Algoritmos I

Marco Aurélio Freitas Santos

M.Sc. Ciência da Computação
UNIGRAN - Dourados/MS
marcoafs@unigran.br
marco@unigran.br
marco@dourados.br

APRESENTAÇÃO DO AUTOR

Marco Aurélio Freitas Santos, nascido em Coronel Fabriciano-MG é graduado em Ciência da Computação e mestre em Ciência da Computação pela UFRGS-Universidade Federal do Rio Grande do Sul. Analista de Sistemas, atua no mercado de informática de Mato Grosso do Sul e em outros Estados desde 1992, desenvolvendo sistemas para empresas de diversos ramos de atividade. É professor no curso de Ciência da Computação da Unigran desde 1995. Já ministrou aulas nas disciplinas de Linguagens e técnicas de programação, Inteligência Artificial, Programação Comercial e Análise de Algoritmos. Atualmente ministra aulas de Engenharia de Software, Análise e Projeto de Sistemas, Tópicos Especiais em Sistemas de Informação, Algoritmos e Programação de Computadores, Paradigmas de Linguagens de Programação e Banco de Dados. É Analista de Sistemas do Departamento de Informática da Unigran, projetando e desenvolvendo sistemas desta instituição. Foi professor da Universidade Estadual de Mato Grosso do Sul e de instituições particulares.

APRESENTAÇÃO DA DISCIPLINA

Caros alunos,

A disciplina de Algoritmos I possui uma carga horária de 80 horas. Os objetivos da disciplina são: Estudar os conceitos de algoritmos, estruturas de controle e princípios de programação estruturada, modularização e estruturas de dados.

OBJETIVOS:

- Gerais:

Incentivar a capacidade de pesquisa e desenvolvimento científico. Capacitar o aluno a aplicar o conhecimento adquirido de forma independente e inovadora e que possa acompanhar as constantes evoluções para contribuir na busca de soluções nas diferentes áreas de aplicação.

- Específicos:

Propiciar ao aluno o desenvolvimento da lógica de programação através da matemática e da elaboração de algoritmos. Dotar o aluno com princípios do bom desenvolvimento de algoritmos através do estudo de algoritmos básicos e sua correção.

COMPETÊNCIAS E HABILIDADES:

A disciplina introduz o aluno em uma lógica de programação proporcionando o conhecimento básico em estruturas que serão necessários para o aprendizado de qualquer linguagem de programação tendo como objetivo desenvolver suas habilidades para a resolução de problemas computacionais independentemente da linguagem de programação. Também temos como objetivo exercitar o raciocínio lógico e desenvolver a capacidade de abstração do aluno.

CONTEÚDO PROGRAMÁTICO DA DISCIPLINA:

1. ALGORITMOS E ETAPAS DE PROGRAMAÇÃO
    1.1. Introdução à Computação
    1.2. Tipos Primitivos
    1.3. Variáveis e Funções Pré-Definidas
    1.4. Comandos de Atribuição, Entrada e Saída
    1.5. Estruturas de Controle
        1.5.1. Seqüencial
        1.5.2. Condicional
        1.5.3. Repetição
    1.6. Modularização
        1.6.1. Procedimentos
        1.6.2. Funções

CRITÉRIOS DE AVALIAÇÃO:

A avaliação será feita com trabalhos em classe e extra-classe, individuais e em equipe e provas bimestrais teóricas e práticas.

A média bimestral será calculada da seguinte forma:

Mb = (Mt + Pb) / 10

Onde: Mb = Média bimestral

Pb = Prova bimestral multiplicada pelo peso (que pode variar em cada prova)

Mt = Média dos trabalhos multiplicados pelo peso (que pode variar em cada trabalho)

Sejam bem-vindos!

