



Escola Nacional de Ciências Estatísticas

Apostila de Introdução à Programação
Prof. Eduardo Corrêa

Capítulo III – Linguagens de Programação de Alto Nível e Algoritmos

Sumário

III. Linguagens de Programação e Algoritmos.....	2
III.1 Linguagens de alto nível.....	2
III.2 Algoritmos	5
III.3 Técnicas para a elaboração de algoritmos	6
Exercícios propostos	13



III. Linguagens de Programação e Algoritmos

Um **algoritmo** pode ser definido como uma sequência ordenada de passos (instruções) que deve ser seguida para a resolução de um problema. Os programas de computador nada mais são do que algoritmos implementados numa determinada linguagem de programação, normalmente, uma **linguagem de alto nível**. Este capítulo tem como principal objetivo apresentar o conceito de algoritmo, explicar porque ele é tão importante na computação e introduzir suas técnicas básicas de elaboração.

III.1 Linguagens de alto nível

Conforme visto nos capítulos anteriores, computadores são máquinas capazes de solucionar problemas através da execução de instruções que lhe são fornecidas em programas. A CPU é o componente do computador responsável por decodificar as instruções. No entanto, ela pode entender diretamente apenas instruções em linguagem de máquina (binária). Uma **instrução de máquina** é formalizada por uma sequência de 0's e 1's:

0100 00000001 11010001

No exemplo acima **0100** é a instrução responsável por somar dois números de uma dada CPU hipotética. Os números a serem somados devem ser indicados logo após a instrução (neste caso eles são 00000001 e 11010001, respectivamente, 1 e 209 em representação decimal). Isto quer dizer que toda vez que a CPU recebe uma instrução que possui “0100” no início ela sabe que deve realizar a soma de dois inteiros. Um **programa executável** (por exemplo, um arquivo **.exe** no Windows) é constituído de um conjunto de instruções de máquina.

A linguagem de máquina é extremamente inconveniente para seres humanos, conforme demonstra a Figura 1 que exhibe um trecho de programa em linguagem de máquina para o processador MIPS. O trecho em questão serve apenas para trocar o valor de duas variáveis. Veja como tudo fica complicado nesse emaranhado de 0's e 1's!

```
00000000101000100000000100011000
00000000100000100001000000100001
10001101111000100000000000000000
100011100001001000000000000000100
10101110000100100000000000000000
101011011110001000000000000000100
0000001111100000000000000000001000
```

Figura 1 – Programa em linguagem de máquina¹

¹ Exemplo retirado do livro “Computer Organization and Design”, de D. A. Patterson e J. L. Hennessy, Ed. Morgan Kaufmann, 5ª Edição (2017).



Escola Nacional de Ciências Estatísticas

A linguagem de máquina possui diversas desvantagens:

☹ Não é nada amigável para o programador. É difícil de escrever e entender programas nessa linguagem.

☹ A implementação fica mais sujeita à erros.

☹ Cada família de processadores tem a sua própria linguagem de máquina. Ex.: a instrução de adição pode ter o código “0100” para computadores com a CPU do modelo x, mas pode ter código diferente para uma CPU de modelo y, produzida por outro fabricante.

Para resolver estes problemas, foi adotada a seguinte solução: criar um outro conjunto de instruções que seja **mais dirigido às pessoas e menos dirigido à máquina**. Ou seja, instruções numa linguagem que se aproxime mais de uma linguagem natural (no caso, a língua Inglesa). Este novo conjunto de instruções compõe o que se chama de **linguagem de alto nível**. Nos dias atuais existem centenas de linguagens de alto nível. A relação a seguir apresenta apenas algumas das mais populares²:

- Python
- C
- Java
- C++
- C#
- Visual Basic
- JavaScript
- SQL
- R
- Go
- Ruby
- Julia
- Lua
- MATLAB
- SAS

A Figura 2 demonstra o mesmo trecho de programa apresentado na Figura 1, mas desta vez implementado em uma linguagem de alto nível (Python) ao invés da linguagem de máquina. Veja como ele se torna muito mais legível para um ser humano.

