Algoritmos e Programação

Teoria e Prática

Marco Medina
Cristina Fertig

Novatec Editora

Capítulo 1

Introdução

Neste capítulo, faremos uma introdução geral sobre algoritmos, suas aplicações e alguns exemplos reais. Mostraremos as diferenças entre algoritmo e programa e também explicaremos o que são compiladores e montadores. Em seguida, descreveremos algumas formas populares de estruturação de algoritmos e apresentaremos a notação que adotaremos.

1.1 Conceituação

Muitas definições podem ser dadas à palavra algoritmo. Atualmente, tem-se associado algoritmo à computação, mas este não é um termo restrito à computação ou que tenha nascido com ela. Na realidade, a palavra algoritmo vem do nome do matemático iraniano Abu Abdullah Mohammad Ibn Musa al-Khawarizmi, nascido em Khawarizm (Kheva), ao sul do mar Aral, que viveu no século XVII. A influência de Khawarizmi no crescimento da ciência em geral, particularmente na matemática, astronomia e geografia, é bastante reconhecida. Também é considerado o fundador da álgebra, cujo nome derivou de seu livro Al-Jabr wa-al-Muqabilah. Mais informações a respeito de al-Khawarizmi podem ser encontradas na bibliografia (apêndice B).

O termo algoritmo também é utilizado em outras áreas, como engenharia, administração, entre outras. Vejamos algumas definições de algoritmo:

- Um procedimento passo a passo para a solução de um problema.
- Uma seqüência detalhada de ações a serem executadas para realizar alguma tarefa.

Assim, as ações que são necessárias para se fazer um balancete, por exemplo, constituem um algoritmo. Outro exemplo clássico de algoritmo é a receita culinária. Veja o exemplo a seguir de um bolo de chocolate:

- **Ingredientes**
    - 4 xícaras (chá) de farinha de trigo.

- 2 xícaras (chá) de açúcar cristal.
- 2 xícaras (chá) de achocolatado.
- 2 colheres (sopa) de fermento em pó.
- 1 pitada de sal.
- 3 ovos.
- 2 xícaras (chá) de água morna.
- 1 xícara (chá) de óleo.
- Óleo para untar.
- Farinha de trigo para polvilhar.

- **Modo de preparo**
  - Numa vasilha, misture 4 xícaras (chá) de farinha de trigo, 2 xícaras (chá) de açúcar cristal, 2 xícaras (chá) de achocolatado, 2 colheres (sopa) de fermento em pó e 1 pitada de sal. Junte 3 ovos, 2 xícaras (chá) de água morna e 1 xícara (chá) de óleo. Misture bem. Unte uma forma retangular de 25 cm x 37 cm com óleo e polvilhe farinha de trigo e despeje a massa. Asse em temperatura média (de 170°C a 180°C) por 30 minutos.
  - A receita tem todas as características de um algoritmo. Ela tem uma seqüência detalhada de passos, descrita no modo de preparo. Apresenta a tarefa a ser realizada, que no caso é o bolo de chocolate. Além disto, podemos identificar na receita entradas (no caso os ingredientes) e uma saída, que é o próprio bolo.
  - Poderíamos, então, nos perguntar por que a palavra algoritmo ficou tão associada à computação? Para compreendermos melhor os motivos, é preciso entender, mesmo que superficialmente, o funcionamento dos computadores.

1.2 Programas de computador

Nesta seção, veremos o processo necessário para se criar um programa e executá-lo. Primeiramente introduziremos os principais conceitos para a melhor compreensão de como um programa é visto pelo computador. Depois nos aprofundaremos nos detalhes de como um programa é transformado em um código que pode ser executado pelo computador.

1.2.1 O que é um programa

Os computadores das mais variadas arquiteturas têm funcionamento similar. A figura 1.1 apresenta a arquitetura simplificada de um computador.

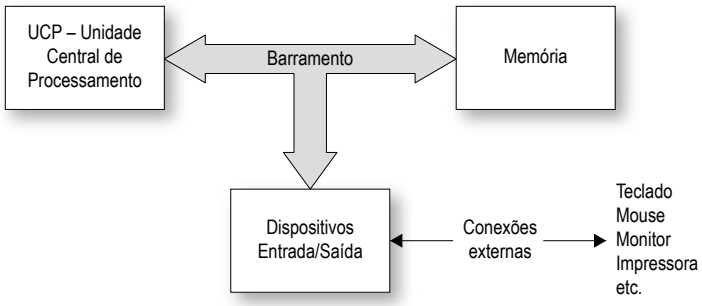

Figura 1.1 – Arquitetura simplificada de um computador.

