

# Programação estruturada

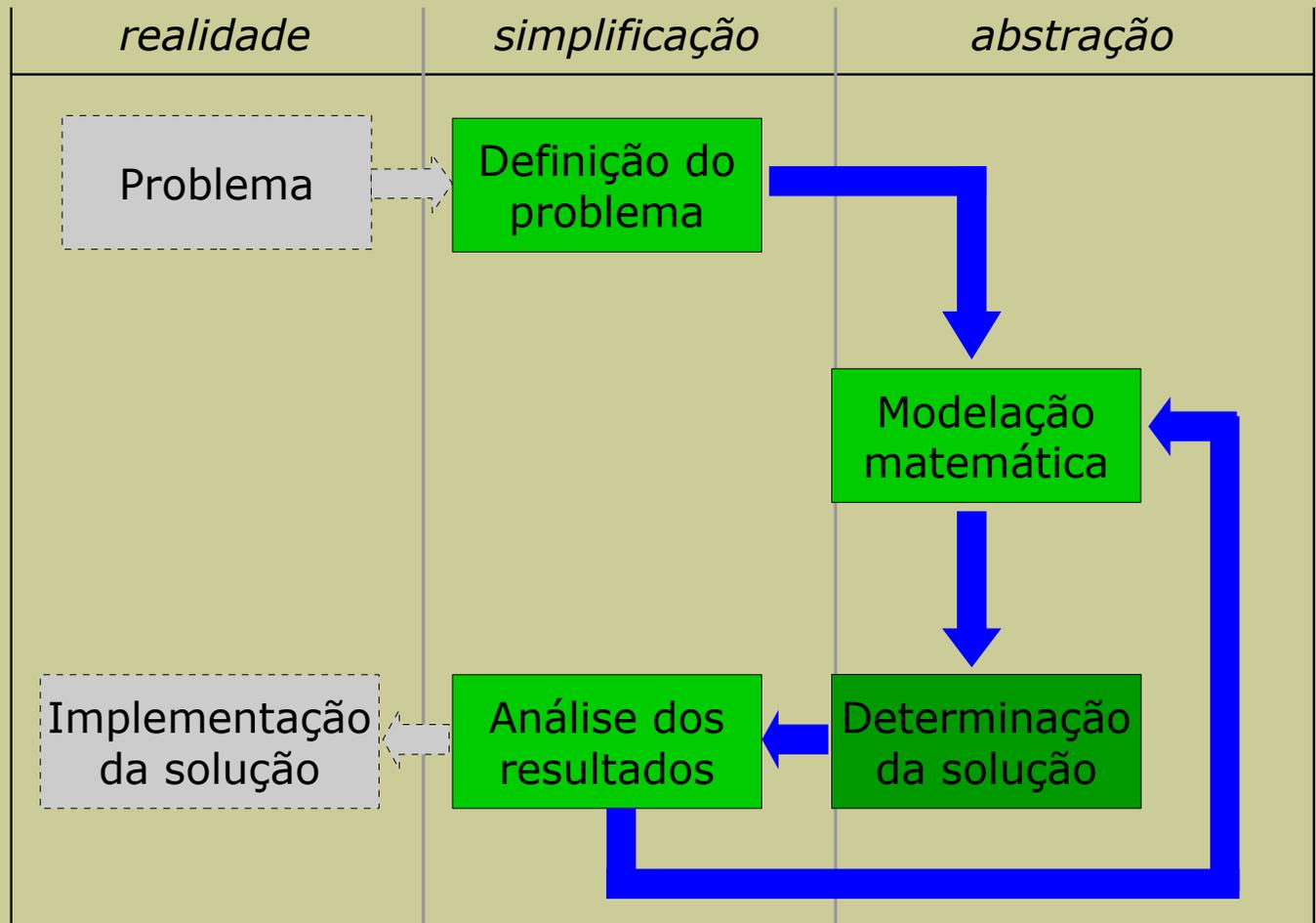
## #

# Desenho de algoritmos

## Resolução de um problema real

### A solução de um problema real pode ser obtida em 4 etapas

- Definição do problema
- Modelação matemática  
(construção do modelo)
- Determinação da solução
- Análise dos resultados
  - validação do modelo
  - análise da solução

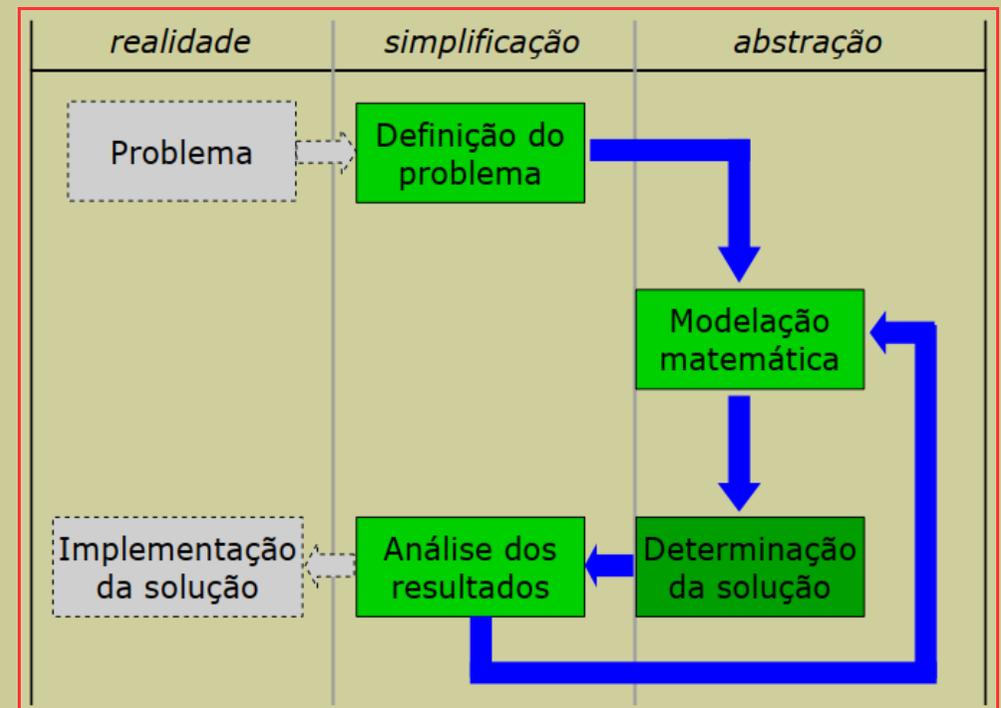


## Definição do problema

- Definir/descrever o problema real que se pretende resolver

## Modelação matemática (construção do modelo)

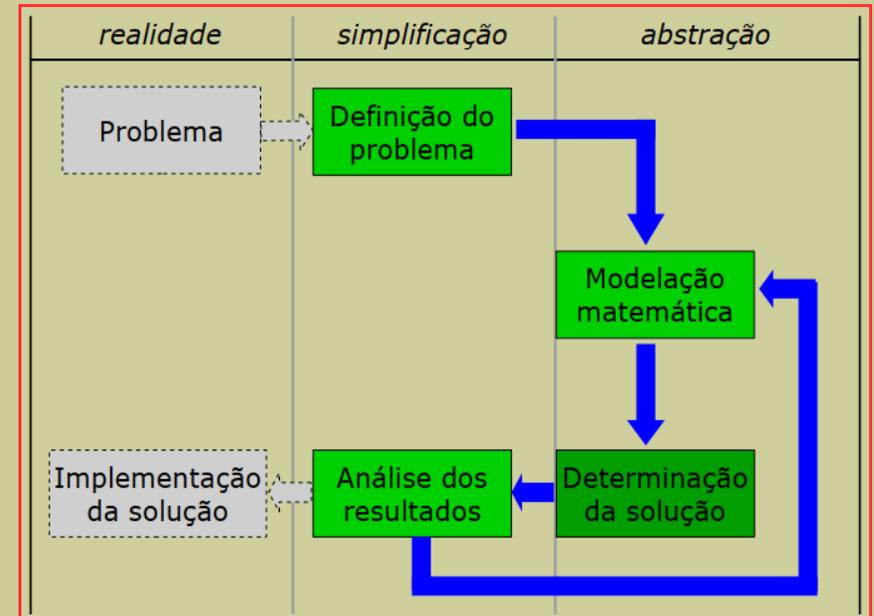
- O problema real é transformado num problema matemático, através de uma formulação matemática denominada **modelo matemático**
- Geralmente, o modelo matemático possui mais soluções que o problema real



