5 *Iniciando um Agoritmo*

5.1 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.

5.1.1 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.

5.1.1.1 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 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, isto é, seis bytes são necessários para guadar 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[1].

A Tabela 7 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.

Tabela 7: Armazenamento da palavra "banana" na memória.

caractere	**código ASCII**	**Endereço**	**bites**
'b'	98	0	01100010
'a'	97	1	01100001
'n'	110	2	01101110
'a'	97	3	01100001
'n'	110	4	01101110
'a'	97	5	01100001

[1]Compilador é um programa que traduz um outro escrito em alguma linguagem de programação para a linguagem de máquina do computador.

5.1.1.2 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.

5.1.1.3 Armazenamento de dados do tipo inteiro

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

$$Z = \{-\infty, \cdots, -3, -2, -1, 0, 1, 2, 3, \cdots, +\infty\} \tag{5.1}$$

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:

$$-127, -126, \cdots, -2, -1, 0, 1, 2, \cdots, 127, 128 \tag{5.2}$$

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:

$$-32767, -32766, \cdots, -2, -1, 0, 1, 2, \cdots, 32767, 32768 \tag{5.3}$$

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.

5.1.1.4 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.

5.1.2 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.

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 o sub-linha '_' (*underline*).

Tabela 8: Exemplos de nomes de variáveis.

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)

5.1.3 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 sempre presente na definição de um conjunto de uma ou mais variáveis;
- Em uma mesma linha poderão ser definidas uma ou mais variáveis do mesmo tipo, separando-se 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. A seguir, tem-se um exemplo de definição de variáveis.

```
VAR Nome: Caractere[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 um nível lógico.

5.1.4 Exercícios Propostos

Questão 01: Assinale com 'C' os identificadores corretos e com 'E' os errados, explicando o que está errado.

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

Questão 02: 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.

5.2 Expressões

5.2.1 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.

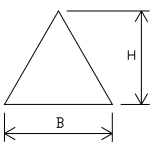

Figura 30: Área do triângulo.

Por exemplo, a fórmula de cálculo da área do triângulo da Fig. 30 é dada por:

$$Area = 0.5 \times B \times H \tag{5.4}$$

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 ($\times$), 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.

5.2.2 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.

5.2.3 Tipos de Expressões

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

5.2.3.1 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 Tab. 9.

Tabela 9: Operadores aritméticos.

Operador	Tipo	Operação	Prioridade
`+`	Binário	Adição	4
`-`	Binário	Subtração	4
`*`	Binário	Multiplicação	3
`/`	Binário	Divisão	3
`**`	Binário	Exponencição	2
`SQR`	Unário	Raiz quadrada	2
`-`	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 ($\times$), 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:

Tabela 10: Exemplos de expressões.

Expressão	Descrição
`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

5.2.3.2 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 Tab. 11.

Tabela 11: Operadores lógicos.

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

Existem outros operadores lógicos, como por exemplo o `.OU_EXCLUSIVO.`, mas suas funções podem ser exercidas por combinações dos três tipos de operadores da Tab. 11.

Para exemplificar o uso de operadores lógicos, a Tab. 12 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`.

Este tipo de tabela é chamado de tabelas-verdade. A Tabela 12 pode ser extraída por observação dos termos da Tab. 11:

- O operador lógico .NÃO. sempre inverte o valor de seu operando. Ex.: `.NÃO.V = F` e `.NÃO.F = V`;

Tabela 12: Tabela-verdade.

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

- 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.

Nas expressões lógicas onde aparecem apenas os operadores lógicos da Tab. 11 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 Tab. 13.

Tabela 13: 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.

5.2.3.3 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".

5.2.4 Exercícios Propostos

Questão 01: Dada a declaração de variáveis abaixo, 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).