Prof. Marco Aurélio Freitas Santos

Capítulo 1

CONCEITO DE ALGORITMO

1 INTRODUÇÃO

Algoritmos são seqüências de ações ou instruções organizadas logicamente para resolver um problema. Os algoritmos são o primeiro passo para a construção de programas. Através deles desenvolvemos o raciocínio lógico necessário para a resolução do problema proposto. Guimarães & Lages definem algoritmo como:

> "[....] é a descrição de um padrão de comportamento, expressado em termos de um repertório bem definido e finito de ações primitivas, das quais damos por certo que elas podem ser executadas." (GUIMARÃES; LAGES, 1994)

ou ainda:

> "[...] uma norma executável para estabelecer um certo efeito desejado, que na prática será geralmente a obtenção de uma solução a um certo tipo de problema." (GUIMARÃES, 1994)

Para todas as tarefas que executamos no dia a dia, nosso cérebro constrói algoritmos, porém isso acontece de uma forma tão natural que nem percebemos, mas para cada problema ou atividade a ser trabalhada, definimos uma seqüência lógica de ações. Podemos afirmar então que construímos vários algoritmos todos os dias, mas não nos damos conta disso.

Os algoritmos são utilizados em praticamente todas as áreas do conhecimento existentes, pois as pessoas envolvidas na resolução dos mais diversos tipos de problemas precisam formular uma seqüência de ações para chegar a um resultado satisfatório.

Vamos imaginar uma situação da vida real, onde podemos perceber a criação de um algoritmo. Por exemplo, para assistir um filme em DVD, algumas seqüências de ações deverão ser executadas. Vamos considerar que o aparelho de DVD já está conectado a uma TV e que ambos estão prontos para funcionar, deveremos então executar os seguintes passos:

1 – Ligar a TV
2 – Ligar o DVD
3 – Abrir o compartimento do disco
4 – Inserir o disco
5 – Fechar o compartimento do disco
6 – Pressionar a tecla PLAY para iniciar o filme

Esta seqüência de instruções precisou ser definida para que fosse possível assistir o filme no aparelho de DVD. Pra nós pode parecer uma coisa muito simples, mas tivemos que montar uma seqüência de instruções para que fosse possível atender à necessidade (problema) em questão.

Em nossa vida cotidiana encontraremos diversas situações em que teremos que desenvolver algoritmos para resolver determinados problemas, tais como: trocar o pneu de um carro, preparar um bolo, arrumar o filho para a escola, etc.

É importante notar que para cada problema a ser resolvido, existem diversos caminhos que levam à solução desejada, ou seja, um problema pode ser resolvido de duas ou mais maneiras diferentes, obtendo o mesmo resultado, ou ainda poderemos ter umas soluções melhores que outras pra atingir o mesmo objetivo. No exemplo do DVD, por exemplo, ao invés de primeiro ligar a TV, poderíamos ligar o DVD e depois a TV, isto seria uma outra seqüência para a solução do problema, porém teríamos o mesmo resultado. Outra solução para o problema seria criar uma lista de ações com mais detalhes, como por exemplo acrescentar a ação de ajustar o canal da TV ou pressionar a tecla TV/VÍDEO para ajustar a sintonia com o aparelho de DVD.

O nível de detalhamento do algoritmo varia de acordo com o problema a ser resolvido. Isto não quer dizer que uma solução mais detalhada seja melhor ou pior que outra menos detalhada. A decisão de usar mais ou menos ações para atingir o objetivo deve ser analisada em cada situação separadamente.

No computador vale a mesma regra, um problema poderá ter duas ou mais soluções diferentes para chegar ao objetivo, que é a solução do problema proposto.

1.1 Percepção do problema

Diferente do que muitos profissionais que trabalham com algoritmos e programação, a primeira coisa a se pensar não é na solução do problema, o primeiro passo para a criação de um algoritmo é a percepção do problema. Muitas pessoas sentem dificuldade em iniciar o desenvolvimento do algoritmo porque focam suas idéias iniciais na solução e não no problema. Nenhum problema poderá ser resolvido se não for minuciosamente entendido, portanto o foco inicial deve ser em cima do problema para depois se pensar na solução.