## Determinação da solução

- Etapa composta pelas seguintes fases:

- **escolha do método** matemático mais apropriado para resolver o modelo matemático obtido na etapa da modelação matemática
  - **elaboração do algoritmo**, descrevendo o método matemático escolhido antes
  - **implementação do algoritmo**, traduzindo o algoritmo usando uma linguagem de programação de computadores (codificação do programa)
  - **obtenção da solução**, através da execução do programa
- A escolha do método mais apropriado deve considerar os seguintes aspetos:
- precisão desejada para os resultados
  - capacidade do método em conduzir aos resultados desejados
  - **esforço computacional despendido** (tempo de processamento e memória)



## Determinação da solução

### - Elaboração do algoritmo

- a descrição do algoritmo é feita através de um **conjunto de comandos** que, quando ativados (executados), resultam numa sucessão finita de ações (acontecimentos)
- em vez de se implementar um método diretamente numa linguagem de programação, é preferível descrevê-lo através de uma **notação algorítmica**
  - é possível abstrair-se dos detalhes da linguagem de programação do computador e concentrar-se apenas nos aspetos matemáticos do método (raciocínio matemático)
  - melhora o entendimento do algoritmo, facilitando assim a implementação do método em qualquer linguagem de programação

## Determinação da solução

### - Implementação do algoritmo (codificação do programa)

- o algoritmo é implementado na linguagem de programação escolhida
- uma vez que os aspetos matemáticos do método já foram pensados na fase de elaboração do algoritmo, agora apenas é necessário preocupar-se com os detalhes de implementação na linguagem adotada

### - Obtenção da solução (execução do programa)

- o código obtido da implementação do algoritmo deve ser executado pelo computador
- se for detetado algum erro lógico na fase de processamento (a execução do programa produzir resultados inesperados), então deve-se regressar à fase de elaboração do algoritmo para o corrigir
- se não for detetado qualquer erro lógico na fase de processamento, então foram obtidos os resultados do modelo matemático (a solução)

## Análise dos resultados

### - Verificar a consistência da solução obtida

- para o modelo matemático (validação do modelo), e
- a sua adequação ao *problema real* (análise da solução).

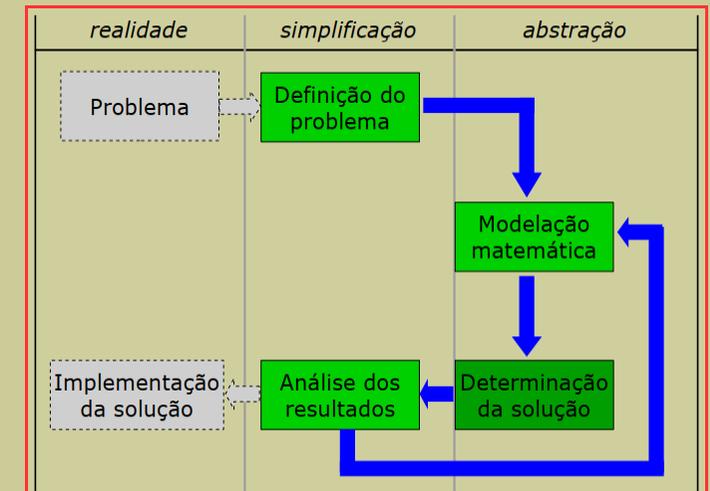
### - Validação do modelo

- se a solução não for satisfatória para o modelo (modelo não válido), então deve-se

- construir um novo *modelo matemático* (uma nova formulação matemática), e
- determinar as soluções do novo modelo matemático

### - Análise da solução

- alguns modelos matemáticos podem
  - produzir várias soluções (e não apenas uma), e
  - algumas delas (ou todas) não terem sentido (ex: tempo negativo, ...)
- um dos objetivos desta etapa é identificar qual a solução válida para o problema real entre as várias obtidas para o modelo matemático (se existirem)



## Programação estruturada

### História

- Nas décadas de 1950 e 1960, o desenvolvimento do hardware era o responsável pela expansão dos computadores
  - a maioria do investimento era feito no hardware,
  - a programação era vista como uma arte
- Na década de 1970, incentivados pela melhoria das características do hardware (diminuição do tamanho e do custo), os informáticos foram confrontados com projetos cada vez mais sofisticados, fazendo com que
  - houvesse uma inversão dos custos dispendidos com hardware (menos) e software (mais)
  - passasse a ser uma preocupação o problema da fiabilidade do software

## História

- Surge então a necessidade de transformar a tarefa de construir software numa atividade com rigor, comparável a uma disciplina de engenharia
  - nasce assim uma nova disciplina, a Engenharia de Software, cujo objectivo é a produção de Software de modo eficiente com custos controlados e segurança
- A produção de software passa por diferentes fases, como sejam: planeamento, análise, projeto, programação, implementação e manutenção
  - para cada uma das fases do desenvolvimento do software foram estudadas métodos e técnicas específicas
  - a **programação estruturada** é uma dessas técnicas, que permite fasear o processo de construção de um programa descrevendo-o de um modo não ambíguo (Algoritmo)

## Definição

- A **programação estruturada** (Paradigma da Programação Estruturada),
  - define um conjunto de regras para elaboração de programas/algoritmos,
  - baseia-se no desenho modular dos programas/algoritmos (programação modular), e
  - baseia-se no refinamento gradual do topo para a base (método descendente)
- Segundo este paradigma um programa pode ser definido da seguinte forma:

Programa = Estrutura de Dados + Algoritmo

- a estrutura de dados é o modo como os dados são organizados, acedidos e alterados
- o algoritmo manipula os dados, que podem ser de diversos tipos
- A Programação Estruturada permite o desenvolvimento de algoritmos usando um número restrito de estruturas básicas de controle, para a construção da lógica de um programa
  - qualquer algoritmo, independentemente da área de aplicação, da sua complexidade e da linguagem de programação usada na sua codificação, pode ser descrito através destas estruturas básicas de controlo

## Definição

- De acordo com o paradigma da programação estruturada, qualquer algoritmo pode ser descrito utilizando exclusivamente três estruturas básicas de controle:
  - de sequência (atribuição de valores, cálculo, entrada de dados e saída de dados)
  - de seleção/condição (condicional simples e condicional composta)
  - de repetição (ciclos)
- Com apenas estas três estruturas básicas, é possível construir algoritmos sem o uso de desvios incondicionais
  - os comandos de desvio incondicional (saltos), como o **goto**, não foram totalmente abolidos, mas o seu uso deve ser evitado sempre que possível
- A programação estruturada foi apresentada no início dos anos 1970 pelo suíço Niklaus Wirth, com o objetivo de suprimir ou limitar o uso do comando **goto**