```
VAR A, B, C: Inteiro;
    X, Y, Z: Real;
    NOME, RUA: Caractere[20];
    L1, L2: Lógico;
```

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

Questão 02: Para as variáveis declaradas na questão 01, assumindo os valores dados a seguir, determine o resultado da avaliação das expressões abaixo:

```
A = 1  X = 2.0   L1 = V  NOME = "Zé das Couve"
B = 2  Y = 10.0  L2 = F  RUA = "Travessa do Sebo"
C = 3  Z = -1.0
```

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)) ⇒

X Y.E.C = B ⇒

(C-3*A) (X+2*Z) ⇒

(L1.E.(.NÃO.L2)).OU.(L2.E.(.NÃO.L1)) ⇒

5.3 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 é incapaz de se comunicar com o mundo exterior.

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.

5.3.1 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 Fig. 31.

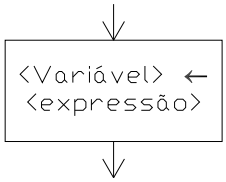

Figura 31: Forma de representação de comandos de atribuição em fluxogramas.

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

Exemplo 1. Algoritmo exemplificando o uso da instrução primitiva de atribuição.

O fluxograma para este exemplo é mostrado na Fig. 32.

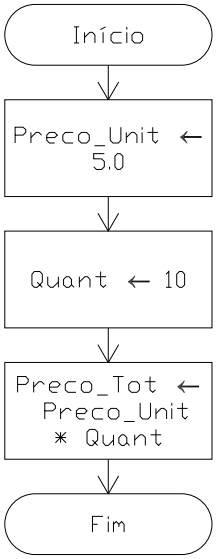

Figura 32: Exemplo representado por fluxograma.

As linhas de código abaixo, representam o algoritmo em português estruturado para o exemplo da Fig. 32.

```
Programa Exemplo_1
  Var PRECO_UNIT, PRECO_TOT: Real;
      QUANT: Inteiro;
Início
  PRECO_UNIT ← 5.0;
  QUANT ← 10;
  PRECO_TOT ← PRECO_UNIT*QUANT;
Fim
```

5.3.2 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>);
Escreva("Texto");
```

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.

Em termos de fluxograma, uma instrução de saída de dados é representada como na Fig. 33.

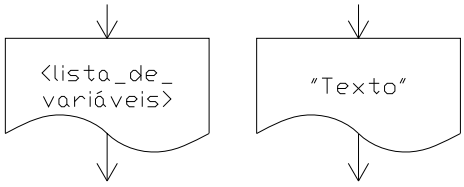

Figura 33: Forma de representação de uma instrução de saída de dados em fluxogramas.

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.

Ainda há a possibilidade de se misturar nomes de variáveis com texto 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.

Exemplo 2. Utilizando as instruções de saída de dados, o algoritmo do Exemplo 1 mostra o resultado para o preço total calculado.

O fluxograma para este exemplo é mostrado na Fig. 34.

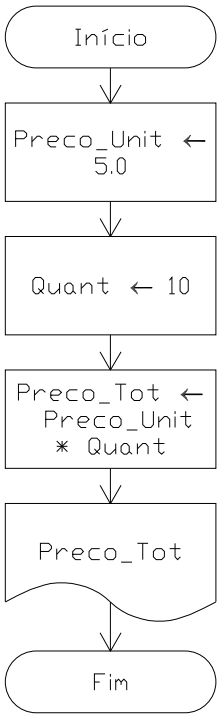

Figura 34: Exemplo representado por fluxograma.

As linhas de código abaixo, representam o algoritmo em português estruturado para o exemplo de aplicação da instrução primitiva de saída de dados da Fig. 34.

```
Programa Exemplo_2
  Var PRECO_UNIT, PRECO_TOT: Real;
      QUANT: Inteiro;
Início
  PRECO_UNIT ← 5.0;
  QUANT ← 10;
  PRECO_TOT ← PRECO_UNIT*QUANT;
  Escreva(PRECO_TOT);
Fim
```

5.3.3 Instrução Primitiva de Entrada de Dados

O algoritmo da Fig. 34 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 35 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.

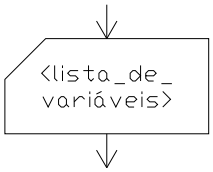

Figura 35: Forma de representação de uma instrução de entrada de dados em fluxogramas.

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>`.

Exemplo 3. Modificação do algoritmo do Exemplo 2,para que os valores das variáveis `PRECO_UNIT` e `QUANT` sejam lidos no dispositivo de entrada.

O fluxograma para este exemplo é mostrado na Fig. 36.

