UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO

ALGORITMO E LÓGICA DE PROGRAMAÇÃO

1ª PARTE

NATAL-RN / 2004

Capítulo 1 3
Introdução 3
1. Conceito de Algoritmo 3
1.1 Algumas Definições de Algoritmo 3
Capítulo 2 4
Formas de Representação de Algoritmos 4
2.1 Descrição Narrativa 4
2.2 Fluxograma Convencional 4
2.3 Pseudocódigo 5
2.4 Síntese 6
Capítulo 3 7
Tipos de Dados 7
3.1 Dados Numéricos 7
3.2 Dados Literais 8
3.3 Dados Lógicos 9
3.4 Síntese 9
3.5 Exercício Proposto 10
Capítulo 4 11
Variáveis 11
4.2 Armazenamento de Dados na Memória 11
4.3 Conceito e Utilidade de Variáveis 13
4.4 Definição de Variáveis em Algoritmos 14
4.5 Síntese 15
4.6 Exercícios Propostos 15
Capítulo 5 24
Expressões 24
5.1 Conceito 24
5.2 Operadores 24
5.3 Tipos de Expressões 25
5.4 Síntese 27
5.6 Exercícios Propostos 28
Capítulo 6 27
Instruções Primitivas 27
6.1 Instrução Primitiva de Atribuição 28
6.2 Instrução Primitiva de Saída de Dados 29
6.3 Instrução Primitiva de Entrada de Dados 1
6.4 Síntese 32
6.5 Exercícios Resolvidos 32
6.6 Exercícios Propostos 36

Capítulo 1

Introdução

1. Conceito de Algoritmo

A automação é o processo em que uma tarefa deixa de ser desempenhada pelo homem e passa a ser realizada por máquinas, sejam estes dispositivos mecânicos, eletrônicos (como os computadores) ou de natureza mista.

Para que a automação de uma tarefa seja bem-sucedida é necessário que a máquina que passará a realizá-la seja capaz de desempenhar cada uma das etapas constituintes do processo a ser automatizado com eficiência, de modo a garantir a **repetibilidade** do mesmo. Assim, é necessário que seja especificado com clareza e exatidão o que deve ser realizado em cada uma das fases do processo a ser automatizado, bem como a seqüência em que estas fases devem ser realizadas.

À especificação da seqüência ordenada de passos que deve ser seguida para a realização de uma tarefa, garantindo a sua repetibilidade, dá-se o nome de **algoritmo**.

Ao contrário do que se pode pensar, o conceito de algoritmo **não** foi criado para satisfazer às necessidades da computação. Pelo contrário, a programação de computadores é apenas um dos campos de aplicação dos algoritmos. Na verdade, há inúmeros casos que podem exemplificar o uso (involuntário ou não) de algoritmos para a padronização do exercício de tarefas rotineiras (vide exemplos da Seção 2.1). No entanto, daqui adiante a atenção desta apostila estará voltada à automação de tarefas utilizando computadores.

Para que um computador possa desempenhar uma tarefa é necessário que esta seja detalhada passo-a-passo, numa forma compreensível pela máquina, utilizando aquilo que se chama de **programa**. Neste sentido, um programa de computador nada mais é que um algoritmo escrito numa forma compreensível pelo computador (linguagem de programação).

1.1 Algumas Definições de Algoritmo

"Serve como modelo para programas, pois sua linguagem é intermediária à linguagem humana e às linguagens de programação, sendo então, uma boa ferramenta na validação da lógica de tarefas a serem automatizadas."

“Os algoritmos, servem para representar a solução de qualquer problema, mas no caso do Processamento de Dados, eles devem seguir as regras básicas de programação para que sejam compatíveis com as linguagens de programação.”

Capítulo 2

Formas de Representação de Algoritmos

Existem diversas formas de representação de algoritmos, mas não há um consenso com relação à melhor delas.

O critério usado para classificar hierarquicamente estas formas está diretamente ligado ao nível de detalhe ou, inversamente, ao grau de abstração oferecido.

Algumas formas de representação de algoritmos tratam os problemas apenas em nível lógico, abstraindo-se de detalhes de implementação muitas vezes relacionados com alguma linguagem de programação específica. Por outro lado existem formas de representação de algoritmos que possuem uma maior riqueza de detalhes e muitas vezes acabam por obscurecer as idéias principais do algoritmo, dificultando seu entendimento.

Dentre as formas de representação de algoritmos mais conhecidas podemos citar:

- **Descrição Narrativa**;
- **Fluxograma Convencional**;
- **Pseudocódigo**, também conhecido como Linguagem Estruturada ou Portugol.

2.1 Descrição Narrativa

Nesta forma de representação os algoritmos são expressos diretamente em **linguagem natural**. Como exemplo, têm-se os algoritmos seguintes:

- *Receita de bolo:*

  Misture os ingredientes
  Unte a forma com manteiga
  Despeje a mistura na forma
  Se houver coco ralado
  então despeje sobre a mistura
  Leve a forma ao forno
  Enquanto não corar
  deixe a forma no forno
  Retire do forno
  Deixe esfriar

- *Troca de um pneu furado:*

  Afrouxar ligeiramente as porcas
  Suspender o carro
  Retirar as porcas e o pneu
  Colocar o pneu reserva
  Apertar as porcas
  Abaixar o carro
  Dar o aperto final nas porcas

- *Tomando um banho:*

  Entrar no banheiro e tirar a roupa
  Abrir a torneira do chuveiro
  Entrar na água
  Ensaboar-se
  Sair da água
  Fechar a torneira
  Enxugar-se
  Vestir-se

- *Cálculo da média de um aluno:*

  Obter as suas 2 notas de provas
  Calcular a média aritmética
  Se a média for maior que 7,
  o aluno foi aprovado,
  senão ele foi reprovado

Esta representação é pouco usada na prática porque o uso da linguagem natural muitas vezes dá oportunidade a más interpretações, ambigüidades e imprecisões.

Por exemplo, a instrução "afrouxar ligeiramente as porcas" no algoritmo da troca de pneus está sujeita a interpretações diferentes por pessoas distintas. Uma instrução mais precisa seria: "afrouxar a porca, girando-a 30º no sentido anti-horário".

2.2 Fluxograma Convencional

É uma representação gráfica de algoritmos onde formas geométricas diferentes implicam ações (instruções, comandos) distintos. Tal propriedade facilita o entendimento das idéias contidas nos algoritmos e justifica sua popularidade.

Esta forma é aproximadamente intermediária à descrição narrativa e ao pseudocódigo (subitem seguinte), pois é menos imprecisa que a primeira e, no entanto, não se preocupa com detalhes de implementação do programa, como o tipo das variáveis usadas.

Nota-se que os fluxogramas convencionais preocupam-se com detalhes de nível físico da implementação do algoritmo. Por exemplo, figuras geométricas diferentes são adotadas para representar operações de saída de dados realizadas em dispositivos distintos, como uma fita magnética ou um monitor de vídeo. Como esta apostila não está interessada em detalhes físicos da implementação, mas tão somente com o nível lógico das instruções do algoritmo, será adotada a notação simplificada da Figura 2.1 para os fluxogramas. De qualquer modo, o Apêndice A contém uma tabela com os símbolos mais comuns nos fluxogramas convencionais.

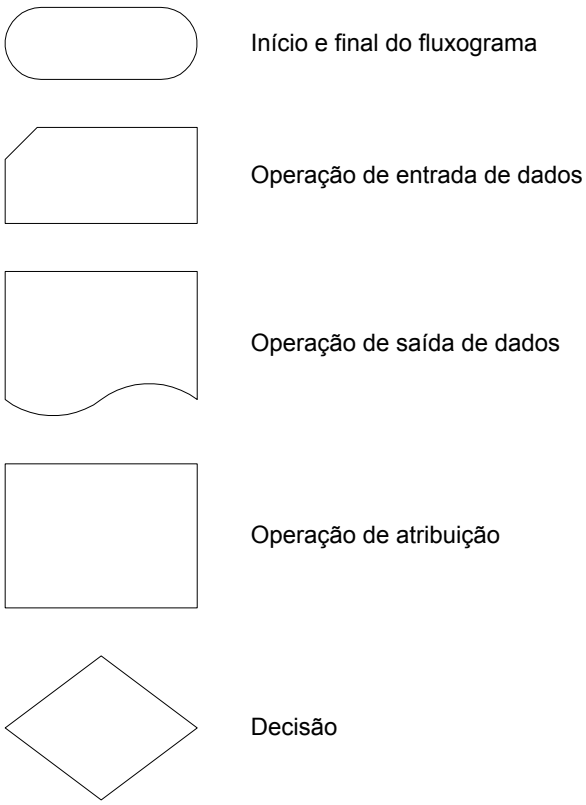


Figura 2.1 Principais formas geométricas usadas em fluxogramas.

