

CURSO DE INFORMÁTICA

Algoritmos

Ricardo José Cabeça de Souza

Parte 3

Sumário

UNIDADE I - ALGORITMOS

1.6 Pseudo-Linguagem (Pseudocódigo)

1.6.1 Pseudo-Linguagem

1.6.2 Identificadores

1.6.2.1 Nomeação de Identificadores

1.6.2.2 Definição de Tipos para os Identificadores

1.6.2.3 Definição de Tipos de Dados

1.6.2.4 Declaração de Identificadores (variável ou constante)

1.6.3 Expressões

1.6.3.1 Operadores Aritméticos

1.6.3.2 Expressões Aritméticas

1.6.3.3 Expressões Lógicas

1.6.4 Comandos Básicos

1.6.4.1 Comando de Atribuição

1.6.4.2 Comando de Entrada

1.6.4.3 Comando de Saída

1.7 REGRAS PRÁTICAS E METODOLOGIA PARA ELABORAÇÃO DE ALGORITMOS

1.7.1 Regras Práticas

1.7.2 Metodologia no Desenvolvimento de Algoritmos

1.6 PSEUDO-LINGUAGEM (PSEUDOCÓDIGO)

Este item tem por objetivo apresentar os elementos componentes da pseudo-linguagem (pseudocódigo) utilizada para escrever algoritmos de forma padronizada.

1.6.1 Pseudo-Linguagem

Considerando que para se escrever algoritmos devemos definir a seqüência lógica das ações a serem executadas, precisamos utilizar uma forma padronizada de realizar essa tarefa. Para isso, utilizaremos uma linguagem apropriada, definida aqui como pseudo-linguagem, de forma a estabelecer um padrão de escrita para representar as ações a serem executadas nos nossos algoritmos. Essa linguagem nada mais é do que o uso de uma linguagem intermediária entre a linguagem natural e uma linguagem de programação para descrever os passos a serem realizados para solução de determinado problema.

Os elementos utilizados nessa linguagem são:

- Identificadores;
- Tipos de dados;
- Operadores aritméticos;
- Operadores relacionais;
- Operadores lógicos;
- Comandos básicos:
 - Comando de atribuição;
 - Comando de entrada;
 - Comando de saída;
 - Escrita de comentários;

Veremos ainda algumas regras práticas para elaboração de algoritmos, com o objetivo de torná-los mais práticos e simples, bem como uma metodologia de desenvolvimento de algoritmos.

1.6.2 Identificadores

O identificador é um nome usado para designar uma entidade em um algoritmo ou programa. Qualquer objeto utilizado para representar um elemento no algoritmo, precisa ser identificado e realizamos essa identificação através de identificadores.

Exemplos de identificadores: NOME, A35, SOMA, N1, etc.

1.6.2.1 Nomeação de Identificadores

Os identificadores possuem sempre um nome, usado para representar de forma única em um algoritmo ou programa, um elemento qualquer. Desta forma, precisamos definir um nome para cada objeto ou elemento utilizado no algoritmo. Contudo, nem todos os nomes podem ser utilizados.

Segue abaixo a regra que deveremos utilizar na definição dos identificadores:

- a) O nome dos identificadores deve começar por caractere alfabético (letra), maiúsculo ou minúsculo. É importante observar que maiúsculas e minúsculas definem identificadores diferentes ($A \neq a$);
- b) Se o identificador possuir mais de um caractere, podemos utilizar caracteres alfabéticos (letras) e números em sua composição;
- c) É recomendável que você defina nomes significativos de acordo com a função do identificador no algoritmo, relacionando o nome ao conteúdo a ser armazenado no identificador;
- d) Não devemos utilizar caracteres especiais, como por exemplo, %, &, *, ç, etc. O único caractere válido na escrita de nomes é o sublinhado (_).

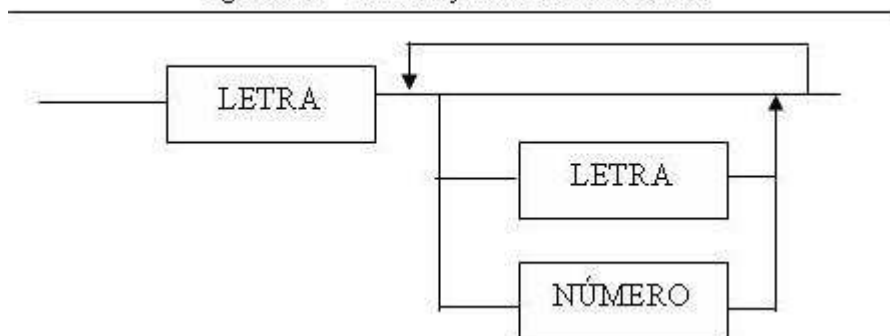
Na escrita de algoritmos, não temos a intenção de executar efetivamente o programa. Em tese, não há necessidade de se evitar utilizar caracteres especiais, contudo, os algoritmos serão posteriormente transformados em programas, utilizando uma linguagem de programação. A idéia então é, desde já, estabelecer correspondência com a linguagem, criando regras que **obrigatoriamente** deverão ser seguidas na sintaxe da linguagem.

- e) Não devemos utilizar espaços em branco, pelos mesmos motivos descritos no item anterior;
- f) Não podemos utilizar nomes para identificadores que correspondem a palavras reservadas na linguagem de programação a ser utilizada na elaboração do programa. Neste caso, torna-se necessário que você observe atentamente, durante o processo de transformação de seu algoritmo em um programa fonte, se os nomes utilizados no algoritmo correspondem a algum nome reservado na linguagem.

