

# Instruções de repetição (ciclos)

## Instrução de repetição (Ciclo)

### Definição

- Uma **instrução de repetição** faz com que uma instrução (ou bloco de instruções) seja executada de forma repetida
- Uma **instrução de repetição** designa-se normalmente por **ciclo**
- Um ciclo utiliza uma **condição**, cujo resultado (verdadeiro ou falso) determina se a repetição deve continuar ou terminar
  - esta condição denomina-se por **condição do ciclo** ou **condição de paragem**
- Sempre que a instrução (ou bloco de instruções) de um ciclo é executada, diz-se que se realizou uma **iteração** (ou um **passo**) do ciclo
- A condição do ciclo será testada a cada iteração/passo do ciclo

## Funcionamento

- Um ciclo pode funcionar de duas formas:
  - Caso 1 (teste da condição à entrada do ciclo):
    - primeiro testa a condição do ciclo,
    - depois, em função do resultado daquela condição,
      - executa a instrução (ou bloco de instruções), ou
      - pára o processo
  - Caso 2 (teste da condição à saída do ciclo):
    - primeiro executa a instrução (ou bloco de instruções),
    - depois testa a condição do ciclo para verificar se
      - continua para mais uma iteração, ou
      - termina o processo

## Instrução (ciclo) “while”

### Definição

- A **instrução “while”** executa (“repete”) uma dada instrução (ou bloco de instruções) enquanto uma determinada condição é verdadeira (Caso 1)
  - enquanto a condição for verdadeira, executa a instrução (ou bloco de instruções)

## Sintaxe

- Na linguagem C:

```
while (condição)
```

```
    instrução;
```

em que,

- **condição**

- é uma expressão lógica (o resultado é “verdadeiro/sim” ou “falso/não”),
- tem **obrigatoriamente** que estar **entre parentesis**

- **instrução**

- é uma instrução de qualquer tipo, incluindo instruções de repetição
- é uma instrução simples ou um bloco de instruções

## Sintaxe

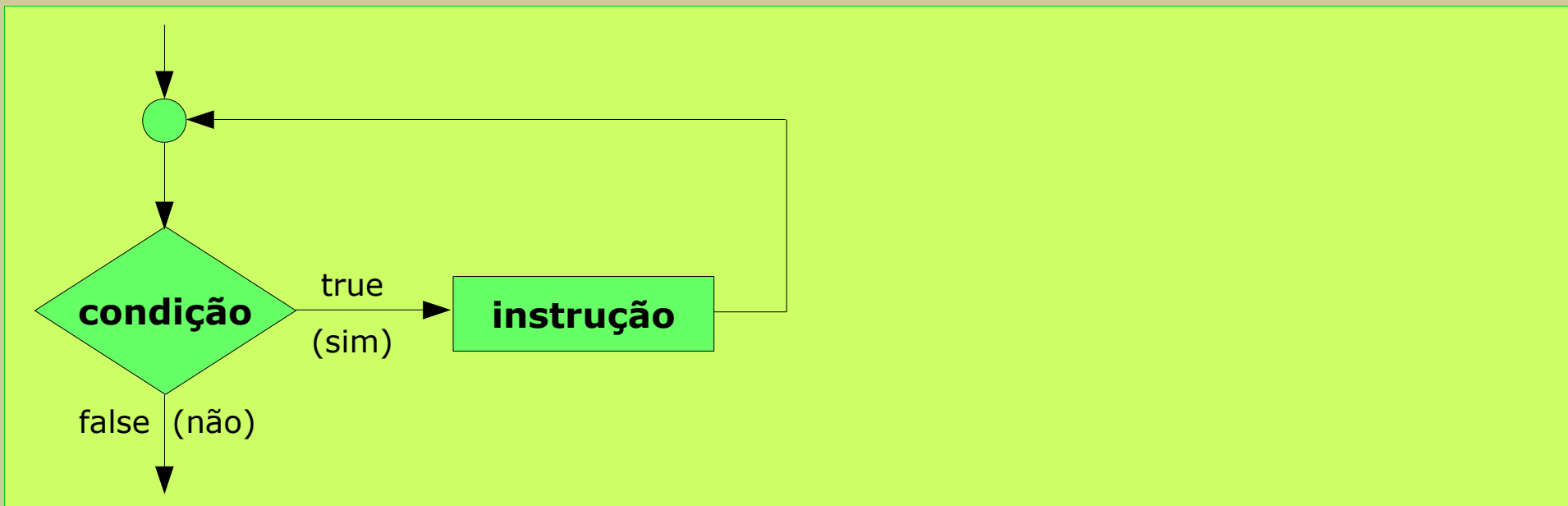
- Em pseudocódigo

```
enquanto <condição> fazer
```

```
  <comandos>
```

```
fim_enquanto
```

- Em fluxograma



## Observações

- Geralmente a instrução do ciclo é um bloco de instruções
- Neste bloco de instruções deve estar incluído uma instrução que faça alterar o valor de uma variável que controle a continuação ou paragem do ciclo (variável de controlo)
- A alteração de valor da variável de controlo pode ser feita usando
  - uma instrução de atribuição, ou
  - uma instrução de leitura

## Exemplo

- Enunciado:

Construir um programa para mostrar no monitor várias vezes (repetidas) a mensagem “Bom dia”

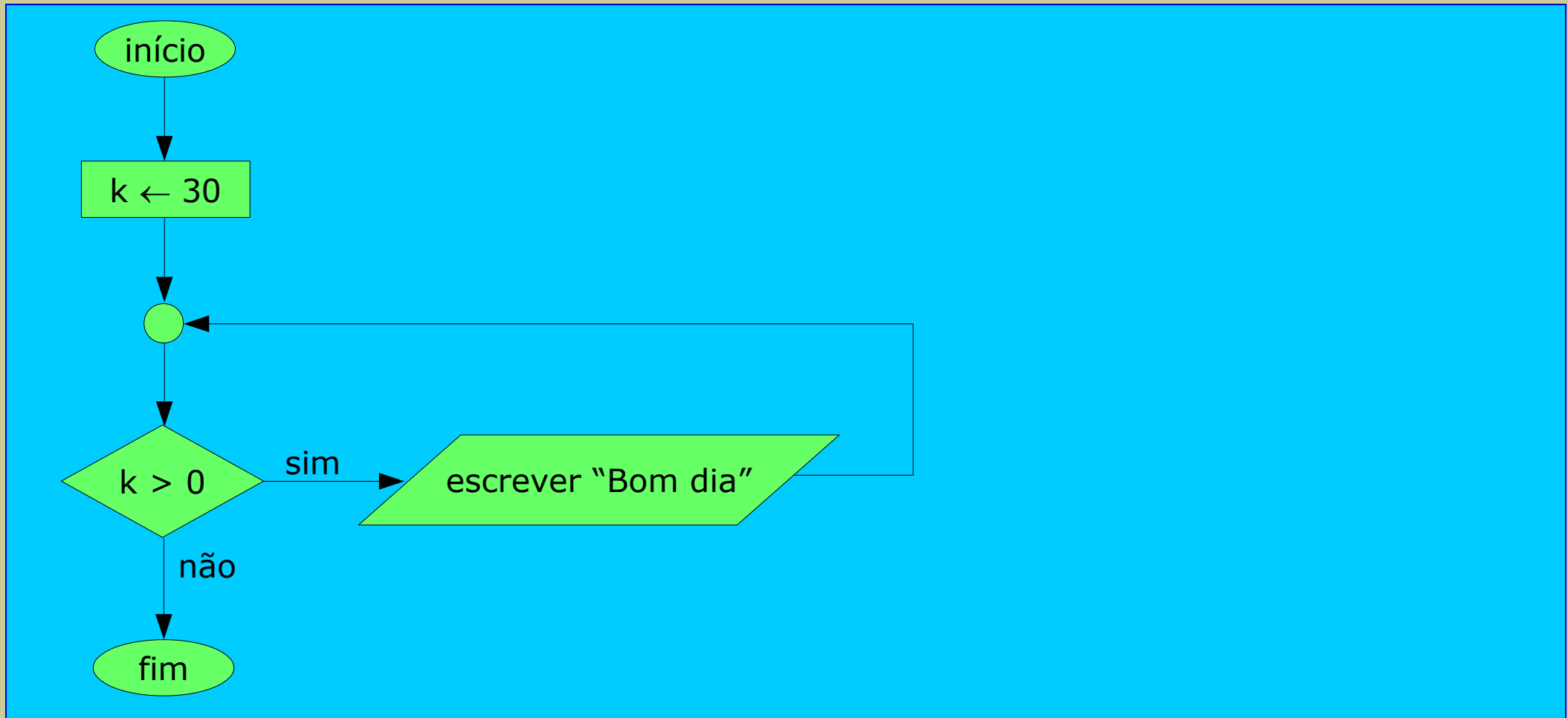
- Estratégia 1:

- usar uma variável com valor inicial igual ao número de vezes que se pretende repetir a mensagem (por exemplo, 30)



## Exemplo

- Algoritmo em fluxograma



## Exemplo

- Algoritmo em pseudocódigo

```
algoritmo mensagem30  
  k ← 30  
  enquanto (k > 0) fazer  
    escrever: “Bom dia”  
  fim_enquanto  
fim_algoritmo
```

## Exemplo

- Programa

```
#include <stdio.h>
void main ()
{
    int k;
    k = 30;
    while (k > 0)
        printf("Bom dia\n");
}
```

- Questão 1: O resultado deste programa é o pretendido?
- Questão 2: Se não é, o que faz então?
- Questão 3: Se não é, como resolver?

## Exemplo

- Resposta à questão 1: O resultado é o pretendido?
  - não faz o pretendido
- Resposta à questão 2: O que faz então?
  - o programa escreve “Bom dia” um número indefinido de vezes (nunca pára)
    - diz-se que o programa entra em ciclo infinito (está em “looping”),  
provocado por um erro de lógica do algoritmo (CUIDADO)

## Exemplo

- Resposta à questão 3: Como resolver?
  - começar por escolher o número de repetições,
    - que corresponde ao número de vezes que se pretende escrever a mensagem
    - por exemplo, 30
  - usar uma variável de controlo
    - para contabilizar o número de iterações do ciclo, que tem que ser igual a 30
    - ou seja, funciona como um “contador”

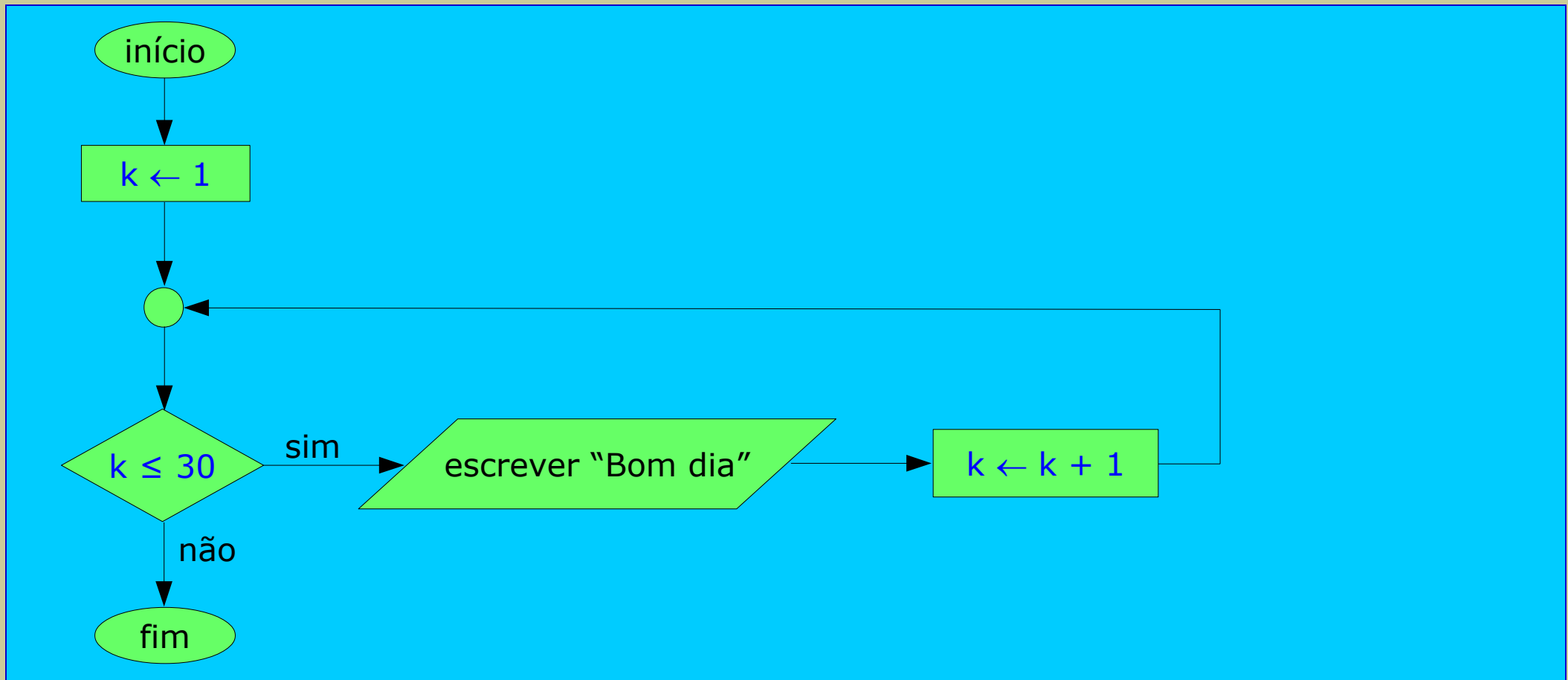
## Exemplo

- Resposta à questão 3: Como resolver?
- Estratégia 1
  - atribuir à variável de controlo o valor 1
  - em cada iteração do ciclo, incrementar a variável de controlo em uma unidade
  - terminar o ciclo quando a variável de controlo ultrapassar o valor 30

## Exemplo

- Estratégia 1

Algoritmo em fluxograma



## Exemplo

- Estratégia 1

Algoritmo em pseudocódigo

```
algoritmo mensagem30
   $k \leftarrow 1$ 
  enquanto ( $k \leq 30$ ) fazer
    escrever: “Bom dia”
     $k \leftarrow k + 1$ 
  fim_enquanto
fim_algoritmo
```



## Exemplo

- Estratégia 1

Programa em C

```
#include <stdio.h>
void main ()
{
    int k;
    k = 1;
    while (k <= 30)
    {
        printf("Bom dia\n");
        k = k + 1;
    }
}
```

## Exemplo

- Estratégia 1
  - Simulação:
    - a variável de controlo do ciclo (**k**) inicia-se com o valor 1 e, em cada iteração do ciclo, é incrementada em 1 unidade, até 30 (inclusive)
    - ou seja, o bloco de duas instruções é repetido “30 vezes”
  - Pergunta:
    - Qual o valor final da variável de controlo do ciclo (**k**) no final do programa?

