Introdução à Programação

João Manuel R. S. Tavares

Universidade do Porto
Faculdade de Engenharia
FEUP

DEMec
Departamento de Engenharia Mecânica

Sumário

1. Ciclo de desenvolvimento de um programa;
2. Descrição de algoritmos;
3. Desenvolvimento modular de programas;
4. Estruturas de controlo de um programa.

Execução de uma tarefa no computador

Passos até escrever as instruções para executar uma determinada tarefa:

1 - Determinar qual deve ser a saída.

2 - Identificar os dados, ou entrada, necessária para obter a saída.

3 - Determinar como processar a entrada para obter a saída desejada.

Execução de uma tarefa no computador

Exemplos de execução de tarefas

1 - Um exemplo do dia a dia: fazer um bolo de maçã:

- ***Saída**:* bolo de maça.
- ***Entrada**:* ingredientes e respetivas quantidades.
  - ➔ Os ingredientes e quantidades são determinados por aquilo que se quer fazer.
- ***Processamento**:* a receita indica como proceder.

2 - Um problema de cálculo: determinar o valor do selo de uma carta:

- ***Saída**:* valor do selo.
- ***Entrada**:* peso da carta, escalões de peso, custo/escalão.
- ***Processamento**:* o algoritmo indica como proceder.

Ciclo de desenvolvimento de um programa

1. Analisar o problema;
2. Planear a solução;
3. Escolher a interface;
4. Codificar;
5. Testar e corrigir erros;
6. Completar a documentação.

1 - Analisar o problema

- Compreender o que o programa deve fazer, qual deve ser a saída.
- Ter uma ideia clara de que dados (entrada) são fornecidos.
- Perceber muito bem qual a relação entre a entrada e a saída desejada.

2 - Planear a solução

- Encontrar uma sequência lógica e precisa de passos para resolver o problema.
  - Tal sequência de passos é chamada um algoritmo.
  - O algoritmo deve incluir todos os passos, mesmo aqueles que parecem óbvios.
  - Existem vários métodos de especificar o algoritmo:
    - diagramas de fluxo ou fluxogramas;
    - pseudocódigo;
    - diagramas *top-down*.
- O planeamento também envolve um teste "manual" do algoritmo, usando dados representativos.

3 - Escolher a interface

- Determinar como é que a entrada será obtida e como é que a saída será apresentada.

- Por exemplo, em *Visual Basic/C++*:

  - Criar objetos para receber a entrada e apresentar a saída.

  - Criar botões de comando apropriados para que o utilizador possa controlar o programa.

4 - Codificar

- Traduzir o algoritmo para uma linguagem de programação (ex.: *Visual Basic/Fortran*).

  $\Rightarrow$ Temos então um programa.

- Introduzir o programa no computador.

5 - Testar o programa e corrigir erros (*debugging* / depuração)

- Localizar e remover eventuais erros do programa.
  - Os erros sintáticos resultam do facto de o utilizador não ter escrito o programa de acordo com as regras da gramática da linguagem de programação utilizada; são detetados pelo compilador/interpretador da linguagem.
    O computador não executará nenhuma instrução sintaticamente incorreta.
  - Os erros semânticos resultam do facto de o programador não ter expressado corretamente, através da linguagem de programação, a sequência de ações a ser executada.
    Estes erros têm de ser detetados pelo programador através de testes exaustivos com dados variados para os quais a saída deve ser conhecida.

6 - Completar a documentação

- A documentação serve para que outra pessoa ou o próprio programador, mais tarde, entenda o programa.
- A documentação consiste em incluir comentários no programa que especificam:
  - o objetivo do programa;
  - como usar o programa;
  - a função das variáveis;
  - a natureza dos dados guardados nos ficheiros;
  - as tarefas executadas em certas partes do programa;
  - …
- Em programas comerciais, a documentação inclui, normalmente, um manual de instruções.
- Outros tipos de documentação: pseudocódigo, fluxograma, diagrama *top-down*.