Por exemplo, se o algoritmo for transformado em um programa fonte utilizando a linguagem de programação C, a palavra **main** não poderá ser utilizada, mesmo seguindo todas as regras estabelecidas para definição do nome do identificador, ou seja, iniciar por caractere alfabético, possuir outros caracteres alfabéticos em sua composição, não possuir caracteres especiais, etc. Porém, **main** é um nome reservado da **linguagem de programação C**, representando uma função. Contudo, a palavra **main** poderia ser um identificador para programação na linguagem Pascal, por não possuir qualquer relação.

Com essa última observação, terminamos então a definição das regras para você definir o nome de seus identificadores, servindo tanto para algoritmo quanto para programas. A figura 1.13 representa a forma geral para se definir o nome de um identificador.

Figura 1.13 – Nomeação de Identificadores



1.6.2.2 Definição de Tipos para os Identificadores

Os identificadores usados em algoritmos e programas podem ser do tipo **constante** ou **variável**.

Conforme o nome sugere, identificadores do tipo constante são identificadores que não podem ter seus valores alterados em algoritmos e programas. Da mesma forma, identificadores do tipo variável são identificadores que podem ter seus valores alterados em algoritmos ou programas.

Para exemplificar, considere um algoritmo onde será calculada a área de uma circunferência. Para executar essa tarefa, com certeza você precisará utilizar a fórmula matemática para realizar o cálculo: $A = \pi * RAIO^2$.

O valor de π é definido como o número que representa o quociente entre o perímetro de uma circunferência e o seu diâmetro, com valor **fixo** aproximadamente igual a 3,141592653589793238462643, normalmente arredondado para 3,14.[10]

Considerando que o valor de π não pode ser alterado, qualquer identificador utilizado para representar π , como por exemplo, **PI**, será do tipo **constante**. Da mesma forma, o valor para o raio, definido através do identificador **RAIO**, pode assumir qualquer valor em um algoritmo ou programa. Esse identificador, considerando que pode assumir qualquer valor, será do tipo **variável**.

Normalmente, na maior parte dos algoritmos, são utilizados identificadores do tipo variável, sendo reduzido o uso de identificadores do tipo constante. Por esta razão, é comum você encontrar na literatura disponível no mercado, a referência dos identificadores simplesmente por **variável**, ou seja, na definição de um problema, são apresentadas as **variáveis** utilizadas.

1.6.2.3 Definição de Tipos de Dados possíveis para armazenamento em Identificadores (variável ou constante)

Após a definição do nome do identificador (constante ou variável), ou como muitos costumam utilizar, o **nome da variável**, você precisa definir quais os tipos de dados efetivamente podem ser armazenados no identificador.

A informação referente ao tipo de um dado identificador é utilizada tanto pelo programador como pelo computador. O programador utiliza esta informação para determinar quais as operações que pode efetuar com

esse identificador. O computador utiliza essa informação em duas fases distintas: durante a tradução para a linguagem de máquina onde verifica a legalidade e compatibilidade das instruções do programa (por exemplo, não é possível somar um valor lógico com um valor inteiro) e durante a execução do programa onde essa informação é utilizada para “interpretar” as suas ações internas.[2]

Os identificadores podem assumir os seguintes tipos:

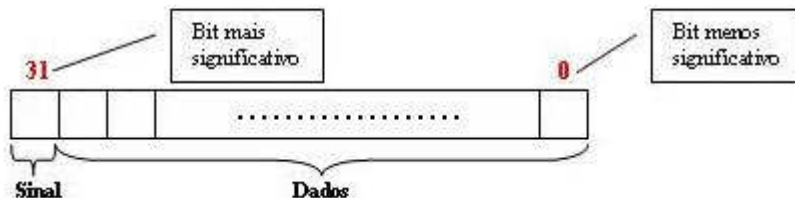
- Inteiro;
- Real;
- Caractere;
- Lógico.

Os identificadores do tipo **inteiro** são representados pelo conjunto dos números inteiros (**Z**), aqueles que não possuem componentes decimais ou fracionários, podendo ser positivos ou negativos.

Exemplo de números inteiros: 48, 0, -59.

A figura 1.14 apresenta como o dado do tipo inteiro é armazenado na memória do computador. Por padrão, as linguagens de programação reservam 2 (dois) bytes para armazenar um número do tipo inteiro. Contudo, algumas linguagens de programação disponibilizam 4 (quatro) bytes para realizar a mesma tarefa, possibilitando uma quantidade maior de representação.

Figura 1.14 – Representação do Inteiro na Memória do Computador



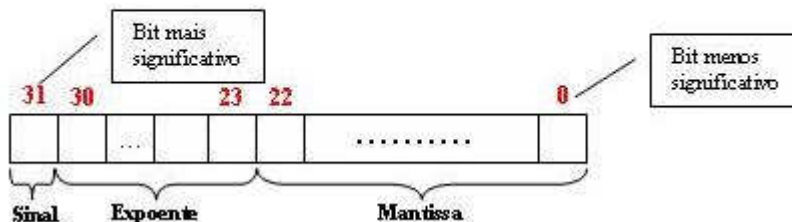
Identificadores do tipo **real** são representados pelo conjunto dos números reais (**R**), onde podem possuir componentes decimais ou

fracionários, e podem ser positivos ou negativos. Os identificadores do tipo real também são chamados de **ponto flutuante**. Ponto flutuante é um formato de representação digital de números reais, que é usada nos computadores. Ao armazenar um dado do tipo real na memória de um computador, a informação é armazenada na forma mantissa (M) e um expoente (E). O valor representado é obtido pelo produto: $M * 2^E$.