O entendimento detalhado do problema é um fator que diferencia uma solução algorítmica da outra. Quanto mais se conhece sobre o problema em questão, maiores as chances de se criar soluções melhores. Daí surgem alguns desafios para a criação do algoritmo, o primeiro é o de usar uma boa parcela de criatividade para criar uma ou mais alternativas de solução para o problema, em seguida, transformar esta idéia abstrata em algo em forma de seqüência de ações.

A arte de criar algoritmos não deve ser encarada apenas como a criação de uma solução para o problema, mas sim como a criação da melhor solução possível. Como já foi comentado anteriormente, podem existir várias soluções para o mesmo problema, cabe então ao desenvolvedor do algoritmo pensar em duas ou mais soluções diferentes para poder analisá-las, compará-las e decidir qual delas é a melhor. Vale lembrar que o tamanho ou complexidade do algoritmo não é relativamente proporcional à sua qualidade.

Paulo A. S. Veloso afirma que:

> "A construção metódica de programas confiáveis é, sem dúvida, uma das questões centrais da Ciência da Computação". (VELOSO, 1987).

Para cada problema a ser levado ao computador, deve-se planejar as operações correspondentes. O automatismo exige que o planejamento destas operações seja feito previamente, antes de se utilizar o computador.

Então, a utilização de um computador para se resolver qualquer problema exige, antes de mais nada, que se desenvolva um algoritmo, isto é, que se faça a descrição do conjunto de comandos ordenados que, quando obedecidos, resultarão na realização da tarefa desejada obtendo os resultados esperados. Os algoritmos encontram-se entre a idéia de se resolver um problema através do computador e o sistema desenvolvido pelas linguagens de programação.

A elaboração de algoritmos é, para estudantes de Ciência da Computação e áreas afins, o passo inicial para a solução de qualquer problema computacional. Isto significa que, identificado o problema, será necessária a escrita de uma série de instruções, normalmente escritas em português, ou bem próximas dele, numa ordem logicamente definida, para que se obtenha a solução final do problema. É deste modo que se torna possível a construção de programas em qualquer linguagem de computador.

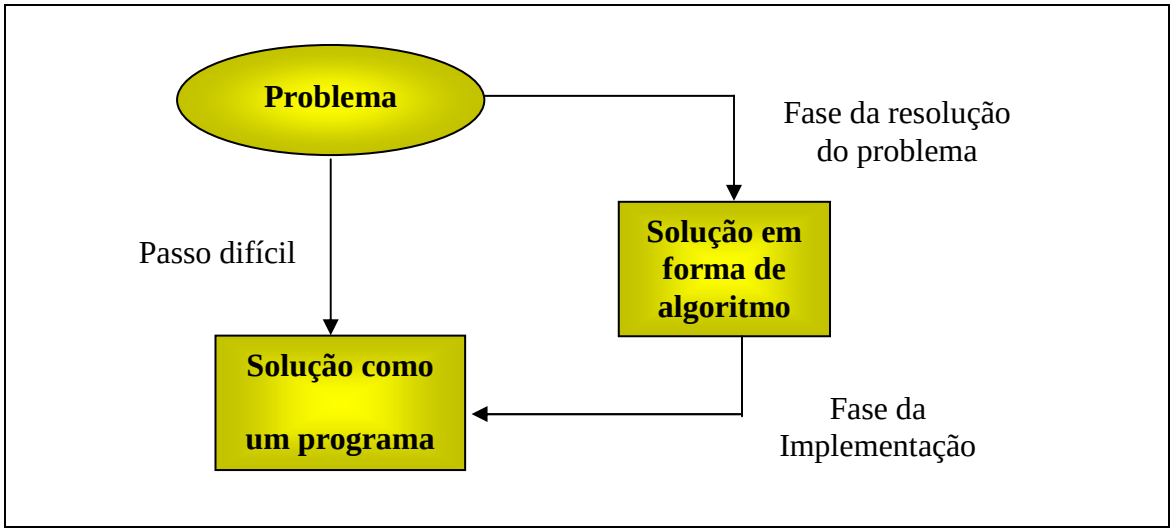

Figura 1.1 - Resolução do problema e programação.