## Exemplo

- Estratégia 1
  - Quantas vezes o bloco de instruções do ciclo é executado no programa que se segue (número de iterações do ciclo)?

```
#include <stdio.h>
void main ()
{
    int k;
    k = 1;
    while (k < 30)
    {
        printf("Bom dia\n");
        k = k + 1;
    }
}
```

## Exemplo

- Estratégia 1
  - Quantas vezes o bloco de instruções do ciclo é executado no programa que se segue (número de iterações do ciclo)?

```
#include <stdio.h>
void main ()
{
    int k;
    k = 0;
    while (k < 30)
    {
        printf("Bom dia\n");
        k = k + 1;
    }
}
```

## Exemplo

- Estratégia 1
  - Outras variantes que seguem esta estratégia.
    - a variável de controlo iniciar-se com o valor 10
    - a variável de controlo iniciar-se com o valor -1
  - Para cada caso, como seria a condição do ciclo?

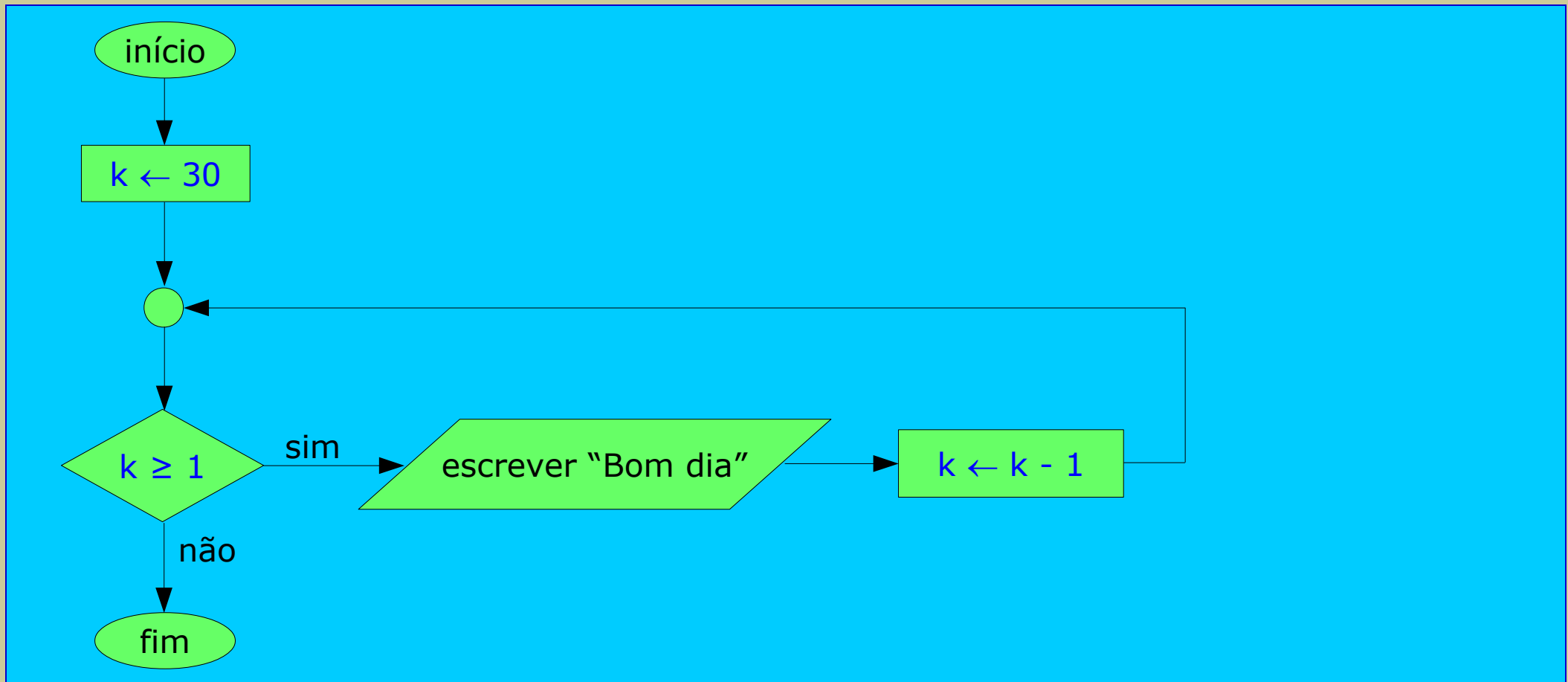
## Exemplo

- Como resolver?
- Estratégia 2
  - atribuir à variável de controlo o valor 30
  - em cada iteração do ciclo, decrementar a variável de controlo em uma unidade
  - terminar o ciclo quando a variável de controlo ficar abaixo do valor 1 (chegar a 0)

## Exemplo

- Estratégia 2

Algoritmo em fluxograma



## Exemplo

- Estratégia 2

Algoritmo em pseudocódigo

```
algoritmo mensagem30  
  k ← 30  
  enquanto (k ≥ 1) fazer  
    escrever: “Bom dia”  
    k ← k - 1  
  fim_enquanto  
fim_algoritmo
```



## Exemplo

- Estratégia 2

Programa em linguagem C

```
#include <stdio.h>
void main ()
{
    int k;
    k = 30;
    while (k >= 1)
    {
        printf("Bom dia\n");
        k = k - 1;
    }
}
```

## Exemplo

- Estratégia 2
  - Simulação:
    - a variável de controlo do ciclo (**k**) inicia-se com o valor 30 e, em cada iteração do ciclo, é decrementado em 1 unidade, até 1 (inclusive)
    - ou seja, o bloco de instruções é repetido “30 vezes”
  - Pergunta:
    - Qual o valor final da variável de controlo do ciclo (**k**) no final do programa?

## Exemplo

- Estratégia 2
  - Quantas vezes o bloco de instruções do ciclo é executado no programa que se segue (número de iterações do ciclo)?

```
#include <stdio.h>
void main ()
{
    int k;
    k = 30;
    while (k > 1)
    {
        printf("Bom dia\n");
        k = k - 1;
    }
}
```

## Exemplo

- Estratégia 2
  - Quantas vezes o bloco de instruções do ciclo é executado no programa que se segue (número de iterações do ciclo)?

```
#include <stdio.h>
void main ()
{
    int k;
    k = 30;
    while (k > 0)
    {
        printf("Bom dia\n");
        k = k - 1;
    }
}
```

## Exemplo

- Estratégia 2
  - Outras variantes que seguem esta estratégia:
    - a variável de controlo a iniciar-se com o valor 29
    - a variável de controlo a iniciar-se com o valor 50
  - Para cada caso, como seria a condição do ciclo?

## Exercício 1

- Enunciado:

- Construir um programa para escrever no monitor os números inteiros de 1 a 50, um em cada linha.

- Simulação:

1

2

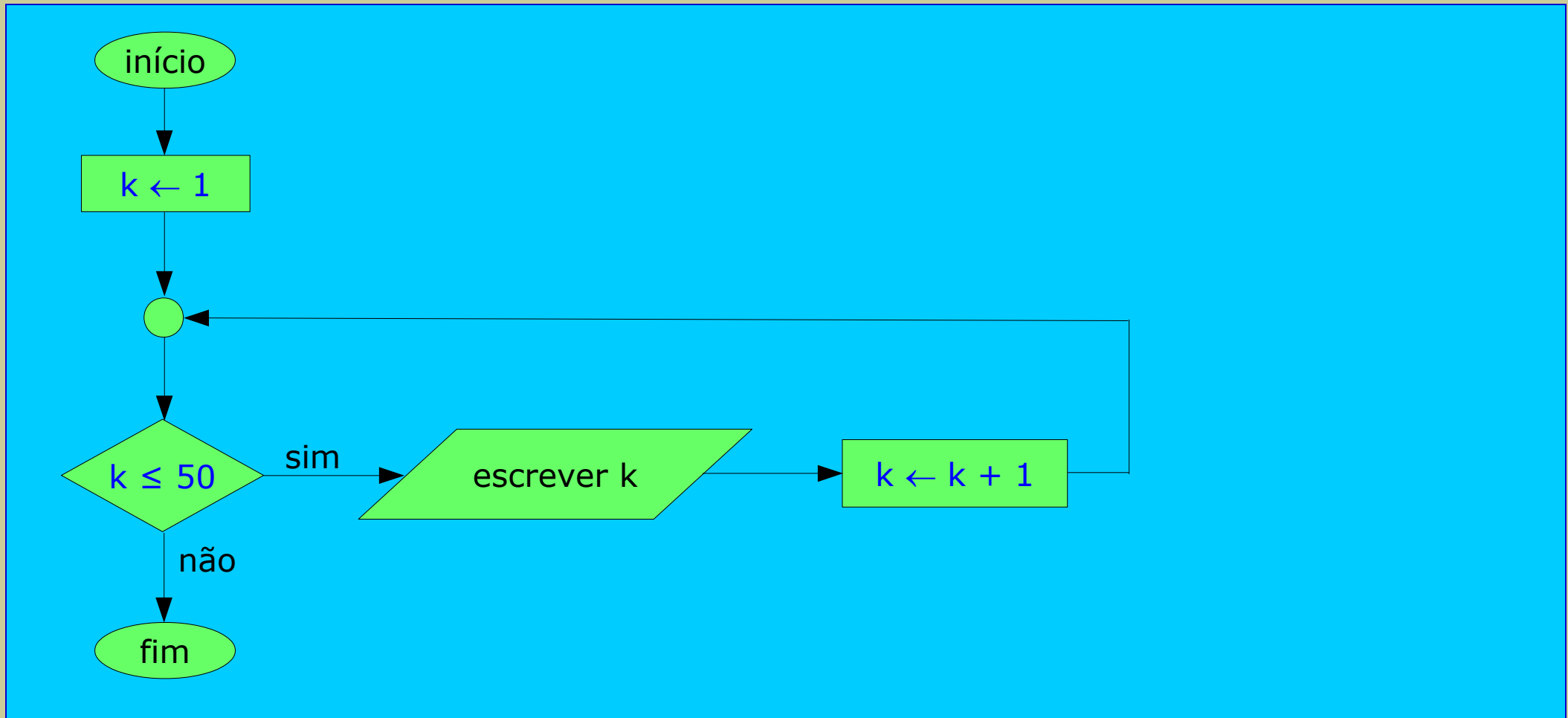
...

49

50

**Exercício 1**

Algoritmo em fluxograma



## Exercício 1

Algoritmo em pseudocódigo

```
algoritmo numeros_1a50  
  k ← 1  
  enquanto (k ≤ 50) fazer  
    escrever: k  
    k ← k + 1  
  fim_enquanto  
fim_algoritmo
```



## Exercício 1

Programa em linguagem C

```
#include <stdio.h>
void main ()
{
    int k;
    k = 1;
    while (k <= 50)
    {
        printf(“%d\n”, k);
        k = k + 1;
    }
}
```

## Exercício 2

- Enunciado:

- Construir um programa para escrever no monitor os números inteiros de -10 a 20, um em cada linha.

- Simulação:

-10

-9

...

