Daniel Hayashida Simão
Wellington José dos Reis

Lógica de Programação
Conhecendo Algoritmos e Criando Programas

editora
VIENA

1ª Edição
Bauru/SP
Editora Viena
2015

Sumário

Lista de Siglas e Abreviaturas

FIFO	*First In, First Out.*
IDE	*Integrated Development Environment.*
LIFO	*Last In, First Out.*
PEPS	*Primeiro que Entra, Primeiro que Sai.*
POO	*Programação Orientada a Objetos.*
RAD	*Rapid Application Development.*
UML	*Unified Modeling Language.*

1
Introdução

1. Introdução

A lógica de programação está ligada diretamente ao contexto da informática, mais precisamente no segmento da programação, que é utilizada para desenvolver softwares, sistemas, sites e aplicativos. No entanto, a lógica está muito mais presente no nosso cotidiano do que imaginamos. Para qualquer coisa que fazemos é necessário realizar tomadas de decisões, e estas seguem um fluxo lógico, que são os passos que executamos para atingir um determinado objetivo. Na lógica de programação, também é necessário analisar o problema e a partir daí ordenar instruções em uma sequência lógica. Isso é feito utilizando uma linguagem de programação que possui suas particularidades.

É importante informar que não existe somente uma maneira correta para se atingir determinado objetivo, pois, cada pessoa possui sua lógica, sua maneira de pensar e de executar determinados passos.

Nesta etapa de aprendizado, o maior desafio é desenvolver uma mentalidade lógica e adequá-la às rotinas básicas da programação, com o intuito de analisar e resolver problemas da melhor maneira possível, pois, possuindo uma boa lógica de programação é possível adequá-la a qualquer linguagem de programação.

O presente livro é o ponto de partida para iniciar os estudos da programação, desde o seu início e evoluir até o ponto de adequar os conhecimentos em lógica para linguagens de programação, para que, assim, possam se aprofundar em linguagens de programação específicas e obter resultados mais relevantes. Este livro é destinado a pessoas que desejam iniciar os estudos em programação e que não possuam nenhuma noção do assunto.

1.1. Aplicação da Matemática à Linguagem de Programação

A matemática está presente em todas as tarefas realizadas diante do computador e liga-se diretamente à lógica de programação. As atividades relacionadas à programação exigem que o programador possua conhecimentos das principais funções e propriedades da matemática, como cálculo de vetores, matrizes e diversos princípios de álgebra, os quais são transmitidos ao indivíduo durante seu período acadêmico.

O desenvolvimento de algoritmos que servirão de base para a criação de programas é estritamente dependente do uso de técnicas matemáticas, como testes de mesa, uso de estruturas condicionais, de decisão, utilização de operadores lógicos, além de outros fundamentos matemáticos. Portanto, se o interesse é se aprofundar no mundo da lógica de programação, é necessário dedicação e empenho na área de exatas.

1.2. Algoritmos

O estudo da lógica de programação tem como objetivo principal construir algoritmos válidos e eficazes. Mas o que são algoritmos? Podemos defini-los como uma sequência de procedimentos que visam atingir um determinado objetivo. Portanto, na medida em que é necessária a criação de novos algoritmos para resolver novos problemas, é de extrema importância a utilização da lógica.

Antes de entrar na parte mais técnica dos algoritmos, vamos criar algoritmos para resolver problemas cotidianos, como o bom e velho exemplo da troca de lâmpadas.

Problema: Troca de lâmpadas.

Exemplo de algoritmo:

1. Localize a escada.
2. Pegue-a.
3. Posicione-a abaixo da lâmpada.
4. Pegue a lâmpada nova.
5. Suba na escada.
6. Retire a lâmpada velha.
7. Coloque a lâmpada nova.
8. Desça da escada.

Desta forma, involuntariamente, analisamos um problema, criamos procedimentos e os executamos, ou seja, foi utilizada uma lógica para criar um algoritmo.

O algoritmo descrito anteriormente é apenas uma possível resolução para o problema proposto. Existem diversas outras soluções, pois, como mencionado anteriormente, cada pessoa possui sua lógica, ou seja, sua maneira de elaborar procedimentos para atingir o objetivo.