² <https://www.tiobe.com/tiobe-index/>



```
def trocar(x, y):  
    temp = x  
    x = y  
    y = temp  
    return x, y
```

Figura 2 – Programa em linguagem de alto nível

Há duas diferentes maneiras de transformar um programa escrito em linguagem de alto nível (também chamado de programa fonte – *source program*) para um programa em linguagem de máquina. São elas:

- **Compilação:** nesta abordagem, o programa fonte é traduzido uma única vez gerando um arquivo executável (ex.: arquivo “.exe” do Windows) contendo as instruções de máquina. O arquivo executável possui esse nome porque pode ser levado para qualquer máquina e executado diretamente (não é preciso nenhum outro programa para executá-lo). Se o programa fonte for alterado, precisará ser compilado novamente para que um novo executável seja gerado. O programa que realiza a tradução é chamado de **compilador**. C, Pascal e Java são exemplos de linguagens compiladas.
- **Interpretação:** nesta abordagem, é realizada uma tradução “em tempo real” do programa fonte sempre que desejamos executá-lo. O programa responsável por realizar este tipo de tradução é chamado **interpretador**. Na interpretação, não ocorre a geração de um arquivo executável. Desta forma, não basta distribuir o código fonte para uma pessoa para que ela possa executar o programa, pois ela precisará também do interpretador para executá-lo. PHP, Python e R são exemplos de linguagens interpretadas.

Devido a razões muito particulares, uma determinada linguagem de programação de alto nível é projetada para se enquadrar em uma dessas duas categorias. Normalmente, as linguagens interpretadas como Python e R são mais adequadas para aplicações estatísticas, uma vez que permitem a execução de um programa de forma **interativa**, onde cada comando digitado é imediatamente traduzido e executado. Isto oferece aos programadores uma forma ágil e simples para examinar em tempo real os resultados intermediários obtidos em qualquer passo de um processo de análise de dados. É por isso que existem tantas linguagens: algumas possuem características adequadas para um tipo de problema, outras para outros tipos.

As principais características das linguagens de programação de alto nível são apresentadas a seguir:

☺ São mais amigáveis para o ser humano.

☺ Minimizam as chances de erro.



Escola Nacional de Ciências Estatísticas

😊 O programador não precisa conhecer a linguagem de máquina de um modelo específico de CPU para programar numa linguagem de alto nível.

Por outro lado, é importante deixar claro que cada linguagem possui um interpretador ou compilador específico. Ou seja, um interpretador Python traduz apenas programas escritos em Python para código de máquina (Figura 3); um interpretador Javascript traduz apenas programas escritos em Javascript. Mas o interpretador Python não “fala” Javascript e o interpretador Javascript não “fala” Python!

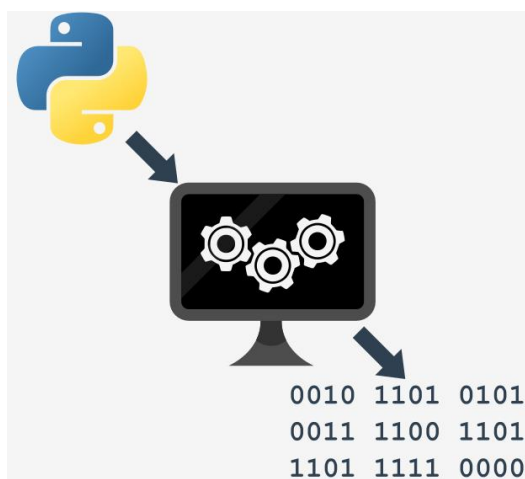


Figura 3 – Interpretador Python

III.2 Algoritmos

As linguagens de programação de alto nível são ótimas para as pessoas, mas ainda não representam o nível mais alto de abstração³ para a solução de um problema. No entanto, esta solução de “altíssimo nível” pode ser obtida através da construção de **algoritmos**.

Um algoritmo pode ser definido da seguinte maneira: “uma **sequência finita e não ambígua** de instruções para executar uma tarefa específica”. Um algoritmo é semelhante a uma **receita** (ex.: receita culinária ou instruções para instalar um ventilador de teto), **porém é mais complexo**, pois: (i) pode incluir **repetição** de passos até que uma condição seja satisfeita; (ii) pode incluir **testes de condições lógicas** para executar a tarefa. A Figura 4 ilustra a ideia:

³ Resumidamente, pode-se definir **abstração** como um processo mental onde procura-se desenvolver a solução para um problema, sem que haja preocupação em detalhar com extrema precisão a forma de implementar esta solução.