## Características

- Uma técnica independente da linguagem de programação usada, que tem como objetivo construir programas claros, legíveis, eficientes e de fácil manutenção
- Programa escrito com indentação (realce), espaços em branco e comentários para facilitar a leitura do mesmo
- Desenho descendente e segmentação em módulos (subprogramas/funções)
- Construção de módulos (subprogramas/funções) de tamanho adequado
- Declaração do domínio de ação das variáveis locais (dentro das funções) e globais (no programa inteiro)
- Documentação do programa

## Observações

- O domínio da programação estruturada é a base para a aprendizagem de outras técnicas de programação como: a programação modular e a orientada a objetos
- Antes da programação estruturada, a técnica usada era a **programacao linear**, que usava desvios incondicionais (saltos) para construir a lógica do programa
  - a instrução **goto** era usada para implementar tais desvios incondicionais e o seu uso foi descontinuado com o aparecimento da programação estruturada
  - dependendo do tamanho do programa, esta abordagem podia tornar muito difícil a manutenção do programa para correção de erros ou simples evolução do programa
  - linguagens que usam este paradigma: FORTRAN

## Elaboração de Algoritmos

### Algoritmo, Estrutura de dados e Programa

- Um **algoritmo** é uma descrição detalhada (passo a passo) do método escolhido para determinar a solução do modelo matemático que representa o problema real
  - o problema real tem dados que têm que ser tratados computacionalmente
  - o modelo matemático, que foi construído para representar matematicamente o problema real, tem que definir os dados e os resultados do problema real matematicamente
  - o método que foi criado para resolver o modelo matemático, é definido através de expressões aritméticas e lógicas para tratar os dados e os resultados (solução)
- Uma **estrutura de dados** é a forma ou processo como os dados são organizados e tratados (accedidos e alterados)
- Um **programa** é a tradução de um algoritmo numa determinada linguagem de programação, de modo que possa ser executado pelo computador
  - um **programa** é a expressão de um **algoritmo**

## Algoritmo, Estrutura de dados e Programa

- Um programa pode ser definido da seguinte forma (segundo o PPE):

Programa = Estrutura de Dados + Algoritmo

- as estruturas de dados tratam os dados
- o algoritmo manipula os dados
- Processo de organização dos dados pode determinar a eficiência do algoritmo
  - algoritmos simples podem requerer estruturas de dados complexas
  - algoritmos complexos podem requerer estruturas de dados simples
- Para maximizar a eficiência do algoritmo
  - definir as estruturas de dados a usar em simultâneo com o desenvolvimento do algoritmo
- Os algoritmos eficientes são aqueles que usam “boas” estruturas de dados

## Passos na elaboração de um algoritmo

- A elaboração de algoritmos é uma das **fases mais importantes** na resolução de um problema real, pois traduz o método escolhido para resolver o modelo matemático construído numa forma próxima da linguagem de programação
- Passos na elaboração do algoritmo:
  - identificar os dados de entrada
  - identificar os dados de saída (resultados)
  - identificar o que é preciso para transformar os dados de entrada nos resultados (método dividir-para-conquistar)
  - construir o algoritmo
  - testar manualmente o algoritmo
  - executar o algoritmo

## Método Cartesiano de Dividir-para-Conquistar

- Também conhecido por **método descendente** (“*top-down method*”) ou **método de refinamento passo-a-passo**
- Consiste em
  - decompor o problema principal em subproblemas (modularização)
  - refinar os subproblemas até encontrar problemas mais simples,
  - refinar até se chegar a um nível de detalhe que permita implementar o algoritmo na linguagem de programação
- A modularização permite dividir o programa em módulos, com subprogramas claramente delimitados, que podem ser implementados, separadamente, por diversos programadores de uma equipa
- **Passo-a-passo**, significa que cada passo deve ser terminado antes de se iniciar o próximo passo
  - exemplo: é impossível “somar dois valores” antes de executar completamente o passo de introduzir/receber aqueles valores

## Características fundamentais de um Algoritmo

### - **Finitude**

- um algoritmo deve sempre terminar após um número finito de passos

### - **Definição**

- cada passo de um algoritmo deve ser definido com precisão
- as ações devem ser definidas rigorosamente e sem ambiguidades

### - **Entradas**

- um algoritmo deve ter 0 ou mais entradas (os dados que lhe são fornecidos no início)

### - **Saídas**

- um algoritmo deve ter 0 ou mais saídas (os dados obtidos e que tem uma relação específica com as entradas)

### - **Eficiência**

- todas as operações devem ser suficientemente básicas de modo que cada uma delas possa ser, em princípio, executada com precisão num tempo finito por um ser humano

## Representação de um algoritmo

- Existem várias formas de representar um algoritmo (**notação algorítmica**)
  - descrição narrativa (usando uma linguagem natural)
  - fluxograma (ou diagrama de fluxo)
  - pseudocódigo (ou pseudo-linguagem)
- Não existe consenso entre os especialistas sobre qual é a melhor maneira de representar um algoritmo
- No entanto, atualmente a maneira mais comum de representar algoritmos é através de fluxogramas e pseudocódigo

## Variáveis

### Definição

- Para resolver problemas no computador é necessário manipular dados, sejam números ou caracteres
- Se for necessário calcular o resultado de uma única operação, então provavelmente o melhor seria utilizar uma calculadora
- A utilidade de se escrever um programa aparece quando usamos **variáveis** que possuem a capacidade de guardar valores; é então possível calcular o resultado de várias operações
- As **variáveis** são identificadas por um nome (o chamado *identificador*)
- A denominação de **variável** deriva da possibilidade dos valores guardados nas variáveis poderem variar (poderem ser substituídos)

## Atribuição de valores

- Para um programa, uma variável é um pedaço de memória identificado por um nome, onde são guardados/armazenados dados
- Existem várias maneiras de indicar-se a atribuição de um valor a uma variável:

**A = 5**

**d ← 8**

- Numa linguagem de programação, isto é feito pela instrução mais importante das instruções, a denominada de “instrução de atribuição”
- A sintaxe (maneira como se escreve) varia de linguagem para linguagem
- O funcionamento é simples: após a execução da instrução pelo computador, a variável recebe/guarda o valor indicado
- Exemplo: a variável x recebe/guarda o valor 10
  - x ← 10** (linguagem algorítmica)
  - x = 10** (linguagem C)

## Notações algorítmicas

### Descrição narrativa

- É uma representação diretamente numa linguagem natural, usando expressões que evidencie como o problema pode ser resolvido computacionalmente
- Apesar de pouco usada na prática, pois o uso da linguagem natural muitas vezes causa má interpretação, ambiguidades e imprecisões, pode ser útil quando não se consegue abstrair uma solução computacional para o problema apresentado
- Não há um padrão da forma de representar um algoritmo
- Exemplo: calcular a soma de dois números e mostrar o resultado

1. inserir/ler um número
2. inserir/ler outro número
3. calcular a soma entre os dois números inseridos/lidos
4. mostrar/escrever o resultado da soma

## Fluxograma

- É uma representação gráfica de algoritmos, em que símbolos (ou formas) geométricos diferentes implicam ações (instruções, comandos) distintas
- Cada símbolo geométrico define a sua função genérica e no seu interior haverá uma descrição do passo do algoritmo, donde se inclui as representações das três estruturas básicas de controlo mencionadas antes: sequência, seleção e repetição
- Os símbolos geométricos são ligados entre si por setas que indicam o sentido do fluxo das ações
- É uma forma intermédia entre a descrição narrativa e o pseudocódigo, mas é mais precisa do que a primeira
- Não se preocupa com detalhes de implementação do programa, como o tipo das variáveis usadas
- Preocupa-se com detalhes de nível físico da implementação do algoritmo, como distinguir dispositivos onde ocorrem as operações de entrada e de saída de dados