0

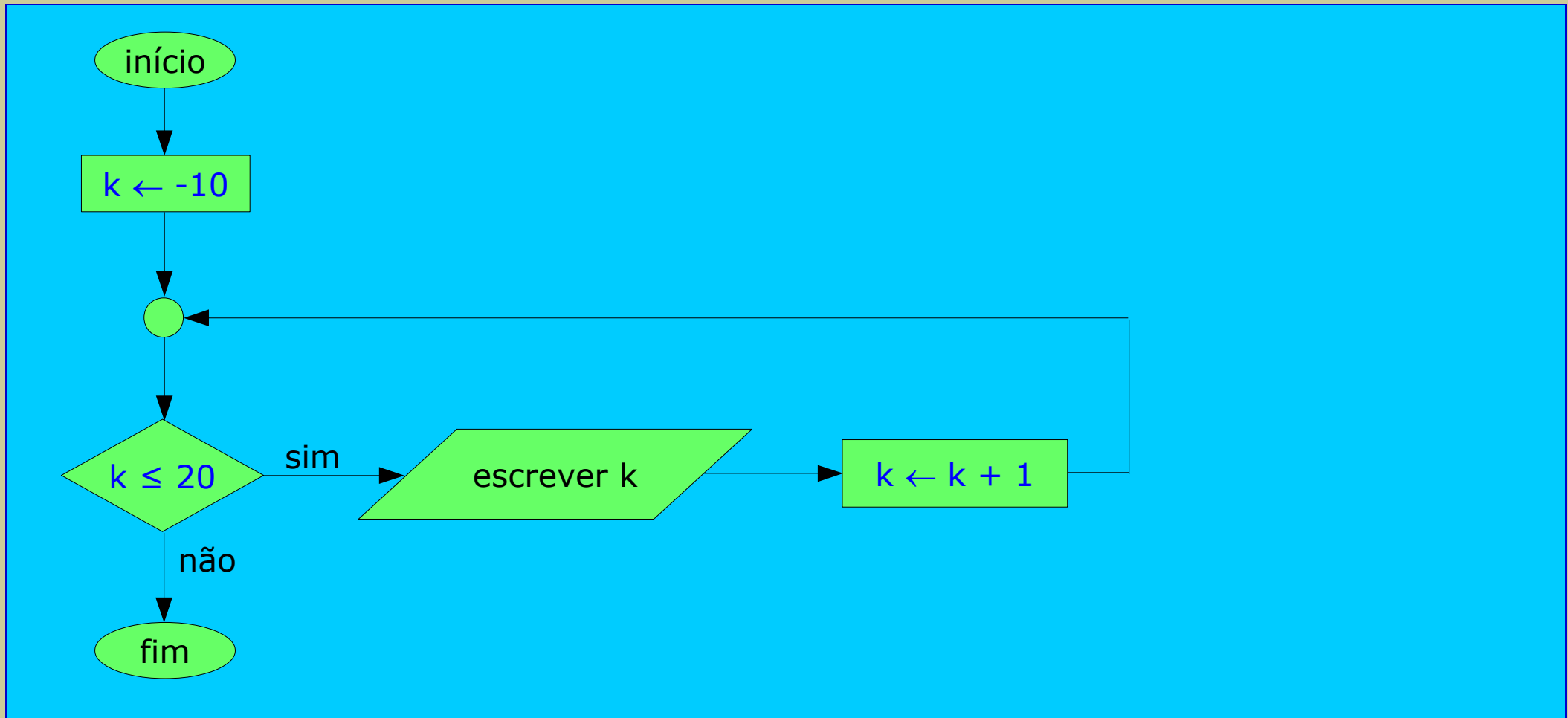
...

19

20

## Exercício 2

Algoritmo em fluxograma



## Exercício 2

Algoritmo em pseudocódigo

```
algoritmo numeros_-10a20  
  k ← -10  
  enquanto (k ≤ 20) fazer  
    escrever: k  
    k ← k + 1  
  fim_enquanto  
fim_algoritmo
```

## Exercício 2

Programa em linguagem C

```
#include <stdio.h>
void main ()
{
    int k;
    k = -10;
    while (k <= 20)
    {
        printf("%d\n", k);
        k = k + 1;
    }
}
```

## Exercício 3

- Enunciado:

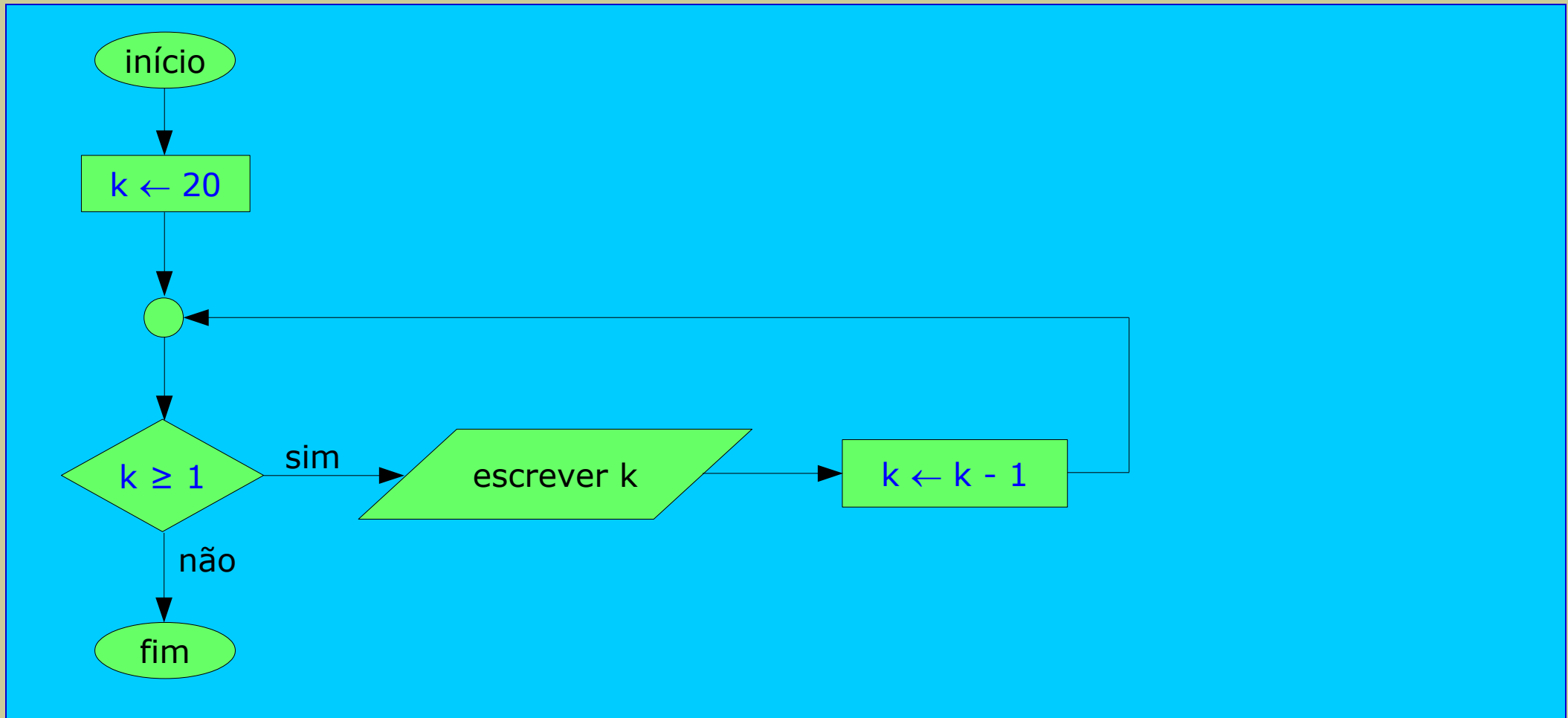
- Construir um programa para escrever no monitor os números inteiros de 20 a 1 (ordem decrescente), todos na mesma linha e separados por um espaço em branco.

- Simulação:

20 19 ... 2 1

**Exercício 3**

Algoritmo em fluxograma



## Exercício 3

Algoritmo em pseudocódigo

```
algoritmo numeros_20a1  
  k ← 20  
  enquanto (k ≥ 1) fazer  
    escrever: k  
    k ← k - 1  
  fim_enquanto  
fim_algoritmo
```



## Exercício 3

Programa em linguagem C

```
#include <stdio.h>
void main ()
{
    int k;
    k = 20;
    while (k >= 1)
    {
        printf("%d ", k);
        k = k - 1;
    }
}
```

## Exercício 4

- Enunciado:

- Construir um programa para escrever no monitor os números inteiros pares de 1 a 100, um em cada linha.

- Simulação:

2

4

...

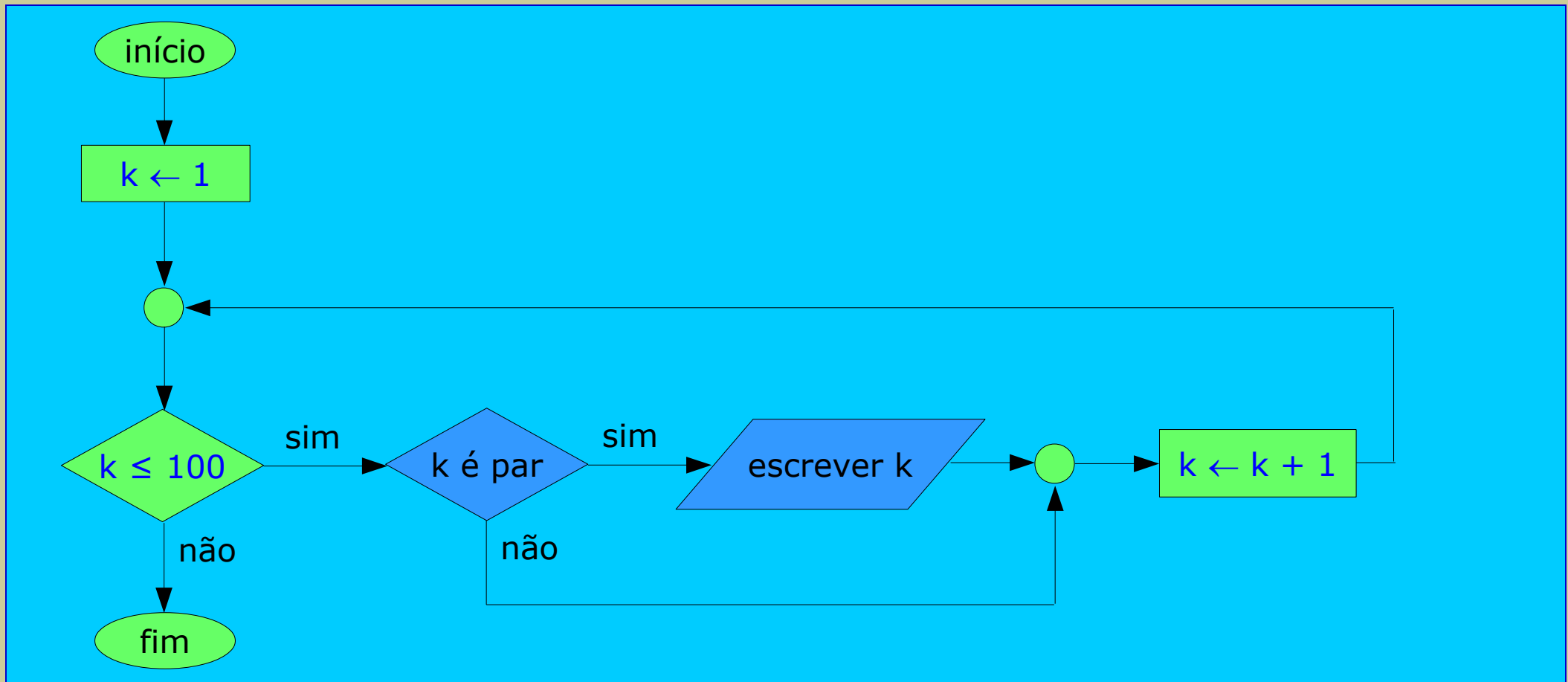
98

100

**Exercício 4**

- Estratégia 1

Algoritmo em fluxograma



## Exercício 4

- Estratégia 1

Algoritmo em fluxograma

- Note-se a existência de duas condições:
  - a **primeira** ( $k \leq 100$ ) é a condição de um **ciclo**
  - a **segunda** (k é par) é a condição de uma **instrução condicional**
- Note-se que a **instrução condicional** faz parte da instrução do **ciclo**

## Exercício 4

- Estratégia 1

Algoritmo em pseudocódigo

```
algoritmo numerosPares_1a100
   $k \leftarrow 1$ 
  enquanto ( $k \leq 100$ ) fazer
    se (k é par) então
      escrever: k
    fim_se
     $k \leftarrow k + 1$ 
  fim_enquanto
fim_algoritmo
```