De modo geral, um fluxograma se resume a um único símbolo inicial por onde a execução do algoritmo começa, e um ou mais símbolos finais, que são pontos onde a execução do algoritmo se encerra. Partindo do símbolo inicial, há sempre um único caminho orientado a ser seguido, representando a existência de uma única seqüência de execução das instruções. Isto pode ser melhor visualizado pelo fato de que, apesar de vários caminhos poderem convergir para uma mesma figura do diagrama, há sempre um único caminho saindo desta. Exceções a esta regra são os símbolos finais, dos quais não há nenhum fluxo saindo, e os símbolos de decisão, de onde pode haver mais de um caminho de saída (usualmente dois caminhos), representando uma bifurcação no fluxo.

A Figura 2.2 mostra a representação do algoritmo de cálculo da média de um aluno sob a forma de um fluxograma.

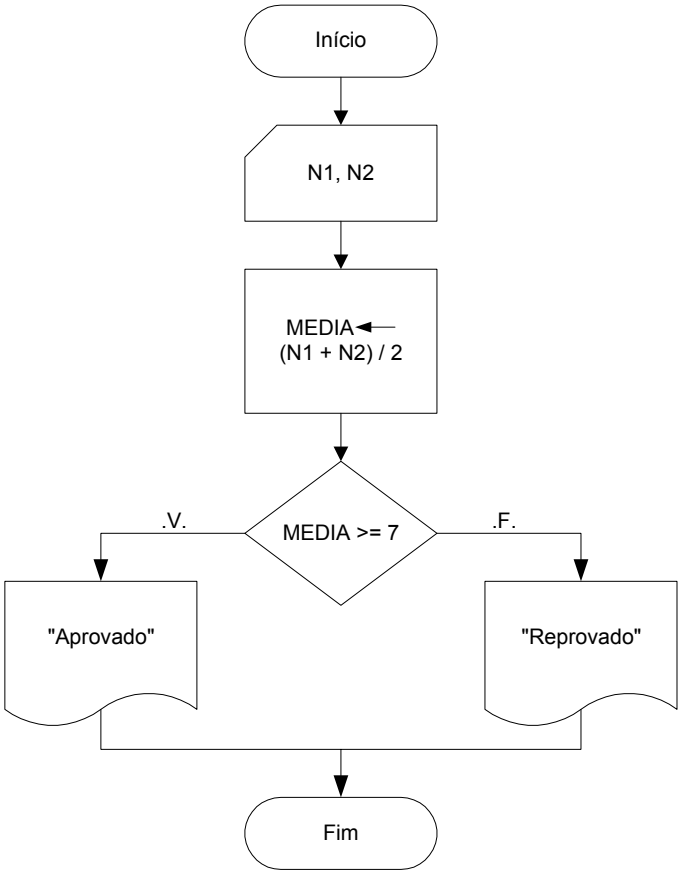


Figura 2.2 Exemplo de um fluxograma convencional.

2.3 Pseudocódigo

Esta forma de representação de algoritmos é rica em detalhes, como a definição dos tipos das variáveis usadas no algoritmo. Por assemelhar-se bastante à forma em que os programas são escritos, encontra muita aceitação.

Na verdade, esta representação é suficientemente geral para permitir a tradução de um algoritmo nela representado para uma linguagem de programação específica seja praticamente direta.

A forma geral da representação de um algoritmo na forma de pseudocódigo é a seguinte:

```
Algoritmo <nome_do_algoritmo>
        <declaração_de_variáveis>
        <subalgoritmos>
Início
        <corpo do algoritmo>
Fim
```

Algoritmo é uma palavra que indica o início da definição de um algoritmo em forma de pseudocódigo.

<nome_do_algoritmo> é um nome simbólico dado ao algoritmo com a finalidade de distingui-los dos demais.

<declaração_de_variáveis> consiste em uma porção opcional onde são declaradas as variáveis globais usadas no algoritmo principal e, eventualmente, nos subalgoritmos.

<subalgoritmos> consiste de uma porção opcional do pseudocódigo onde são definidos os subalgoritmos (Capítulo 8).

Início e **Fim** são respectivamente as palavras que delimitam o início e o término do conjunto de instruções do corpo do algoritmo.

Como exemplo, a Figura 2.3 mostra a representação do algoritmo do cálculo da média de um aluno, na forma de um pseudocódigo.

```
Algoritmo Calculo_Media
Var N1, N2, MEDIA: real
Início
Leia N1, N2
MEDIA ← (N1 + N2) / 2
Se MEDIA >= 7 então
    Escreva "Aprovado"
Senão
    Escreva "Reprovado"
Fim_se
Fim
```

Figura 2.3 Exemplo de um pseudocódigo.

2.4 Síntese

Há diversas formas de representação de algoritmos que diferem entre si pela quantidade de detalhes de implementação que fornecem ou, inversamente, pelo grau de abstração que possibilitam com relação à implementação do algoritmo em termos de uma linguagem de programação específica.

Dentre as principais formas de representação de algoritmos destacam-se: a **descrição narrativa**, o **fluxograma convencional** e o **pseudocódigo** (ou linguagem estruturada).

Capítulo 3

Tipos de Dados

Todo o trabalho realizado por um computador é baseado na manipulação das informações contidas em sua memória. Grosso modo, estas informações podem ser classificadas em dois tipos:

- As **instruções**, que comandam o funcionamento da máquina e determinam a maneira como devem ser tratados os dados. As instruções são específicas para cada modelo de computador, pois são funções do tipo particular de processador utilizado em sua implementação.

- Os **dados** propriamente ditos, que correspondem à porção das informações a serem processadas pelo computador.

A maior parte das pessoas não ligadas à área de informática ignora o potencial dos computadores e imagina que eles são capazes de tratar apenas com dados numéricos. Na realidade, a capacidade dos mesmos se estende a outros tipos de dados.

O objetivo deste capítulo é justamente o de classificar os dados de acordo com o tipo de informação contida neles. A classificação apresentada não se aplica a nenhuma linguagem de programação específica; pelo contrário, ela sintetiza os padrões utilizados na maioria das linguagens.

3.1 Dados Numéricos

Antes de apresentar formalmente os tipos de dados numéricos, é conveniente recordar alguns conceitos básicos relacionados à teoria dos números e conjuntos.

O conjunto dos números **naturais** é representado por **N** e é dado por:

N = {1, 2, 3, 4, ...}

Algumas correntes de matemáticos teóricos convencionam que o número 0 está contido neste conjunto; contudo, não convém perder tempo em tais discussões filosóficas, uma vez que isto não influenciará de forma alguma este estudo.

Na seqüência, encontramos o conjunto dos números **inteiros**:

Z = {..., -3, -2, -1, 0, 1, 2, 3, ...}

O conjunto **Z** contém todos os elementos de **N**, bem como alguns números que não pertencem a **N** (os números negativos e o zero). Portanto, dizemos que **N** está contido em **Z**, ou então, que **Z** contém **N**.

Englobando o conjunto dos números inteiros, existe o conjunto dos números **fracionários** (**Q**), dado pelo universo dos números que podem ser expressos na forma de uma fração, isto é, um quociente onde o numerador e o denominador são números inteiros. Mais formalmente:

Q = (p/q | p, q pertencem a **Z**}

Por último, surge o conjunto dos números **reais** (**R**), formado pela união do conjunto dos números fracionários **Q** com o conjunto dos números que não podem ser expressos na forma de uma fração (os números irracionais). Ex.: 2 = 1.1412... , PI = 3.14159...

3.1.1 Dados Numéricos Inteiros

Os números **inteiros** são aqueles que não possuem componentes decimais ou fracionários, podendo ser positivos ou negativos.

Os elementos pertencentes aos conjuntos **N** e **Z**, apesar de serem representáveis na classe dos números reais, são classificados como dados do tipo **inteiro**, por não possuírem parte fracionária. Esta possibilidade é interessante por permitir uma economia do espaço de memória, como veremos adiante.

Por sua vez, os elementos dos conjuntos **Q** e **R**, por possuírem parte fracionária, **não** podem ser representados na classe inteira, pertencendo necessariamente aos tipos de dados ditos **reais**.

Como exemplos de números **inteiros** temos:

24 - número inteiro positivo

0 - número inteiro

-12 - número inteiro negativo

3.1.2 Dados Numéricos Reais

Os dados de tipo **real** são aqueles que podem possuir componentes decimais ou fracionários, e podem também ser positivos ou negativos.

Como dito anteriormente, os elementos dos conjuntos de números **fracionários** e **reais** são necessariamente representados no computador por dados do tipo **real**.

Exemplos de dados do tipo **real**:

24.01 - número real positivo com duas casas decimais

144. - número real positivo com zero casas decimais

-13.3 - número real negativo com uma casa decimal

0.0 - número real com uma casa decimal

0. - número real com zero casas decimais

Observe que há uma diferença entre “0”, que é um dado do tipo inteiro “0.” (ou “0.0”) que é um dado do tipo real. Portanto, a simples existência do ponto decimal serve para diferenciar um dado numérico do tipo **inteiro** de um do tipo **real**.