2 ALGORITMOS ESTRUTURADOS

Desenvolver algoritmos estruturados é basicamente utilizar uma metodologia de projeto de programas, com os seguintes objetivos:

- Facilitar a escrita dos algoritmos;
- Facilitar a leitura e o entendimento dos algoritmos;
- Facilitar a manutenção e modificação dos algoritmos.

O grande desafio da escrita de programas e reduzir a complexidade dos mesmos. Muitos programas são escritos de uma forma tão complicada que até o próprio autor tem dificuldade de interpretá-lo depois. Para reduzir a complexidade dos algoritmos deve-se:

- Escrever o algoritmo fazendo refinamentos sucessivos (chamado de **Desenvolvimento *Top-Down***) – esta técnica consiste em ir escrevendo as funções principais do programa e depois detalhar cada uma delas em funções menores, até que se tenha um último nível que não é mais possível detalhar;
- Decompor o algoritmo todo em módulos funcionais – é mais fácil compreender uma solução se a analisarmos por partes, ao invés de olharmos a solução como um todo. A este processo chamamos de **Modularização**;
- Usar soluções simples e não muito extensas em cada um dos módulos, com poucas **estruturas de controle** para facilitar o entendimento.

2.1 Desenvolvimento Top-Down

Depois de entendermos o problema a ser solucionado, precisamos formular alternativas de solução para o mesmo. Para isto, formulamos uma possível solução, porém, esta solução precisa ser refinada, ou seja, detalhada em partes menores até que se tenha uma visão de todos os detalhes.

2.2 Modularização

A modularização consiste em dividir a solução do problema em partes, ou módulos, cada um com funções bem definidas, dando maior agilidade ao desenvolvimento.

2.3 Estruturas de controle

O uso de estruturas de controle adequadas afeta diretamente a qualidade do algoritmo. Muitos programadores tem o costume de usar comandos de desvio

incondicional ou de interrupção do algoritmo. Um desvio incondicional é aquele em que o programa muda seu fluxo de execução arbitrariamente. Este artifício é muito usado quando não se tem um completo domínio da lógica do programa, mas deve ser evitado, pois ele provoca a quebra da estrutura lógica do algoritmo.

3 PROGRAMAS E LINGUAGENS DE PROGRAMAÇÃO

Escrever um programa é o mesmo que traduzir um algoritmo para uma linguagem de programação qualquer. As linguagens de programação são softwares que permitem transformar um algoritmo em um programa de computador.

Aprender uma nova linguagem de programação é uma tarefa fácil quando se tem um bom conhecimento de algoritmos, pois o maior problema na criação de um programa não é a linguagem em si, mas sim as dificuldades na percepção do problema e na formulação de uma solução, ou melhor, de uma boa solução.

Existem linguagens para os mais diversos domínios de aplicação, cada uma com seus propósitos. Algumas são destinadas ao desenvolvimento de aplicações comerciais, outras têm propósito científico, para aplicações em inteligência artificial e também aquelas para criação de programas para internet, etc.

As linguagens podem ser interpretadas, compiladas ou híbridas.

3.1 LINGUAGENS INTERPRETADAS

Nas linguagens interpretadas, o interpretador lê, analisa e executa cada instrução do programa fonte, sem traduzir para uma linguagem de máquina. Cada linha ou instrução é executada na seqüência. Quando um erro é encontrado, a execução do programa é interrompida.

O interpretador simula por software uma máquina virtual, onde o ciclo de execução entende os comandos da linguagem de alto nível.

Este tipo de linguagem oferece algumas desvantagens, como a necessidade da presença do código fonte para a execução do programa. Também o interpretador da linguagem precisa estar instalado no computador onde está o programa. A execução do programa é bem mais lenta que nas linguagens compiladas, pois o interpretador precisa analisar as instruções do programa sempre que vai executá-las.