## Exercício 4

- Estratégia 1

Programa em linguagem C

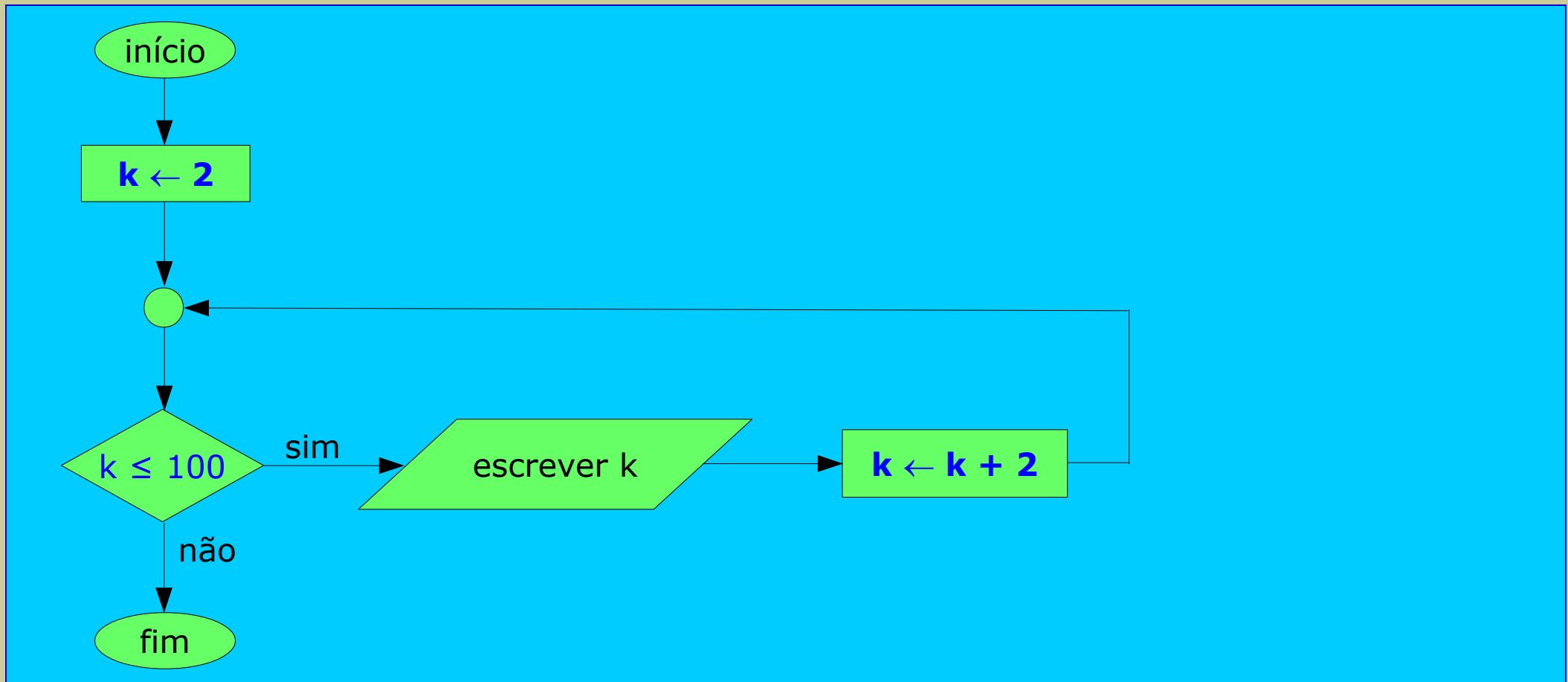
```
#include <stdio.h>
void main ()
{
    int k;
    k = 1;
    while (k <= 100)
    {
        if (k % 2 == 0)
            printf("%d\n", k);
        k = k + 1;
    }
}
```

- Questão: Quantas iterações tem o ciclo?

## Exercício 4

- Estratégia 2

Algoritmo em fluxograma



## Exercício 4

- Estratégia 2:

Algoritmo em pseudocódigo

```
algoritmo numerosPares_1a100
  k ← 2
  enquanto (k ≤ 100) fazer
    escrever: k
    k ← k + 2
  fim_enquanto
fim_algoritmo
```



## Exercício 4

- Estratégia 2

Programa em linguagem C

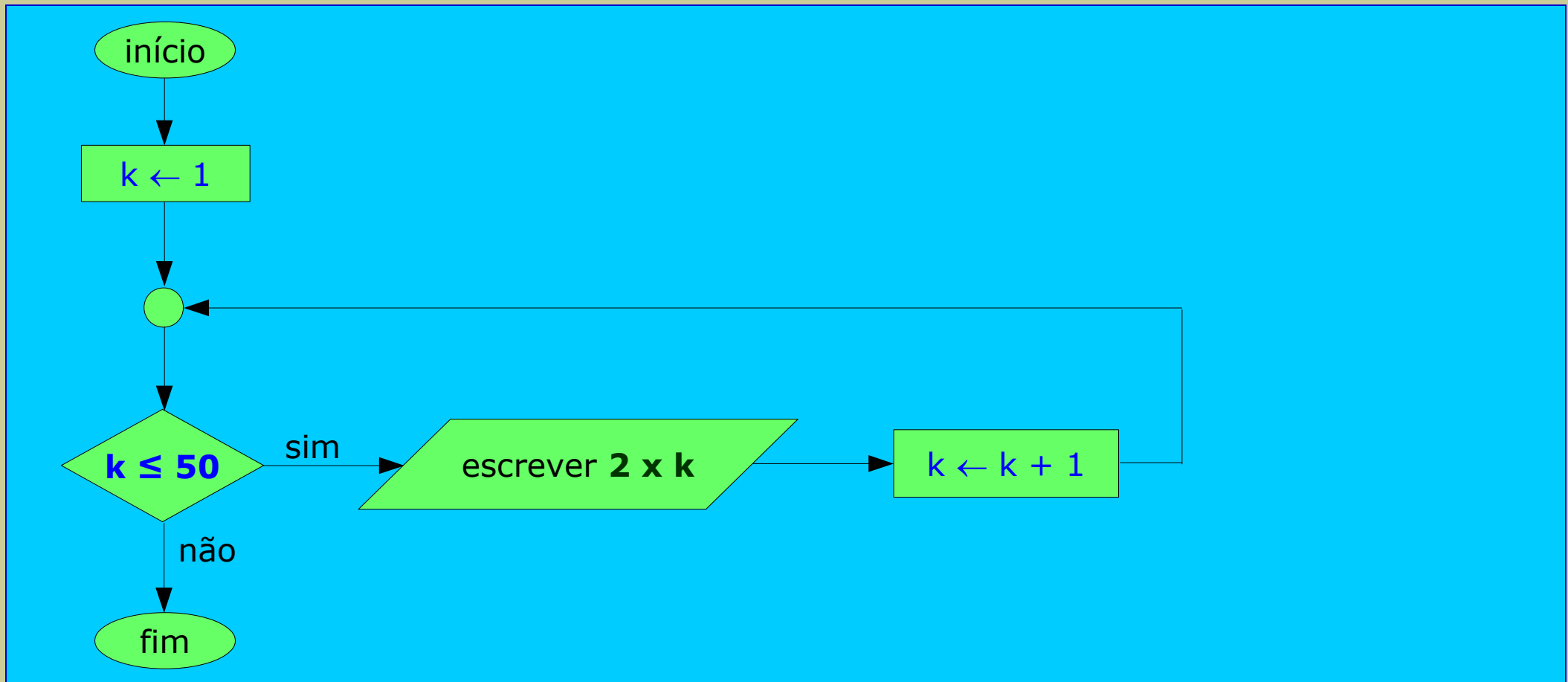
```
#include <stdio.h>
void main ()
{
    int k;
    k = 2;
    while (k <= 100)
    {
        printf("%d\n", k);
        k = k + 2;
    }
}
```

- Questão: Quantas iterações tem o ciclo?

## Exercício 4

- Estratégia 3

Algoritmo em fluxograma



## Exercício 4

- Estratégia 3:

Algoritmo em pseudocódigo

```
algoritmo numerosPares_1a100
   $k \leftarrow 1$ 
  enquanto ( $k \leq 50$ ) fazer
    escrever:  $2 \times k$ 
     $k \leftarrow k + 1$ 
  fim_enquanto
fim_algoritmo
```

## Exercício 4

- Estratégia 3

Programa em linguagem C

```
#include <stdio.h>
void main ()
{
    int k;
    k = 1;
    while (k <= 50)
    {
        printf("%d\n", 2*k);
        k = k + 1;
    }
}
```

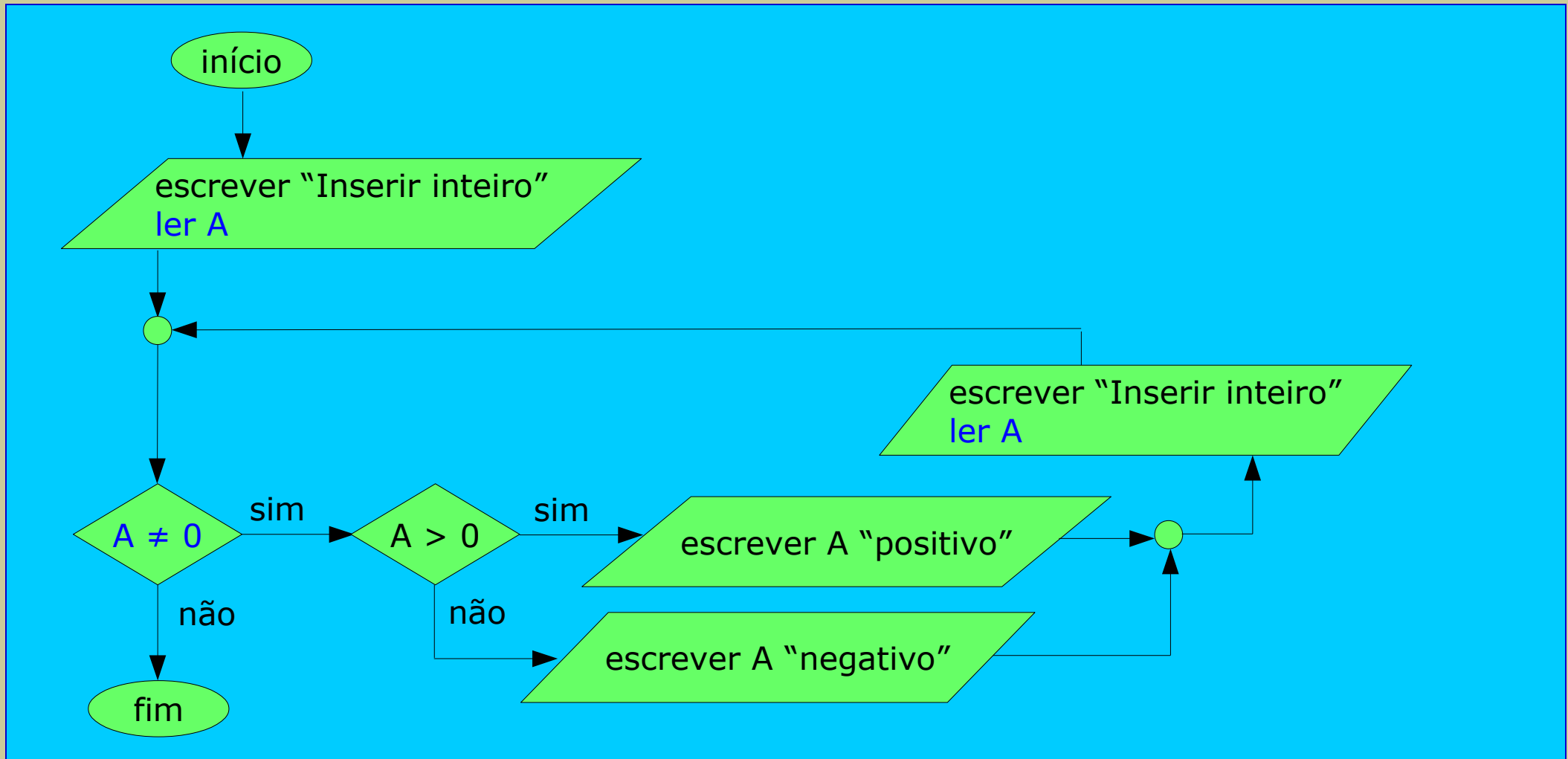
- Questão: Quantas iterações tem o ciclo?

## Exercício 5

- E se o número de repetições for desconhecido?
- Existem muitos problemas que usam ciclos, cujo número de iterações a realizar
  - não é possível determinar previamente,
  - dependem, por exemplo, de um “aviso” do utilizador para que as repetições terminem
- Enunciado:
  - Construir um programa para inserir/ler números inteiros e, para cada número inserido/lido escrever uma mensagem no monitor a informar se é positivo ou negativo; o processo deve terminar quando o número inserido/lido for o número zero (0).