3.2 Dados Literais

O tipo de dados **literal** é constituído por uma seqüência de caracteres contendo letras, dígitos e/ou símbolos especiais. Este tipo de dados é também muitas vezes chamado de **alfanumérico**, **cadeia** (ou **cordão**) **de caracteres**, ainda, do inglês, **string**.

Usualmente, os dados literais são representados nos algoritmos pela coleção de caracteres, delimitada em seu início e término com o caractere aspas (**"**).

Diz-se que o dado do tipo literal possui um comprimento dado pelo número de caracteres nele contido.

Exemplos de dados do tipo literal:

"QUAL ?" - literal de comprimento 6

" " - literal de comprimento 1

"qUaL ?!$" - literal de comprimento 8

" AbCdefGHi" - literal de comprimento 9

"1-2+3=" - literal de comprimento 6

“0” - literal de comprimento 1

Note que, por exemplo, "1.2" representa um dado do tipo **literal** de comprimento 3, constituído pelos caracteres "1", "." e "2", diferindo de 1.2 que é um dado do tipo **real**.

3.3 Dados Lógicos

A existência deste tipo de dado é, de certo modo, um reflexo da maneira como os computadores funcionam. Muitas vezes, estes tipos de dados são chamados de **booleanos**, devido à significativa contribuição de BOOLE à área da lógica matemática.

O tipo de dados **lógico** é usado para representar dois únicos valores lógicos possíveis: **verdadeiro** e **falso**. É comum encontrar-se em outras referências outros tipos de pares de valores lógicos como **sim/não**, **1/0**, **true/false**.

Nos algoritmos apresentados nesta apostila os valores lógicos serão delimitados pelo caractere ponto (**.**).

Exemplo:

.V. - valor lógico verdadeiro

.F. - valor lógico falso

3.4 Síntese

Os dados numéricos dividem-se em duas classes:

- **inteiros**, que não possuem parte fracionária e podem ser positivos ou negativos;
- **reais**, que podem possuir parte fracionária e podem ser positivos ou negativos.

Os dados do tipo **literal** podem conter seqüências de letras, dígitos ou símbolos especiais, delimitados por aspas (**"**). Seu comprimento é dado pelo número de caracteres em **string**.

Os dados do tipo **lógico** só possuem dois valores possíveis (**.V.** e **.F.**).

A árvore abaixo resume a classificação dos dados com relação aos tipos de dados apresentados.

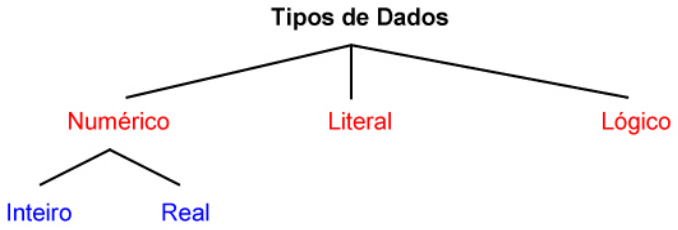


Figura 3.2 Representação dos diversos tipos de dados

3.5 Exercício Proposto

1. Classifique os dados especificados abaixo de acordo com seu tipo, assinalando com **I** os dados do tipo inteiro, com **R** os reais, com **L** os literais, com **B** os lógicos (booleanos), e com **N** aqueles para os quais não é possível definir *a priori* um tipo de dado.

( ) 0.21	( ) .T.	( ) .V.
( ) 1	( ) +3257	( ) .V
( ) V	( ) “a”	( ) “abc”
( ) “0.”	( ) “+3257”	( ) F
( ) 1%	( ) +3257.	( ) C
( ) “José”	( ) “-0.0”	( ) Maria
( ) 0,35	( ) “.F.”	( ) +36
( ) .F.	( ) ± 3	
( ) -0.001		

Capítulo 4

Variáveis

A todo momento durante a execução de qualquer tipo de programa os computadores estão manipulando informações representadas pelos diferentes tipos de dados descritos no capítulo anterior. Para que não se "esqueça" das informações, o computador precisa guardá-las em sua memória.

Este capítulo é destinado ao estudo da forma como os computadores armazenam e acessam informações contidas em sua memória.

4.2 Armazenamento de Dados na Memória

Cada um dos diversos tipos de dados apresentados no capítulo anterior necessita de uma certa quantidade de memória para armazenar a informação representada por eles.

Esta quantidade é função do tipo de dado considerado, do tipo da máquina (computador) e do tipo de linguagem de programação. Por isso, o que será exposto nos subitens seguintes não deve ser tomado como padrão, apenas como exemplo.

4.2.1 Armazenamento de Dados do Tipo Literal

Devemos sempre ter em mente que um byte consegue representar 256 (2^8) possibilidades diferentes.

Uma informação do tipo literal nada mais é do que um conjunto de caracteres que podem ser letras, dígitos ou símbolos especiais.

A união de todos os caracteres existentes nos computadores resulta num conjunto com um número de elementos menor que 256. Deste resultado surgiu a idéia de associar a cada caractere um número (código) variando de 0 a 255 (256 possibilidades). No princípio, cada fabricante de computador adotava uma convenção diferente para este código. Mais recentemente, esta convenção foi padronizada a fim de facilitar a portabilidade (migração) de programas entre máquinas diferentes. Esta convenção é representada na forma de uma tabela de mapeamento de caracteres em números. O padrão mais universalmente aceito é o **ASCII**, cuja tabela é mostrada no Apêndice B.

Assim, cada célula de memória (byte) pode conter um caractere, representado pelo seu código **ASCII**.

Retornando à questão do armazenamento de informações do tipo literal na memória, deve-se lembrar que um dado deste tipo possui um certo comprimento dado pelo número de caracteres nele contido. Portanto, para guardar um dado do tipo literal devemos alocar (reservar) um espaço contíguo de memória igual ao comprimento do mesmo, destinando um byte para cada caractere da informação.

Exemplificando, a informação do tipo literal "banana" possui seis caracteres e, portanto, seis bytes são necessários para reter a referida informação na memória. A princípio, estes bytes podem estar em qualquer lugar da memória, mas é conveniente que estejam juntos (posições contíguas). A primeira posição deste conjunto de bytes é absolutamente arbitrária e sua escolha geralmente é feita automaticamente pelo compilador (isto é, pelo programa que

traduz um outro escrito em alguma linguagem de programação para outra geral, a linguagem de máquina do computador com que se trabalha).

A Figura 4.3 mostra o caso em que se armazena a **literal** "banana" no conjunto de seis bytes contíguos de memória iniciando pela posição de memória 0. Na verdade, ao invés dos caracteres da **literal**, os códigos correspondentes aos mesmos é que são guardados na memória.

Endereço	Informação
0	b (98)
1	a (97)
2	n (110)
3	a (97)
4	n (110)
5	a (97)

Figura 4.3 Armazenamento da literal “banana” na memória de um computador.

4.2.2 Armazenamento de Dados do Tipo Lógico

Uma informação do tipo lógico só possui dois valores possíveis: **.V.** ou **.F.**. Assim, a princípio, um único bit seria suficiente para armazenar uma informação deste tipo. Contudo, deve-se lembrar que a menor porção de memória que se pode acessar é o byte. Portanto, uma informação do tipo lógico é armazenada em um byte de memória. De certa forma, se por um lado isto pode ser como um "desperdício" de memória, por outro simplifica bastante a arquitetura de memória dos computadores (por motivos que fogem ao contexto desta apostila). Além do mais, isto não é tão relevante, uma vez que na prática o número de ocorrências de dados do tipo lógico é bastante inferior ao de ocorrências de dados do tipo literal ou numérico.

4.2.3 Armazenamento de Dados do Tipo Inteiro

O conjunto dos números inteiros (**Z**) contém um número infinito de elementos:

Z = { $-\infty$, ..., -3, -2, -1, 0, 1, 2, 3, ..., $+\infty$}

Obviamente é inviável o armazenamento de todos os números deste conjunto num computador. Faz-se necessário realizar um estudo para que se limite o número de elementos representáveis deste conjunto.

Se apenas um byte fosse utilizado para armazenar os dados do tipo inteiro, existiriam apenas 256 números diferentes neste conjunto:

{-127, -126, ..., -2, -1, 0, 1, 2, ..., 127, 128)

Esta restrição é bastante forte, uma vez que boa parte das aplicações práticas necessitam de números inteiros maiores que estes.