## Fluxograma

### - Símbolos inicial e terminal

indica os pontos de início e de fim do fluxograma



**início/fim**

### - Símbolos de entrada/leitura e saída/escrita (*estruturas de sequência*)

indica passos envolvendo troca de dados com o exterior



**ler ...**  
**escrever ...**

### - Símbolo de sentido do fluxo

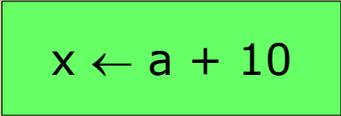
indica a origem e o destino do fluxo



## Fluxograma

- **Símbolo de processamento** (*estruturas de sequência*)

indica cálculo ou manuseio de dados



$x \leftarrow a + 10$

- **Símbolo de comparação** (*estruturas de seleção e de repetição*)

indica a seleção de uma de duas ações, segundo o resultado de uma condição lógica (**sim/verdadeiro** ou **não/falso**)



- **Símbolo de conexão**

indica a ligação de dois pontos do fluxograma (pode-se omitir)



## Pseudocódigo

- É a forma de representar algoritmos, podendo já indicar alguns requisitos que as linguagens de programação necessitam, tais como os tipos de variáveis e as instruções (comandos) similares aos usados em linguagens de programação
- O algoritmo é descrito numa forma muito próxima das linguagens de programação de computadores
- É também uma linguagem estruturada, pois deve seguir uma estrutura (ou padrão) que possui um formato semelhante ao das linguagens de programação, facilitando a codificação em qualquer linguagem de programação

## Pseudocódigo

### - Estrutura do algoritmo

- O algoritmo deve-se iniciar e terminar, respetivamente, com

```
algoritmo <nome-do-algoritmo>
```

```
fim_algoritmo
```

- Os dados necessários à execução do algoritmo são indicados pelo comando (opcional)

```
parâmetros de entrada: <lista-de-variáveis>
```

- onde <lista-de-variáveis> são os nomes das variáveis que recebem os dados iniciais
- Os dados obtidos pelo algoritmo (resultados) são indicados pelo comando (opcional)

```
parâmetros de saída: <lista-de-variáveis>
```

- onde <lista-de-variáveis> são os nomes das variáveis que recebem os resultados
- A escrita de comentários para clarificar o algoritmo (parte não executável)

```
{ <comentários> }
```

## Pseudocódigo

### - **Comando de atribuição** (estrutura de sequência)

- O símbolo ← é usado para atribuir o resultado de uma expressão a uma variável,

**<variável> ← <expressão>**

- onde <expressão> é uma expressão matemática ou um caráter

### - **Estruturas de entrada e saída de dados** (estruturas de sequência)

- Para a entrada/leitura de dados

**ler:** <lista-de-variáveis>

- indica a <lista-de-variáveis> que está disponível para receber valores

- Para a saída/escrita de dados

**escrever:** <lista-de-variáveis, mensagens>

- deve ser utilizado para
  - indicar quais e onde certos valores devem ser escritos,
  - mostrar mensagens

## Pseudocódigo

### - Estruturas alternativas ou condicionais (estruturas de seleção)

- Permite a escolha de um conjunto de comandos a ser executado, mediante se uma **condição lógica** é satisfeita ou não
- Estrutura simples

```
se <condição> então  
    <comandos>  
fim_se
```

- Estrutura composta

```
se <condição> então  
    <comandos_1>  
senão  
    <comandos_2>  
fim_se
```

## Pseudocódigo

### - Ciclos (estruturas de repetição)

- Permite que um conjunto de comandos seja executado repetidamente, mediante se uma **condição lógica** é satisfeita ou não (executa os comandos se a condição for satisfeita)
- Verifica a *condição* lógica antes de executar os comandos (verifica à entrada do ciclo)

```
enquanto <condição> repetir
```

```
    <comandos>
```

```
fim_enquanto
```

- Verifica a *condição* lógica depois de executar os comandos (verifica à saída do ciclo)

```
repetir
```

```
    <comandos>
```

```
enquanto <condição>
```

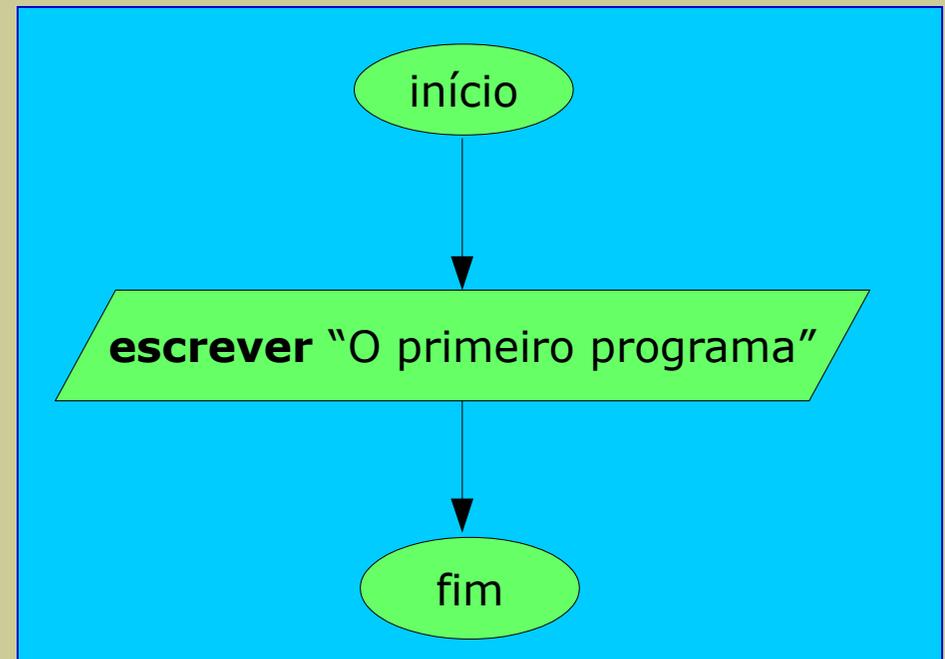
## Expressões

- Uma expressão é uma sequência de operandos e operadores
- Existem, essencialmente, dois tipos de expressões, dependendo do tipo de operadores e de operandos (valores e variáveis) envolvidos
  - aritméticas, e
  - lógicas
- Uma expressão aritmética é uma sequência de
  - operadores aritméticos
  - valores numéricos (inteiros e reais)
  - variáveis do tipo numérico com valores
  - funções matemáticas predefinidas: trigonométricas e numéricas
- Uma expressão lógica (**condição**) é uma sequência de
  - operadores relacionais e lógicos
  - expressões matemáticas e caracteres

## Exemplo 1

- Enunciado:  
escrever a mensagem: "O primeiro programa"
- Algoritmo:

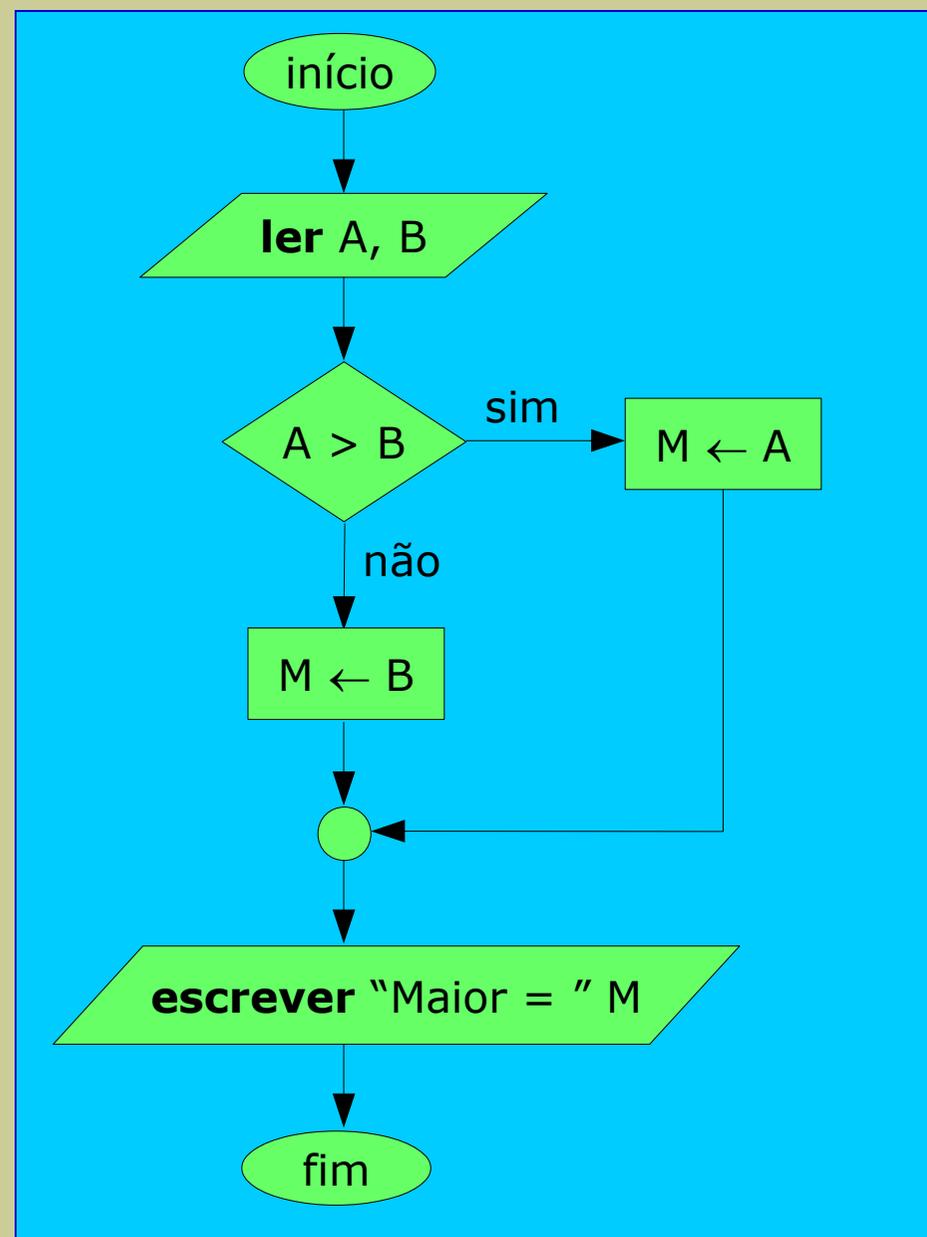
```
algoritmo primeiroPrograma  
  escrever: "O primeiro programa."  
fim_algoritmo
```



## Exemplo 2

- Enunciado:  
determinar o maior de dois números
- Algoritmo:

```
algoritmo maiorDe2Numeros  
parâmetros de entrada: A, B (inteiro)  
parâmetros de saída: M (inteiro)  
ler: A, B  
se (A > B) então  
    M ← A  
senão  
    M ← B  
fim_se  
escrever: "Maior = ", M  
fim_algoritmo
```



### Exemplo 3

- Enunciado:  
somar quatro números
- Algoritmo 1:

**algoritmo** soma4Numeros

**parâmetros de entrada:** A, B, C, D (real)

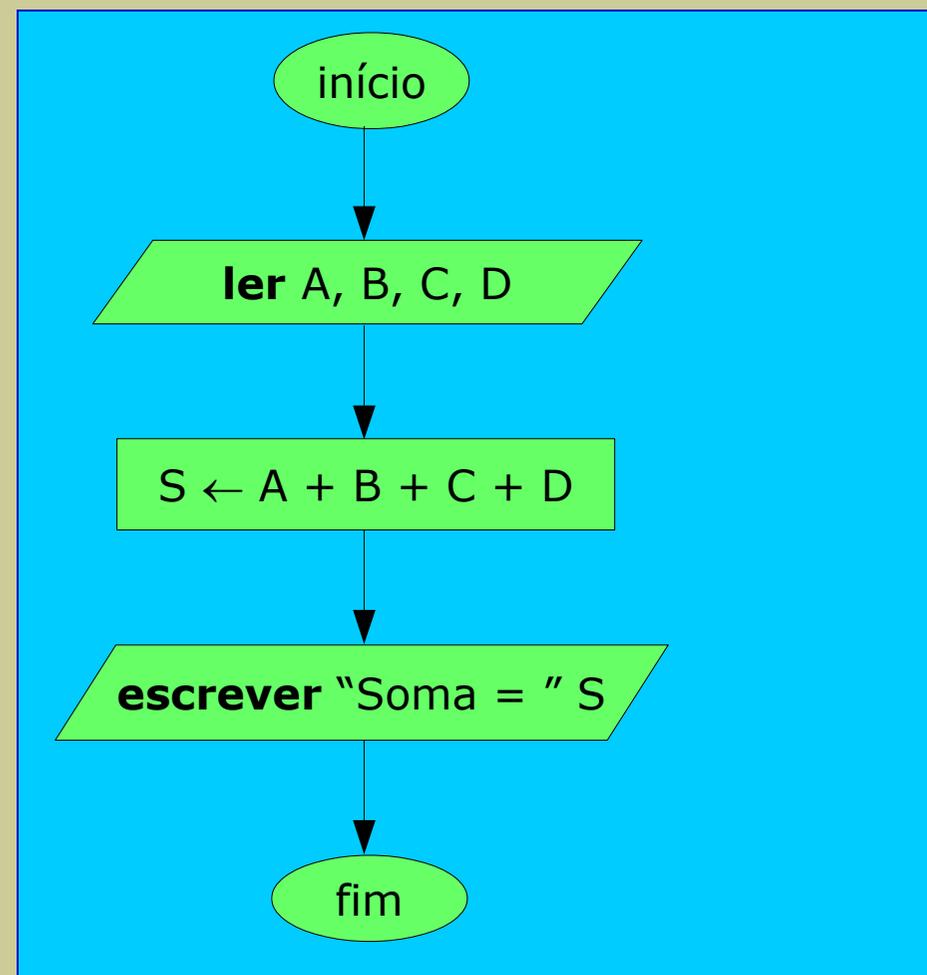
**parâmetros de saída:** S (real)

**ler:** A, B, C, D

$S \leftarrow A + B + C + D$

**escrever:** "Soma = ", S

**fim\_algoritmo**



### Exemplo 3

- Enunciado:  
somar quatro números
- Algoritmo 2:

**algoritmo** soma4Numeros

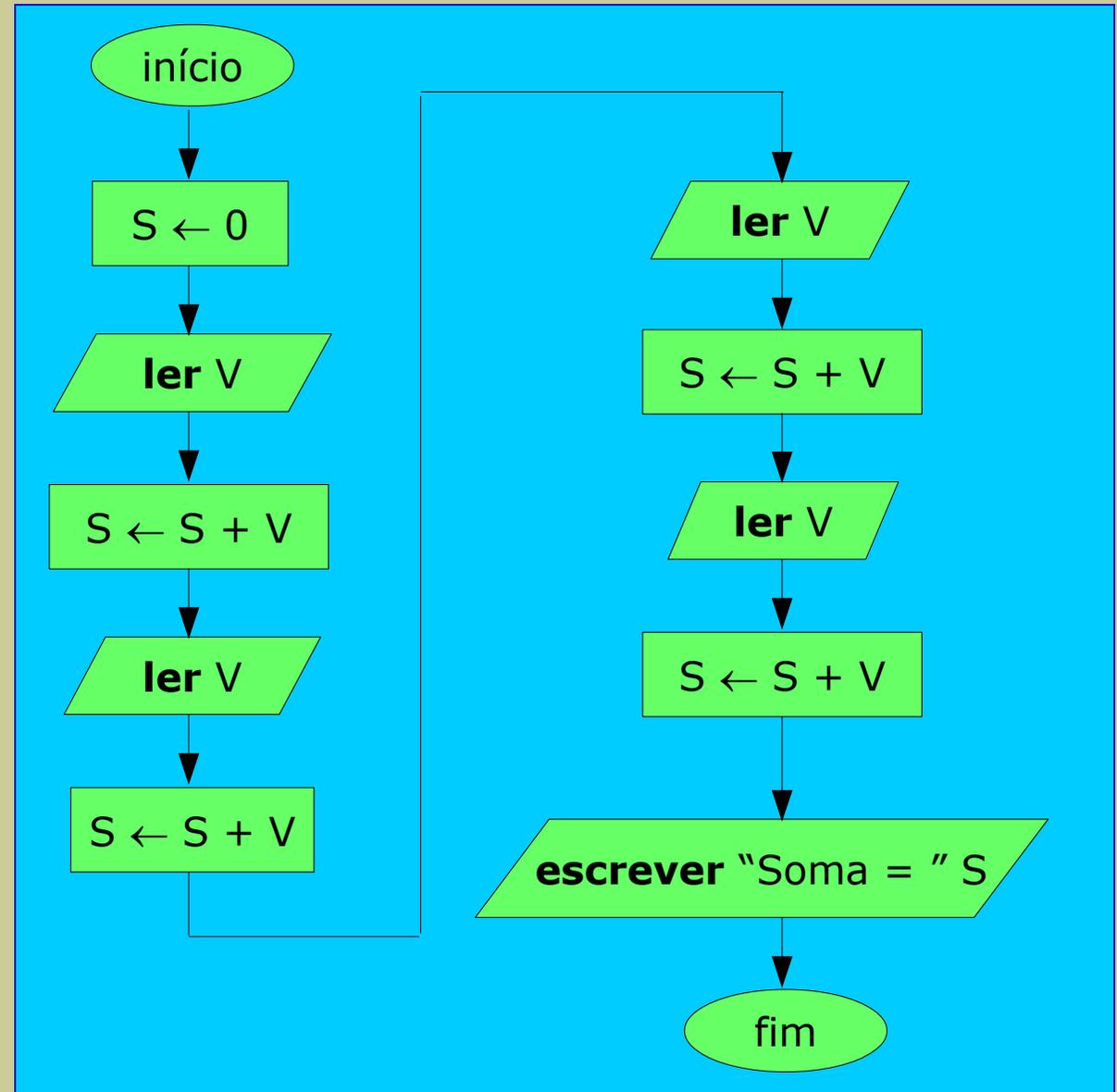
$S \leftarrow 0$

**ler:** V

$S \leftarrow S + V$

**escrever:** "Soma = ", S

**fim\_algoritmo**



## Exemplo 3

- Enunciado:

somar quatro números

- Pergunta:

- Será que não existe um algoritmo mais simples do que os anteriores?
- Imagine-se que em vez de somar 4 números, pretendia-se somar 100 números.

### Exemplo 3

- Enunciado:  
somar quatro números
- Algoritmo 3:

**algoritmo** soma4Numeros

$S \leftarrow 0$

$I \leftarrow 1$

**enquanto** ( $I \leq 4$ ) **repetir**

**ler:** V

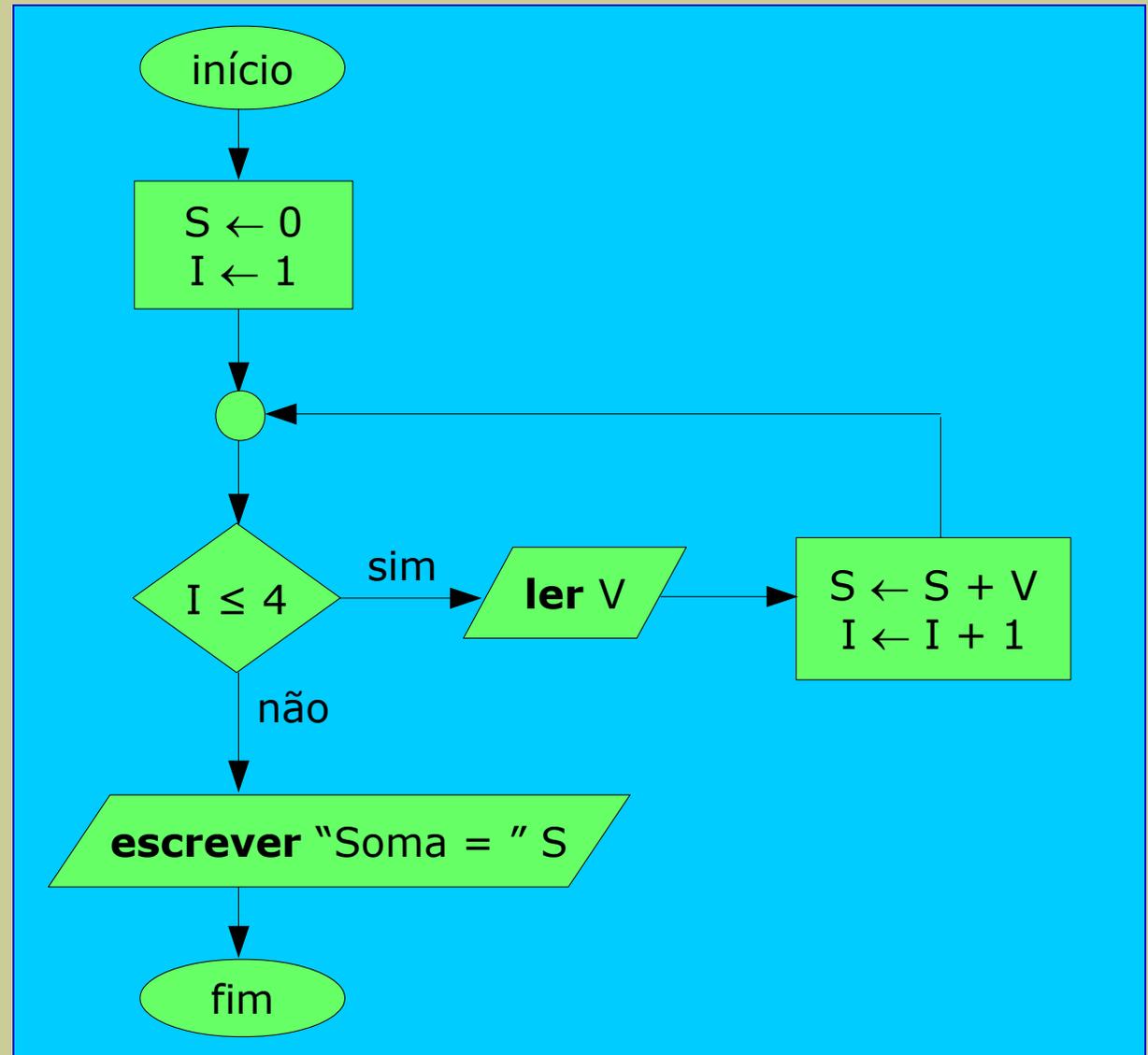
$S \leftarrow S + V$

$I \leftarrow I + 1$

**fim\_enquanto**

**escrever:** "Soma = ", S

**fim\_algoritmo**



### Exemplo 3

- Algoritmo 3:

- observe-se que se utilizou um **ciclo** (estrutura de repetição), que

- primeiro testa a condição para continuar a somar ou terminar

$$I \leq 4$$

- só depois é que são realizadas as acções

ler V

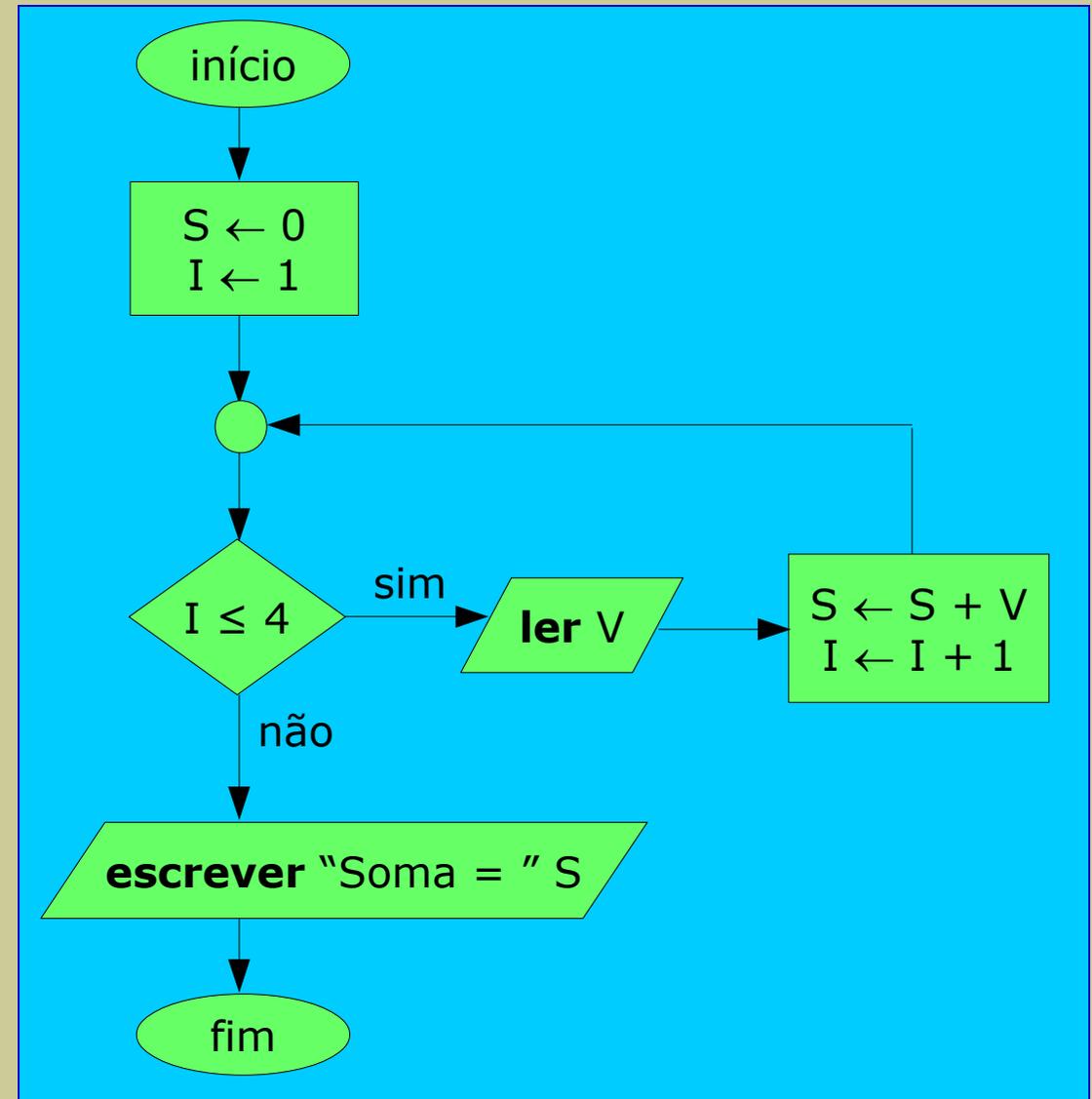
$S \leftarrow S + V$  (somar)

$I \leftarrow I + 1$  (contar números)

- Seria possível fazer ao contrário?

- primeiro realizar as acções,

- e só depois testar a condição para continuar a somar ou terminar



### Exemplo 3

- Enunciado:  
somar quatro números
- Algoritmo 4:

**algoritmo** soma4Numeros

$S \leftarrow 0$

$I \leftarrow 1$

**repetir**

**ler:** V

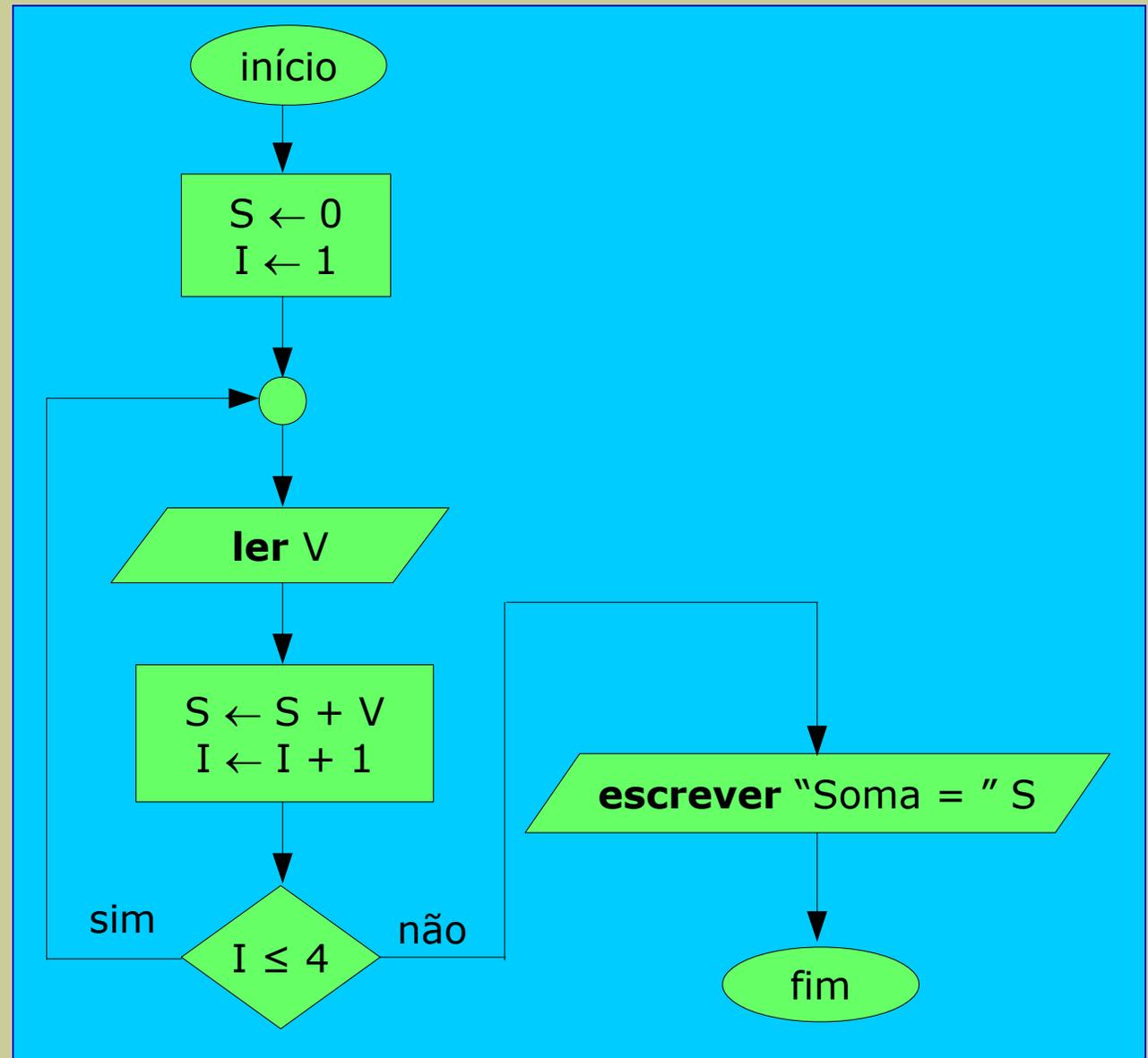
$S \leftarrow S + V$

$I \leftarrow I + 1$

**enquanto** ( $I \leq 4$ )

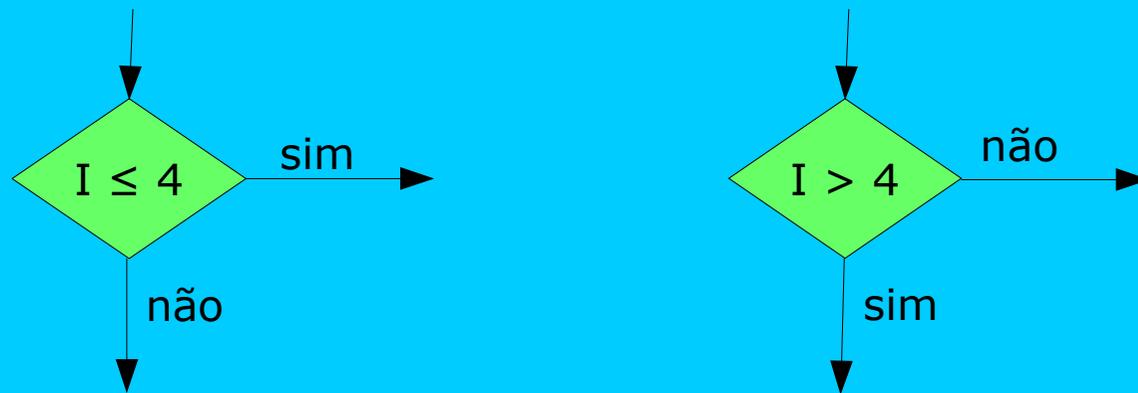
**escrever:** "Soma = ", S

**fim\_algoritmo**



### Exemplo 3

- Se na comparação pretender-se trocar o "sim" pelo "não", basta fazer a pergunta ao contrário ("a negação")
- As comparações que se seguem são equivalentes.



## Exercício

- Enunciado:

calcular o produto de 20 números

- Resposta 1

- usar um ciclo (estrutura de repetição) com verificação da sua condição no início (à entrada) do ciclo
- usar um ciclo (estrutura de repetição) com verificação da sua condição no fim (à saída) do ciclo

## Exercício

- Enunciado:  
calcular produto de 20 números
- Algoritmo 1:

**algoritmo** produto20Numeros

$P \leftarrow 1$

$I \leftarrow 1$

**enquanto** ( $I \leq 20$ ) **repetir**

**ler:** A

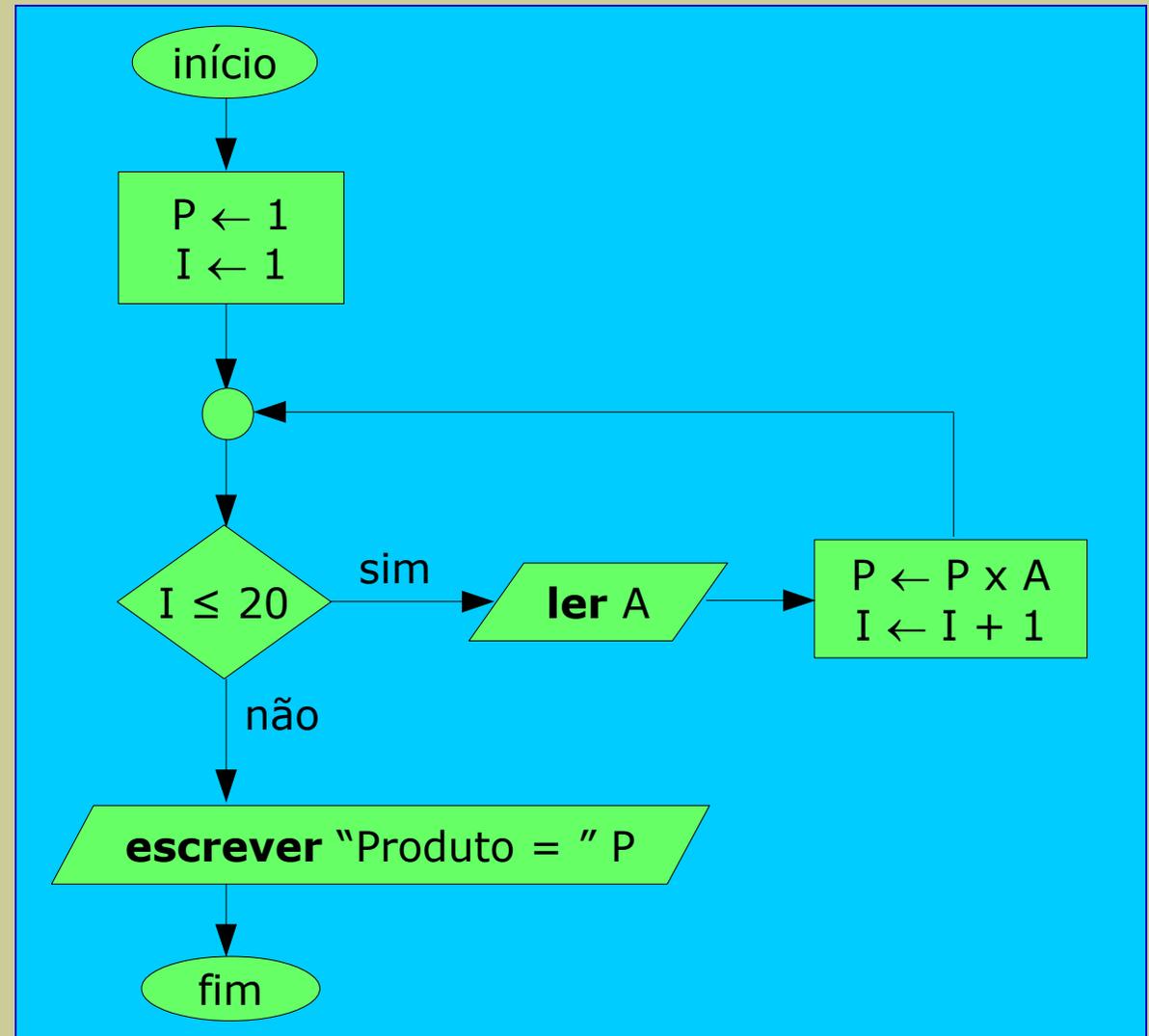
$P \leftarrow P \times A$

$I \leftarrow I + 1$

**fim\_enquanto**

**escrever:** "Produto = ", P

**fim\_algoritmo**



## Exercício

- Enunciado:  
calcular produto de 20 números
- Algoritmo 2:

**algoritmo** produto20Numeros

$P \leftarrow 1$

$I \leftarrow 1$

**repetir**

**ler:** A

$P \leftarrow P \times A$

$I \leftarrow I + 1$

**enquanto** ( $I \leq 20$ )

**escrever:** "Produto = ", P

**fim\_algoritmo**