Escola Nacional de Ciências Estatísticas

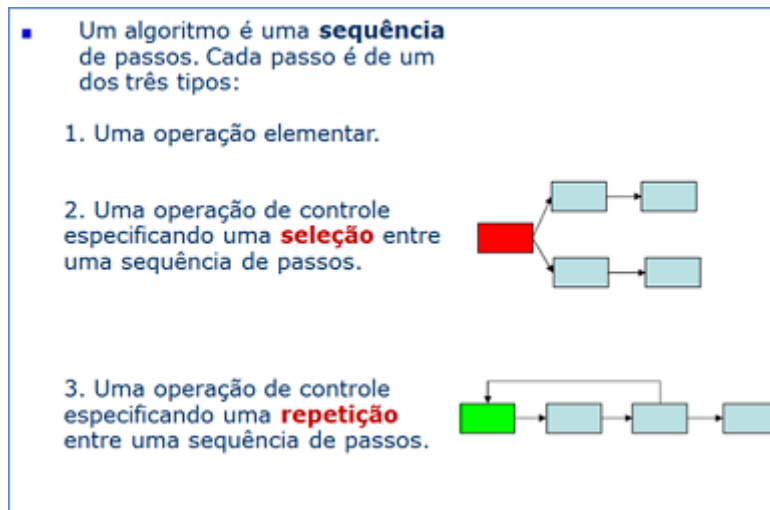


Figura 4 – O conceito de algoritmo

Um programa de computador nada mais é do que um algoritmo escrito numa dada linguagem. Os programas de computadores também são formados por sequências de passos dos três tipos apresentados na Figura 4: operações elementares, instruções de seleção e instruções de repetição.

III.3 Técnicas para a elaboração de algoritmos

As principais técnicas para a elaboração de algoritmos são:

- Descrição Narrativa.
- Fluxograma.
- Pseudocódigo.

III.3.1 Descrição Narrativa

Consiste em analisar o enunciado de um problema e escrever, **utilizando linguagem natural** (Língua Portuguesa), os passos a serem seguidos para a solução do problema. A Figura 5 apresenta o algoritmo para a solução do seguinte problema: preparar um ovo frito.



Exemplo: preparar um ovo frito

1. Colocar margarina na frigideira;
2. Acender o fogão;
3. Quebrar a casca do ovo;
4. Despejar o ovo na frigideira;
5. Aguardar algum tempo;
6. Retirar o ovo da frigideira;
7. Apagar o fogão.

Figura 5. Algoritmo implementado com a técnica de descrição narrativa

☺ A principal vantagem da descrição narrativa é o fato de que, para utilizar esta técnica, não é necessário aprender nenhum conceito novo, pois todo mundo conhece uma linguagem natural.

☹ No entanto a descrição narrativa possui um problema que inviabiliza seu uso na resolução de problemas complexos: a língua natural é **ambígua** e **imprecisa**. Considere, por exemplo, o passo 5 do algoritmo para fritar um ovo. O que significa “aguardar algum tempo”? Dependendo do nível de conhecimento sobre o problema cada pessoa poderá assumir que é um período de tempo diferente (5 minutos? 10 minutos? 10 horas?).

☹ Outro problema é que a descrição narrativa não se assemelha muito ao formato de um programa, ao contrário das duas outras técnicas que ainda serão apresentadas (fluxograma e pseudocódigo).

Apesar das desvantagens, a descrição narrativa é útil como ponto de partida para o desenvolvimento de um programa (para fazer aquele primeiro “rascunho” da solução). Antes de pensar em detalhes, é mais fácil (especialmente em dias de prova!) escrever uma solução com os passos em Português para depois convertê-la em pseudocódigo ou num programa de computador.

III.3.2 Fluxograma

Consiste em analisar o enunciado do problema e escrever, utilizando **símbolos gráficos predefinidos**, os passos a serem seguidos para a resolução do mesmo. Esses símbolos possuem função semelhante aos dos comandos disponíveis em linguagens de programação. Os exemplos a seguir ilustram a elaboração de algoritmos com o uso de fluxogramas e introduzem alguns dos principais símbolos utilizados.