Exemplo de números reais: 46.9, -18.4, 0.4.

A figura 1.15 apresenta como o dado do tipo real é armazenado na memória do computador. Por padrão, as linguagens de programação reservam 4 (dois) bytes para armazenar um número do tipo real.

Figura 1.15 – Representação do Real na Memória do Computador



Os identificadores do tipo **caractere** são representados por qualquer letra do alfabeto, dígito numérico, código de controle ou símbolo especial, pertencente a um sistema específico de codificação. Esses identificadores reservam na memória do computador uma única posição (um byte) para armazenar o dado.

Exemplo de caracteres: A, b, '4', '\$'.

Observe que os números também podem ser definidos como caractere, e para distinguir o valor numérico do tipo caractere é utilizar aspas simples (` `) em sua representação, como mostrado no exemplo acima. Contudo, neste caso, esse caractere não pode ser utilizado para operações matemáticas, pois não é possível executar operações matemáticas com valores tipo caractere.

Podemos também associar à informação do tipo caractere ao tipo estruturado de informação chamado **cadeia de caracteres (dado literal, vetor de caracteres, ou em inglês "string")**. Uma cadeia de

caracteres é definida como uma seqüência de caracteres. O tamanho da cadeia vai ser definido pelo número de caracteres que compõe a cadeia. Em sua representação, utilizamos aspas duplas (“ ”).

Exemplo de cadeia de caracteres: “Maria”, “ESTADO”, “A casa”.

Os dados do tipo **lógico** são caracterizados com os seguintes valores: verdadeiro (em inglês *true*) ou falso(em inglês *false*), sendo que esse tipo de dado somente poderá representar esses dois valores. Esse tipo também é denominado de **booleano**, devido à contribuição do filósofo e matemático inglês George Boole na área da lógica matemática. Para facilitar a citação de um dado do tipo lógico e diferenciar entre nomes de variáveis, alguns autores e professores apresentam estes valores delimitados por ponto (.).

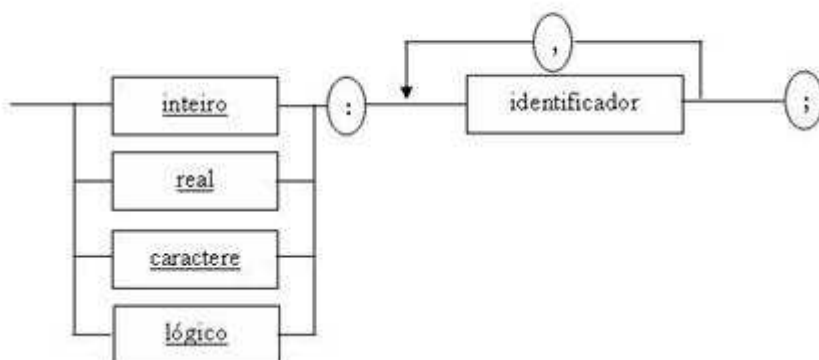
Valores possíveis em um tipo lógico: **.verdadeiro.** ou **.falso.** .

1.6.2.4 Declaração de Identificadores (variável ou constante)

Em todos os algoritmos ou programa, a primeira tarefa a ser realizada é a declaração dos identificadores (chamado muitas vezes de **declaração de variáveis**). Nesta tarefa o programador define os nomes dos identificadores que serão utilizados no algoritmo ou programa, se serão do tipo variável ou constante e os tipos de dados que poderão armazenar.

Para efetivamente realizar a declaração de variáveis, seguiremos a sintaxe mostrada na figura 1.16.

Figura 1.16 – Declaração de Identificadores



Exemplos de declaração de variáveis:

a) Declaração de duas variáveis do tipo inteiro:

inteiro: A, B;

b) Declaração de uma variável do tipo real e duas variáveis do tipo caractere:

real: SOMA;

caracatere: Letra, Sinal;

c) Declaração de duas variáveis do tipo cadeia de caracteres e uma variável lógica:

caractere: NOME[60], ENDERECO[100];

lógico: X;

Observe que na declaração de variáveis do tipo cadeia de caracteres (vetor de caracteres ou *string*) você precisa definir o número de posições necessárias para armazenar a informação desejada.

1.6.3 Expressões

Para Araújo (2005) 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 relaciona-se por meio de operadores, compondo uma fórmula que, uma vez avaliada, resulta em um valor.

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 em um valor.[3]

1.6.3.1 Operadores Aritméticos

Araújo (2005) define operadores como sendo elementos funcionais que atuam sobre operandos e produzem um determinado resultado. Os operadores aritméticos são utilizados para realização de cálculos matemáticos.[3]

1.6.3.2 Expressões Aritméticas

De acordo com Araújo (2005) expressões aritméticas são aquelas cujo resultado da avaliação é do tipo numérico, seja inteiro ou real. Somente o uso de operadores aritméticos e variáveis numéricas são permitidos em expressões deste tipo.[3]

A tabela 1 apresenta os operadores que nós utilizaremos na escrita de nossos algoritmos, contendo ainda exemplos de utilização desses operadores.