**Exercício 5**

Algoritmo em fluxograma



## Exercício 5

Algoritmo em pseudocódigo

```
algoritmo positivosNegativos
  escrever: “Inserir um número inteiro (0 para terminar):”
  ler: A
  enquanto (A ≠ 0) fazer
    se (A > 0) então
      escrever: A, “ positivo.”
    senão
      escrever: A, “ negativo.”
    fim_se
  escrever: “Inserir um número inteiro (0 para terminar):”
  ler: A
fim_enquanto
fim_algoritmo
```

## Exercício 5

Programa em linguagem C

```
void main ()
{
    int A;
    printf("Inserir um número inteiro (0 para terminar): ");
    scanf("%d", &A);
    while (A != 0)
    {
        if (A > 0)
            printf("%d positivo\n", A);
        else
            printf("%d negativo\n", A);
        printf("Inserir um número inteiro (0 para terminar): ");
        scanf("%d", &A);
    }
}
```



## Observação

- Em todos os exercícios apresentados que usaram uma instrução “while” bem definida, a variável de controlo do ciclo aparece sempre em três situações:
  - **Inicialização** (valor inicial ou primeiro valor) da variável de controlo
    - Ex: `k = 1; k = 0; k = 30; k = -10; k = 20; k = 2; scanf("%d", &A);`
  - **Condição** do ciclo (que contém a variável de controlo)
    - Ex: `(k < 30), (k >= 1), (k > 0), (k <= 50), (k <= 20), (k <= 100), (A != 0)`
  - **Atualização** da variável de controlo
    - Ex: `k = k + 1; k = k - 1; k = k + 2; scanf("%d", &A);`

## Instrução (ciclo) “do ... while”

- A **instrução (ciclo) “do ... while”** executa (“repete”) uma dada instrução (ou bloco de instruções) até uma determinada condição ser falsa (Caso 2)
  - executa a instrução (ou bloco de instruções) enquanto a condição for verdadeira
- Relembrar:
  - Caso 2:
    - primeiro executa a instrução (ou bloco de instruções),
    - depois testa a condição do ciclo para verificar se
      - continua para mais uma iteração, ou
      - termina o processo

## Sintaxe

- Na linguagem C:

```
do{  
    instrução;  
}while (condição);
```

em que,

- **instrução**
  - é uma instrução de qualquer tipo, incluindo instruções de repetição
  - é uma instrução simples ou várias instruções
- **condição**
  - é uma expressão lógica (o resultado é “verdadeiro/sim” ou “falso/não”),
  - tem **obrigatoriamente** que estar **entre parentesis**
- termina **obrigatoriamente** com ponto e vírgula (;)

## Sintaxe

- Em pseudocódigo

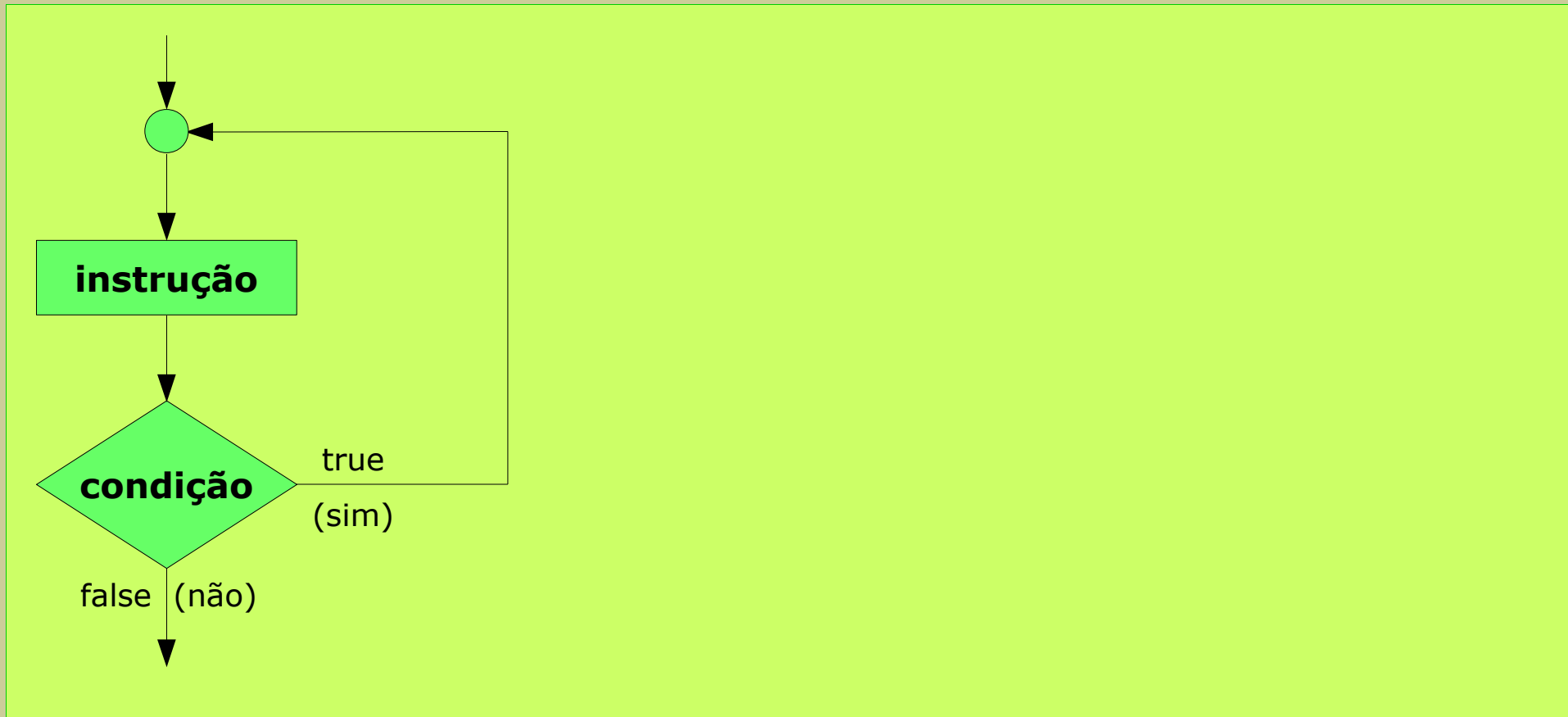
**fazer**

<comandos>

**enquanto** <condição>

## Sintaxe

- Em fluxograma



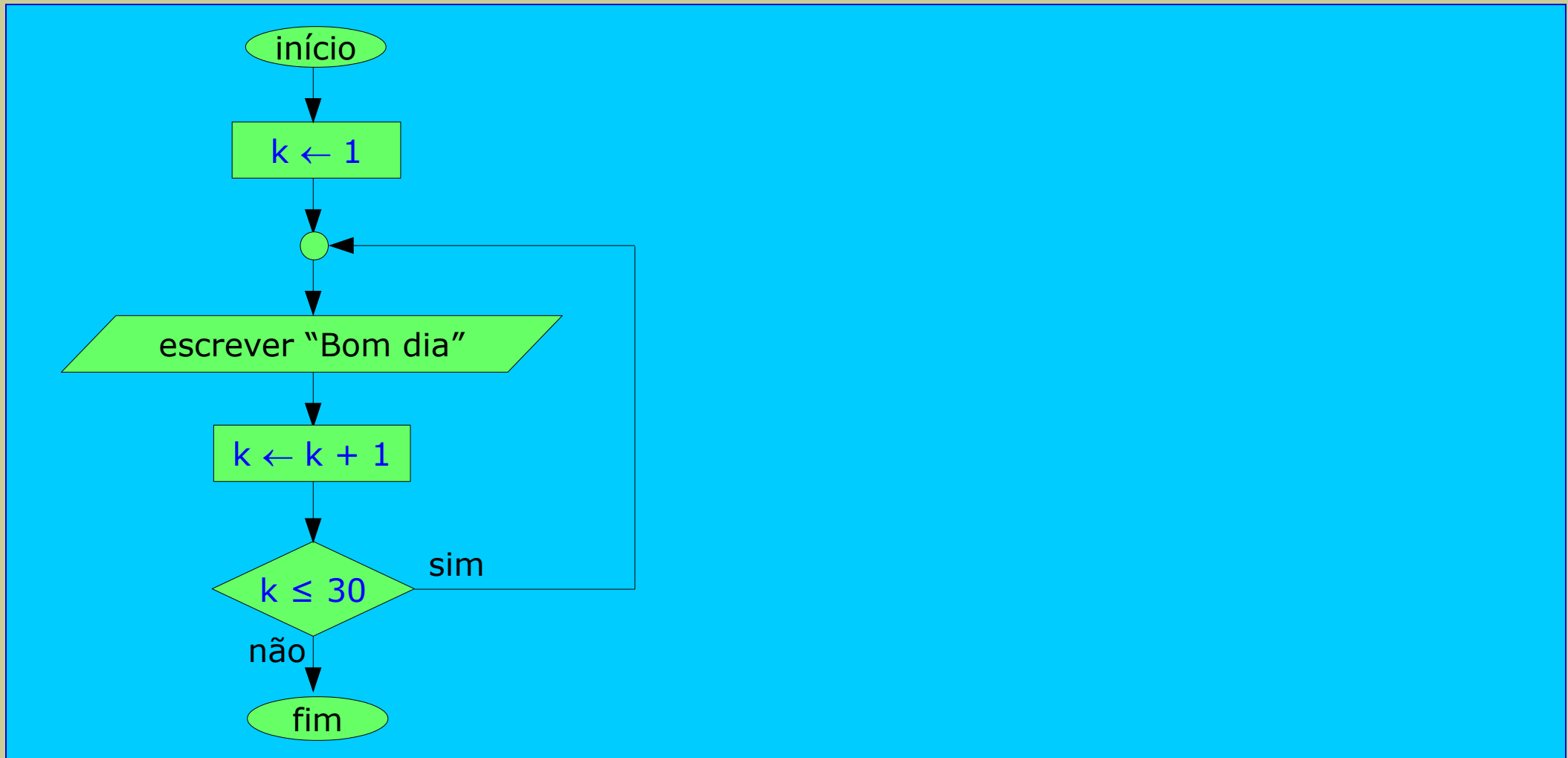
## Exemplo

- Enunciado:

Construir um programa para escrever no monitor várias vezes (repetidas) a mensagem “Bom dia”. Por exemplo, repetir 30 vezes.

## Exemplo

- Algoritmo em fluxograma



## Exemplo

- Algoritmo em pseudocódigo

```
algoritmo mensagem30
  k ← 1
  fazer
    escrever: “Bom dia”
    k ← k + 1
  enquanto (k ≤ 30)
fim_algoritmo
```



## Exemplo

- Programa em C

```
#include <stdio.h>
void main ()
{
    int k;
    k = 1;
    do{
        printf("Bom dia\n");
        k = k + 1;
    }while (k <= 30);
}
```

## Instrução (ciclo) "for"

### Definição

- A **instrução (ciclo) for** executa ("repete") uma dada instrução (ou bloco de instruções) um número de vezes previamente conhecido (Caso 1)
  - é mais apropriada para os casos em que se conhece previamente o número de iterações (por exemplo: 10 vezes ou N vezes)
- Tem a vantagem de conter no seu formato os elementos essenciais para este tipo de repetição:
  - a "inicialização" da variável de controlo (o contador),
  - o "teste de paragem" (a condição do ciclo)
  - a "atualização" do valor da variável de controlo (o contador)

## Sintaxe

- Na linguagem C:

```
for (inicialização; condição; atualização)  
    instrução;
```

em que,

- **inicialização** é uma instrução de atribuição para atribuir à variável de controlo o seu valor inicial ou primeiro valor
- **condição**
  - é uma expressão lógica (o resultado é “verdadeiro/sim” ou “falso/não”),
  - deve conter a variável de controlo
- **atualização** é uma instrução de atribuição para atualizar o valor da variável de controlo, cuja alteração deverá alguma vez permitir que o ciclo termine
- **instrução**
  - é uma instrução de qualquer tipo, incluindo instruções de repetição
  - é uma instrução simples ou um bloco de instruções

## Sintaxe

- Em pseudocódigo

inicialização

**enquanto** <condição> **fazer**

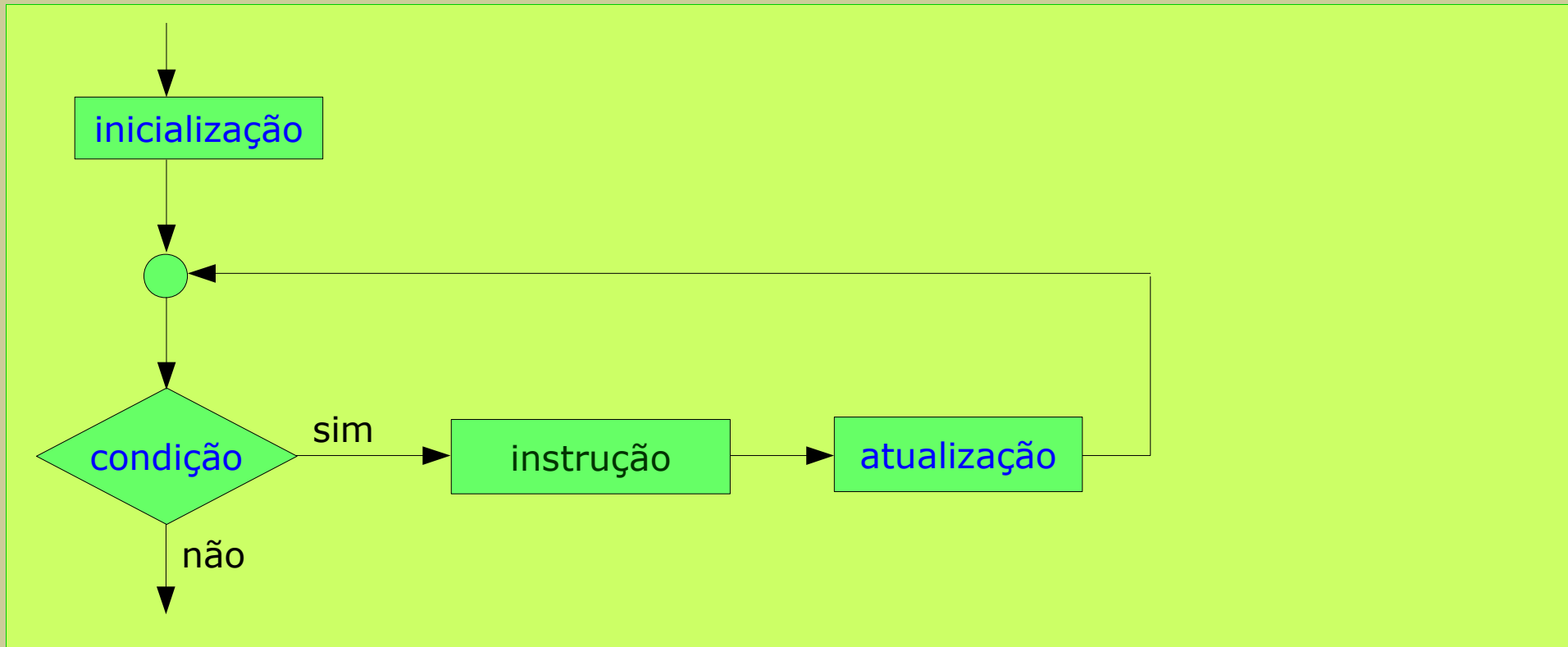
<comandos>

<atualização>

**fim\_enquanto**

## Sintaxe

- Em fluxograma



## Funcionamento

- Primeiro é feita a inicialização da variável de controlo (apenas uma vez)
- A seguir é testada a condição do ciclo que,
  - se é verdadeira, a instrução é executada
  - se é falsa, as repetições terminam
- Após a execução da instrução
  - a variável de controlo é atualizada (recebe outro valor), e
  - depois regressa ao passo anterior (a condição é testada novamente)

