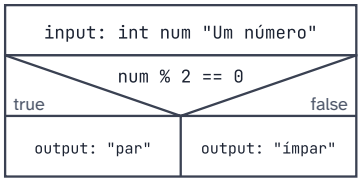
Cábula de estruturogramas · Bases da programação

Elementos base · Fluxo · Subprogramas · Ajuda

Holger Sebastião · www.sebastiao.org
Versão 1.1 · 2026-10-25

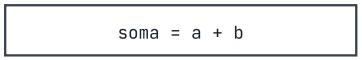
Algoritmo — exemplo



Uma sequência exata de passos para resolver uma tarefa. Cada passo é inequívoco e, ao fim de um número finito de passos, termina.

Um **estruturograma** mostra um algoritmo como uma imagem feita de caixas e lê-se de cima para baixo (Nassi-Shneiderman, DIN 66261).

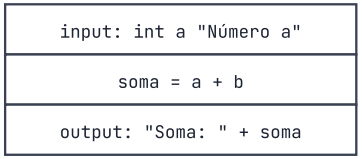
Instrução — statement



Um único passo: uma atribuição, um cálculo, uma entrada ou uma saída.

```
soma = a + b;
```

Sequência — sequence



As instruções correm uma após outra, de cima para baixo, cada uma exatamente uma vez.

`input:` lê um valor, `output:` mostra algo.

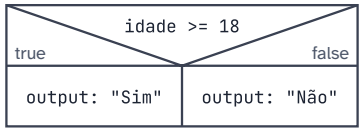
Chamada — subprograma



Inicia um subprograma, ou seja, um estruturograma próprio. Os parâmetros vão entre parênteses. Uma função devolve um valor de retorno com `return`.

```
int e = dobro(5);
```

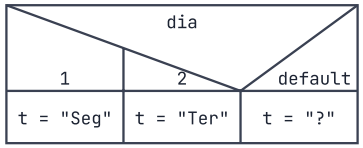
Decisão — if-else



A condição é verificada. Se for `true`, corre o ramo da esquerda; senão, o da direita. Um ramo pode ficar vazio (ø).

```
if (idade >= 18) { … }
else { … }
```

Escolha múltipla — switch



Compara o valor de uma variável com casos fixos. É executado exatamente um caso. Se nenhum corresponder, corre o `default`.

```
switch (dia) {
  case 1 -> t = "Seg";
  case 2 -> t = "Ter";
  default -> t = "?";
}
```

Marcador — elemento vazio



Aqui não acontece nada, por exemplo num ramo false vazio. No VisuStruct arrastas um elemento para cima dele e o marcador é substituído.

Regras para bons estruturogramas

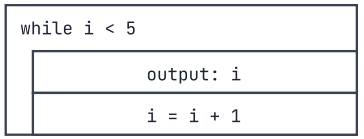
Cada elemento tem exatamente uma entrada em cima e uma saída em baixo. Os elementos só se **encaixam uns nos outros**, nunca se ligam com setas. Assim não há código esparguete.

> Escreve as condições como comparações reais: `num % 2 == 0` em vez de «num par?». Assim, a simulação consegue verificá-las e o código é gerado diretamente.

Variáveis com nomes que digam o que guardam, contador `i`, encaixado `j`.

> **Testar:** simulação passo a passo ou teste de secretária com uma tabela de valores.

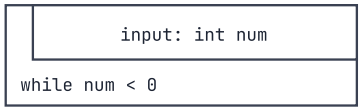
Ciclo com teste no início — while



Verifica a condição **antes** de cada passagem. Se logo no início for false, o corpo não corre nem uma vez.

```
while (i < 5) { … }
```

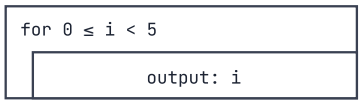
Ciclo com teste no fim — do-while



Executa primeiro o corpo e verifica **depois**. Corre, portanto, pelo menos uma vez e repete enquanto a condição for true.

```
do { … } while (num < 0);
```

Ciclo de contagem — for



Repete um número fixo de vezes. O contador `i` muda a cada passagem. Verifica antes, por isso tem teste no início.

```
for (int i = 0; i < 5;
i++) { … }
```

Ciclo infinito — endless



Não tem condição e nunca para sozinho, por exemplo num robô que anda sempre. Só se sai com um break.

```
while (true) { … }
```

Interrupção — break



Sai imediatamente do ciclo em que está. O programa continua por baixo do ciclo. Usar com moderação.

```
break;
```