A parte física do computador é chamada de hardware, que é formado basicamente por uma Unidade Central de Processamento (UCP), pela memória e pelos dispositivos de entrada e saída. O barramento faz a ligação desses componentes.

A UCP (ou processador) contém um conjunto relativamente pequeno de instruções que é capaz de executar. Cada processador contém um conjunto diferente de instruções, apesar de similares entre si. As instruções que o processador executa são buscadas da memória. Essas instruções podem ser desde operações matemáticas a interações com os dispositivos de entrada e saída. Chamamos de programa de computador um conjunto de instruções que será executado pelo processador em uma determinada seqüência. Esse programa leva o computador a executar alguma tarefa.

Como podemos perceber, um programa nada mais é que um tipo de algoritmo. Sua particularidade é que suas operações são específicas para o computador e restritas ao conjunto de instruções que o processador pode executar. Podemos considerar esse conjunto de instruções como a primeira linguagem de programação do computador, também chamada de linguagem de máquina. Classificamos as linguagens de programação segundo a sua proximidade com a linguagem de máquina. Quanto maior a semelhança com a linguagem de máquina, mais baixo é o nível da linguagem. As linguagens de programação mais semelhantes à linguagem de máquina são conhecidas como linguagens de baixo nível. Analogamente, linguagens de programação "distantes" da linguagem de máquina são conhecidas como linguagens de programação de alto nível. Linguagens de programação de alto nível são mais próximas da linguagem natural e guardam pouca similariedade com a linguagem da máquina em que serão executadas.

A linguagem de programação que um computador é capaz de compreender é composta apenas de números. Assim, fazer algoritmos na linguagem de programação do computador ou em sua linguagem de máquina é um processo extremamente complicado para nós, seres humanos, acostumados com a nossa própria linguagem. Por isso, para facilitar a programação de computadores, foi necessária a criação de um código que relacionasse a linguagem de máquina a uma linguagem mais fácil de ser compreendida. A linguagem de montagem (ou assembly) é um código que tem uma instrução alfanumérica (ou mnemônica) para cada instrução numérica em linguagem de máquina.

Para que um programa escrito em linguagem de montagem possa ser executado pelo computador, é necessário que seu código seja traduzido para o código de máquina. Isto é feito por meio de um programa chamado assembler. A figura 1.2 apresenta o esquema de tradução feita por um assembler.

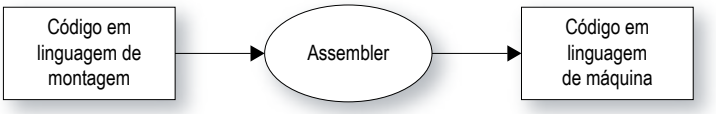

Figura 1.2 – Tradução para a linguagem de máquina.

A linguagem de montagem é muito próxima da linguagem de máquina e, por isso, é uma linguagem de programação de baixo nível. Para cada processador, há uma linguagem de montagem já que há uma relação direta entre as instruções em linguagem de montagem e em linguagem de máquina. Isto faz com que o código tenha que ser refeito se quisermos executar um programa em um processador não compatível com o qual ele foi escrito inicialmente. Neste caso, dizemos que o código não é portável.

A implementação de programas nesse tipo de linguagem ainda é muito complexa e dependente do conhecimento das instruções do processador. Para aumentar a produtividade dos programadores e a portabilidade dos programas, foram criadas as linguagens de programação de alto nível. Essas linguagens são independentes do processador em que serão executadas. Suas características principais são que seu código é mais elaborado, contemplando operações mais complexas e mais próximas da "lógica humana". Para que possam ser processadas por um computador, os comandos da linguagem precisarão ser traduzidos para a linguagem de máquina. Essa tradução é feita por meio de um compilador ou de um interpretador, dependendo do caso, como veremos a seguir.

1.2.2 Executando um programa

O compilador, a partir do código em linguagem de alto nível, chamado de código-fonte, gera um arquivo com o código em linguagem de máquina, conhecido como código-objeto. Esse código-objeto fica em disco e só é carregado em memória no momento da execução. O interpretador faz o mesmo trabalho, porém não gera o arquivo em código-objeto. As instruções são traduzidas para linguagem de máquina em tempo de execução, instrução a instrução. A figura 1.3 mostra os passos para a execução de um código em linguagem de alto nível por meio da compilação. A figura 1.4 apresenta o esquema similar para a interpretação.