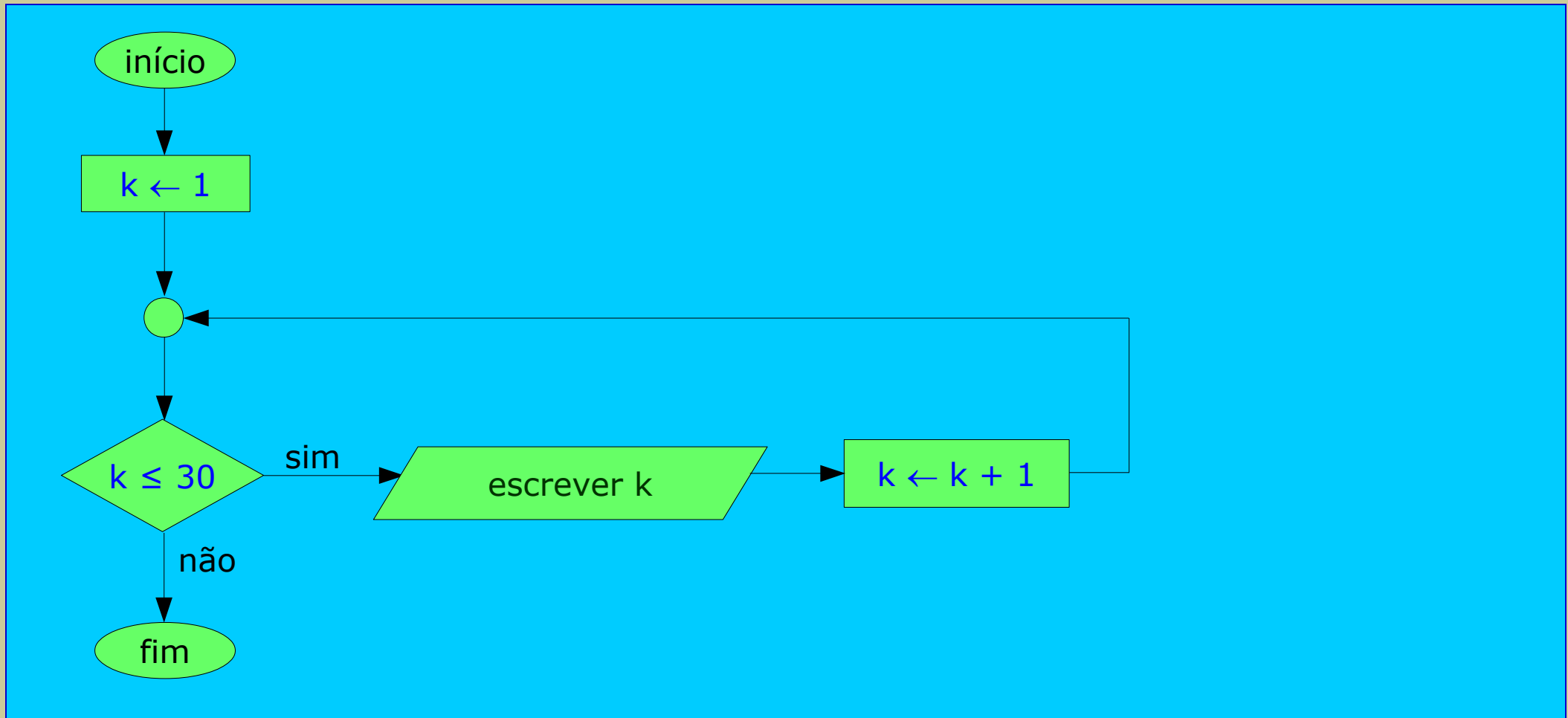
## Exemplo 1

- Enunciado

Construir um programa para escrever no monitor os números inteiros de 1 a 30 (ordem crescente)

**Exemplo 1**

Algoritmo em fluxograma





## Exemplo 1

- Programa em C

```
#include <stdio.h>
void main ()
{
    int k;
    for (k = 1; k <= 30; k = k + 1)
        printf("%d\n", k);
}
```

- Simular programa

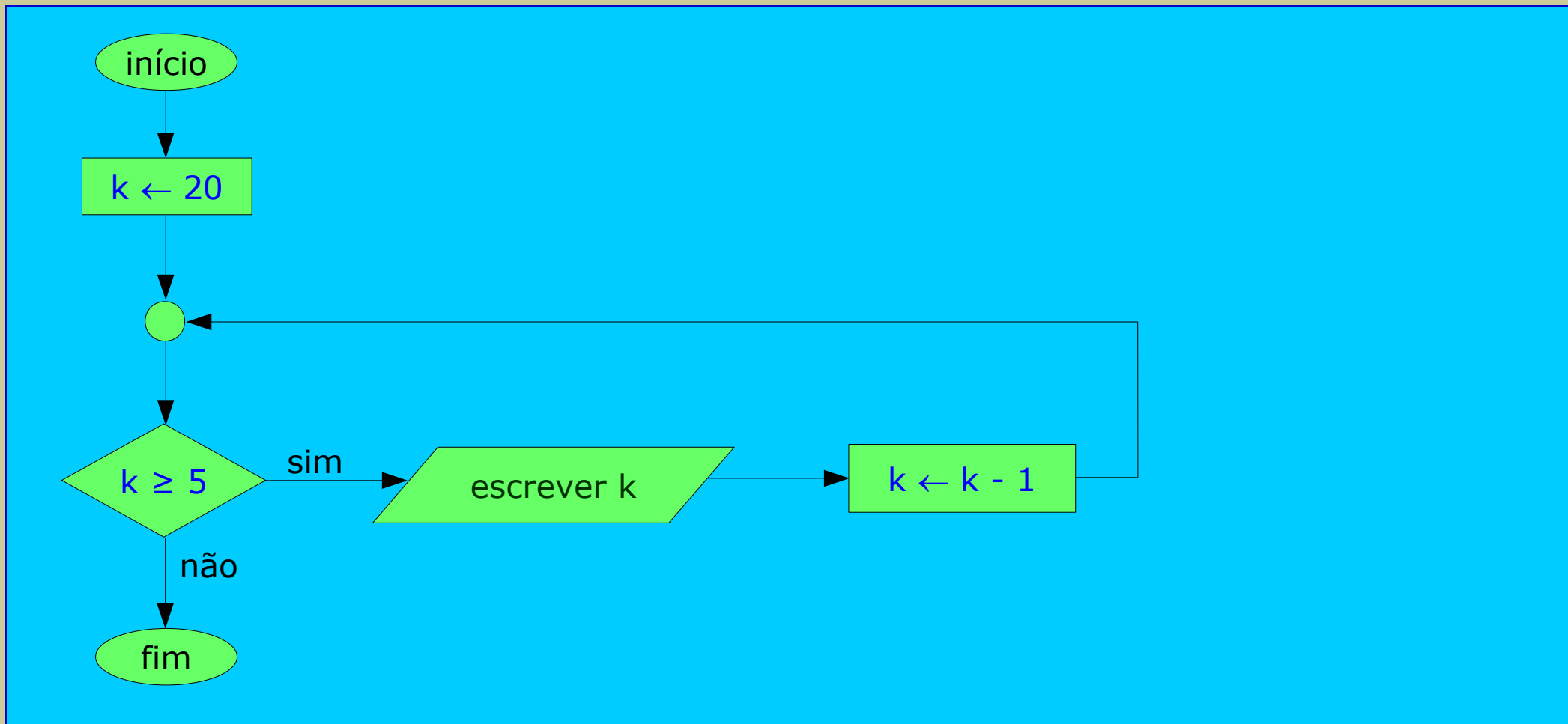
## Exemplo 2

- Enunciado

Construir um programa para escrever no monitor os números inteiros de 20 a 5 (ordem decrescente)

## Exemplo 2

Algoritmo em fluxograma



## Exemplo 2

- Programa em C

```
#include <stdio.h>
void main ()
{
    int k;
    for (k = 20; k >= 5; k = k - 1)
        printf("%d\n", k);
}
```

- Simular programa

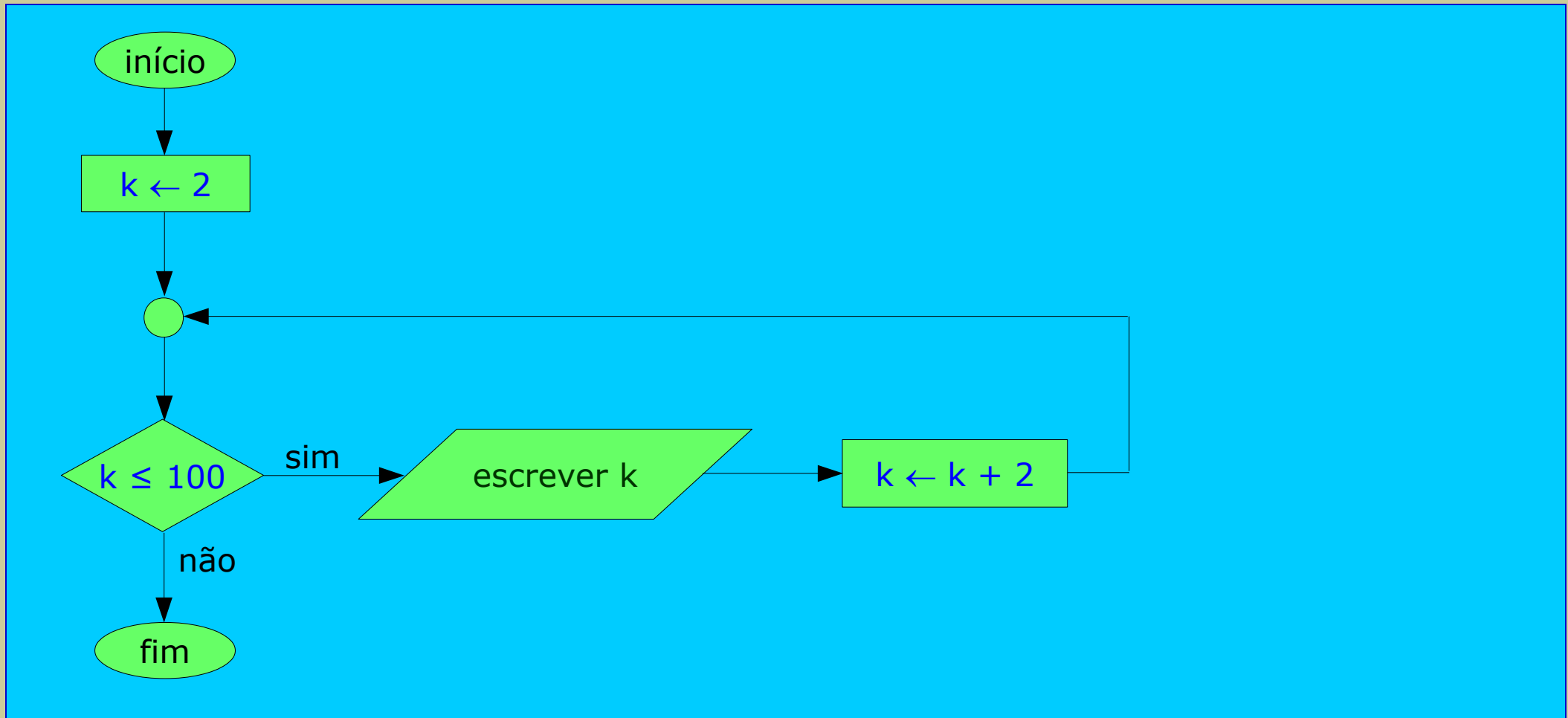
## Exemplo 3

- Enunciado

Construir um programa para escrever no monitor os números inteiros pares de 1 a 100