Descrição de algoritmos

- Duas formas utilizadas:

Pseudocódigo

Descreve a sequência de passos usando uma linguagem parecida com a linguagem comum (Inglês, Português, …) mas usando frases com construções próximas das que são aceites por muitas linguagens de programação.

Exemplos de construções:

1 - Se *condição* então fazer *ações_1* senão fazer *ações_2*

2 - Repetir *ações* até que *condição*

Fluxograma ou diagrama de fluxo

Descreve graficamente a sequência de passos a executar para resolver um determinado problema e como os passos estão interligados. É constituído por um conjunto de símbolos geométricos ligados por setas.

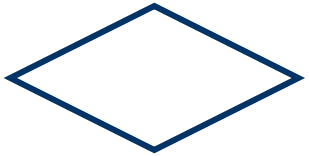
Símbolos *ANSI* usados em fluxogramas

Linha de fluxo - usado para ligar os outros símbolos indicando a sequência de operações

Terminal - usado para representar o início ou o fim de uma tarefa

Entrada/Saída - usado para operações de entrada/saída tais como ler ou imprimir
(os dados a ler/escrever são indicados no interior)

Processamento - usado para operações de manipulação dos dados ou operações aritméticas



Decisão - usado para indicar operações de teste
(tem uma entrada e duas saídas correspondentes ao resultado do teste ser verdadeiro ou falso)

Símbolos *ANSI* usados em fluxogramas

Processo pré-definido - usado para representar um grupo de operações que constituem uma tarefa

Conector - usado para ligar diferentes linhas de fluxo

Conector para fora da página - usado para indicar que o fluxograma continua noutra página

Comentário - usado para fornecer informação adicional acerca de outro símbolo do fluxograma

Exemplo

PROBLEMA:

Calcular as raízes reais de uma equação do 2º grau.

equação : $Ax^2+Bx+C = 0$

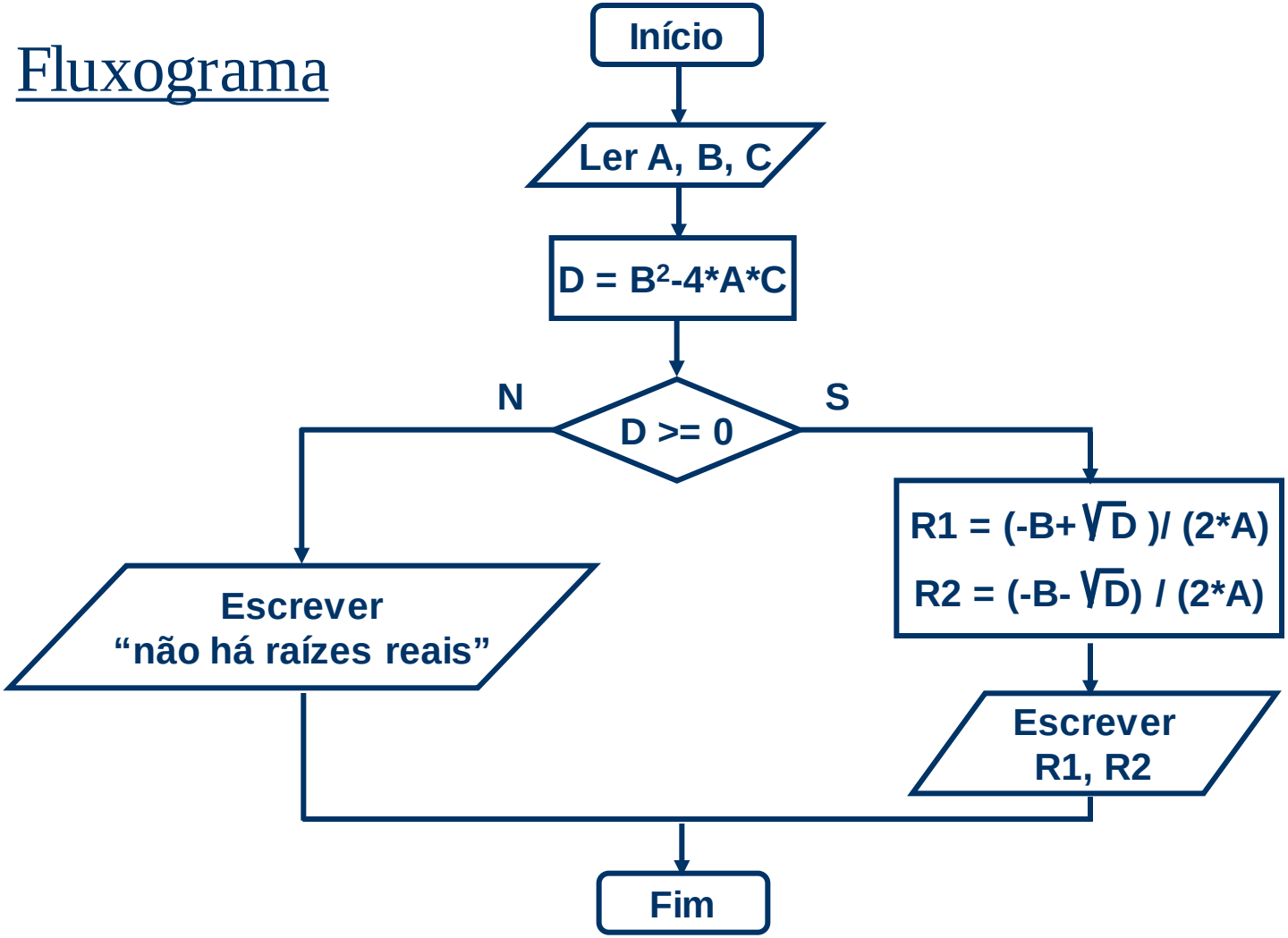
raízes : $x = (-B \pm \sqrt{B^2- 4AC}\ ) / (2A)$