Tabela 1 – Operadores Aritméticos

Operador	Operação	Exemplo de Expressões
+	Soma	$5 + 3 \rightarrow 8$
-	Subtração	$8 - 2 \rightarrow 6$
*	Multiplificação	$9 * 2 \rightarrow 18$
/	Divisão real	$5 / 2 \rightarrow 2.5$
<u>div</u>	Divisão inteira	$5 \text{ div } 2 \rightarrow 2$ Observação: Neste caso, se faz a divisão até não ser mais possível realizá-la sem a utilização de recursos adicionais, apresentando como resultado o quociente.

Operador	Operação	Exemplo de Expressões
<u>mod</u>	Resto da divisão inteira	<u>5 mod 2</u> → 1
**	Potenciação (<u>Exponenciação</u>)	4 ** 2 → 16 Observação: Conforme a regra matemática: <ul style="list-style-type: none"> ▪ <u>a¹</u> = a ▪ <u>a⁰</u> = 1 ▪ <u>a⁻¹</u> = 1/a ▪ <u>a⁻ⁿ</u> = 1/<u>aⁿ</u>

<u>RAIZ()</u>	Raiz quadrada	<u>RAIZ(25)</u> → 5 Observação: Somente é possível calcular a raiz quadrada de um número não negativo.
---------------	---------------	--

A **prioridade dos operadores** define a ordem de execução das operações a ser seguida, e acompanha a regra matemática, executando inicialmente as operações entre parênteses e funções pré-definidas (como por exemplo, raiz quadrada), em seguida operações de potenciação, multiplicação e divisão, e por fim operações de soma e subtração, convencionalmente da esquerda para a direita para casos de operadores de mesma prioridade.

Observe que você, conforme descrito por GUIMARÃES e LAGES (1985), tem plena liberdade de introduzir novos operadores ou nomes de funções para adaptar a pseudo-linguagem às necessidades específicas do problema a ser resolvido, conforme a área de aplicação, primando por sua definição clara para não deixar margem para ambigüidades.[4]

Exemplos de utilização de expressões com necessidade de análise da prioridade de operadores:

a) Dada a expressão: $4 + 5 - 9 * 2 / 3 + 6 ** 0$

Conforme a prioridade de operadores matemáticos, inicialmente serão realizadas as operações de multiplicação, divisão e potenciação:

$$- 9 * 2 / 3 = - 6$$

$$6 ** 0 = 1$$

Recompondo a expressão e executando as operações de soma e subtração o resultado fica da seguinte forma:

$$4 + 5 - 6 + 1 = 4$$

b) Dada a expressão $(3 + 5) - (5 * (50 / 10)) + (RAIZ(100))$

Conforme a prioridade de operadores matemáticos, serão realizadas inicialmente as expressões definidas nos parênteses, como a seguir:

$$(3 + 5) = 8$$

$$(50 / 10) = 5 \rightarrow -(5 * (5))$$

$$-(5 * 5) = - 25$$

$$RAIZ(100) = 10$$

Recompondo a expressão e executando as operações de soma e subtração o resultado fica da seguinte forma:

$$8 - 25 + 10 = -7$$

1.6.3.3 Expressões Lógicas

Araújo define **expressão lógica** como aquela cujo resultado é um valor lógico (**.verdadeiro.** ou **.falso.**). [3]

De acordo com FARRER (1989) denomina-se expressão lógica a expressão cujos operadores são lógicos e cujos operandos são relações, constantes e/ou variáveis do tipo lógico.[1]

Durante a elaboração da seqüência de ações a serem executadas na resolução de problemas, em várias situações você terá necessidade de estabelecer **condições** para realizar determinada tarefa, e na avaliação dessas condições utilizamos expressões lógicas.

Para efetuar operação com expressões lógicas, utilizamos operadores lógicos e operadores relacionais.[3]

A tabela 2 apresenta os operadores lógicos e relacionais utilizados na pseudo-linguagem para elaboração de algoritmos.

Tabela 2 – Operadores Relacionais e Lógicos

Relações		
Operador	Operação	Exemplo de Expressões
=	Igual a	$6 = 5 \rightarrow$.falso. $9 = 9 \rightarrow$.verdadeiro.
<>	Diferente de	$6 <> 5 \rightarrow$.verdadeiro. $9 <> 9 \rightarrow$.falso.
>	Maior que	$6 > 5 \rightarrow$.verdadeiro. $9 > 9 \rightarrow$.falso.
<	Menor que	$6 < 5 \rightarrow$.falso. $9 < 9 \rightarrow$.falso.
>=	Maior ou igual a	$6 >= 5 \rightarrow$.verdadeiro. $9 >= 9 \rightarrow$.verdadeiro.
<=	Menor ou igual a	$6 <= 5 \rightarrow$.falso. $9 <= 9 \rightarrow$.verdadeiro.

Lógicos		
Operador	Operação	Exemplo de Expressões
<u>e</u>	Conjunção	$6 = 5 \text{ e } 9 \leq 9 \rightarrow \text{.falso.}$
<u>ou</u>	Disjunção	$6 = 5 \text{ ou } 9 \leq 9 \rightarrow \text{.verdadeiro.}$
<u>não</u>	Negação	$\text{não}(9 \leq 9) \rightarrow \text{.falso.}$ $\text{não}(6 = 5) \rightarrow \text{.verdadeiro.}$

Para facilitar o entendimento das expressões relacionais e lógicas, a tabela 3 apresenta todos os resultados possíveis da combinação dos operadores lógicos quando se avalia duas ou mais expressões.