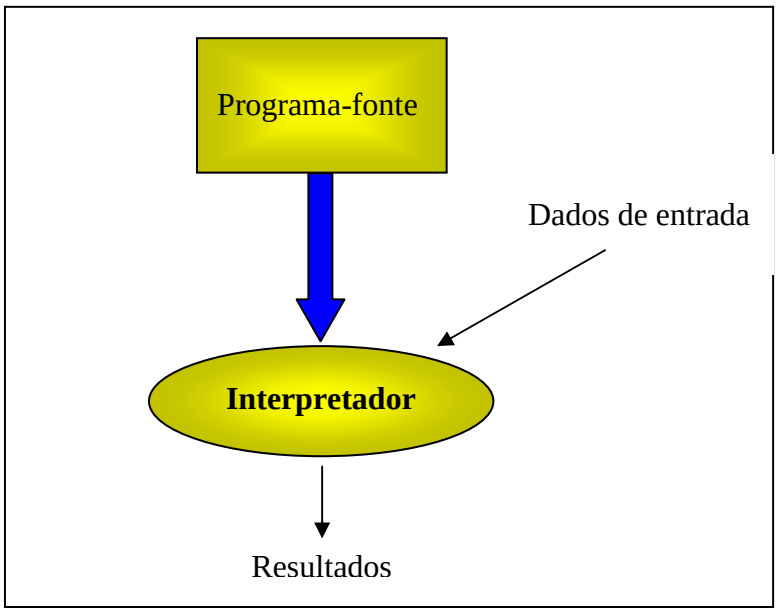

Figura 1.2 – Funcionamento das linguagens interpretadas

3.2 LINGUAGENS COMPILADAS

Os compiladores tem a tarefa de ler e analisar o programa escrito em uma linguagem de programação, o programa fonte ou código fonte, e traduzi-los para a linguagem de máquina, executando-os diretamente no computador. Diferente da interpretação, a compilação analisa todo o código fonte a procura de erros, só depois que esta análise termina e que nenhum erro tenha sido encontrado, é que será criado um código intermediário chamado de "código objeto". Este código objeto será então linkeditado (ou ligado) e um código executável será gerado.

Nos programas compilados, não há a necessidade da presença do código fonte para a sua execução, assim como o compilador não precisa estar instalado para isto. O programa executável criado é independente da linguagem. Neste caso, a execução do programa é mais rápida, uma vez que o programa fonte não precisará ser analisado em cada execução.

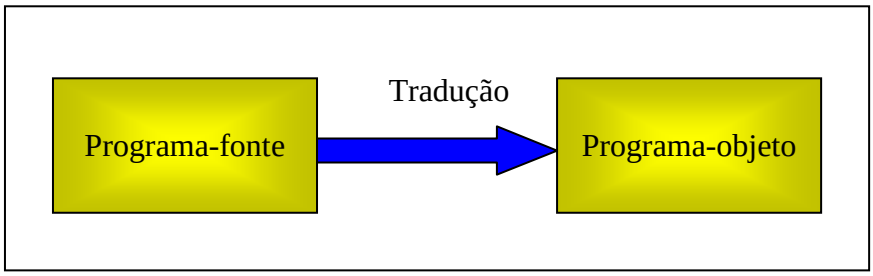

Figura 1.3 – Processo de compilação

3.3 LINGUAGENS HÍBRIDAS

As linguagens híbridas usam a interpretação e a compilação. O compilador tem o papel de converter o código fonte em um código conhecido por *byte code*, que é uma linguagem de baixo nível, que depois é interpretada. Como exemplo disto temos a linguagem JAVA, que gera um código chamado de ***Java Bytecode***.

Programas escritos em uma linguagem híbrida são mais rápidos que os de uma linguagem interpretada, instruções intermediárias são projetadas para serem interpretadas facilmente.

ATIVIDADES

Responda às questões a seguir e envie e-mail para: marco@unigran.br ou marcoafs@unigran.br. No cabeçalho do e-mail coloque as seguintes informações: “Algoritmos I – RGM – Nome”. No corpo do e-mail, informe qual a atividade ou exercício que está sendo enviado.