Figura 1.3 – Compilação de um programa.

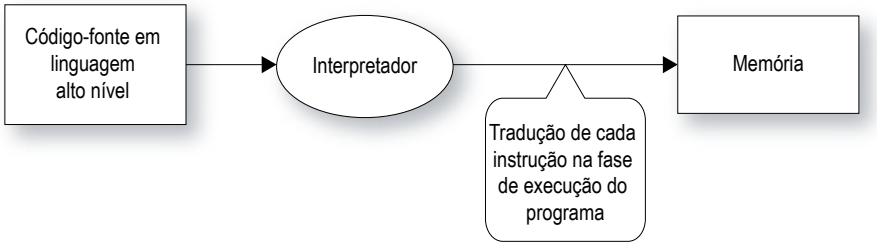

Figura 1.4 – Interpretação de um programa.

Podemos notar que programar um computador tornou-se muito mais fácil do que anteriormente. A introdução de linguagens de alto nível como Pascal, C, Cobol, Java, entre outras, aumentou a produtividade dos programadores, permitindo que programas mais elaborados fossem feitos, atendendo a demandas mais complexas. Além disso, os códigos são portáveis, ou seja, independentes da plataforma mediante nova compilação do código-fonte.

1.2.3 Linguagens de programação e sistemas operacionais

A realização de determinadas operações em linguagem de máquina pode ser extremamente complexa. Operações que necessitem de interface com os dispositivos de entrada e saída, como gravação ou leitura de dados em disco, são um exemplo. Esse tipo de operação não é importante apenas em linguagens de programação. Este é um dos motivos pelos quais os sistemas operacionais são construídos.

Na realidade, os sistemas operacionais constroem uma camada entre hardware e software, facilitando a utilização dos recursos da máquina real por meio da criação de uma máquina virtual. Essa máquina virtual é muito mais simples de ser utilizada e provê uma série de mecanismos ao usuário que facilitam a utilização dos recursos do computador. Um exemplo são os arquivos, que permitem a manipulação de dados em disco sem que seja necessário o conhecimento dos detalhes do funcionamento dos discos, como trilhas, setores, velocidade da rotação, posicionamento dos dados no disco e outros mais. Além disso, o sistema operacional gerencia os recursos da máquina, evitando a má utilização e até mesmo a danificação de seus componentes por um programador.

Além de os sistemas operacionais implementarem operações complexas e extremamente úteis ao uso do computador, também fornecem uma interface para que outros programas possam utilizá-las. Essas interfaces são conhecidas como chamadas de sistema, ou system calls, e o conjunto de operações disponibilizadas são as bibliotecas do sistema operacional. Os compiladores utilizam essas interfaces em vez de implementar as operações complexas por si mesmos. Quando o compilador encontra chamadas ao sistema operacional no código-fonte em linguagem de alto nível, transforma-as em referências "não resolvidas" no código-objeto em linguagem de máquina. Para que esse código possa ser executado, precisa ser ligado ao código do sistema operacional que efetivamente

implementa a operação. Esta é a segunda etapa da compilação. O processo é conhecido como ligação do código ou linking. A figura 1.5 apresenta as etapas da transformação de um código em linguagem de alto nível até que este possa ser executado.

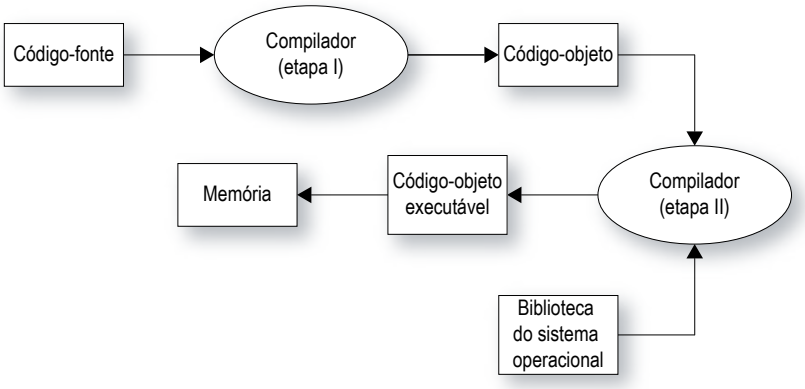

Figura 1.5 – Etapas para a execução de um programa em linguagem de alto nível.