Tabela 3 – Resultado de Operações Lógicas

Proposição 1	Operador	Proposição 2	Resultado
V	<u>e</u>	V	V
V	<u>e</u>	F	F
F	<u>e</u>	V	F
F	<u>e</u>	F	F
V	<u>ou</u>	V	V
V	<u>ou</u>	F	V
F	<u>ou</u>	V	V
F	<u>ou</u>	F	F
Operador		Proposição	Resultado
<u>não</u>		V	F
<u>não</u>		F	V

1.6.4 Comandos Básicos

Os comandos básicos são utilizados para você definir determinadas ações que serão executadas com os operandos dentro de um algoritmo.

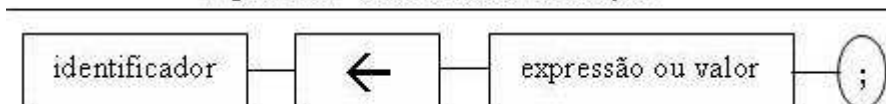
1.6.4.1 Comando de Atribuição

O comando de atribuição é utilizado quando se deseja atribuir determinado valor ou expressão a uma variável, ou seja, se deseja armazenar certo conteúdo em uma variável para posterior manipulação.

Para realizar a operação de atribuição, no pseudocódigo utilizamos o símbolo gráfico seta voltada para a esquerda (\leftarrow), representando desta forma que o valor ou resultado da expressão mostrada à direita do símbolo deve ser armazenado na variável definida à esquerda do símbolo.

A figura 1.17 apresenta a sintaxe a ser utilizada na formatação do comando de atribuição.

Figura 1.17 – Comando de Atribuição



Exemplos de comandos de atribuição:

a) Atribuição de um valor a uma variável:

inteiro: A;

real: B;

caractere: LETRA;

caractere: NOME[60];

lógico: TESTE;

A \leftarrow 5;

B \leftarrow 3.6;

LETRA \leftarrow 'm';

NOME \leftarrow "Governo Federal";

Se (A > B)

então

TESTE ← .verdadeiro.

senão

TESTE ← .falso.

fimse;

Observe que para atribuir determinado valor a uma variável, inicialmente a variável precisa ser criada (declarada) para posterior utilização.

b) Atribuição de expressões a uma variável:

inteiro: A;

real: B;

$A \leftarrow 8 + 5;$

O conteúdo armazenado em A = 13

$B \leftarrow 5.7 - 4.3;$

O conteúdo armazenado em B = 1.4

Conforme os exemplos mostrados acima, verifica-se que não cabe comando de **atribuição de expressões** para valores do tipo caractere e cadeia de caracteres, por não haver possibilidade de se realizar operações matemáticas com esses tipos de dados.

Outra observação importante descrita por GUIMARÃES e LAGES (1985) se refere ao resultado da expressão do lado direito de um comando de atribuição. O resultado de uma expressão deve ser coerente com o tipo declarado para a variável do lado esquerdo. Por exemplo, o comando de atribuição:

$M \leftarrow K < = P;$

O resultado armazenado em **M** só tem sentido se **M** for declarada como tipo lógico, pois os valores possível para o resultado da expressão **K <= P** somente pode ser **.verdadeiro.** ou **.falso.,** ou seja, uma dado do tipo **lógico.**[4]

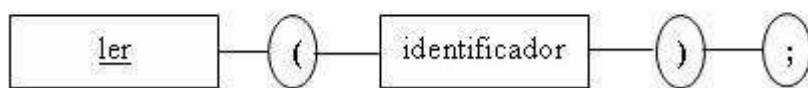
1.6.4.2 Comando de Entrada

Para executar operação no computador, fatalmente será necessário obter dados do exterior para efetuar a manipulação desses dados. A obtenção de valores do exterior é realizada através das instruções de leitura de dados(entrada de dados).[2]

Os **comandos de entrada** são usados para representar a entrada de dados com o uso dos dispositivos de entrada do computador, como por exemplo, o teclado. O comando de entrada representa a digitação de um dado para ser armazenado na memória principal do computador.

Para representar o comando de entrada usamos a palavra **ler,** indicando sempre entrada de dados. O formato geral da entrada de dados é mostrado na figura 1.14.

Figura 1.18 – Comando de Entrada



As entradas de dados estão intimamente relacionadas com os **tipos** definidos para as variáveis. Considere a entrada de um valor **tipo real** para armazenamento no computador. Para que isso seja possível, torna-se necessário que a variável criada para recebê-lo tenha sido criada com esse tipo específico, caso contrário, não será possível armazenar este valor na variável.

