela

```
valor de x 3  
valor de y 6  
valor de x 5  
valor de y 7  
valor de x 7  
valor de y 8  
valor de x 9  
valor de y 9_
```

Tipos de Estruturas de Repetição

- Existem três tipos básicos de estruturas de repetição:
 - Com número definido de repetições
 - Estrutura **PARA**
 - Com número indefinido de repetições e teste no início
 - Estrutura **ENQUANTO**
 - Com número indefinido de repetições e teste no final
 - Estrutura **REPITA**

Tipos de Estruturas de Repetição

REPITA

- Nº indefinido de repetições e teste no final: **REPITA**
 - Vantagem: não necessita conhecer o nº de repetições
 - Sintaxe:

REPITA

comandos

ATÉ condição

- Os **comandos** se **repetirão** até que a **condição** se torne **verdadeira**

Tipos de Estruturas de Repetição

REPITA

- Exemplo: Pseudocódigo

```
ALGORITMO
  DECLARE X, Y NUMÉRICO
  X ← 1
  Y ← 5
  REPITA
    X ← X + 2
    Y ← Y + 1
  ATÉ X >= Y
FIM_ALGORITMO
```

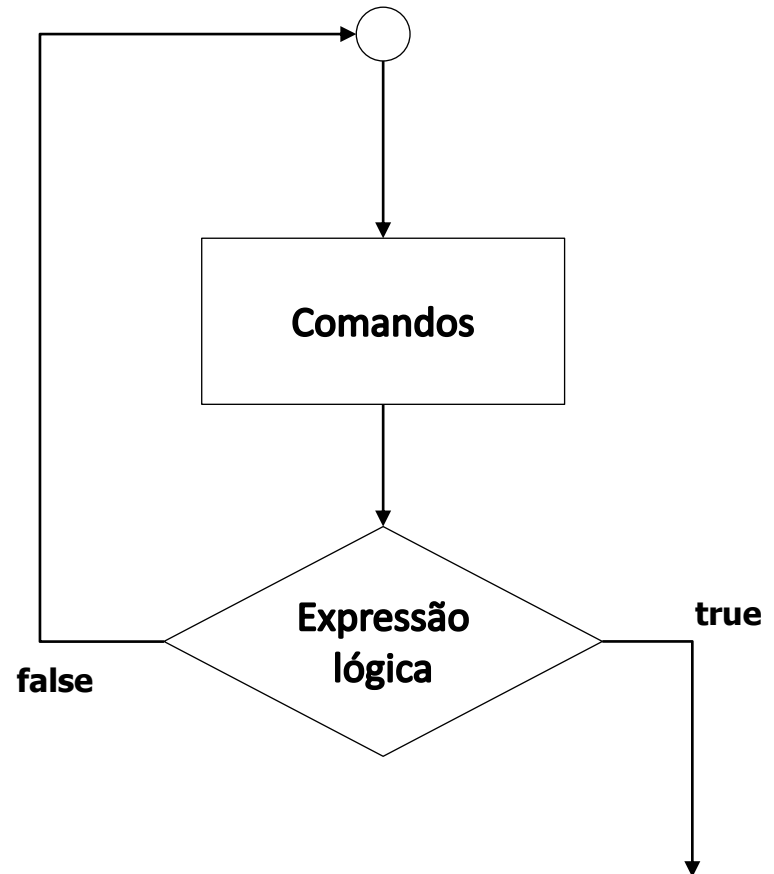
- A estrutura de repetição **REPITA**, no exemplo acima, repetirá os comandos **até que** X seja maior ou igual a Y

Tipos de Estruturas de Repetição

REPITA

- Estrutura: **Repita**

Na figura ao lado, os comandos **repetem** quando uma expressão lógica é **falsa**; já quando for **verdadeira** o fluxo segue para outros comandos fora do laço de repetição.