Se forem utilizados dois bytes para armazenar um número inteiro, o universo de números representáveis cresce para $2^8 \times 2^8 = 2^{16} = 65.536$ possibilidades:

{-32767, -32766, ..., -2, -1, 0, 1, 2, ..., 32767, 32768}

Este conjunto satisfaz à grande maioria das necessidades práticas. Assim, em geral utilizam-se dois bytes para representar os números inteiros em computadores. Contudo, restam algumas aplicações muito específicas em que se precisa de um conjunto ainda maior. Para

estes casos, algumas linguagens de programação fornecem mecanismos para trabalhar números inteiros com quatro bytes. Nestes casos os dados são ditos **inteiros longos** ou **estendidos**.

4.2.4 Armazenamento de Dados do Tipo Real

O conjunto dos números **reais** (**R**) contém um número infinito de elementos e, pelas mesmas razões que o conjunto dos números **inteiros**, precisa ser limitado.

Para dados deste tipo julgou-se apropriado adotar quatro bytes para sua representação interna nos computadores.

São muito comuns situações como as aplicações científicas em que é necessária uma maior precisão de cálculo, intimamente ligada ao número de casas decimais dos dados. Para este caso, em analogia com o que acontece com os dados do tipo inteiro, algumas linguagens de programação decidiram criar dados do tipo **real estendido** (com oito bytes).

4.3 Conceito e Utilidade de Variáveis

Como visto anteriormente, informações correspondentes a diversos tipos de dados são armazenadas na memória dos computadores. Para acessar individualmente cada uma destas informações, a princípio, seria necessário saber o tipo de dado desta informação (ou seja, o número de bytes de memória por ela ocupados) e a posição inicial deste conjunto de bytes na memória.

Percebe-se que esta sistemática de acesso a informações na memória é bastante ilegível e difícil de se trabalhar. Para contornar esta situação criou-se o conceito de **variável**, que é uma entidade destinada a guardar uma informação.

Basicamente, uma variável possui três atributos: um **nome**, um **tipo de dado** associado à mesma e a **informação** por ela guardada.

Figura 4.4 Atributos de uma variável.

Toda variável possui um **nome** que tem a função de diferenciá-la das demais. Cada linguagem de programação estabelece suas próprias regras de formação de nomes de variáveis. Adotaremos nesta apostila as seguintes regras:

- um nome de variável deve necessariamente começar com uma letra;
- um nome de variável não deve conter nenhum símbolo especial exceto a sublinha (_).

Exemplos:

SALARIO	=	correto
1ANO	=	errado (não começou com uma letra)
ANO1	=	correto
A CASA	=	errado (contém o caractere espaço em branco)
SAL/HORA	=	errado (contém o caractere "/")
SAL_HORA	=	correto
_DESCONTO	=	errado (não começou com uma letra)

4.4 Definição de Variáveis em Algoritmos

Todas as variáveis utilizadas em algoritmos devem ser definidas antes de serem utilizadas. Isto se faz necessário para permitir que o compilador reserve um espaço na memória para as mesmas.

Nos algoritmos apresentados nesta apostila será adotada a seguinte convenção:

- todas as variáveis utilizadas em algoritmos serão definidas no início do mesmo, por meio de um comando de uma das formas seguintes:

  VAR <nome_da_variável> : <tipo_da_variável>

  VAR <lista_de_variáveis> : <tipo_das_variáveis>

- a palavra-chave **VAR** deverá estar presente sempre e será utilizada uma única vez na definição de um conjunto de uma ou mais variáveis;

- numa mesma linha poderão ser definidas uma ou mais variáveis do mesmo tipo. Para tal, deve-se separar os nomes das mesmas por vírgulas;

- variáveis de tipos diferentes devem ser declaradas em linhas diferentes.

A forma de utilização deste comando ficará mais clara quando da utilização da representação de algoritmos em linguagem estruturada (pseudocódigo).

Esta convenção é válida para a representação de algoritmos na forma de pseudocódigo. Em termos de fluxograma, não é usual adotar-se qualquer forma de definição de variáveis.

Exemplo de definição de variáveis:

VAR	NOME	:	**literal[**10**]**
	IDADE	:	**inteiro**
	SALARIO	:	**real**
	TEM_FILHOS	:	**lógico**

No exemplo acima foram declaradas quatro variáveis:

- a variável **NOME**, capaz de armazenar dados literais de comprimento 10 (dez caracteres);

- a variável **IDADE**, capaz de armazenar um número inteiro;

- a variável **SALARIO**, capaz de armazenar um número real;

- a variável **TEM_FILHOS**, capaz de armazenar uma informação lógica.

4.5 Síntese

A memória dos computadores é composta por células numeradas ordenadamente denominadas **bytes**. Cada byte é constituído por 8 **bits**.

Cada tipo de dado requer um número diferente de bytes para armazenar a informação representada por ele na memória. Esta quantidade também pode variar em função do tipo de computador considerado.

Uma **variável** é uma entidade dotada de um **nome** para diferenciá-la das demais e um **tipo de dado** que define o tipo de informação que ela é capaz de guardar. Uma vez definidos, o nome e o tipo de uma variável não podem ser alterados no decorrer de um programa. Por outro lado, a informação útil da variável é objeto de constante modificação durante o decorrer do programa, de acordo com o fluxo de execução do mesmo.

4.6 Exercícios Propostos

1. Assinale com **C** os identificadores corretos e com **I** os incorretos. Explique o que está errado nos identificadores incorretos.

( ) valor	( ) km/h
( ) _b248	( ) xyz
( ) nota*do*aluno	( ) nome empresa
( ) a1b2c3	( ) sala_215
( ) 3 x 4	( ) “nota”
( ) Maria	( ) ah!

2. Supondo que as variáveis NB, NA, NMAT e SX sejam utilizadas para armazenar a nota do aluno, o nome do aluno, o número da matrícula e o sexo, declare-as corretamente, associando o tipo adequado ao dado que será armazenado.

Capítulo 5

Expressões

5.1 Conceito

O conceito de **expressão** em termos computacionais está intimamente ligado ao conceito de expressão (ou fórmula) matemática, onde um conjunto de variáveis e constantes numéricas relacionam-se por meio de operadores aritméticos compondo uma fórmula que, uma vez avaliada, resulta num valor.

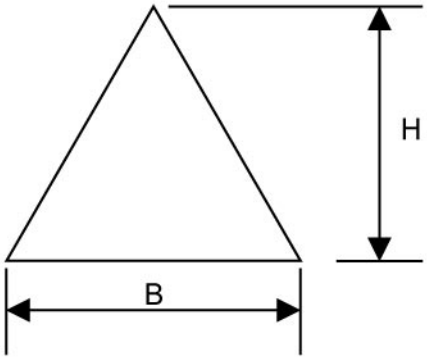


Figura 5.1 Triângulo de base (**B**) e altura (**H**).

Por exemplo, a fórmula de cálculo da área do triângulo da Figura 5.1 é dada por:

AREA = 0.5 x B x H

Esta fórmula utiliza três variáveis: **B** e **H**, que contêm as dimensões do triângulo, e **AREA**, onde é guardado o valor calculado (resultado da avaliação da expressão). Há, também, uma constante (**0.5**) e o operador de multiplicação (**x**), que aparece duas vezes na expressão.

O conceito de **expressão** aplicado à computação assume uma conotação mais ampla: uma **expressão** é uma combinação de variáveis, constantes e operadores, e que, uma vez avaliada, resulta num valor.

5.2 Operadores

Operadores são elementos funcionais que atuam sobre **operandos** e produzem um determinado resultado. Por exemplo, a expressão **3 + 2** relaciona dois operandos (os números 3 e 2) por meio do operador (**+**) que representa a operação de adição.

De acordo com o número de operandos sobre os quais os operadores atuam, os últimos podem ser classificados em:

- **binários**, quando atuam sobre dois operandos. Ex.: os operadores das operações aritméticas básicas (soma, subtração, multiplicação e divisão);

- **unários**, quando atuam sobre um único operando. Ex.: o sinal de (**-**) na frente de um número, cuja função é inverter seu sinal.

Outra classificação dos operadores é feita considerando-se o tipo de dado de seus operandos e do valor resultante de sua avaliação. Segundo esta classificação, os operadores dividem-se em **aritméticos**, **lógicos** e **literais**. Esta divisão está diretamente relacionada com o tipo de expressão onde aparecem os operadores.

Um caso especial é o dos operadores **relacionais**, que permitem comparar pares de operandos de tipos de dados iguais, resultando sempre num valor lógico.

Mais adiante serão apresentados os operadores dos diversos tipos acima relacionados.

5.3 Tipos de Expressões

As expressões são classificadas de acordo com o tipo do valor resultante de sua avaliação.

5.3.1 Expressões Aritméticas

Expressões **aritméticas** são aquelas cujo resultado da avaliação é do tipo numérico, seja ele inteiro ou real. Somente o uso de operadores aritméticos e variáveis numéricas é permitido em expressões deste tipo.

Os operadores aritméticos relacionados às operações aritméticas básicas estão sumarizados na Tabela 5.1.