1) Qual a primeira coisa a se fazer para se desenvolver um algoritmo? Explique o motivo da resposta.

2) Comente algumas diferenças entre linguagens compiladas e interpretadas.

Capítulo 2

LINGUAGEM ALGORITMICA

1 PORTUGOL

Uma linguagem algorítmica é uma pseudo-linguagem de programação, que utiliza comandos e instruções em Português para representar as ações dos algoritmos. A esta pseudo-linguagem damos o nome de **Portugol**, também conhecida como **Português Estruturado**.

A necessidade de facilitar o trabalho em computador por parte dos profissionais de informática é constante e uma das formas para se conseguir este objetivo é fazer com que o computador, cada vez mais, compreenda a linguagem escrita ou falada, reduzindo ao máximo a quantidade de códigos e símbolos que precisam ser aprendidos e memorizados. Por este motivo, estamos sempre buscando formas para fazer com que o computador aprenda a nossa língua, ao invés de nós aprendermos a "língua" dele.

Os programadores teriam seu trabalho facilitado se os programas fossem escritos em sentenças padronizadas da linguagem humana; infelizmente, isso não acontece. Os programas têm de ser escritos em uma linguagem de programação e há muitas dessas linguagens.

Sebesta afirma que, em tese, o estudante de programação deve focalizar seus esforços no entendimento e resolução do problema, bem como no desenvolvimento do raciocínio lógico necessário e da abstração, ou seja, a capacidade de definir e usar estruturas ou operações complicadas, sem visualizar muitos detalhes (SEBESTA, 1999). Isto só será possível se o mesmo não tiver que se preocupar com a tradução de cada um

dos comandos que ele deve utilizar como também não perder tempo por causa de uma mensagem de erro cujo significado não foi entendido corretamente.

Por este motivo, torna-se muito importante o aprendizado de algoritmos para a elaboração de programas estruturados e aprimoramento da lógica de programação, sem a preocupação com o idioma em que se apresentam os programas da linguagem estrutura, bem como a necessidade de conhecimento de uma linguagem de programação específica.

O processo de aprender uma linguagem de programação para resolver os problemas do algoritmo pode ser uma tarefa extensa e difícil. A programação em uma linguagem algorítmica é na verdade uma simples transcrição de palavras-chave, o que torna o processo muito mais fácil. Uma vez pré-definidas as seqüências lógicas das tarefas, ou instruções, a serem realizadas passo a passo, necessita-se apenas traduzi-las em uma linguagem própria, que o computador reconheça, para então submetê-las à máquina para análise e obter o seu resultado.

A vantagem dos algoritmos, portanto, não está no fato de eliminar a adoção de regras comuns na programação, mas no fato do usuário estar escrevendo seu programa em português, o que dará ao programador maior facilidade para compreender e assimilar a lógica do programa, ao mesmo tempo em que exigirá o cumprimento de regras obrigatórias para confecção dos mesmos.

2 GRAMÁTICA DO PORTUGOL

Segundo Leland L. Beck,

> “A gramática de uma linguagem de programação é uma descrição formal da sintaxe, ou forma, dos programas e instruções individuais escritas nessa linguagem”. (BECK, 1997).

O Portugol é constituído de letras maiúsculas e minúsculas ("A" - "Z", "a" - "z"), caracter sublinhado ("_"), os dígitos de 0 a 9 e os símbolos especiais +-*/=<>(){}.,;:'. Seus operadores e delimitadores são os seguintes:

Adição	+
Negação, subtração	-
Multiplicação	*

Divisão	/
Igualdade, declaração	=
Menor que	<
Maior que	>
Maior ou igual a	>=
Menor ou igual a	<=
Desigualdade (diferente de)	<> ou ≠
Precedência	( )
Delimitadores de comentários	{}
Fim do programa, decimal	.
Separador de lista	,
Separador de comando	;
Delimitador literal de seqüência de caracteres	' ou “
Atribuição	←

2.1 CONSTANTES

Constante é um identificador que armazena um valor fixo, ou seja, este valor não se modifica no decorrer do algoritmo. As constantes podem ser do tipo numérica, lógica ou literal.