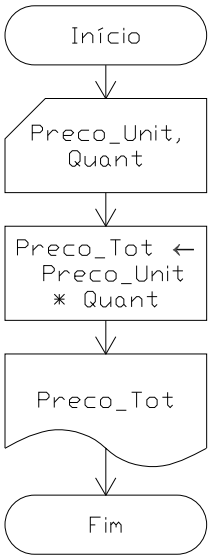

Figura 36: Exemplo representado por fluxograma.

As linhas de código abaixo, representam o algoritmo em português estruturado para o exemplo de aplicação da instrução primitiva de entrada de dados da Fig. 36.

```
Programa Exemplo_3
  Var PRECO_UNIT, PRECO_TOT: Real;
      QUANT: Inteiro;
Início
  Leia(PRECO_UNIT, QUANT);
  PRECO_TOT ← PRECO_UNIT*QUANT;
  Escreva(PRECO_TOT);
Fim
```

O algoritmo do Exemplo 3 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.

Exemplo 4. Versão final do algoritmo do Exemplo 3.

O fluxograma para este exemplo é mostrado na Fig. 37.

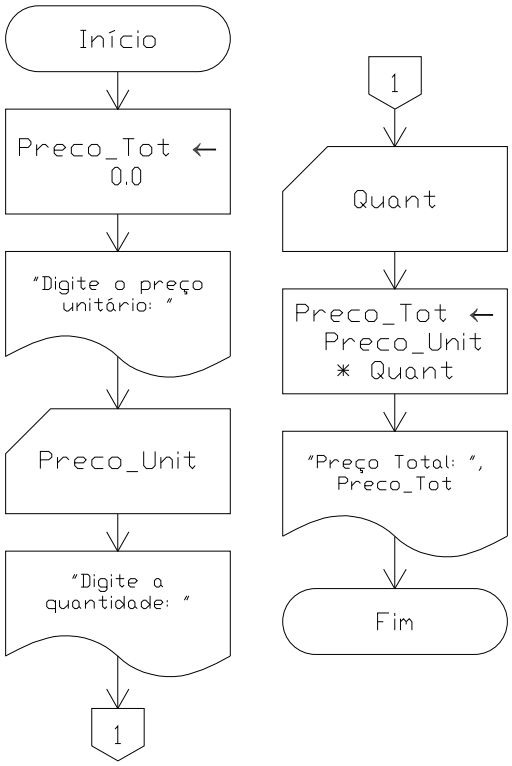

Figura 37: Exemplo representado por fluxograma.

As linhas de código abaixo, representam o algoritmo em português estruturado para o exemplo de aplicação das instruções primitivas de atribuição, entrada e saída de dados da Fig. 37.

```
Programa Exemplo_4
  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
```

5.3.4 Exercícios Resolvidos

Exercício 01: 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:

$$M = \frac{N1 + N2}{2} \tag{5.5}$$

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.

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

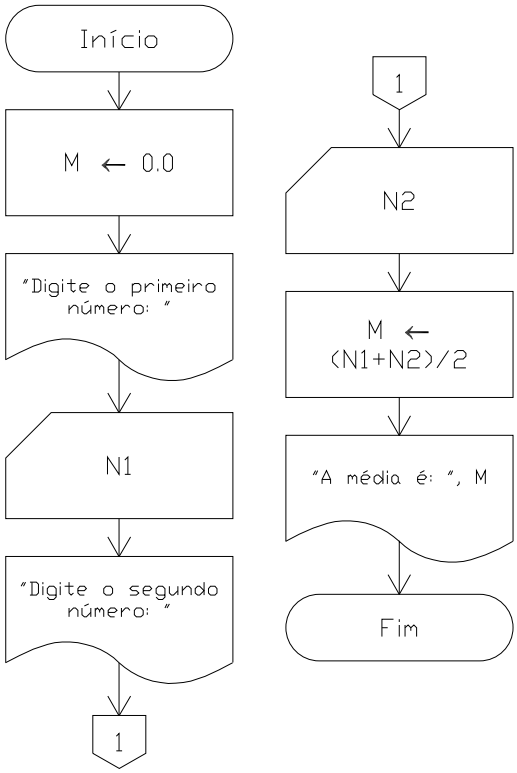

Figura 38: Fluxograma para o calculo da média.

também deve ser do tipo real.