Escola Nacional de Ciências Estatísticas

Exemplo III.1: Elabore um algoritmo para a resolução do seguinte problema: receber um número como entrada e imprimir o valor do dobro deste número.

SOLUÇÃO (Figura 6):

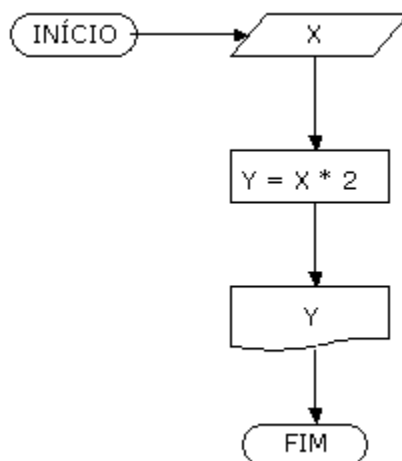


Figura 6. Algoritmo que contém apenas comandos elementares.

Para a elaboração da solução acima, foram utilizados os símbolos apresentados na Figura 7:

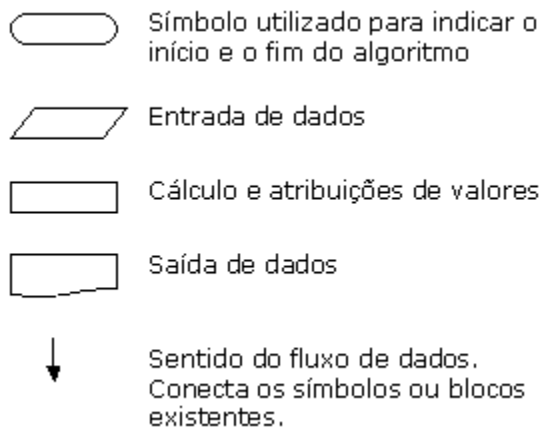
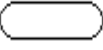


Figura 7. Símbolos dos comandos elementares.

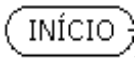
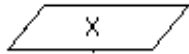
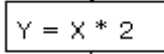
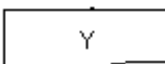
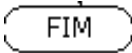
Um fluxograma sempre começa com o símbolo , com a palavra “INÍCIO” escrita dentro deste símbolo. De maneira análoga, o algoritmo deve terminar com o mesmo símbolo, mas com a palavra “FIM” escrita.



Escola Nacional de Ciências Estatísticas

O símbolo ↓ é utilizado para ligar todos os símbolos do algoritmo, indicando a sequência em que cada passo é executado.

O algoritmo da Figura 6 será comentado a seguir:

- Ele começa com o símbolo de início , o que é padrão para qualquer fluxograma.
- O passo a seguir representa um comando de **entrada de dados**: . Este comando serve para indicar que o usuário deverá digitar um valor (no caso um número) e esta informação será referenciada como **X** ao longo do fluxograma (X é o que chamamos de **variável** – conceito introduzido no Capítulo I).
- Em seguida realiza-se uma **operação aritmética** sobre o valor de X: . Esta instrução indica que o valor de X está sendo multiplicado por 2 e que o resultado desta operação é **armazenado** em outra variável, chamada Y.
- O passo seguinte indica um comando de **saída de dados**: . A instrução representa a impressão do valor de Y para o usuário.
- Por fim, o algoritmo é encerrado com o uso do símbolo: .

Observe que os símbolos são dispostos na sequência adequada à resolução do problema (afinal de contas, por definição, um algoritmo representa uma sequência ordenada de passos para a resolução de um problema).

Exemplo III.2: Elabore um algoritmo para resolver o seguinte problema: receber uma nota de um aluno como entrada. Caso o valor da nota seja maior do que 7 imprimir “aprovado”. Caso contrário imprimir “reprovado”.