Analisando o algoritmo anterior, é possível observar que ele tem um objetivo claro, que é trocar a lâmpada, o que foi executado com êxito. Mas será que a lâmpada está funcionando? Quais os procedimentos que devem ser tomados caso a lâmpada esteja funcionando e quais os procedimentos que devem ser tomados caso a lâmpada não esteja funcionando? Essas são situações que não foram pensadas na criação do primeiro algoritmo. Portanto, é possível criar um algoritmo mais completo, envolvendo teste e situações condicionais.

Problema: Trocar a lâmpada com teste.

Exemplo de algoritmo:

1. Localize a escada.
2. Pegue-a.
3. Posicione-a abaixo da lâmpada.
4. Pegue a lâmpada nova.
5. Suba na escada.
6. Retire a lâmpada velha.
7. Coloque a lâmpada nova.
8. Desça da escada.
9. Aperte o interruptor. A luz acendeu?
  9.1. Sim.
    9.1.1. Guarde a escada.
  9.2. Não.
    9.2.1. Pegue uma lâmpada nova.
    9.2.2. Suba na escada.
    9.2.3. Retire a lâmpada velha.
    9.2.4. Coloque a lâmpada nova.
    9.2.5. Desça da escada.
    9.2.6. Aperte o interruptor. A luz acendeu?

9.2.6.1. Sim.
9.2.6.1.1. Guarde a escada.
9.2.6.2. Não.
9.2.6.2.1. Pegue outra lâmpada nova.
...
...
...

O algoritmo acima se aproxima um pouco mais de uma solução completa, pois o fluxo só será finalizado quando a lâmpada for trocada e a luz acender, ou seja, só assim a troca será realizada com êxito, do contrário, o fluxo será realizado infinitas vezes. Esse algoritmo está mais completo, no entanto possui dois pontos negativos: não saber o número necessário de lâmpadas para atingir o objetivo e ter de ficar repetindo o texto várias vezes. Para não precisar ficar repetindo o texto várias vezes, é necessário expressar uma situação condicional de repetição, a qual será realizada sempre que a lâmpada não acender.

Problema: Trocar a lâmpada com teste e condição.

Exemplo de algoritmo:

1. Localize a escada.
2. Pegue-a.
3. Posicione-a abaixo da lâmpada.
4. Pegue a lâmpada nova.
5. Suba na escada.
6. Retire a lâmpada velha.
7. Coloque a lâmpada nova.
8. Desça da escada.
9. Aperte o interruptor. A luz acendeu?
9.1. Sim.
9.1.1. Guarde a escada.
9.2. Enquanto a lâmpada não acender, faça:
9.2.1. Pegue uma lâmpada nova.
9.2.2. Suba na escada.
9.2.3. Retire a lâmpada velha.
9.2.4. Coloque a lâmpada nova.

Por meio desse algoritmo poderemos atingir o que foi proposto, que é trocar a lâmpada até que ela acenda, além de criar um "código" mais curto utilizando uma condição de parada. Mas é claro que, mediante um simples problema como "trocar uma lâmpada", poderíamos definir diversos outros problemas e condições para aprimorar nosso algoritmo, por exemplo, trocar no máximo 10 lâmpadas e parar mesmo que ela não acenda, testar a mesma lâmpada em vários soquetes, e assim por diante.

Mas o que trocar a lâmpada tem a ver com programação e com desenvolver um software? Assim, como nos procedimentos para trocar uma lâmpada, um programa é desenvolvido por meio de linhas de códigos, que são nada mais do que instruções escritas de acordo com uma linguagem de programação específica. Nos exemplos anteriores, escrevemos nossos códigos utilizando a língua portuguesa, mas poderia ser em **C**, **Java**, **Python**, entre outras linguagens.

A elaboração de um algoritmo para a criação de um programa de computador requer algumas etapas:

- Definir o problema (o que o programa tem que fazer).
- Estudar a situação atual e analisar a forma de resolver o problema.
- Desenvolver o programa utilizando uma linguagem de programação.
- Testar o programa com o intuito de encontrar possíveis erros.
- Após a implementação, analisar junto aos usuários se o programa satisfaz todas as necessidades (se o problema foi solucionado).