Constante numérica

Pode ser um número inteiro ou real, positivo, negativo ou nulo.

Ex.:

a) 32;
b) 3,1415;
c) -54;
d) 0,342;

Constante lógica

Constantes lógicas podem assumir um dos seguintes valores: Verdadeiro (V) ou Falso (F).

Constante literal

São valores do tipo caracter, ou seja, é qualquer seqüência de caracteres (letras, dígitos ou símbolos especiais). A constante literal deve sempre aparecer entre aspas.

Ex.:

a) “Castro Alves”;
b) “X1Y2W3”;
c) “*A!B?-/”;

d) “1234”. (como este número aparece entre aspas, não é uma constante numérica)

2.2 VARIÁVEIS

Variáveis também são identificadores que armazenam valores, porém, ao contrário das constantes, o valor de uma variável pode mudar dentro do algoritmo. As variáveis, assim como as constantes, podem ser dos tipos: numérica (inteiro ou real), lógica ou literal.

2.2.1 Nomeando constantes e variáveis

Os nomes de constantes e variáveis (identificadores) são nomes simbólicos para os objetos referenciados nos algoritmos. Estes nomes são escolhidos pelo usuário para representar endereços de memória onde vão ser alocadas as informações. Os identificadores são formados por um ou mais caracteres e devem seguir algumas regras:

1) Os nomes devem começar sempre por um caractere alfabético ou o símbolo “_” (sublinhado);
2) Podem ser constituídos de caracteres alfabéticos ou numéricos;
3) Não podem conter caracteres especiais, como por exemplo: +-*/=<>(){}.,;:'
4) Não podem ter o mesmo nome que comandos ou palavras reservadas do algoritmo ou da linguagem de programação que será usada.

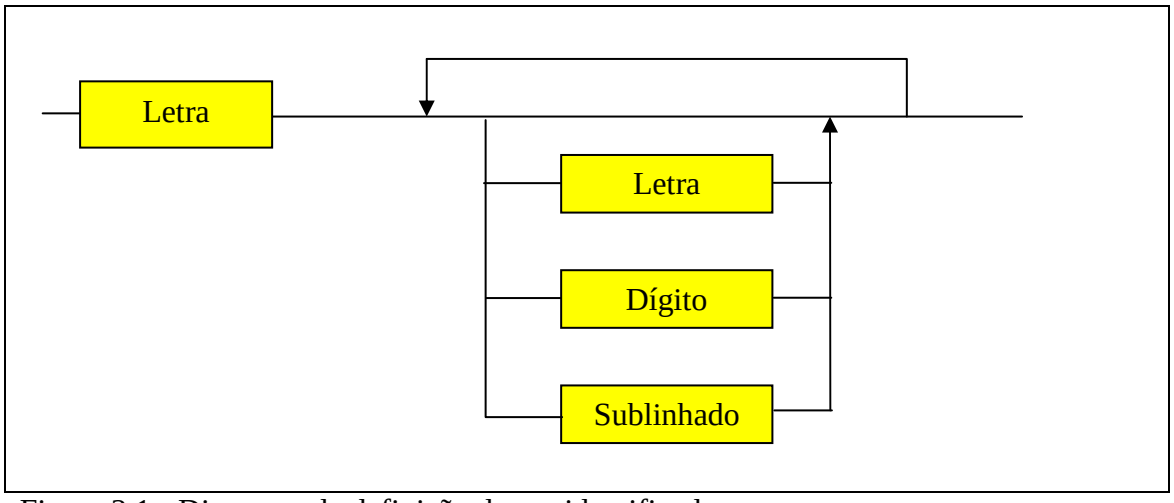

Figura 2.1 - Diagrama da definição de um identificador.