A integração dos compiladores com os sistemas operacionais permite que o código-objeto de um programa seja portável para qualquer máquina que utilize o mesmo sistema operacional, tornando o código-objeto dependente do sistema operacional e não mais da plataforma do processador, visto que o sistema operacional deixa transparente as diferenças entre plataformas diversas. Porém, um código-objeto compilado em um sistema operacional não poderá ser executado em um outro sistema operacional. Assim, um programa executável em Linux provavelmente não poderá ser executado em um sistema operacional HP-UX, SunOS, AIX ou da família Microsoft. Para que esse programa funcione nesses sistemas operacionais, o código-fonte, em linguagem de alto nível, deverá ser compilado neste sistema operacional.

A linguagem de programação Java tem uma arquitetura diferente, o que fornece a seus programas mais portabilidade que outras linguagens de programação de alto nível. A compilação do código Java não gera código executável em nenhum sistema operacional. Em vez disso, gera um código "pseudo-executável", chamado bytecode. Esse código é uma espécie de código de baixo nível que, porém, não é uma linguagem de máquina de algum processador, nem faz interface específica com algum determinado sistema operacional. Para que esse código possa ser executado em algum (e ao mesmo tempo em qualquer) sistema operacional, é necessário que mais uma camada de software esteja aí instalada. A camada funciona como um sistema operacional genérico, deixando transparentes as particularidades do sistema operacional real. Essa camada de software é chamada de Máquina Java Virtual, ou Java Virtual Machine (JVM). A JVM faz a tradução dos bytecodes para código executável naquele sistema operacional. Isto faz com que um programa escrito em Java e que seja compilado para bytecode possa ser executado em qualquer máquina que tenha a JVM adequada instalada. Durante a execução, esse

código será traduzido em tempo real para a linguagem proprietária e executado. A tradução em tempo real pode ser entendida como uma espécie de interpretação. Portanto, é difícil afirmar que a linguagem de programação Java seja uma linguagem puramente compilada ou interpretada.

Essa característica da linguagem Java faz com que esta seja extremamente interessante para o uso na Internet, visto que o código Java que esteja presente em um site poderá ser executado em qualquer plataforma que contenha a JVM instalada. Normalmente, os browsers já contêm uma JVM para que possam trabalhar com sites que contenham programas Java. A figura 1.6 apresenta as etapas necessárias para a execução de um programa Java.

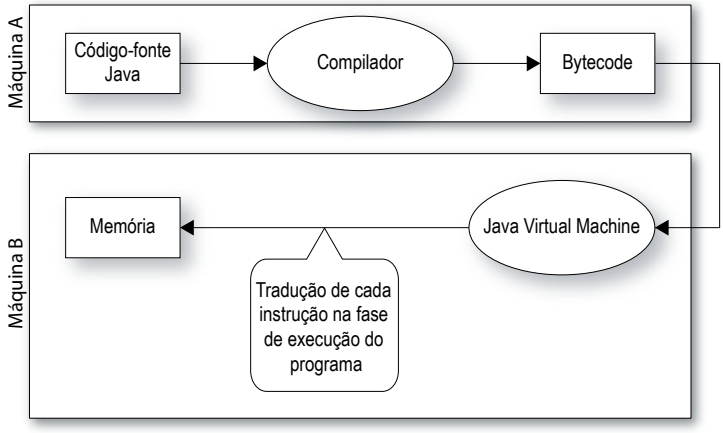

Figura 1.6 – Processo de execução de programas Java.

1.3 Estruturação de algoritmos

1.3.1 Linguagem natural

Vimos a evolução na portabilidade dos programas de alto nível e o quanto tais linguagens de programação se afastaram da linguagem de máquina e se aproximaram da "lógica humana". Porém, para um iniciante, mesmo a linguagem de programação de alto nível pode ser um tanto complexa. Claro que é muito mais simples do que programar em linguagem de máquina ou em linguagem de baixo nível, mas o verdadeiro problema é que as pessoas, em geral, não estão acostumadas a fazer algoritmos.