Pseudocódigo

- Ler (A, B, C) ;
- Calcular D = B^2-4*A*C ;
- Se D >= 0 então
  { Calcular R1 = (-B + $\sqrt{D}$) / (2*A) ;
  Calcular R2 = (-B - $\sqrt{D}$) / (2*A) ;
  Escrever (R1, R2) ; }
  Senão
  Escrever (“não tem raízes reais”) ;
- Fim

Exemplo

Fluxograma



Descrição da estrutura de um programa

- A estrutura de um programa pode ser descrita através de um diagrama de estrutura, diagrama hierárquico ou diagrama *top-down* que descreve a organização do programa, mas omite os pormenores das operações.
- Ele descreve o que cada parte ou módulo do programa faz e mostra como os diferentes módulos estão relacionados entre si.
- O diagrama lê-se do topo para baixo (*top-down*) e da esquerda para a direita.
- Cada módulo pode estar dividido em sub-módulos e assim sucessivamente.
- Estes diagramas são úteis no planeamento inicial do programa e ajudam a escrever programas bem estruturados.

Desenvolvimento modular de programas

- Método usado para lidar com problemas de programação complexos.
- Começa-se por dividir a tarefa inicial em sub-tarefas algumas das quais poderão ser de grande complexidade.
- Cada uma destas sub-tarefas é, por sua vez, dividida em sub-tarefas mais simples e assim sucessivamente, até que todas as tarefas estejam descritas de forma suficientemente elementar para poderem ser facilmente codificadas na linguagem de programação escolhida.
- Vantagens do desenvolvimento modular:
  - um módulo pode ser facilmente reutilizado em várias partes do programa;
  - facilita a deteção e correção de erros (analisando os sintomas de um erro é frequentemente fácil reduzir a causa desse erro a um determinado módulo).