Tabela 5.1 Operadores aritméticos e sua ordem de prioridade.

Operador	Tipo	Operação	Prioridade
+	Binário	Adição	4
-	Binário	Subtração	4
*	Binário	Multiplicação	3
/	Binário	Divisão	3
**	Binário	Exponenciação	2
+	Unário	Manutenção de sinal	1
-	Unário	Inversão de sinal	1

A prioridade entre operadores define a ordem em que os mesmos devem ser avaliados dentro de uma mesma expressão. Este assunto será tratado com maior profundidade numa seção posterior.

O caractere (*****) é adotado na maioria das linguagens de programação para representar a operação de multiplicação, ao invés do caractere (**x**), devido à possibilidade da ocorrência do mesmo no nome de variáveis. Pela mesma razão, o símbolo (******) é adotado para representar a operação de exponenciação. Algumas linguagens de programação adotam o símbolo (**^** - circunflexo) para esta finalidade, mas isto é pouco freqüente.

As variáveis usadas em expressões aritméticas podem somente ser do tipo inteiro ou real. Se todas as variáveis que aparecem numa expressão são do tipo inteiro, então o valor resultante da expressão é também do tipo inteiro. Se ao menos uma das variáveis da expressão aritmética for do tipo real, então o valor resultante da avaliação da expressão é necessariamente do tipo real.

Nos exemplos seguintes, assumiremos que:

- **A**, **B** e **C** são variáveis do tipo inteiro;
- **X**, **Y** e **Z** são variáveis do tipo real.

Exemplos:

A + B * C = expressão de resultado inteiro
A + B + Y = expressão de resultado real
A / B = expressão de resultado real
X / Y = expressão de resultado real

5.3.2 Expressões Lógicas

Expressões **lógicas** são aquelas cujo resultado da avaliação é um valor lógico (**.V.** ou **.F.**).

Os operadores lógicos e suas relações de precedência são mostrados na Tabela 5.2.

Existem outros operadores lógicos, como por exemplo o **OU_EXCLUSIVO.**, mas suas funções podem ser exercidas por combinações dos três tipos de operadores da Tabela 5.2.

Tabela 5.2 Operadores lógicos e suas relações de prioridade.

Operador	Tipo	Operação	Prioridade
.OU.	Binário	Disjunção	3
.E.	Binário	Conjunção	2
.NÃO.	Unário	Negação	1

Para exemplificar o uso de operadores lógicos, a Tabela 5.3 apresenta duas variáveis lógicas **A** e **B**. Uma vez que cada variável lógica possui somente dois valores possíveis, então há exatamente quatro combinações para estes valores, razão pela qual a tabela tem quatro linhas. As diversas colunas contêm os resultados das operações lógicas sobre as combinações possíveis dos valores das variáveis **A** e **B**.

Tabela 5.3 Tabela-Verdade dos operadores apresentados na Tabela 5.2.

A	B	.NÃO. A	.NÃO. B	A .OU. B	A .E. B
.F.	.F.	.V.	.V.	.F.	.F.
.F.	.V.	.V.	.F.	.V.	.F.
.V.	.F.	.F.	.V.	.V.	.F.
.V.	.V.	.F.	.F.	.V.	.V.

Tabelas como a da Figura 5.3 são chamadas de **Tabelas-Verdade**. Convém salientar as seguintes conclusões que podem ser extraídas por observação da Tabela 5.3:

- O operador lógico **.NÃO.** sempre inverte o valor de seu operando. Ex.: **.NÃO. .V. = .F.** e **.NÃO. .F. = .V.**;
- Para que a operação lógica **.OU.** tenha resultado verdadeiro basta que um de seus operandos seja verdadeiro; Para melhor visualizar este efeito, podemos imaginar que as variáveis lógicas **A** e **B** são como dois interruptores ligados em paralelo num circuito de acionamento de uma lâmpada (Figura 5.2).

Nas expressões lógicas onde aparecem apenas os operadores lógicos da Tabela 5.2 somente variáveis do tipo lógico podem ser usadas. Isto parece óbvio, uma vez que os operadores lógicos somente atuam sobre valores (constantes ou variáveis) lógicos.

Há, ainda, outro tipo de operador que pode aparecer em operações lógicas: os operadores **relacionais**, mostrados na Tabela 5.4.

Tabela 5.4 Operadores relacionais.

Operador	Operação
=	Igual
<>	Diferente
<	Menor
<=	Menor ou igual
>	Maior
>=	Maior ou igual

Estes operadores são somente usados quando se deseja efetuar comparações. Comparações só podem ser feitas entre objetos de mesma natureza, isto é, variáveis do mesmo tipo de dado. O resultado de uma comparação é sempre um valor lógico.

O uso de operadores relacionais possibilita o aparecimento em expressões lógicas de variáveis de outros tipos de dados que não o lógico.

5.3.3 Expressões Literais

Expressões **literais** são aquelas cujo resultado da avaliação é um valor literal. Este tipo de expressão é bem menos freqüente que os anteriores. Os tipos de operadores existentes variam de uma linguagem de programação para outra, não havendo uma padronização.

Para que o assunto não passe em branco, considere-se como exemplo a operação de concatenação de **strings**: toma-se duas **strings** e acrescenta-se (concatena-se) a segunda delas ao final da primeira. Em algumas linguagens esta operação é representada pelo símbolo (+). Por exemplo, a concatenação das **strings** "**REFRIGERA**" e "**DOR**" é representada por "**REFRIGERA**" + "**DOR**" e o resultado de sua avaliação é "**REFRIGERADOR**".

5.4 Síntese

Uma **expressão** é uma combinação de variáveis, constantes e operadores, que resulta num valor quando avaliada.

Operadores são elementos funcionais que atuam sobre **operandos**. Segundo o número de operandos sobre os quais atua, um operador pode ser classificado em **unário** ou **binário**. Segundo os tipos de dados de seus operandos e do valor resultante de sua avaliação, os operadores podem ser classificados em aritméticos, lógicos ou literais.

Um tipo especial de operador é o **relacional**, que é usado na comparação de operandos de um mesmo tipo de dado e cujo resultado da avaliação é sempre um valor lógico.

As expressões são classificadas de acordo com o valor resultante de sua avaliação em:

- **Aritméticas**, que resultam num valor numérico (real ou inteiro);

- **Lógicas**, que resultam num valor lógico;
- **Literais**, que resultam num valor literal.

Há três regras básicas que definem a seqüência correta de avaliação passo a passo de expressões:

1. Operadores de maior prioridade devem ser avaliados primeiro. Em caso de empate, a avaliação se faz da esquerda para a direita.
2. O uso de parênteses em subexpressões força a avaliação das mesmas com maior prioridade.
3. Os diversos tipos d operadores devem ser avaliados na seguinte seqüência dentro de uma expressão complexa: primeiro os aritméticos e literais; a seguir, os relacionais e, por último, os lógicos.

5.6 Exercícios Propostos

1. Dada a declaração de variáveis:

```
VAR   A, B, C       :  inteiro
      X, Y, Z       :  real
      NOME, RUA     :  literal[20]
      L1, L2        :  lógico
```

Classifique as expressões seguintes de acordo com o tipo de dado do resultado de sua avaliação, em **I** (inteiro), **R** (real), **L** (literal), **B** (lógico) ou **N** (quando não for possível defini-lo):

( ) A + B + C	( ) L1 .OU. L2	( ) (A = B)
( ) A + B + Z	( ) RUA <> NOME	( ) X + Y / Z
( ) NOME + RUA	( ) A + B / C	( ) X = Z / A
( ) A B	( ) A + X / Z	( ) L1 ** L2
( ) A Y	( ) A + Z / A	( ) A + B / L2
( ) NOME RUA	( ) A B = L1	( ) X < L1 / RUA

2. Para as mesmas variáveis declaradas no exercício 1, às quais são dados os valores seguintes:

A = 1	**Z** = -1.0
B = 2	**L1** = .V.
C = 3	**NOME** = “PEDRO”
X = 2.0	**RUA** = “PEDRINHO”
Y = 10.0	**L2** = .F.

Determine o resultado da avaliação das expressões abaixo:

A + C / B → _____

A + B + C → _____

C / B / A → _____

-X ** B → _____

- (X ** B) → _____

(-X) ** B → _____

NOME + RUA → _____

NOME = RUA → _____

L1 .OU. L2 → _____

(L1 .E.(.NÃO. L2)) → _____

(L2 .E. (.NÃO. L1)) → _____

(L1 .E. (.NÃO. L2)) .OU. (L2 .E. (.NÃO. L1)) → _____

_____

X Y .E. C = B → _____

_____

(C - 3 * A) (X + 2 * Z) → _____

_____

Capítulo 6

Instruções Primitivas

Como o próprio nome diz, **instruções primitivas** são os comandos básicos que efetuam tarefas essenciais para a operação dos computadores, como entrada e saída de dados (comunicação com o usuário e com os dispositivos periféricos), e movimentação dos mesmos na memória. Estes tipos de instrução estão presentes na absoluta maioria das linguagens de programação. De fato, um programa que não utiliza nenhuma instrução primitiva - como as que serão definidas neste capítulo - é incapaz de se comunicar com o mundo exterior e, portanto, não tem utilidade alguma.