Exemplo de entrada de dados:

ler(A);

ler(NOME);

1.6.4.3 Comando de Saída

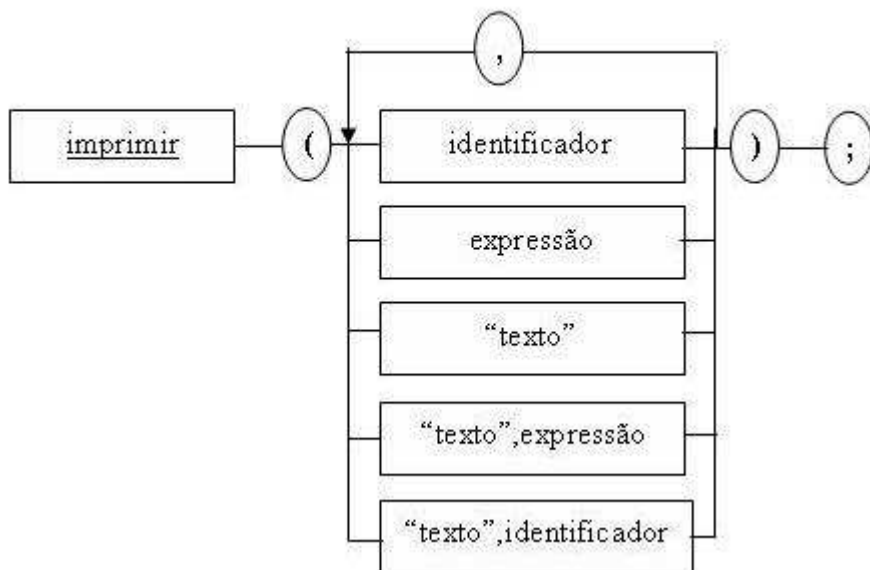
Em todas as operações de manipulação de dados pelo computador, com certeza será necessário a exibição do resultado, comunicando ao mundo exterior a conclusão a que se chegou. Essa operação é realizada através da escrita de dados nas unidades de saída com comandos específicos.

O comando de saída é utilizado para mostrar o resultado das operações executadas pelo computador, apresentando, conforme definido pelo usuário, a resposta esperada.

No comando de saída será exibida a resposta ao usuário conforme o formato especificado no algoritmo, podendo apresentar uma resposta simples, ou ainda, dados adicionais para melhor compreensão dos resultados.

A figura 1.19 mostra o padrão geral para formatação da saída, e em seguida, a função de cada opção e alguns exemplos para melhor entendimento da sintaxe utilizada.


Figura 1.19 – Comando de Saída



Funções dos comandos de saída, conforme modelo escolhido e exemplos de utilização:

a) **Saída com identificador** – Normalmente usada para exibição do conteúdo de uma variável. Nenhuma outra informação além do conteúdo é exibida na unidade de saída do computador.


Exemplo de saída com identificador:

Algoritmo	Unidade de Saída
... <u>inteiro</u> : A; A ← 5; <u>imprimir</u> (A); ...	

b) **Saída com expressão** – As saídas com o uso de expressões são usadas para exibir o resultado da operação definida na expressão sem a necessidade de armazenar o resultado em alguma variável.

Convém analisar com cuidado suas necessidades no algoritmo para evitar o uso de linhas de comandos em seu algoritmo sem a real necessidade ou ainda a falta destes. O bom algoritmo deve conter os dados necessários para execução das tarefas solicitadas com o uso do menor número possível de linhas de código.

Exemplo de saída com expressão:

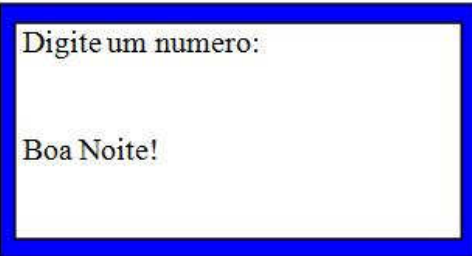
Algoritmo	Unidade de Saída
... <u>inteiro</u> : A,B; A ← 5; B ← 3; <u>imprimir</u> (A+B); ...	

Observe neste exemplo que o comando no algoritmo indica a exibição de **A+B** e o que é exibido na saída é apenas o resultado da operação solicitada (**8**), e não os valores da fórmula apresentada. Neste caso, o valor da somatória não foi armazenado em nenhuma variável, apenas mostrado na unidade de saída.

c) **Saída com “texto”** – Esse tipo de saída é utilizada para mostrar exatamente o texto informado entre as aspas duplas, do mesmo modo que foi escrito. Convém observar que o conteúdo do texto entre aspas não é analisado pelo programa, apenas mostrado na unidade de saída.

Essa opção é muito utilizada para informar ou solicitar alguma tarefa ao usuário, como por exemplo, antes de realizar a entrada de um dado (leitura).

Exemplo de saída com “texto”:

Algoritmo	Unidade de Saída
... <code>imprimir("Digite um numero:");</code> ... <code>imprimir("Boa Noite!");</code> ...	 <p>Digite um numero: Boa Noite!</p>

d) **Saída com “texto” seguido de expressão** – Opção muito utilizada para exibir respostas de operações executadas pelo usuário com informações sobre o conteúdo da resposta.

Exemplo de saída com “texto” seguida de expressão:

Algoritmo	Unidade de Saída
<pre> ... inteiro: A,B; A ← 7; B ← 9; imprimir("Soma=",A+B); ... </pre>	<div style="border: 2px solid blue; padding: 5px; width: fit-content;"> <p>Soma=16</p> </div>

Existe a possibilidade de utilizarmos diversas combinações desse tipo de estrutura, sempre informando o texto entre aspas duplas quando se deseja a exibição de determinado texto na unidade de saída e o uso de identificadores (variáveis) sem aspas, separados por vírgula, quando se deseja apresentar o conteúdo de uma variável.