Exemplo

PROBLEMA:

Calcular a área de um triângulo, dadas as coordenadas dos vértices, usando a fórmula de Heron:

$$AREA = S * (S-A) * (S-B) * (S-C)$$

em que S = semi-perímetro = (A + B + C) / 2 e A, B e C = comprimentos dos lados.

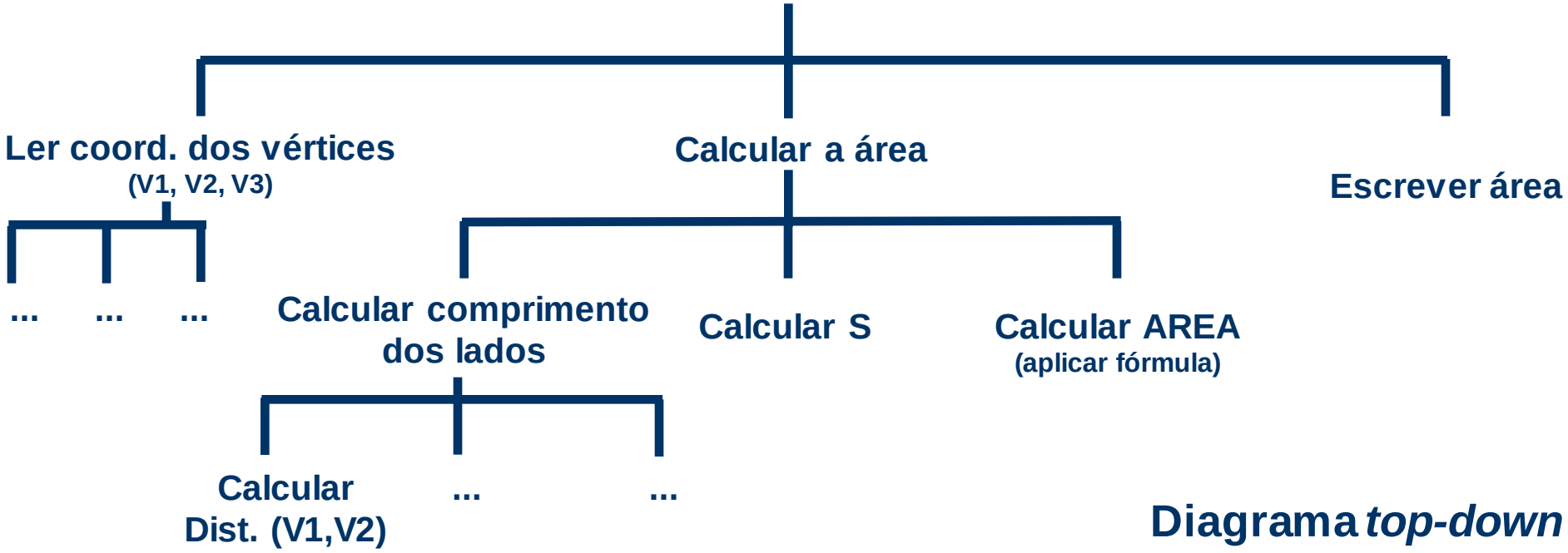


Diagrama *top-down*

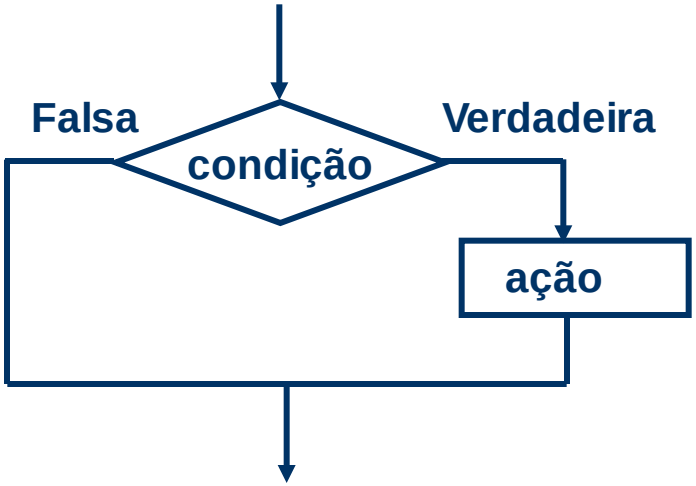
Estruturas de controlo de um programa

- Em geral, um programa não é constituído por uma sequência rígida, linear, de instruções que são executadas sempre do mesmo modo.
- Muitas problemas requerem que seja tomada uma decisão para selecionar entre duas sequências de instruções qual a que vai ser executada.
- Por vezes, é necessário repetir um determinado conjunto de instruções enquanto se verificar uma determinada condição, até que se verifique uma determinada condição, ou um determinado número de vezes.
- A generalidade das linguagens de programação possui além de *instruções simples* de leitura, escrita e atribuição de valores *instruções de controlo* que envolvem ações de seleção ou de repetição de sequências de instruções, permitindo "fugir" a uma sequência rígida, linear, de execução de um programa.