Antes de passar à descrição das instruções primitivas, é necessária a definição de alguns termos que serão utilizados mais à frente:

- **Dispositivo de entrada** é o meio pelo qual as informações (mais especificamente os dados) são transferidas pelo usuário ou pelos níveis secundários de memória ao computador. Os exemplos mais comuns são: o teclado, o cartão perfurado (já obsoleto), as fitas e os discos magnéticos, entre outros;

- **Dispositivo de saída** é o meio pelo qual as informações (geralmente, os resultados da execução de um programa) são transferidas pelo computador ao usuário ou aos níveis secundários de memória. Exemplos: monitor de vídeo, impressora, fitas e discos magnéticos, entre outros;

- **Sintaxe** é a forma como os comandos devem ser escritos, a fim de que possam ser entendidos pelo tradutor de programas. A violação das regras sintáticas é considerada um erro sujeito à pena do não-reconhecimento do comando por parte do tradutor;

- **Semântica** é o significado, ou seja, o conjunto de ações que serão exercidas pelo computador durante a execução do referido comando.

Daqui por diante, todos os comandos novos serão apresentados por meio de sua sintaxe e sua semântica, isto é, a forma como devem ser escritos e a(s) ação(ões) que executam.

6.1 Instrução Primitiva de Atribuição

A **instrução primitiva de atribuição**, ou simplesmente **atribuição**, é a principal maneira de se armazenar uma informação numa variável. Sua sintaxe é:

<nome_de_variável> ← <expressão>

Em termos de fluxograma, os comandos de atribuição são representados como na Figura 6.1.

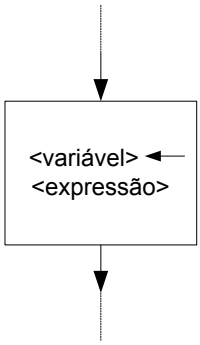
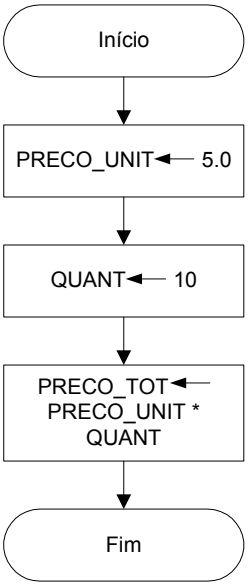


Figura 6.1 Forma de representação de comandos de atribuição em fluxogramas.

O modo de funcionamento (semântica) de uma atribuição consiste **1)** na avaliação da expressão e **2)** no armazenamento do valor resultante na posição de memória correspondente à variável que aparece à esquerda do comando.

Fluxograma



Pseudocódigo

```
Algoritmo EXEMPLO_6.1
Var  PRECO_UNIT, PRECO_TOT : real
     QUANT : inteiro
Início
     PRECO_UNIT ← 5.0
     QUANT ← 10
     PRECO_TOT ← PRECO_UNIT * QUANT
Fim.
```

Figura 6.2 Exemplo de aplicação de comandos de atribuição.

6.2 Instrução Primitiva de Saída de Dados

As instruções primitivas de saída de dados são o meio pelo qual informações contidas na memória dos computadores são colocadas nos dispositivos de saída, para que o usuário possa visualizá-las.

Há duas sintaxes possíveis para esta instrução:

Escreva <lista de variáveis>

ou

Escreva <literal>

Daqui por diante, **Escreva** será considerada uma palavra reservada e não mais poderá ser utilizada como nome de variável, de modo que toda vez que for encontrada em algoritmos será identificada como um comando de saída de dados.

Uma **lista_de_variáveis** é um conjunto de nomes de variáveis separados por vírgulas. Um **literal** é simplesmente um dado do tipo literal delimitado por aspas.

Em termos de fluxograma, uma instrução de saída de dados é representada como na Figura 6.3.

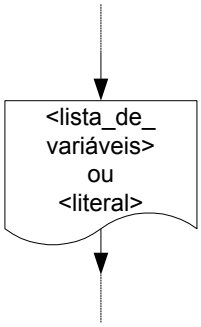


Figura 6.3 Forma de representação de uma instrução de saída de dados em fluxogramas.

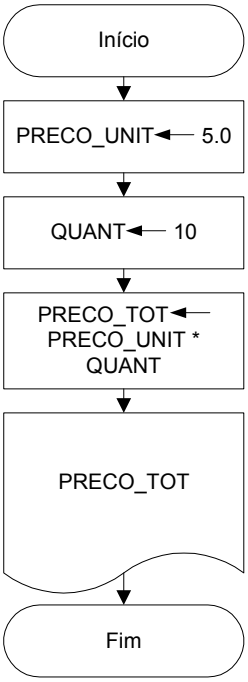
Repare que a representação no fluxograma dispensa o uso da palavra reservada **Escreva**, uma vez que a mesma já está embutida na forma geométrica da figura.

A semântica da instrução primitiva de saída de dados é muito simples: os argumentos do comando são enviados para o dispositivo de saída. No caso de uma lista de variáveis, o conteúdo de cada uma delas é pesquisado na posição de memória correspondente à variável e depois enviado para o dispositivo de saída. No caso de argumentos do tipo **string**, estes são enviados diretamente ao referido dispositivo.

Há, ainda, a possibilidade de se misturar nomes de variáveis com literais na lista de um mesmo comando. O efeito obtido é bastante útil e interessante: a lista é lida da esquerda para a direita e cada elemento da mesma é tratado separadamente; se um nome de variável for encontrado, então a informação da mesma é pega da memória e colocada no dispositivo de saída; no caso de um literal, o mesmo é escrito diretamente no dispositivo de saída.

O exemplo da Figura 6.2 torna-se muito mais interessante com a aplicação de instruções de saída de dados, como na Figura 6.4.

Fluxograma



Pseudocódigo

```
Algoritmo EXEMPLO_6.4
Var  PRECO_UNIT, PRECO_TOT : real
     QUANT : inteiro
Início
     PRECO_UNIT ← 5.0
     QUANT ← 10
     PRECO_TOT ← PRECO_UNIT * QUANT
     Escreva PRECO_TOT
Fim.
```

Figura 6.4 Exemplo de aplicação da instrução primitiva de saída de dados.

6.3 Instrução Primitiva de Entrada de Dados

O algoritmo da Figura 6.4 ainda necessita de uma melhoria essencial. Toda vez que ele é executado, o mesmo valor é calculado, já que os valores das variáveis **PRECO_UNIT** e **QUANT** permanecem inalterados. Seria interessante que estes valores pudessem ser fornecidos ao computador pelo usuário do programa toda vez que o programa fosse executado, para que o usuário tivesse um maior controle sobre o valor calculado. A instrução primitiva de entrada de dados foi criada para suprir esta necessidade.

Sua sintaxe é:

Leia <lista_de_variáveis>

Da mesma forma que **Escreva**, daqui em diante **Leia** será tratada como uma palavra-reservada e não mais poderá ser usada como nome de variável em algoritmos. A **lista_de_variáveis** é um conjunto de um ou mais nomes de variáveis, separados por vírgulas.

A Figura 6.5 mostra como uma instrução de entrada de dados é representada em fluxogramas. Esta representação dispensa o uso da palavra-reservada **Leia**, pelo fato da mesma já estar de certo modo embutida na forma geométrica da figura.

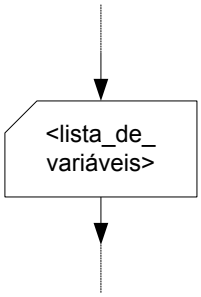


Figura 6.5 Forma de representação de uma instrução de entrada de dados em fluxogramas.

A semântica da instrução de entrada (ou leitura) de dados é, de certa forma, inversa à da instrução de escrita: os dados são fornecidos ao computador por meio de um dispositivo de entrada e armazenados nas posições de memória das variáveis cujos nomes aparecem na **lista_de_variáveis**.

O algoritmo da Figura 6.4, modificado para que os valores das variáveis **PRECO_UNIT** e **QUANT** sejam lidos no dispositivo de entrada, está na Figura 6.6.

Pseudocódigo

Algoritmo EXEMPLO_6.6
Var PRECO_UNIT, PRECO_TOT : **real**
QUANT : **inteiro**
Início
Leia PRECO_UNIT, QUANT
PRECO_TOT ← PRECO_UNIT * QUANT
Escreva PRECO_TOT
Fim.