É necessário salientar que o algoritmo deve possuir procedimentos claros e precisos, que precisam ser seguidos fielmente a fim de que se crie um padrão para garantir que sempre ele seja executado sob as mesmas condições, funcione e produza o resultado esperado.

1.3. Programas

Como já mencionado anteriormente, os programas são desenvolvidos por meio de linguagens de programação específicas. Mas, para que esses programas sejam executados e disponham de todas as suas funcionalidades, é necessário outros programas que traduzam o código do programa desenvolvido para uma linguagem que o computador entenda e execute. Existem diversos programas que auxiliam no desenvolvimento de programas, como o **compilador**, **interpretador** e a **IDE**.

1.3.1. Compilador

O compilador é um programa que traduz o código escrito em uma linguagem de programação para um código capaz de ser interpretado pelo computador. Esse código compilado é chamado de código objeto, que pode ser executado e reproduzir as funcionalidades descritas no programa. Esse tipo de programa é um dos mais utilizados.

Os compiladores analisam o código em três partes: de forma sintática ou hierárquica, análise léxica ou linear e análise semântica.

Veja abaixo o fluxo seguido pelo compilador:

Vantagens:

- O código compilado é mais rápido de ser acessado.
- Dificulta o acesso ao código-fonte original.
- Permite otimização do código por parte do compilador.
- Compila o código somente se ele não possuir erros.

Desvantagens:

- Para ser utilizado, o código precisa passar por muitos níveis de compilação.
- A dificuldade em acessar o código-fonte, que foi citada acima como uma vantagem, também pode ser uma desvantagem.

- Ao realizar uma alteração no código, será necessário recompilar tudo novamente.

1.3.2. Interpretador

Os interpretadores leem um código-fonte de uma linguagem de programação interpretada e o convertem em um código capaz de ser executado pelo computador. Em alguns casos, o interpretador lê linha por linha e converte em código-objeto à medida que vai executando o programa e, em outros casos, converte o código-fonte por inteiro e depois o executa, o que pode variar de acordo com a implementação.

Ao contrário do compilador, o interpretador executa o código-fonte escrito como sendo o código-objeto, traduzindo-o linha por linha. Dessa forma, o programa vai sendo utilizado à medida que vai sendo traduzido. Cada execução do programa exige que ele seja novamente traduzido e interpretado.

Veja abaixo o fluxo seguido pelo interpretador:

Os interpretadores analisam os códigos de duas formas: sintática e semanticamente. Se o código for aprovado mediante essas duas análises, ele poderá ser executado.

Vantagens:

- Rapidez na realização de correções e alterações.
- Eliminação da necessidade de compilar o código para que seja executado.
- Consumo de menos memória em relação ao compilador.

Desvantagens:

- Execução mais lenta do programa.
- Necessidade de ler o código original sempre que for executado.

1.3.3. IDE

A **IDE (Integrated Development Environment — Ambiente Integrado de Desenvolvimento)** é um programa de computador com características e ferramentas que agilizam o processo de desenvolvimento de programas.

Esse tipo de programa favorece a aplicação de uma técnica chamada **RAD (Rapid Application Development)**, em português **Desenvolvimento Rápido de Aplicativos**, que visa maior produtividade por parte dos desenvolvedores.

A **IDE** é a alternativa mais completa para o desenvolvimento de programas. Veja abaixo algumas das funções e ferramentas mais comuns encontradas em uma **IDE**:

- **Editor de código**: Local para desenvolvimento e edição do código do programa.
- **Compilador**: Transforma o código digitado em uma linguagem de programação para a linguagem da máquina.
- **Linker**: Reuni as partes do código-fonte compilados em linguagem de máquina em um programa executável, para que, assim, possa ser executado em um computador.

- **Depurador ou Debugger**: Auxilia no processo de encontrar possíveis erros no código do programa.
- **Modelagem**: Criação do modelo de classes, objetos, interfaces, associações e interações de todo o programa que está sendo desenvolvido.
- **Geração de código**: Gera códigos baseados em códigos comumente utilizados para solucionar problemas rotineiros, uma espécie de templates. Esse recurso torna muito mais rápido o desenvolvimento e a distribuição do programa.
- **Distribuição**: Auxilia na criação do instalador do software.
- **Testes automatizados**: São testes automáticos no programa, baseados em scripts ou programas de testes, gerando um relatório, com o intuito de analisar o impacto das alterações no código.
- **Refatoração**: Esse recurso consiste na melhoria constante do código do programa. A refatoração, juntamente com os testes automatizados, é uma ótima alternativa para tentar eliminar os famosos "bugs".