Instruções condicionais

Permitem uma seleção de sequências alternativas de instruções.

Fluxograma



Pseudocódigo

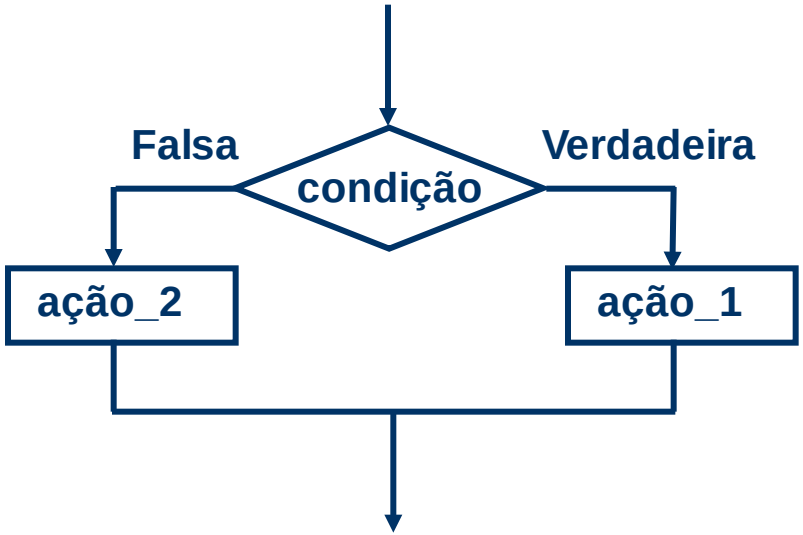
Se *condição* então *ação*

Se a condição for verdadeira a ação é executada.
Se a condição for falsa não é executada nenhuma ação, passa a ser executada a instrução seguinte.

Uma ação pode ser constituída por uma ou mais instruções.

Instruções condicionais

Fluxograma



Pseudocódigo

Se *condição* então
    ação_1
senão
    ação_2

Se a condição for verdadeira é executada a ação_1.

Se a condição for falsa é executada a ação_2.

Instruções de repetição

Usadas quando se pretende executar uma sequência de instruções zero ou mais vezes.

Há 3 variantes de instruções repetitivas:

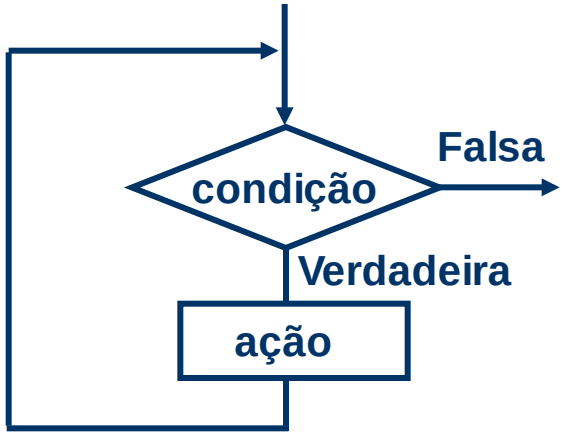
1) Enquanto se verificar uma condição executar uma ação;

2) Repetir uma ação até que se verifique uma condição;

3) Executar uma ação um número conhecido de vezes.