**Exemplo 3**

Algoritmo em fluxograma



## Exemplo 3

- Programa em C

```
#include <stdio.h>
void main ()
{
    int k;
    for (k = 2; k <= 100; k = k + 2)
        printf("%d\n", k);
}
```

- Simular programa

## Exemplo 4

### - Enunciado

Construir um programa para

- ler 20 números inteiros e
- calcular a soma e a média entre eles

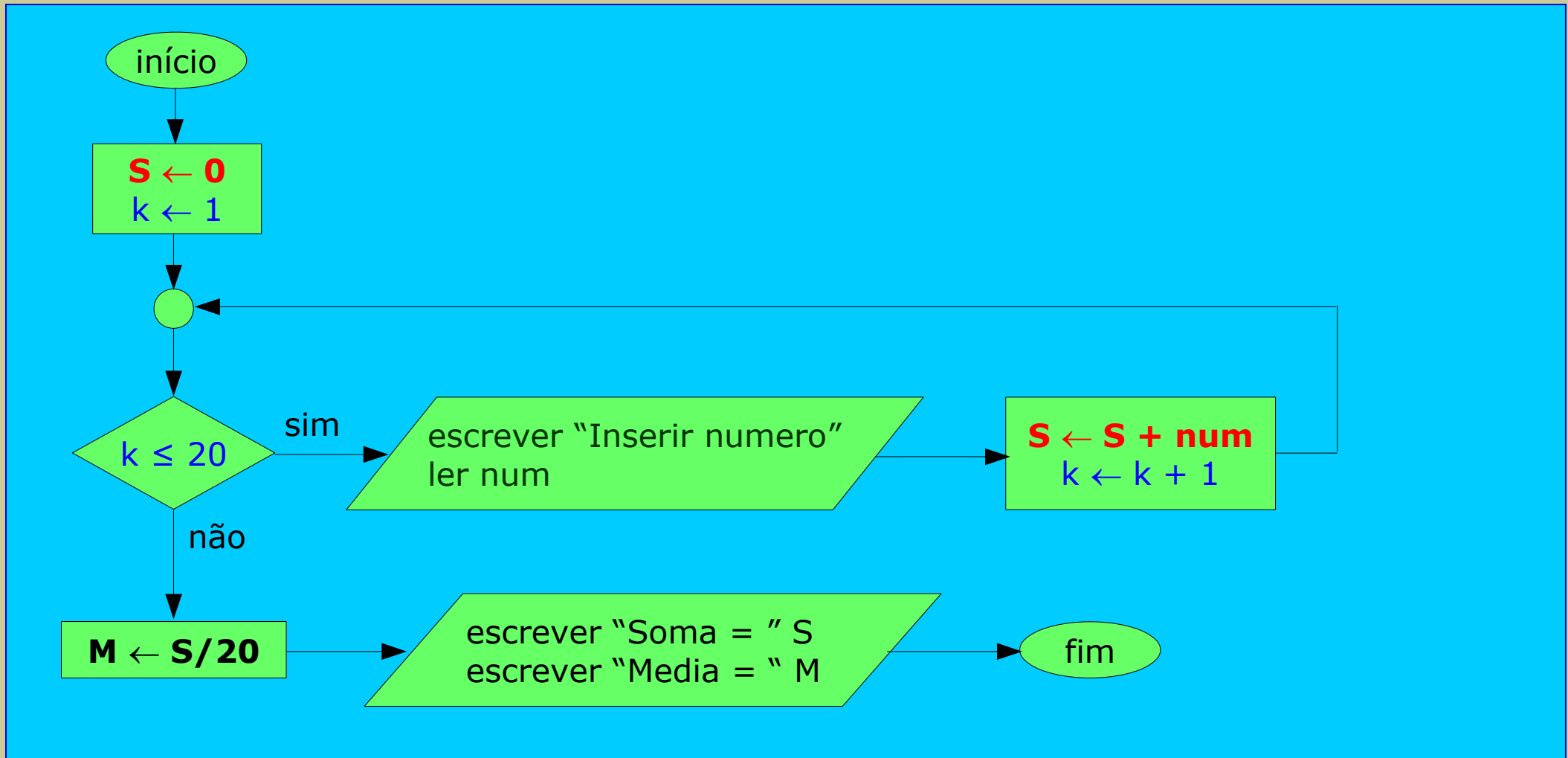
### - Estratégia

- ler um número de cada vez e determinando a soma gradualmente, acumulando aquele número à soma determinada até ao momento



## Exemplo 4

Algoritmo em fluxograma



## Exemplo 4

- Programa em C

```
#include <stdio.h>
void main () {
    int k, num, S;
    float M;
    S = 0;
    for (k = 1; k <= 20; k = k + 1)
    {
        printf("Insira um número inteiro: ");
        scanf("%d", &num);
        S = S + num;
    }
    M = S/20;
    printf("Soma = %d\n", S);
    printf("Media = %f\n", M);
}
```

## Questão

- O cálculo da média está bem definida?

$$M = S/20;$$

- Imagine-se que  $S = 710$ .
  - Qual o valor de  $M$ ?

## Conversão entre tipos de dados (type casting)

- O operador (**cast**) permite fazer a conversão entre tipos de dados
- A sintaxe é a seguinte:
  - de inteiro para real
    - (float) inteiro ==> real com o mesmo valor do inteiro
    - exemplo: (float) 65 = 65.0
  - de inteiro para caracter
    - (char) inteiro ==> caráter cujo código ASCII é o valor de inteiro
    - exemplo: (char) 65 = 'A' (caráter cujo código ASCII é 65)
  - de real para inteiro
    - (int) real ==> parte inteiro do valor representado por real
    - exemplo: (int) 65.76 = 65
  - de caráter para inteiro
    - (int) caráter ==> código ASCII de caráter
    - exemplo: (int) 'A' = 65 (código ASCII do caráter 'A')

## Exemplo 4

- Programa em C

```
#include <stdio.h>
void main () {
    int k, num, S;
    float M;
    S = 0;
    for (k = 1; k <= 20; k = k + 1)
    {
        printf("Insira um número inteiro: ");
        scanf("%d", &num);
        S = S + num;
    }
    M = (float) S / 20;
    printf("Soma = %d\n", S);
    printf("Media = %f\n", M);
}
```

## Questão

- E se a expressão da média fosse a que se segue? Estaria correta também?

```
M = (float) (S/20);
```

- Imagine-se que  $S = 710$ .
  - Qual o valor de  $M$ ?
- Outra solução para este caso

```
M = S / 20.0;
```

## Exercícios

- Usar as outras instruções de repetição para resolver o mesmo problema:
  - instrução “while”
  - instrução “do ... while”

## Repetições dentro de repetições

### Definição

- Alguns problemas mais complexos, podem exigir a utilização de repetições dentro de outras repetições
- O funcionamento mais comum é quando para cada iteração de uma instrução de repetição, for necessário realizar um conjunto de repetições



## Exemplo 1

### - Enunciado

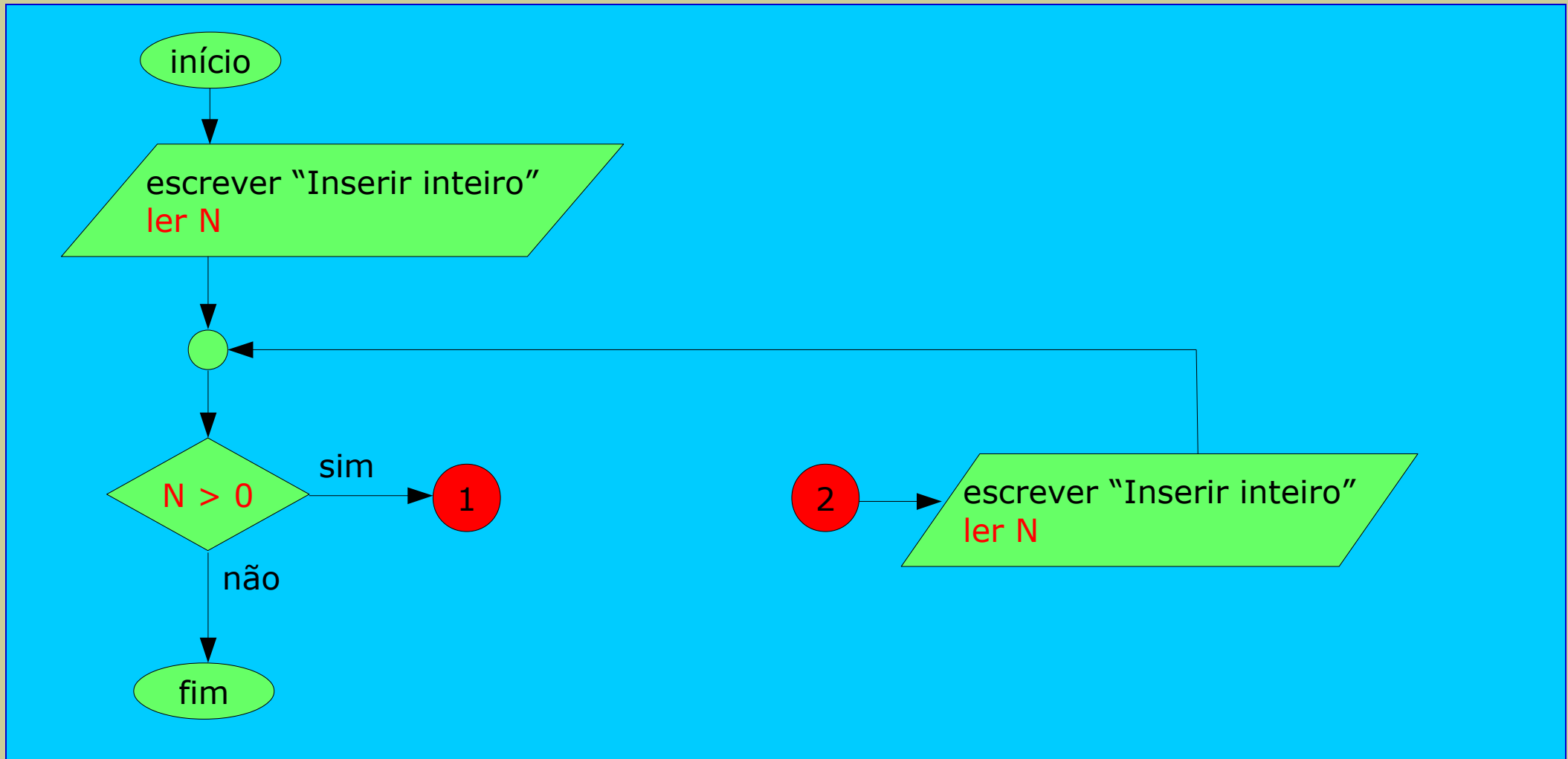
- inserir (ler) números inteiros a partir do teclado e para cada número positivo  $N$  inserido calcular a soma de 1 até  $N$ ; por exemplo: para  $N = 4$ , a soma é 10 ( $= 1 + 2 + 3 + 4$ )
- terminar o cálculo de somas quando for inserido um número não positivo (ou seja,  $\leq 0$ )

## Exemplo 1

- Estratégia para a resolução
  - uma primeira instrução de repetição (ciclo) para
    - tratar os números inteiros inseridos
    - termina quando for inserido um número não positivo ( $\leq 0$ )
  - uma segunda instrução de repetição (ciclo) para
    - calcular a soma de 1 até ao número inserido
  - para cada iteração da primeira instrução de repetição, aplicar a segunda instrução de repetição
- Qual a instrução de repetição (das 3 apresentadas) mais adequada
  - para a primeira instrução de repetição
  - para a segunda instrução de atribuição