Tipos de Estruturas de Repetição

REPITA

ALGORITMO

DECLARE X, Y NUMÉRICO

$X \leftarrow 1$

$Y \leftarrow 5$

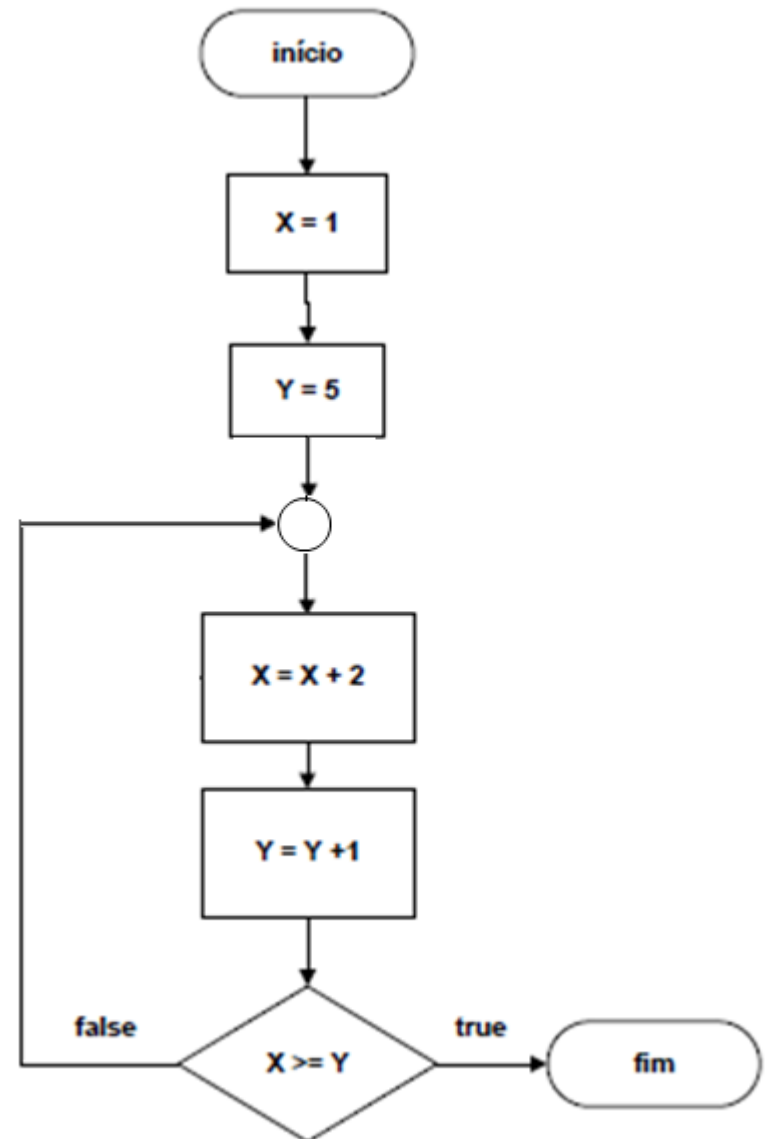
REPITA

$X \leftarrow X + 2$

$Y \leftarrow Y + 1$

ATÉ $X \geq Y$

FIM_ALGORITMO



Tipos de Estruturas de Repetição

REPITA – do while em C

- Estrutura de repetição – **do while**

```
do {  
    comando1;  
    comando2;  
} while ( condição );
```

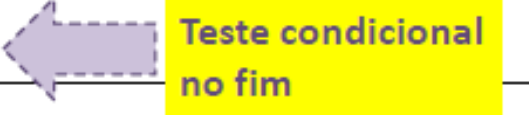
- **comando1** e **comando2** serão executados enquanto a **condição** for verdadeira
- Cuidado!!! Note que **do while** é um pouco diferente do **REPITA**:
 - Nos dois casos a condição é testada no final
 - Mas no **REPITA**, a repetição ocorre para uma **condição falsa**, enquanto no **do while** a repetição ocorre para uma **condição verdadeira**

Tipos de Estruturas de Repetição

REPITA – do while em C

- Estrutura de repetição – **do while**

```
do {  
    comando1;  
    comando2;  
} while ( condição);
```



Exemplo:

```
i = 1;  
j = 3  
do {  
    i = i + 3;  
    j = j - 2;  
} while (i>j);
```

- Existem situações em que o teste condicional (condição) da estrutura de repetição, que fica no **fim**, resulta em **falso** logo na primeira comparação
- Isso significa que os comandos dentro do laço de repetição **serão executados no mínimo uma vez**

Tipos de Estruturas de Repetição

REPITA – do while em C

- Exemplo:

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

```
int main ()
```

```
{
```

```
    int x = 0;
```

```
    do {
```

```
        printf("\n O valor de x = %d", x);
```

```
        x = x + 1;
```

```
    } while (x != 5);
```

```
}
```

Teste de mesa – Exemplo anterior

Tela	x	
	0	Valor inicial
Valor de x = 0	1	Valores obtidos dentro da estrutura da repetição
Valor de x = 1	2	
Valor de x = 2	3	
Valor de x = 3	4	
Valor de x = 4	5	Valor obtido dentro da estrutura de repetição, que torna a condição falsa e interrompe a repetição
Valor de x depois que sair da estrutura = 5		

- **Exercício 1 – Faça um programa que imprima os números 1,4,7,10,...,46,49,52.**

- **Exercício 2 – Faça um programa que imprima os números 52,49,46,...,10,7,4,1.**

- **Exercício 3 – Faça um programa imprima a soma e a média dos salários dos funcionários de uma empresa. Os salários serão digitados pelo usuário. A leitura dos salários é interrompida quando o usuário digitar um salário menor que 0. O programa deve imprimir, também, a quantidade de salários digitados.**

Referências bibliográficas

- **ASCENCIO, A. F. G. e CAMPOS, E. A. V. Fundamentos da Programação de Computadores – Algoritmos, Pascal e C/C++. São Paulo: Pearson Prentice Hall, 2007. 2ª Edição.**
- **SOUZA, A. Furlan; GOMES, Marcelo Marques; SOARES, Marcio Vieira e CONCILIO, Ricardo. Algoritmos e Lógica de Programação. 2ª ed. Ver. e ampl. São Paulo: Cengage Learning 2011.**