Instruções de repetição

Fluxograma



Pseudocódigo

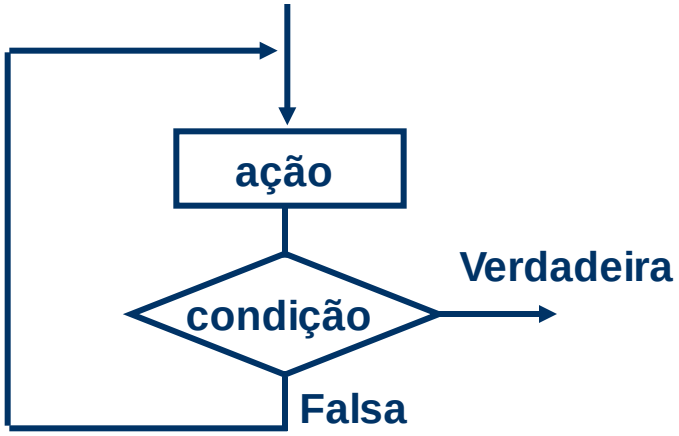
Enquanto *condição* executar
ação

Se a condição for verdadeira é executada a ação e volta-se a testar a condição.
Se a condição for falsa passa-se à execução da instrução seguinte.

A ação pode ser executada zero (se o teste de condição resultar falso logo da 1ª vez) ou mais vezes.

Instruções de repetição

Fluxograma



Pseudocódigo

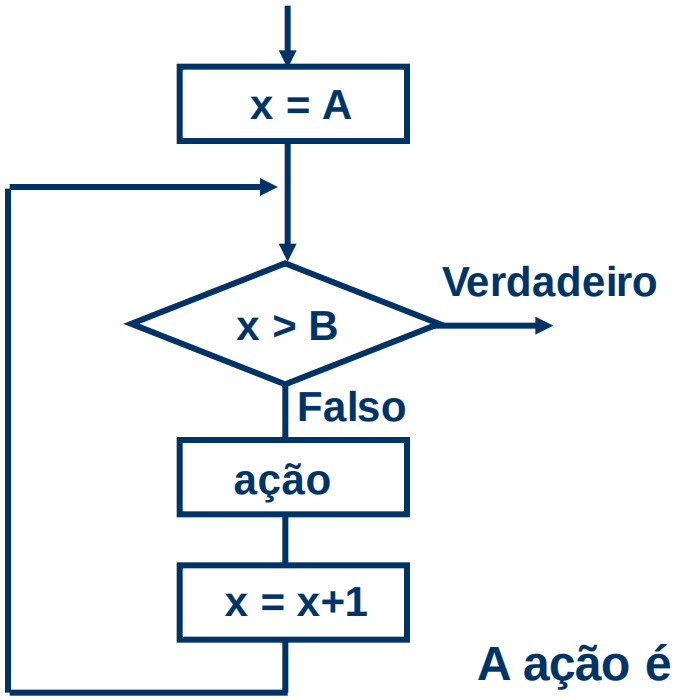
Repetir
ação
até que *condição*

A ação é executada e, a seguir, testa-se a condição.
Se a condição for falsa a ação é repetida e volta-se a testar a condição.
Se a condição for verdadeira passa-se à execução da instrução seguinte.

A ação pode ser executada uma (se o teste de condição resultar verdadeiro logo da 1ª vez) ou mais vezes.

Instruções de repetição

Fluxograma



Pseudocódigo

Para *x* de *A* até *B* executar
ação

A ação é executada um certo número de vezes, desde um valor inicial (*A*) até um valor final (*B*) de uma variável (*x*), designada *contador*, que controla o ciclo.

Se *A* > *B* o ciclo não é executado nenhuma vez.