Diferentemente da linguagem natural, a linguagem de programação é dirigida a orientar uma máquina e não pessoas. Máquinas não podem tomar decisões com base em premissas. Máquinas não podem escolher alternativas, mesmo que estas nos pareçam óbvias. Máquinas não podem corrigir comandos mal redigidos. Máquinas não podem descobrir a intenção do programador, mesmo que ela seja (ou pelo menos pareça) clara

no contexto. Pessoas fazem tudo isso (pelo menos na maior parte das vezes) sem sequer notar. Por isso, a linguagem de programação precisa ter algumas características que a linguagem natural não tem. Veja-as a seguir:

- **Rigidez sintática**: O compilador é um tradutor relativamente limitado, que só consegue fazer as traduções sobre um idioma bastante limitado, com construções muito bem definidas. Apesar de encontrarmos palavras pertencentes à linguagem natural, elas não serão usadas com a mesma liberdade.

- **Rigidez semântica**: O computador definitivamente não pode lidar com ambigüidades. Por isso, não adianta o programador ter uma intenção se não conseguir exprimi-la de forma exata. Podemos dizer que o computador é um ótimo cumpridor de ordens, porém não tem idéia de quais ordens está cumprindo, nem o contexto em que essas ações estão inseridas. Diferentemente da linguagem de programação, a linguagem natural apresenta ambigüidades. Veja o exemplo:

  – "A velhinha ouviu o barulho da janela".

  Essa frase curta pode ser interpretada de pelo menos três maneiras:

  1. A velhinha ouviu o barulho produzido pela janela.

  2. A velhinha estava junto à janela e ouviu o barulho.

  3. A velhinha ouviu o barulho que veio através da janela.

Qualquer máquina seria incapaz de interpretar o que realmente está acontecendo, mesmo que o contexto pudesse ajudar. Por isso, a rigidez semântica é tão crucial e conseqüentemente a linguagem natural não pode ser utilizada como ferramenta para a construção de algoritmos para computador.

A necessidade desses quesitos faz com que a linguagem natural não seja a escolha adequada para a escrita de algoritmos para computador. A segunda alternativa seria escrever o algoritmo diretamente na linguagem de programação. Porém, a rigidez sintática e a semântica tornam a escrita de algoritmos diretamente em uma linguagem de programação real, mesmo de alto nível, uma tarefa bastante difícil, pois as pessoas não estão acostumadas a essas exigências para expressar ordens. Muitas vezes, mesmo em linguagem natural, esta não é uma tarefa trivial.

Realmente chegamos a um dilema: a linguagem natural não é adequada porque não tem rigidez sintática e semântica e a linguagem de programação não é adequada justamente por ter essas características. Parece claro que teremos de encontrar uma terceira alternativa para expressarmos algoritmos para computador.

Na realidade, já existem algumas alternativas. Apresentaremos duas delas: fluxograma e pseudocódigo.

1.3.2 Fluxograma

Os fluxogramas apresentam os algoritmos de forma gráfica. São formados de caixas que contêm as instruções a serem executadas. Tais caixas são ligadas por setas que indicam o fluxo das ações. Algumas caixas especiais indicam a possibilidade de o fluxo seguir caminhos distintos, dependendo de certas situações que podem ocorrer durante a execução do algoritmo. Também há representações para o início do algoritmo e para o seu final, para que o fluxo do algoritmo possa ser determinado do seu princípio até o seu término. A figura 1.7 apresenta o exemplo de um algoritmo na forma de fluxograma para a escolha e apresentação do maior valor entre dois números distintos introduzidos pelo usuário.

De fato, a representação de algoritmos por meio de fluxogramas tem uma série de vantagens. A primeira é a facilidade proporcionada para a compreensão do funcionamento do algoritmo, mesmo para os leigos. Algumas pessoas também se adaptam bem ao desenvolvimento de algoritmos sob essa representação.

Entretanto, a representação gráfica não é prática. A correção de uma linha de pensamento pode implicar a reconstrução de muitas instruções. Além disso, a construção de algoritmos mais complexos e longos pode se tornar extremamente trabalhosa, ocupando várias páginas. Essas características acabam tornando a utilização do fluxograma desaconselhável como ferramenta principal para o desenvolvimento de algoritmos. Todavia, a utilização de fluxogramas continua sendo útil para apresentação de algoritmos em um nível de abstração alto, sem entrar nos detalhes de sua implementação.