Outro exemplo de saída com "texto" seguida de expressão:

Algoritmo	Unidade de Saída
<pre> ... inteiro: A,B; A ← 7; B ← 9; imprimir("Soma entre ",A, "e", B, "e igual a ",A+B); ... </pre>	<div style="border: 2px solid blue; padding: 5px; width: fit-content;"> <p>Soma entre 7 e 9 e igual a 16</p> </div>

A ordem de exibição das informações é definida pelo usuário, tendo sempre em mente que, o texto colocado entre aspas será exibido tal qual foi escrito e onde aparecer o identificador, será mostrado o conteúdo da variável no momento da execução do algoritmo, como mostrado a seguir:

Algoritmo	Unidade de Saída
<pre> ... inteiro: A,B; A ← 7; B ← 9; imprimir(A+B, "e a soma de ", A, "com", B); ... </pre>	<p>16 e a soma de 7 com 9</p>

e) **Saída com “texto” seguido de identificador** – Este formato segue o mesmo padrão da saída de texto seguida de expressão, tendo como única diferença a utilização de identificadores no lugar de expressões.

Exemplo de utilização de “texto” seguido de identificadores:

Algoritmo	Unidade de Saída
<pre> ... inteiro: A,B,RESPOSTA; A ← 7; B ← 9; RESPOSTA ← A + B; imprimir("Soma entre ", A, "e", B, "e igual a ", RESPOSTA); ... </pre>	<p>Soma entre 7 e 9 e igual a 16</p>

Esta opção é muito utilizada para exibir respostas de operações executadas pelo usuário com informações sobre o conteúdo da resposta e da operação executada.

1.7 REGRAS PRÁTICAS E METODOLOGIA PARA ELABORAÇÃO DE ALGORITMOS

Guimarães e Lages (1985) sugerem algumas regras práticas e uma metodologia a serem utilizadas na elaboração de algoritmos, visando um melhor entendimento das operações definidas e também garantir uma estrutura adequada a solução proposta.

1.7.1 Regras Práticas

- a) Procure incorporar comentários no algoritmo, pelo menos para descrever o significado das variáveis utilizadas;
- b) Escolha nomes de variáveis que sejam significativas, isto é, traduzam o tipo de informação a ser armazenada na variável;
- c) Grife todas as palavras-chave (escritas com letras minúsculas) do algoritmo, destacando as estruturas de controle;
- d) Procure alinhar os comandos de acordo com o nível a que pertençam, isto é, destaque a estrutura na qual estão contidos.

1.7.2 Metodologia no Desenvolvimento de Algoritmos

- a) Leia cuidadosamente a especificação do problema até o final;
- b) Enquanto não entender a tarefa solicitada, leia outras vezes e/ou solicite mais informações sobre a solicitação do usuário para que não se tenha dúvidas do que se deseja realizar;
- c) Levantar e analisar todas as entradas necessárias para a resolução do problema, sem as quais você não conseguirá resolver o problema;
- d) Levantar e analisar todas as saídas exigidas na especificação do problema e a forma que será apresentada ao usuário;
- e) Verificar se é necessário gerar valores internamente ao algoritmo e levantar as variáveis necessárias e os valores iniciais de cada uma;
- f) Levantar e analisar todas as transformações necessárias para, dadas as entradas e valores gerados internamente, produzir as saídas especificadas;
- g) Testar cada passo do algoritmo, verificando se as transformações intermediárias executadas estão conduzindo aos objetivos desejados. Utilizar, sempre que possível, valores de teste que permitam prever os resultados;
- h) Fazer uma reavaliação geral, elaborando o algoritmo através da integração das partes.

1.8 EXERCÍCIOS RESOLVIDOS

- a) Fazer um algoritmo para calcular a média final (média aritmética) de um aluno, considerando a realização de quatro avaliações.

inicio

```
real: N1, N2, N3, N4, MEDIA;  
imprimir (“Digite a 1 nota:”);  
ler (N1);  
imprimir (“Digite a 2 nota:”);  
ler (N2);  
imprimir (“Digite a 3 nota:”);  
ler (N3);  
imprimir (“Digite a 4 nota:”);  
ler (N4);  
MEDIA ← ( N1 + N2 + N3 + N4 ) / 4;  
imprimir (“Media final:”,MEDIA);
```

fim.

- b) Faça um algoritmo para ler dois valores inteiros representando, respectivamente, um valor de hora e um de minutos. Calcular e exibir quantos minutos se passou desde o início do dia.

inicio

```
inteiro: HORA, MINUTOS, TMIN;  
imprimir (“Informe a hora:”);  
ler (HORA);  
imprimir (“Informe os minutos:”);  
ler (MINUTOS);  
TMIN ← (HORA * 60) + MINUTOS;  
imprimir (“Se passaram “, TMIN, “ minutos desde o início do dia.”);
```

fim.

- c) Fazer um algoritmo para inverter a ordem de determinado número de quatro dígitos.

inicio

inteiro: NUM, N1, N2, N3, N4;

imprimir (“Informe um número de 4 dígitos:”);

ler (NUM);

$N1 \leftarrow \text{NUM} \text{ div } 1000;$

$N2 \leftarrow (\text{NUM} \text{ mod } 1000) \text{ div } 100;$

$N3 \leftarrow ((\text{NUM} \text{ mod } 1000) \text{ mod } 100) \text{ div } 10;$

$N4 \leftarrow ((\text{NUM} \text{ mod } 1000) \text{ mod } 100) \text{ mod } 10;$

imprimir (N4, N3, N2, N1);

fim.

- d) Considerando duas variáveis inteiras quaisquer, após a entrada de dados, fazer a troca do conteúdo de uma pelo conteúdo da outra.