Fluxograma

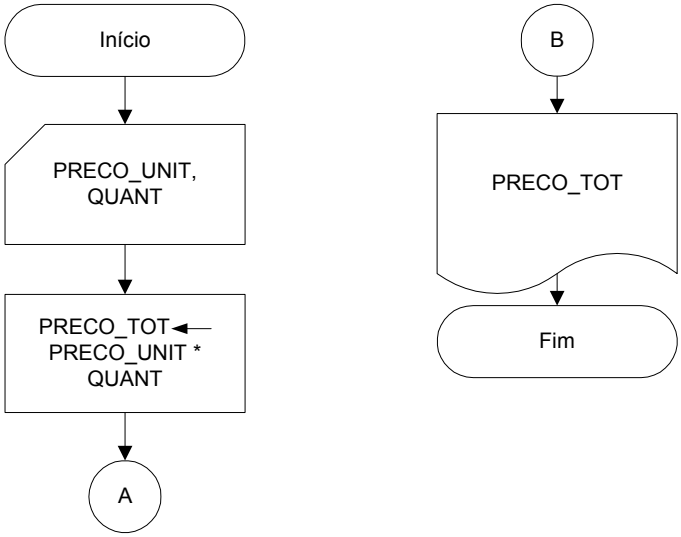


Figura 6.6 Exemplo de aplicação da instrução primitiva de entrada de dados num algoritmo.

O algoritmo da Figura 6.6 ainda precisa sofrer algumas modificações para ficar perfeito. Em sua forma atual, ao início de sua execução, ele procura ler os valores para as variáveis **PRECO_UNIT** e **QUANT**.Um usuário diferente daquele que criou o programa, a não ser que esteja bem treinado no uso do mesmo, poderá encontrar dificuldades na interação com o programa. Ele pode confundir a ordem em que os dados devem ser fornecidos ou simplesmente esquecer o que o programa deseja que ele digite. Ao término da execução o programa escreve como resultado um número que pode não possuir nenhum significado ao usuário se este não souber a finalidade para a qual o algoritmo foi concebido.

Uma preocupação constante de um bom programador deve ser a de conceber um programa "amigo do usuário". Esta preocupação é traduzida no planejamento de urna **interface com o usuário** (meio pelo qual um programa e o usuário "conversam") bastante amigável. Em termos práticos, isto se resume à aplicação de duas regras básicas:

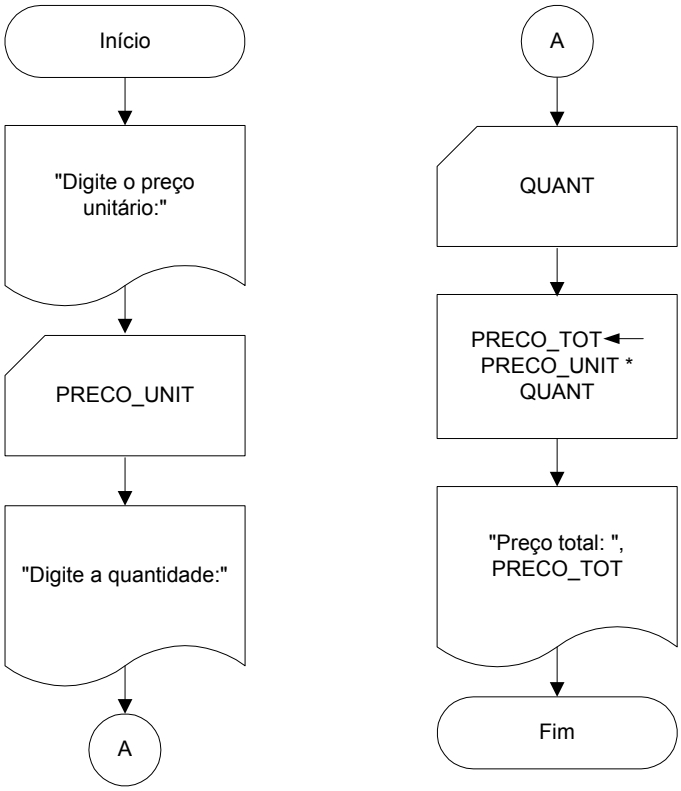
- toda vez que um programa estiver esperando que o usuário forneça a ele um determinado dado (operação de leitura), ele deve antes enviar uma mensagem dizendo ao usuário o que ele deve digitar, por meio de uma instrução de saída de dados;

- antes de enviar qualquer resultado ao usuário, um programa deve escrever uma mensagem explicando o significado do mesmo.

Estas medidas tornam o diálogo entre o usuário e o programador muito mais fácil.

A versão final do algoritmo estudado é mostrada na Figura 6.7.

Fluxograma



Pseudocódigo

Algoritmo EXEMPLO_6.7
Var PRECO_UNIT, PRECO_TOT : **real**
QUANT : **inteiro**
Início
Escreva “Digite o preço unitário: ”
Leia PRECO_UNIT
Escreva “Digite a quantidade: ”
Leia QUANT
PRECO_TOT ← PRECO_UNIT * QUANT
Escreva “Preço total: “, PRECO_TOT
Fim.

Figura 6.7 Exemplo de aplicação das instruções primitivas de atribuição,

entrada e saída de dados num algoritmo.

6.4 Síntese

A **instrução primitiva de atribuição** avalia uma expressão e armazena o valor resultante numa variável. O valor resultante da expressão e a variável devem ter tipos compatíveis.

A **instrução primitiva de saída de dados** admite como argumentos uma lista de variáveis, um literal, ou uma mistura de ambos. No primeiro caso, o valor de cada uma das variáveis é buscado na memória e colocado no dispositivo de saída. No caso de literais, estes são copiados diretamente no dispositivo de saída.

A **instrução primitiva de entrada de dados** busca, no dispositivo de entrada, dados que são guardados nas posições de memória correspondentes às variáveis da lista que lhe são passadas como argumento.

6.5 Exercícios Resolvidos

1. Escreva um algoritmo (fluxograma e pseudocódigo) para calcular a média entre dois números quaisquer.

   Solução:

   A idéia principal do algoritmo está centrada na expressão matemática utilizada no cálculo da média (**M**) entre dois números, **N1** e **N2**, dada por:

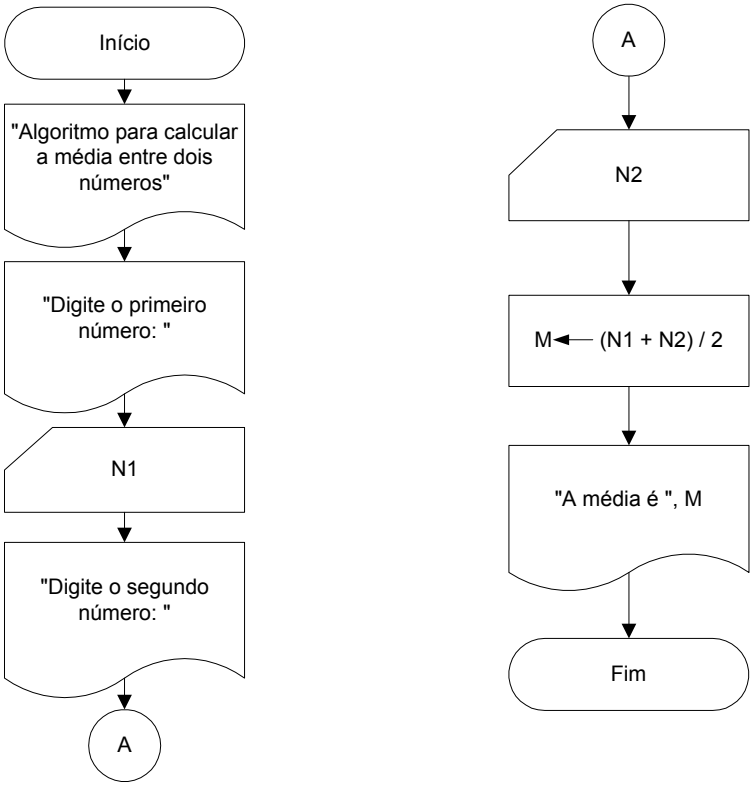
   M = (N1 + N2) / 2

   Para que o valor de **M** possa ser calculado pelo algoritmo, é necessário que os valores de **N1** e **N2** tenham sido fornecidos ao mesmo com antecedência. Portanto, a primeira etapa do algoritmo consiste da obtenção (leitura) dos valores de **N1** e **N2** e armazenamento dos mesmos em posições distintas de memória (variáveis).

   Na seqüência, o valor da média deve ser calculado por meio de uma expressão apropriada e atribuído a uma terceira variável (**M**).

   Por fim, deve-se relatar ao usuário o valor calculado por meio de uma instrução primitiva de saída de dados.

   O fluxograma do algoritmo descrito é mostrado a seguir. Note que ele está enriquecido com instruções para informar sua finalidade, os dados que devem ser fornecido ao usuário e o significado do valor calculado.