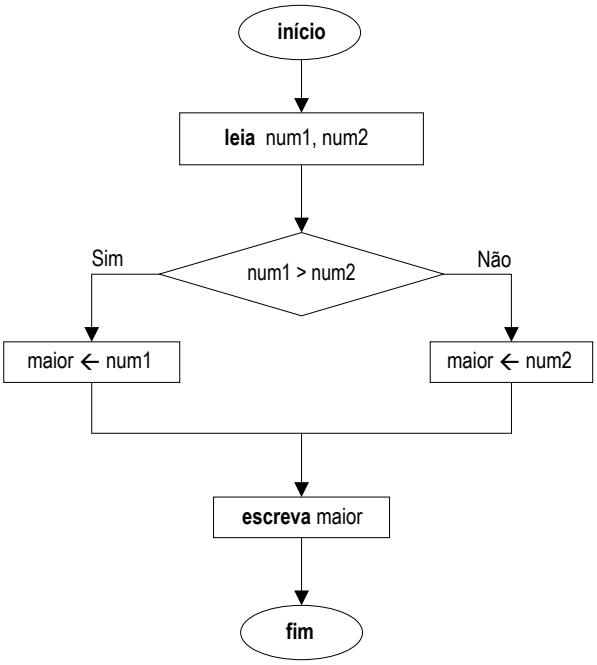

Figura 1.7 – Fluxograma para a escolha do maior de dois números distintos.

1.3.3 Pseudocódigo

Algoritmos podem ser representados em código diretamente em linguagem de programação. Como já vimos no item 1.3.1, a implementação de algoritmos diretamente em uma linguagem de programação apresenta algumas desvantagens. Porém, é fato que temos algumas vantagens como o código pronto para a execução (mesmo que seja após a compilação) e, o mais importante, a rigidez sintática e a semântica, que são imprescindíveis para que o algoritmo possa ser lido e executado pelo computador.

O pseudocódigo visa a trazer o máximo possível desses benefícios, tentando diminuir o ônus da utilização da linguagem de programação. Abre-se mão do código compilável para se ter um código menos rígido, menos dependente das peculiaridades que todo compilador tem. Ao contrário da linguagem de programação, o pseudocódigo tem um grau de rigidez sintática intermediária entre as linguagens natural e de programação. Além disso, podemos utilizar o idioma nativo. Em geral, linguagens de programação são construídas utilizando palavras reservadas em inglês, uma espécie de padrão de mercado.

Porém, o pseudocódigo deve manter, tanto quanto possível, a rigidez semântica. A idéia é que o pseudocódigo seja um passo intermediário entre a linguagem natural, a que os iniciantes estão acostumados, e a linguagem de programação de alto nível.

Após a construção do algoritmo em pseudocódigo, é necessário que mais um passo seja executado para que o algoritmo possa ser compilado e posteriormente executado. É a transformação do pseudocódigo em código de alguma linguagem de programação real. O pseudocódigo é independente do compilador e pode ser traduzido de uma forma quase direta para uma gama de linguagens de programação.

Um pseudocódigo bastante conhecido no Brasil é o Portugol, que apresenta uma aceitação grande e tem suas razões para isso. É bastante simples e atende às características necessárias no que concerne à rigidez semântica e à sintática. Entretanto, o Portugol opta por algumas construções em detrimento de outras que consideramos interessantes para o passo posterior, que é a transformação do pseudocódigo em programa. A questão principal é a ausência de blocos genéricos de comandos, que veremos no capítulo 2. Tais construções estão presentes na maior parte das linguagens de programação de alto nível e são importantes para que o programador iniciante fique acostumado com esse tipo de estruturação.

Para representarmos nossos algoritmos, utilizaremos uma adaptação do Portugol, que incluirá algumas construções e mecanismos que julgamos adequados para o melhor aprendizado do programador, procurando facilitar o processo de construção do algoritmo.

Veja a seguir um exemplo de pseudocódigo, referente ao fluxograma mostrado na figura 1.7.

Algoritmo **Maior**

```
var num1, num2, maior: inteiro;
início
   leia(num1, num2);
   se (num1 > num2) então
      maior ← num1;
   senão
      maior ← num2;
   escreva(maior);
fim
```

1.4 Exercícios do capítulo

1. Defina o que é um algoritmo.

2. Diferencie um algoritmo de um programa.

3. Explique como um programa é executado em um computador.

4. Defina o que é uma linguagem de programação de alto nível e uma linguagem de programação de baixo nível.

5. Dado um programa executável em um sistema operacional, o que é preciso fazer para que tal programa possa ser utilizado em outro sistema operacional?

6. Explique por que um código Java é portável em vários sistemas operacionais.

7. Por que a linguagem natural não é adequada para a construção de algoritmos para computador?

8. Quais as vantagens e desvantagens da utilização de fluxograma e de pseudocódigo na construção de algoritmos?