SOLUÇÃO (Figura 8):

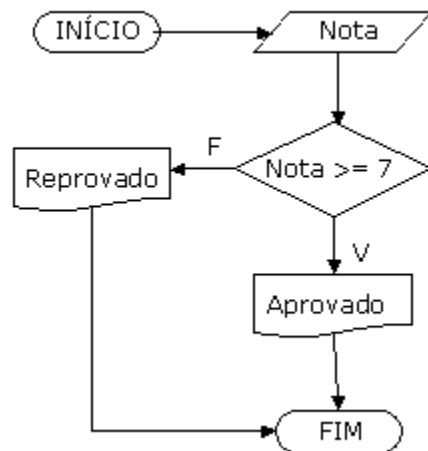



Figura 8. Algoritmo que possui comando de Desvio Condicional

Neste exemplo, foi preciso utilizar o símbolo de **desvio condicional** (ou **alternativa** ou **teste de condição lógica**)  para que fosse possível resolver o algoritmo. O desvio condicional permite a escolha do grupo de instruções a ser executado de acordo com o resultado de um teste. Independentemente da complexidade do teste, ele sempre deverá resultar em apenas um dos seguintes valores: VERDADEIRO (V) ou FALSO (F). O exemplo acima indica que o algoritmo imprimirá a mensagem “Aprovado” apenas se o resultado do teste **Nota >= 7** for VERDADEIRO (veja que “Nota >= 7” é a condição especificada dentro do símbolo de desvio condicional). Se o resultado do teste for FALSO, o valor impresso será “Reprovado”. Observe que o algoritmo possui **dois “caminhos”** diferentes até o seu final: um passando pela impressão da palavra “Reprovado” e outro passando pela impressão da palavra “Aprovado”. É importante deixar claro que um algoritmo pode ser composto por inúmeros comandos de alternativa, dispostos em diferentes passos, caso a solução do problema exija isto.

Exemplo III.3: Elabore um algoritmo que leia um número como entrada e calcule e imprima o valor do quadrado deste número. O algoritmo deve ser executado até que o valor “0” seja especificado como entrada.

SOLUÇÃO (Figura 9):

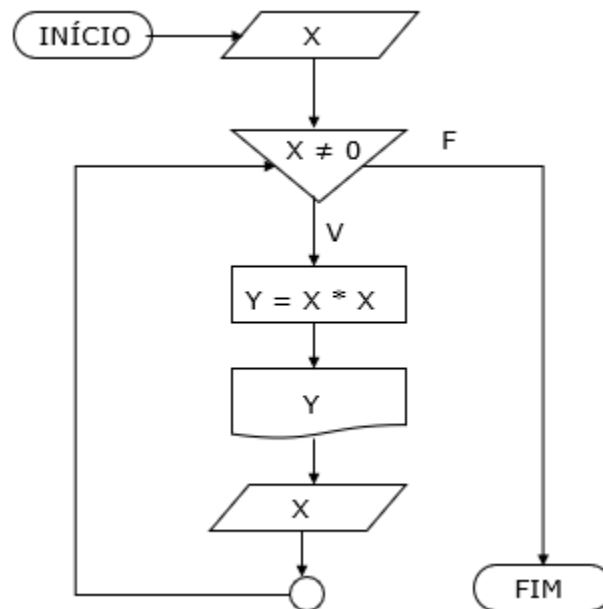


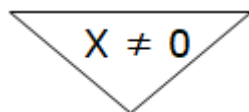




Figura 9. Algoritmo que possui comando de Repetição

Neste exemplo, foi preciso utilizar o símbolo de **repetição** do tipo **enquanto ... faça**:  e o símbolo do fim do **laço de repetição** . Um comando de repetição permite que um conjunto de passos seja executado **repetidas vezes até que uma condição seja atingida**.

Observe o algoritmo acima. Inicialmente o usuário especifica o valor de entrada e este é armazenado em “X”. Em seguida aparece o símbolo do comando de repetição:



Este símbolo testa se o valor de “X” é diferente de zero. Caso seja FALSO (ou seja, se “X” armazena 0), o algoritmo é finalizado (o caminho desvia para o FIM do algoritmo). Caso contrário o algoritmo “**entra no laço de repetição**”. Tudo que está dentro de um laço de repetição é feito repetidas vezes enquanto a sua condição for válida (VERDADEIRA). Os comandos de um laço de repetição do tipo **enquantofaça** sempre são colocados entre os símbolos  e .

No algoritmo da Figura 9, existem três comandos subordinados ao laço de repetição (“dentro” do laço). Um para calcular o “X” ao quadrado, outro para imprimir este valor e um terceiro para solicitar um novo valor para “X” (ou seja, um novo valor para cálculo). Estes comandos serão executados repetidas vezes, até que a **condição de saída** seja especificada pelo usuário. Neste caso, a condição de saída significa o usuário digitar o valor 0. Quando o usuário especifica 0, é porque ele deseja encerrar o algoritmo. Cada laço possui uma condição de saída diferente, dependendo do problema a ser resolvido.