Início
"Algoritmo para calcular a média entre dois números"
"Digite o primeiro número: "
N1
"Digite o segundo número: "
A
A
N2
M ← (N1 + N2) / 2
"A média é ", M
Fim

A transformação do fluxograma em pseudocódigo exige a disponibilidade de algumas informações adicionais concernentes ao tipo das variáveis utilizadas. Como o algoritmo opera apenas com dados numéricos, certamente as variáveis utilizadas serão do tipo inteiro ou real. Como se deseja calcular a média entre dois números quaisquer, então as variáveis **N1** e **N2** devem ser capazes de armazenar números com ou sem parte fracionária e, portanto, é necessário que estas sejam do tipo real. Como o valor médio entre dois números reais é um número que pode ou não ter parte fracionária, então a variável **M** também deve ser do tipo real.

De posse dessa informação, pode-se escrever o pseudocódigo do algoritmo em questão, a partir de seu fluxograma.

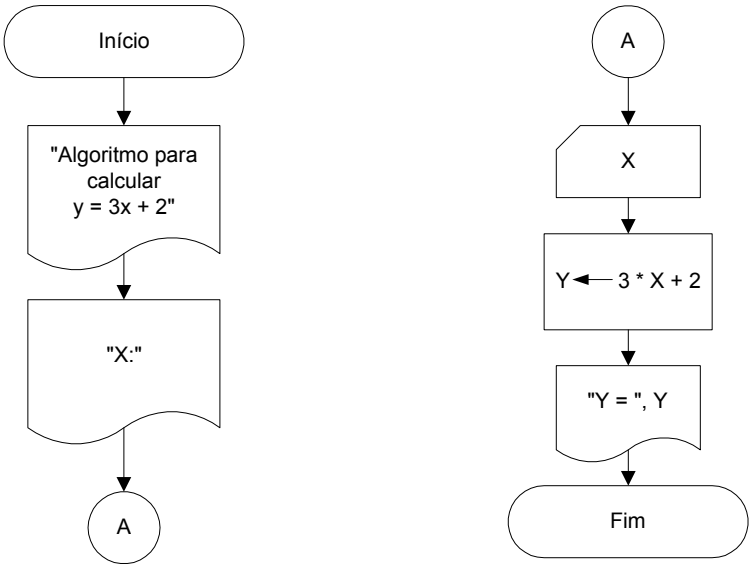
Algoritmo MEDIA
Var N1, N2, M : **real**
Início
    Escreva “Algoritmo para calcular a média entre dois números”
    Escreva “Digite o primeiro número: ”
    Leia N1
    Escreva “Digite o segundo número: ”
    Leia N2
    M ← (N1 + N2) / 2
    Escreva “O valor da média é:”, M
Fim.

2. Escreva um algoritmo para calcular o valor de **y** como função de **x**, segundo a função **y(x) = 3x + 2**, num domínio real.

Solução:

Essencialmente o algoritmo usado na solução deste problema consiste na obtenção do valor de x para o qual se deseja calcular a função, o cálculo desta propriamente dito e a mostra do resultado obtido ao usuário:

Veja fluxograma correspondente a seguir:



Para que se possa escrever o pseudocódigo do algoritmo deve-se decidir qual será o tipo das variáveis **X** e **Y**. Como especificado no enunciado do problema, o algoritmo deve operar num domínio real e, portanto, as variáveis **X** e **Y** devem ser do tipo real. Então, o pseudocódigo fica assim:

```
Algoritmo FUNCAO_DE_X
Var  X, Y : real
Início
      Escreva "Algoritmo para calcular
                y = 3x +2"
      Escreva "X: "
      Leia X
      Y ← 3 * X + 2
      Escreva "Y = ", Y
Fim.
```

3. Escreva um algoritmo para calcular o consumo médio de um automóvel (medido em Km/l), dado que são conhecidos a distância total percorrida e o volume de combustível consumido para percorrê-la (medido em litros).

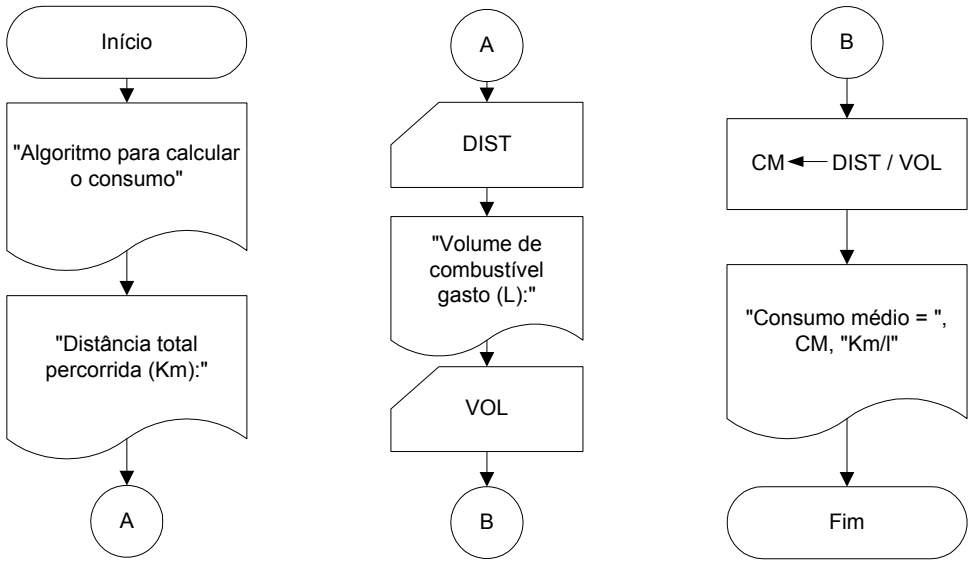
Solução:

A principal questão a ser levantada na obtenção do algoritmo pedido consiste na formulação da expressão usada para calcular o consumo médio (**CM**) a partir da distância total percorrida (**DIST**) e do volume de combustível consumido (**VOL**), que é dada por:

CM = DIST / VOL

Uma vez obtida esta expressão, a formulação do algoritmo desejado consiste em uma simples repetição daqueles apresentados nas questões anteriores: deve-se obter o valor das variáveis **DIST** e **VOL**, calcular o consumo pela expressão acima e, finalmente, mostrar ao usuário o valor calculado.

O fluxograma correspondente ao algoritmo é o seguinte:



Assumindo que todas as variáveis utilizadas (**CM**, **DIST** e **VOL**) são do tipo real, pode-se escrever o pseudocódigo seguinte para o fluxograma anterior:

Pseudocódigo

```
Algoritmo CONSUMO_MEDIO
Var  CM, DIST, VOL : real
Início
     Escreva "Algoritmo para calcular o consumo"
     Escreva "Distância total percorrida (Km): "
     Leia DIST
     Escreva "Volume de combustível gasto (L): "
     Leia VOL
     CM ← DIST / VOL
     Escreva "Consumo médio =", CM, "Km/l"
Fim.
```

6.6 Exercícios Propostos

Para cada um dos problemas propostos a seguir, expresse um algoritmo que pode ser usado em sua solução na forma de fluxograma e pseudocódigo.

1. Calcule a média de quatro números inteiros dados.

2. Leia uma temperatura dada na escala Celsius (C) e imprima o equivalente em Fahrenheit (F). (Fórmula de conversão: F = 9/5 * C + 32)

3. Leia uma quantidade de chuva dada em polegadas e imprima o equivalente em milímetros (25,4 mm = 1 polegada).

4. Calcule o quadrado de um número, ou seja, o produto de um número por si mesmo.

5. O custo ao consumidor de um carro novo é a soma do custo de fábrica com a porcentagem do distribuidor e dos impostos, ambos aplicados ao custo de fábrica. Supondo que a porcentagem do distribuidor seja de 12% e a dos impostos de 45%, prepare um algoritmo para ler o custo de fábrica do carro e imprimir o custo ao consumidor.

6. O cardápio de uma lanchonete é dado abaixo. Prepare um algoritmo que leia a quantidade de cada item que você consumiu e calcule a conta final.

   Hambúrguer..... R$ 3,00
   Cheeseburger..... R$ 2,50
   Fritas..... R$ 2,50
   Refrigerante..... R$ 1,00
   Milkshake..... R$ 3,00

7. Uma companhia de carros paga a seus empregados um salário de R$ 500,00 por mês mais uma comissão de R$ 50,00 para cada carro vendido e mais 5% do valor da venda. Elabore um algoritmo para calcular e imprimir o salário do vendedor num dado mês recebendo como dados de entrada o nome do vendedor, o número de carros vendidos e o valor total das vendas.

8. Calcule a média de um aluno na disciplina de MDS. Para isso solicite o nome do aluno, a nota da prova e a nota qualitativa. Sabe-se que a nota da prova tem peso 2 e a nota qualitativa peso 1. Mostre a média como resultado.