1.4. Linguagens de Programação

Podemos considerar uma linguagem de programação como um idioma qualquer, o alemão ou japonês, por exemplo, pois uma linguagem de programação, assim como os idiomas, é padronizada e possui regras de sintaxe para comunicar instruções, sendo a linguagem de programação utilizada para se comunicar com o computador, permitindo a um programador que especifique precisamente como o computador deverá agir. Linguagens de programação são utilizadas para expressar algoritmos com precisão.

A linguagem de programação tem como objetivo facilitar a comunicação do programador com o computador, permitindo ao programador que expresse melhor suas intenções por meio de uma linguagem de programação, em vez de programar na linguagem do computador, que só entende números binários (0 e 1). Dessa forma, é mais fácil o programador aprender uma linguagem de programação, como o **Java** ou o **C**, que possuem uma sintaxe mais fácil de ser compreendida por nós, humanos, em vez de aprender a linguagem da máquina, os números binários (0110001001010010).

As linguagens de programação tornam os programas mais portáteis, ou seja, menos dependentes de computadores ou ambientes computacionais específicos, pois programas escritos em linguagens de programação são traduzidos para o código de máquina do computador no qual será executado, em vez de ser executado diretamente.

Podemos catalogar as linguagens de programação em dois tipos: de alto nível e baixo nível. Lembramos que isso não significa superioridade de um tipo sobre o outro.

- **Linguagem de programação de alto nível**: Esse tipo de linguagem possui um nível de abstração relativamente elevado, ou seja, é mais compreensível ao ponto de vista humano, longe do código de máquina, que só é compreendida pelos computadores. Portanto, o programador de uma linguagem de alto nível não precisa conhecer características do processador, como instruções e registradores, pois elas são abstraídas na linguagem de alto nível. Esse tipo de linguagem é a mais utilizada atualmente.

- **Linguagem de programação de baixo nível**: Esse tipo de linguagem está diretamente relacionado com a arquitetura do computador, pois esta utiliza somente instruções do processador, sendo necessário conhecer os registradores da máquina. Um programa desenvolvido em linguagem de baixo nível permite máximo controle das instruções, tendendo a ser um programa que exige velocidade de processamento.

1.4.1. Pseudolinguagem

Pseudolinguagem ou pseudocódigo são maneiras genéricas de se escrever um algoritmo utilizando uma linguagem simples, que pode ser entendida por todos, sem necessariamente conhecer uma linguagem de programação. Pelo fato de não ser uma linguagem de programação padronizada, os algoritmos desenvolvidos com pseudolinguagens não podem ser executados em sistemas reais. No entanto, foi criada uma pseudolinguagem denominada **Portugol**, que nada mais é que um português estruturado para escrever algoritmos.

Veja um simples esboço da pseudolinguagem **Portugol**, cujo objetivo é apenas exibir a mensagem **Olá Mundo**:

```
ALGORITMO OláMundo

INÍCIO

ESCREVA ("Olá Mundo")

FIM
```

Os livros e os cursos superiores da área de programação utilizam frequentemente o pseudocódigo para ilustrar os seus exemplos e introduzir os conceitos iniciais sobre programação e novas técnicas.

Observe a seguir um exemplo utilizando o **Portugol** para resolver outro problema, que envolve a necessidade de ler dois números reais, calcular a multiplicação entre eles e exibir o resultado na tela:

```
ALGORITMO multiplicacao
VARIÁVEIS
   m: REAL
   n: REAL
   x: REAL
INÍCIO
   LEIA m
   LEIA n
   x ← m × n
   ESCREVA x
FIM
```

A declaração das variáveis deve ser feita antes da execução das instruções do programa, ou seja, antes do **INÍCIO** do programa. Após a declaração das variáveis, começa-se a montar a lógica para atingir o objetivo: a entrada dos dois números, o processamento da multiplicação e a exibição do resultado.

Após esse processamento, o resultado armazenado na variável **x** é exibido na tela do computador.