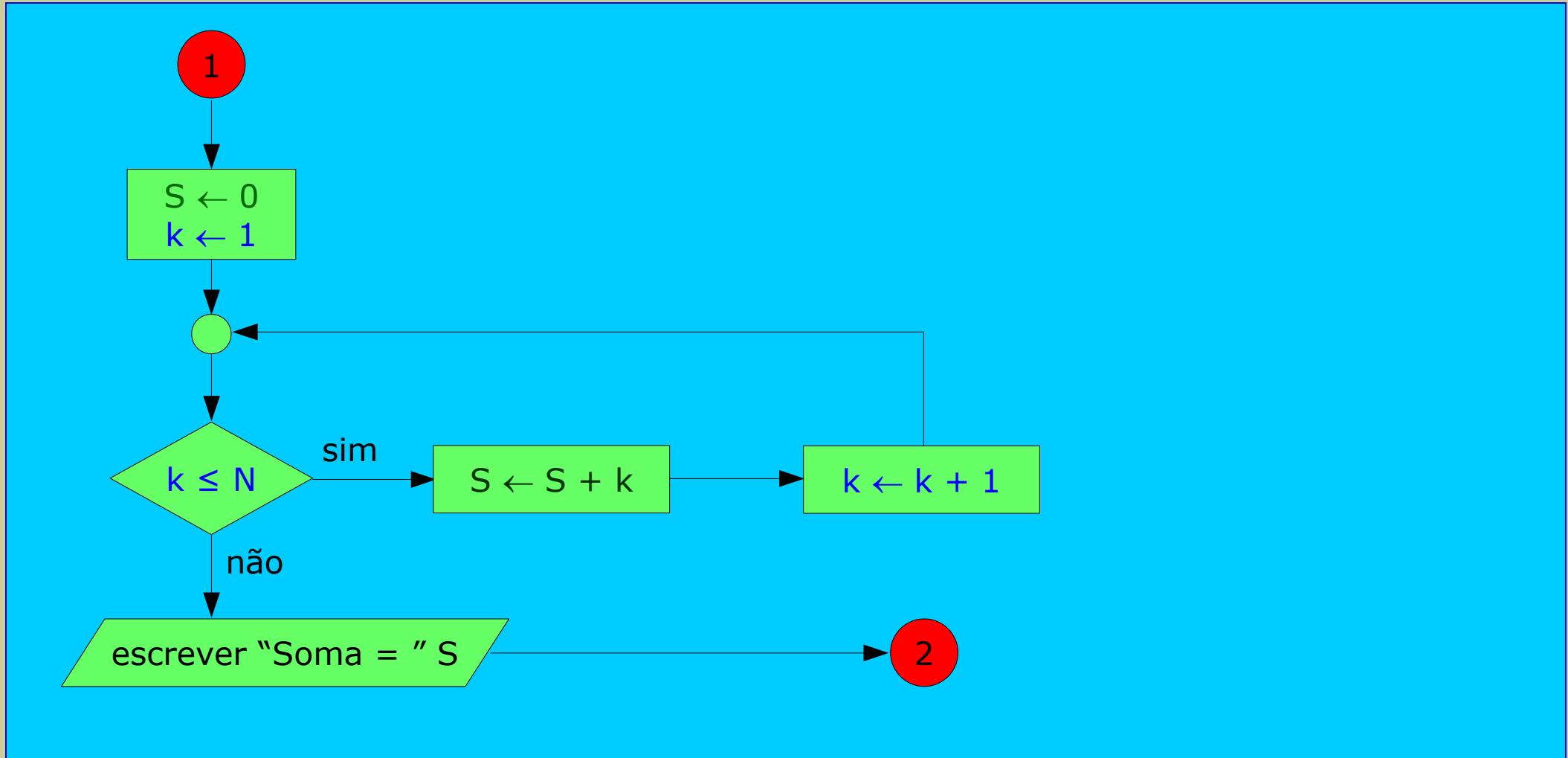
**Exemplo 1**

- Algoritmo em fluxograma



**Exemplo 1**

- Algoritmo em fluxograma (cont)



## Exemplo 1

- Algoritmo em pseudocódigo

```
algoritmo somas_1aN
  escrever: "Inserir um inteiro (<= 0 para terminar)"
  ler: N
  enquanto (N > 0) fazer
    S ← 0
    k ← 1
    enquanto (k <= N) fazer
      S ← S + k
      k = k + 1
    fim_enquanto
  escrever: "Soma = ", S
  escrever: "Inserir um inteiro (<= 0 para terminar)"
  ler: N
fim_enquanto
fim_algoritmo
```

## Exemplo 1

- Programa em C

```
#include <stdio.h>
void main () {
    int k, N, S;
    printf("Insira um número inteiro (<= 0 para terminar): ");
    scanf("%d", &N);
    while (N > 0)
    {
        S = 0;
        for (k = 1; k <= N; k = k + 1)
            S = S + k;
        printf("Soma de 1 a %d = %d\n", N, S);
        printf("Insira um número inteiro (<= 0 para terminar): ");
        scanf("%d", &N);
    }
}
```

## Observações

- A "instrução" do **while** é um bloco de instruções
- Podia-se utilizar um **while** em vez do **for** (isto é sempre possível), mas o **for** é mais simples
- Na primeira instrução de repetição ("while") podia-se usar uma instrução
  - "do ... while"?
  - "for"?
- Na segunda instrução de repetição ("for") podia-se usar uma instrução
  - "do ... while"?

## Exemplo 2

### - Enunciado

- inserir (ler) números inteiros do teclado e para cada número positivo N inserido, calcular o fatorial de N; por exemplo: para  $N = 4$ ,  $\text{fatorial}(4) = 24$  ( $4 \times 3 \times 2 \times 1$ )
- terminar o cálculo de fatoriais quando for inserido (lido) um número não positivo ( $\leq 0$ )

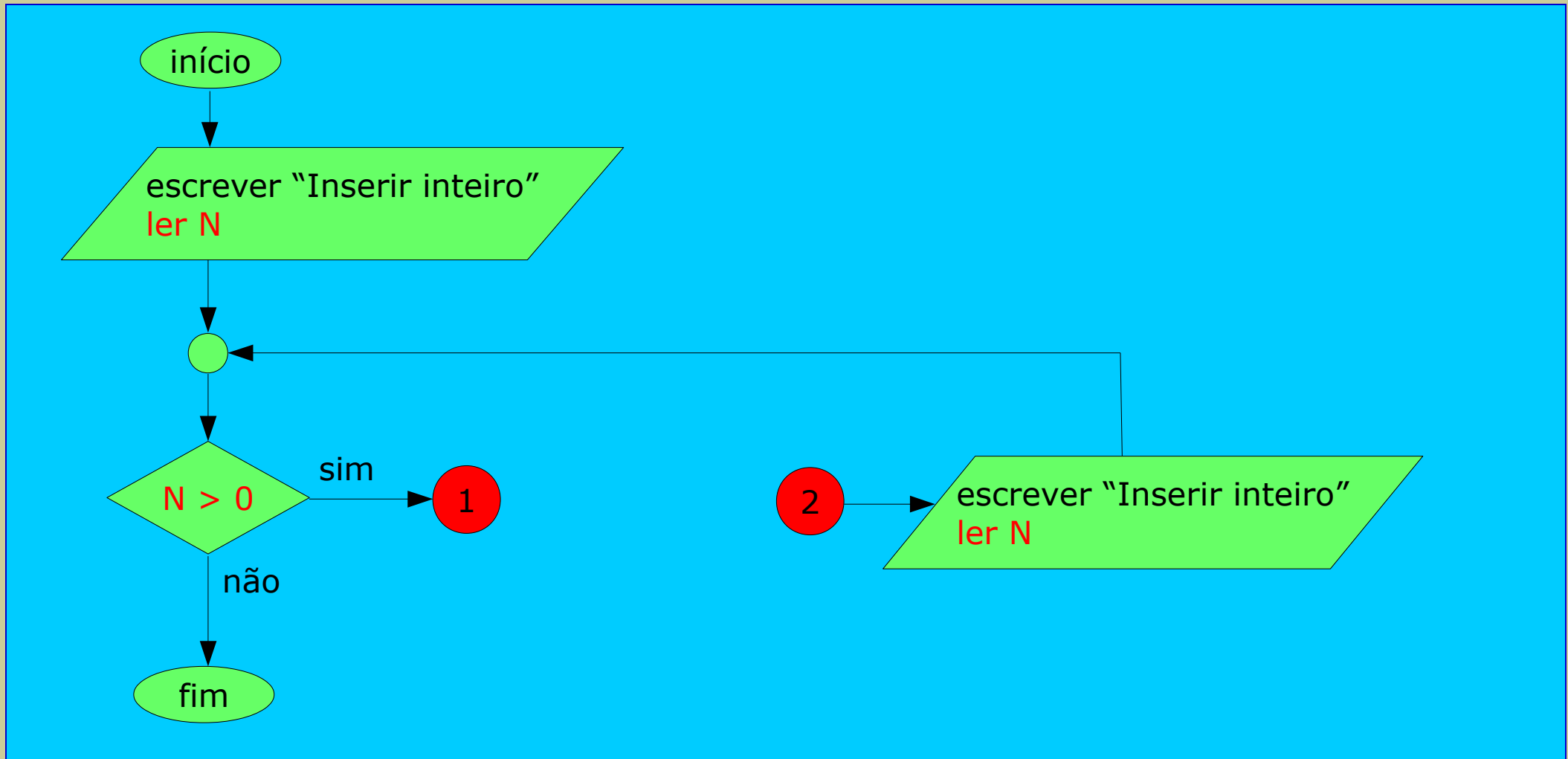


## Exemplo 2

- Estratégia para a resolução
  - uma primeira instrução de repetição para
    - tratar os números inteiros inseridos
    - termina quando for inserido um número não positivo ( $\leq 0$ )
  - uma segunda instrução de repetição para
    - calcular o fatorial do número inserido
  - para cada iteração da primeira instrução de repetição, aplicar a segunda instrução de repetição
- Qual a instrução de repetição (das 3 apresentadas) mais adequada
  - para a primeira instrução de repetição
  - para a segunda instrução de atribuição

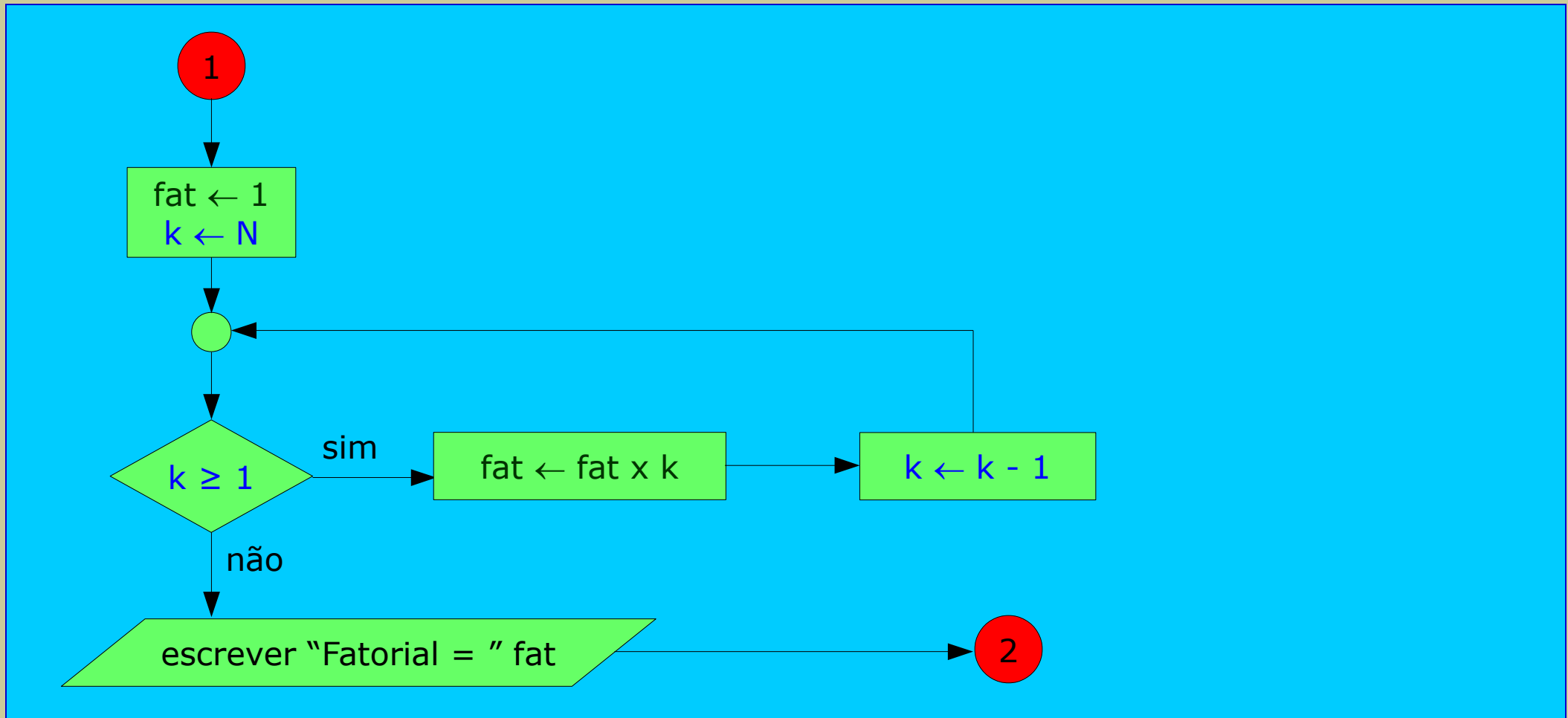
## Exemplo 2

- Algoritmo em fluxograma



## Exemplo 2

- Algoritmo em fluxograma (cont)



## Exemplo 2

- Algoritmo em pseudocódigo

```
algoritmo calcularFatoriais
  escrever: "Inserir um inteiro (<= 0 para terminar)"
  ler: N
  enquanto (N > 0) fazer
    fat ← 1
    k ← N
    enquanto (k >= 1) fazer
      fat ← fat x k
      k = k - 1
    fim_enquanto
    escrever: "Fatorial = ", fat
    escrever: "Inserir um inteiro (<= 0 para terminar)"
  ler: N
fim_enquanto
fim_algoritmo
```

## Exemplo 2

- Programa em C

```
#include <stdio.h>
void main ()
{
    int k, N, fat;
    printf("Insira um número inteiro (<= 0 para terminar): ");
    scanf("%d", &N);
    while (N > 0) {
        fat = 1;
        for (k = N; k >= 1; k = k - 1)
            fat = fat * k;
        printf("Fatorial de %d = %d\n", N, fat);
        printf("Insira um número inteiro (<= 0 para terminar): ");
        scanf("%d", &N);
    }
}
```