Escola Nacional de Ciências Estatísticas

Nos dias atuais, os fluxogramas são raramente utilizados, já que são trabalhosos de se elaborar e geram esquemas muito grandes para problemas mais complexos. Na prática, os programadores preferem especificar algoritmos usando pseudocódigo, conforme apresentado a seguir.

III.3.3 Pseudocódigo

É considerada uma técnica altamente eficiente, pois permite com que o desenvolvedor de programas possa projetar uma solução para seu problema que seja mais facilmente implementada no computador (de maneira diferente do que o ocorre com a Descrição Narrativa) e de uma forma menos trabalhosa em comparação com o Fluxograma.

Há vários tipos de notação para pseudocódigo, e nessa apostila adotaremos uma abordagem bem simples, que consiste basicamente em trocar os símbolos do fluxograma por palavras em Português com nomes similares aos dos comandos reais da linguagem Python. Isso será ilustrado nos próximos três exemplos, que correspondem aos mesmos que havíamos resolvido por fluxograma. Ao analisar os exemplos resolvidos em pseudocódigo e compará-los com a resolução em fluxograma, você perceberá que os símbolos foram trocados pelas seguintes palavras:

- **início** : início do algoritmo.
- **fim** : fim do algoritmo.
- **ler**: para entrada de dados.
- **imprimir**: para saída de dados.
- **se, senão**: para desvio condicional.
- **enquanto**: para laço de repetição.

Exemplo III.4: Elabore um algoritmo em pseudocódigo para a resolução do seguinte problema: receber um número como entrada e imprimir o valor do dobro deste número.

SOLUÇÃO:

```
início  
    ler (X)  
    Y = X * 2  
    imprimir (Y)  
fim.
```

Exemplo III.5: Elabore um algoritmo em pseudocódigo para resolver o seguinte problema: receber uma nota de um aluno como entrada. Caso o valor da nota seja maior ou igual a 7 imprimir “Aprovado”. Caso contrário imprimir “Reprovado”.



SOLUÇÃO:

```
início
    ler(Nota)
    se nota >= 7.0 então:
        imprimir('Aprovado')
    senão:
        imprimir('Reprovado')
fim.
```

No exemplo acima, veja que utilizamos espaçamento, mais precisamente 4 espaços em branco, para caracterizar que o comando **imprimir('Aprovado')** é **subordinado** ao comando **se nota >= 7.0 então:** (ou seja executado é função deste comando). Da mesma forma fizemos com o comando **imprimir('Reprovado')**, que é subordinado ao comando **senão**.

Exemplo III.6: Elabore um algoritmo em pseudocódigo para o seguinte problema: ler um número como entrada e calcular e imprimir o valor do quadrado deste número. O algoritmo deve ser executado até que o valor “0” seja especificado como entrada.

SOLUÇÃO:

```
início
    ler(X)
    enquanto X ≠ 0 faça:
        Y = X * X
        imprimir(Y)
        ler(X)
fim.
```

Mais uma vez, utilizamos espaçamento (alinhamento com 4 espaços em branco), para caracterizar que os comandos **Y = X * X**, **imprimir(Y)** e **ler(X)**, são subordinados ao comando de repetição **enquanto X ≠ 0 faça:**.

Exercícios propostos

Especifique algoritmos para resolver os problemas abaixo utilizando pseudocódigo.

(1) Receber como entrada o raio da esfera e imprimir o valor do volume.



Escola Nacional de Ciências Estatísticas

(2) Ler a distância a ser percorrida em Km e o consumo de um carro (em Km por litro) e, em seguida, calcular e exibir quantos litros são necessários para completar a distância.

(3) ler 2 notas, calcular e exibir a média. Se a média for maior ou igual a 7, imprimir “Aprovado”. Se for menor do que 7 e maior ou igual a 3 imprimir “Prova Final”. Se for menor do que 3 imprimir “Reprovado”.

(4) ler o valor de N como entrada e calcular e imprimir o valor de H, sendo:

$$H = 1 + (1 / 2) + (1 / 3) + \dots + (1 / N)$$