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

```
Programa 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
```

Exercício 02: 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:

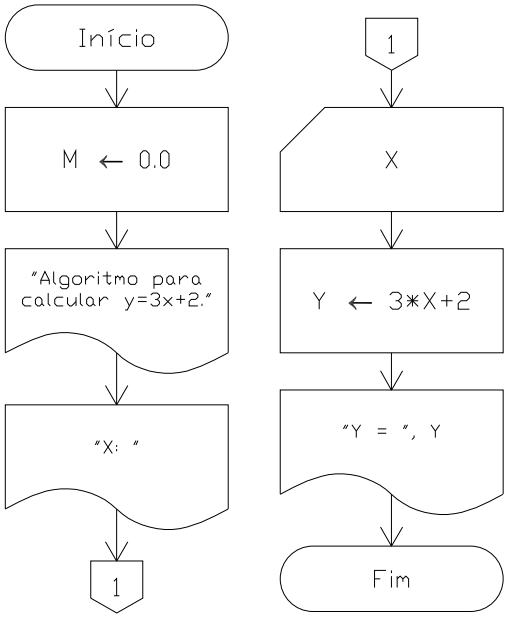

Figura 39: Fluxograma para calculo de uma função.

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:

```
Programa f_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
```

Exercício 03: 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:

$$CM = \frac{DIST}{VOL} \tag{5.6}$$

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:

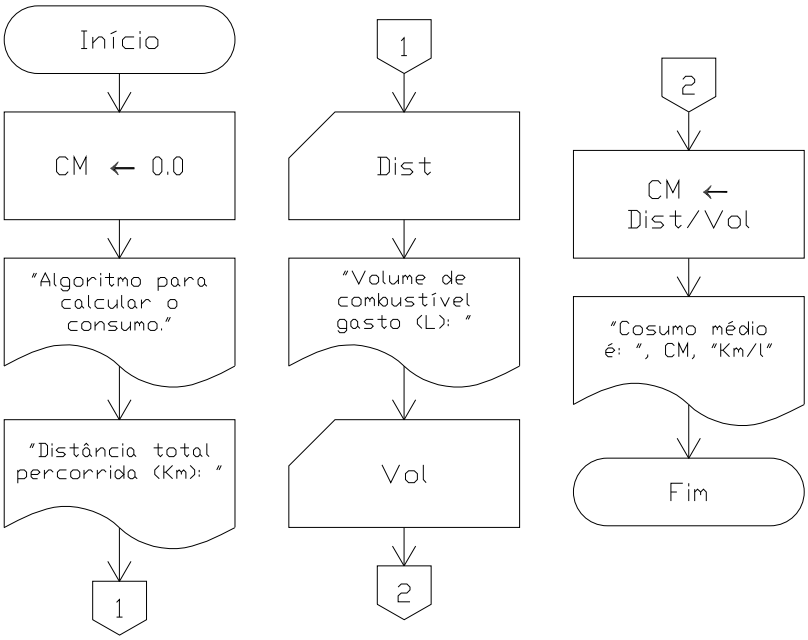

Figura 40: Fluxograma para calculo de consumo.

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

```
Programa 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
```

5.3.5 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.

Questão 01: Calcule a média de quatro números inteiros dados.

Questão 02: Leia uma temperatura dada na escala Celsius (C) e imprima o equivalente em Fahrenheit (F). (Fórmula de conversão: F = 9/5 * C + 32).

Questão 03: Leia uma quantidade de chuva dada em polegadas e imprima o equivalente em milímetros (25,4 mm = 1 polegada).

Questão 04: Calcule o quadrado de um número, ou seja, o produto de um número por si mesmo.

Questão 05: 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.

Questão 06: 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.

Tabela 14: Cardápio.

Seboso	**Preço**
Hambúrguer	R$ 3,00
Cheeseburger	R$ 2,50
Fritas	R$ 2,50
Bebida	
Refrigerante	R$ 1,00
Milkshake	R$ 3,00

Questão 07: 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.

Questão 08: 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.