1.4.2. Teste de Mesa

Após desenvolver um algoritmo, não é possível testá-lo diretamente no computador, pois ele é representado por meio de pseudocódigos. No entanto, é possível verificar a exatidão de um algoritmo por meio do chamado **teste de mesa**.

O teste de mesa simula de maneira simples e precisa todas as instruções descritas no algoritmo desenvolvido, informando se ele possui algum erro em sua concepção que poderá impedir sua posterior implementação, independente da linguagem de programação que será utilizada.

Veja a seguir um exemplo de aplicação do teste de mesa a partir de um algoritmo desenvolvido com o propósito de receber duas notas, realizar a soma e, em seguida, apresentar o resultado do cálculo da média final.

Algoritmo:

1. Receba a nota da prova 1.
2. Receba a nota da prova 2.
3. Some as notas e divida o resultado por 2.
4. Exiba o resultado da divisão.

Teste de mesa:

Prova 1	Prova 2	Média
8,0	9,0	8,5

Obseve que, somando a nota de **Prova 1** (8,0) e **Prova 2** (9,0), obtemos como resultado da soma o valor 17,0. Para realizar o cálculo da **Média**, basta dividir o valor da soma (17,0) por 2 (número de notas lançadas), tendo como saída o valor 8,5, ou seja, o algoritmo foi desenvolvido de forma correta e poderá ser posteriormente implementando em qualquer linguagem de programação, como na linguagem **C**, construindo um programa que realiza o cálculo da média de alunos de uma sala de aula, por exemplo.

1.4.3. Boas Práticas para Construção de um Algoritmo

Na hora de escrever um algoritmo, é importante adotar como rotina métodos que representem o objetivo dos passos descritos de maneira eficaz.

Veja a seguir algumas boas práticas que devem ser seguidas durante o processo de desenvolvimento de um algoritmo:

- Leia atentamente o enunciado do problema.
- Descreva os passos de maneira didática e usando linguagem de fácil entendimento, desenvolvendo um texto que seja compreendido principalmente por indivíduos que não trabalham com informática e não possuem conhecimento avançado na área.
- Utilize somente um verbo por frase.
- Evite a utilização de frases longas e termos complexos que possam desviar o foco principal da situação.
- Evite palavras que possam causar dupla interpretação.

- A inserção de comentários, principalmente em algoritmos longos, auxilia a posterior consulta e entendimento do algoritmo, lembrando que outro indivíduo pode utilizar ou terminar um algoritmo criado por você, e, portanto, presença de comentários facilita o entendimento e função dos passos.
- Ao concluir uma construção de um algoritmo, peça a alguém que leia e tente interpretá-lo, retornando um feedback.
- Sempre faça o teste de mesa ao final da construção do algoritmo.

Anotações

Anotações

2
Diagrama de Bloco

2. Diagrama de Bloco

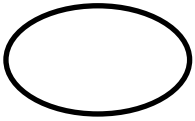
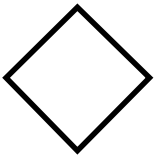
O diagrama de bloco, também chamado de diagrama de fluxo, é uma forma padronizada para representar ou descrever os passos lógicos de um determinado processamento de dados de um sistema computacional. Esse diagrama utiliza os símbolos de um fluxograma, que tem como função descrever o fluxo de uma ação de um trabalho, seja manual ou automático.

O diagrama de bloco também é uma forma de escrever algoritmos, pois com ele é possível definir claramente uma sequência de símbolos com significados bem definidos, facilitando a visualização dos passos a serem seguidos.

2.1. Simbologia

Um diagrama de bloco é formado por símbolos. Cada símbolo possui um significado que indica uma determinada ação, o que facilita a interpretação do fluxo descrito.

Existem diversos símbolos, no entanto não utilizaremos todos. Veja a seguir uma tabela com os principais símbolos e suas respectivas descrições:

Símbolo	Descrição
	Utilizado para indicar o início e o fim do algoritmo.
	Utilizado para representar a entrada de dados.
	Utilizado para representar a saída de dados.
	Utilizado para representar cálculos e atribuição de valores.
	Utilizado para representar tomadas de decisões e desvios condicionais.
	Utilizado para conectar os blocos existentes, indicando o fluxo dos dados.