Ex: A = 25 e B = 4 → Após a execução do algoritmo: A=4 e B=25

inicio

inteiro: A, B, C;

imprimir (“Informe um número:”);

ler (A);

imprimir (“Informe outro número:”);

ler (B);

$C \leftarrow A;$

$A \leftarrow B;$

$B \leftarrow C;$

imprimir (“A=”,A);

imprimir (“B=”,B);

fim.

REFERÊNCIAS BÁSICAS

- [1] FARRER, Harry. et. al. **Programação estruturada de Computadores. Algoritmos estruturados.** 2 Ed. Rio de Janeiro: LTC Livros Técnicos e Científicos Editora Ltda, 1989.
- [2] MARTINS, J. Pavão. **Introdução à programação usando Pascal.** Lisboa: Editora McGraw-Hill de Portugal Lda, 1994.
- [3] ARAÚJO, Everton Coimbra de. **Algoritmos: Fundamentos e Prática.** 2 Ed. ampl. e atual. Florianópolis: VisualBooks Editora, 2005.
- [4] GUIMARÃES, Ângelo de Moura. LAGES, Newton Alberto de Castilho. **Algoritmos e estruturas de dados.** Rio de Janeiro: Campus, 1985.
- [5] SCHILDT, Herbert. **C Completo e total.** São Paulo: MAKRON BOOKS, 1997.
- [6] LOUDON, Kyle. **Dominando algoritmos com C.** São Paulo: CIENCIA MODERNA COMPUTAÇÃO, 2000.
- [7] LAUREANO, Marcos. **Programando em C para Linux, Unix e Windows.** Rio de Janeiro: Brasport Livros, 2005.
- [8] MEDINA, Marco. FERTIG, Cristina. **Algoritmos e programação: teoria e prática.** São Paulo: NOVATEC INFORMATICA, 2005.
- [9] FERREIRA, Aurélio Buarque de Holanda. **Novo Dicionário Aurélio Século XXI.** Rio de Janeiro: Nova Fronteira, 1999.
- [10] INFORMAÇÃO. In: Wikipédia: a enciclopédia livre. Disponível em: <<http://pt.wikipedia.org/wiki/Informa%C3%A7%C3%A3o>>. Acesso em: 15/07/2009.
- [11] PEREZ, Anderson Luiz Fernandes. **Linguagens de programação: sintaxe e semântica de linguagens de programação e conceitos de linguagens compiladas e interpretadas.** Disponível em <<http://www.univasf.edu.br/~anderson.perez/ensino/intprog/>> acesso em 16/07/2009.
- [12] BARBOSA, Lisbete Madsen. **Ensino de algoritmos em cursos de computação.** São Paulo: EDUC, 2001.
- [13] BIT. In: Wikipédia: a enciclopédia livre. Disponível em: <<http://pt.wikipedia.org/wiki/Bit>>. Acesso em: 15/07/2009.
- [14] CÓDIGO-FONTE. In: Wikipédia: a enciclopédia livre. Disponível em: <<http://pt.wikipedia.org/wiki/C%C3%B3digo-fonte>>.
- Acesso em: 15/07/2009.
- [15] PI (Π). In: Wikipédia: a enciclopédia livre. Disponível em: <<http://pt.wikipedia.org/wiki/Pi>>. Acesso em: 17/07/2009.
- [16] BUGS. In: Wikipédia: a enciclopédia livre. Disponível em: <<http://pt.wikipedia.org/wiki/Bug>> . Acesso em: 15/08/2009.

[17] FREEDMAN, Alan. **Dicionário de Informática**. São Paulo: Makron Books, 1995.

[18] RAIZ QUADRADA. Wikipédia a Enciclopédia Livre. Disponível em <http://pt.wikipedia.org/wiki/Raiz_quadrada> acesso em 30/07/2009.

REFERÊNCIAS COMPLEMENTARES

SCHILDT, Herbert. **C Completo e total**. São Paulo: MAKRON BOOKS, 1997.

DAMAS, Luís. **Linguagem C**. 10. ed. Rio de Janeiro: LTC, 2007.

LOUDON, Kyle. **Dominando algoritmos com C**. São Paulo: CIENCIA MODERNA COMPUTAÇÃO, 2000.

JAMSA, Kris. **Programando em C/C++: a bíblia**. São Paulo: Makron Books, 2000.

LOPES, Anita. **Introdução a programação: 500 algoritmos resolvidos**. Rio de Janeiro: Campus, 2002.

LAUREANO, Marcos. **Programando em C para Linux, Unix e Windows**. Rio de Janeiro: BRASPORT LIVROS, 2005.

DEITEL, Paul J.; DEITEL, Harvey M. **C++ como programar**. Porto Alegre: Bookman, 2001.

PINTO, Wilson Silva. **Introdução ao desenvolvimento de algoritmos e estrutura de dados**. São Paulo: Érica, 1990. KERNIGHAN, Brian W.; RITCHIE, Dennis M. **C: a linguagem de programação**. Rio de Janeiro: Campus, 2000. (005.133 K39c).

MANZANO, José Augusto N. G.; OLIVEIRA, Jayr Figueiredo. **Estudo dirigido de algoritmos**. São Paulo: Érica, 1997. SALVETTI, Dirceu Douglas; BARBOSA, Lisbete Madsen. **Algoritmos**. São Paulo: Makron Books, 1998. (005.1. S183A).

MIZRAHI, Victorine Viviane. **Treinamento em linguagem C: curso completo**. Módulo I. São Paulo: McGraw-Hill, 1990.