No nome dos identificadores não há distinção entre letras maiúsculas ou minúsculas.

Exemplos de nomes permitidos:

a) cliente b) delta c) X d) BC4R e) MEDIA

f) NOTA_ALUNO g) A h) A123B

Exemplos de nomes **não** permitidos:

a) 5X b) E(13) c) X-Y d) NOTA/2 e) 123TESTE

f) NOME DO CLIENTE

2.2.2 Declaração de variáveis

As variáveis devem ser declaradas logo no início do algoritmo (ou no início do procedimento ou função como veremos mais adiante). A declaração de variáveis deve iniciar pela palavra reservada “declare” e segue a sintaxe apresentada na figura a seguir:

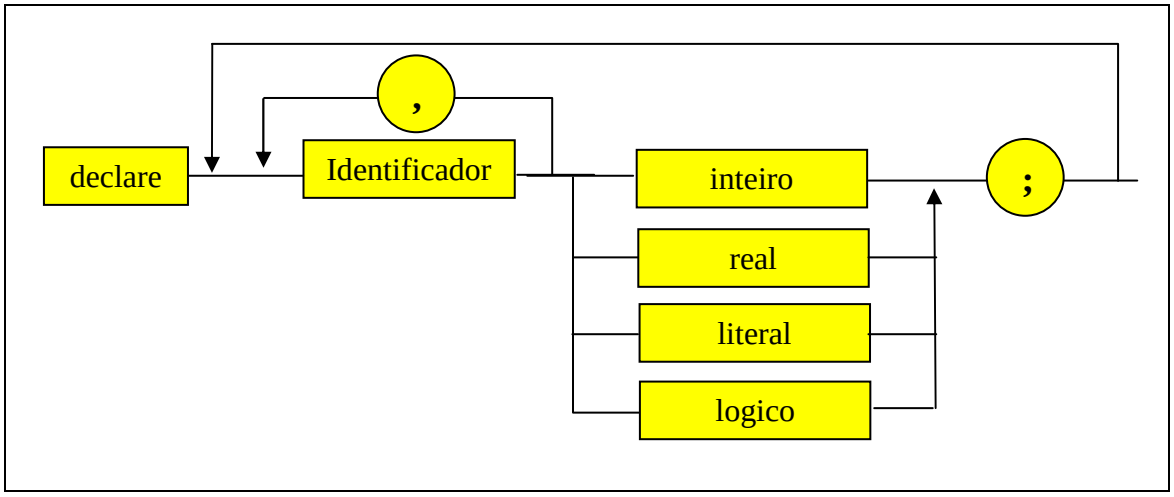

Figura 2.2 - Diagrama da definição de uma variável.

Exemplo:

```
Algoritmo
    Declare VARIAVEL_1, VARIAVEL_2 inteiro;
    Declare VARIAVEL_3, VARIAVEL_4 real;

    Leia VARIAVEL_1, VARIAVEL_2;
    Leia VARIAVEL_3, VARIAVEL_4;
fim-algoritmo
```

Uma variável só pode receber valores do mesmo tipo declarado, ou seja, variáveis do tipo inteiro só podem receber valores inteiros e assim por diante.

2.3 PALAVRAS RESERVADAS

As palavras reservadas são nomes utilizados pelo Algoritmo que tem um sentido predeterminado no mesmo, portanto não podem ser redefinidas pelo usuário como identificadores ou utilizados de outra forma senão para a que foram criadas. Algumas delas são:

se	de	até
então	declare	escreva
senão	fim-algoritmo	procedimento
enquanto	Fim-se	função
faça	Fim-enquanto	início
repita	Fim-para	fim
até-que	para	Inteiro
real	literal	logico

2.4 COMANDOS BÁSICOS

Os comandos especificam as ações a serem realizadas pelo computador, como comparações e atribuições. Eles se constituem por expressões, palavras-chave e operadores.

2.4.1 Atribuição

Para atribuirmos um valor a uma variável usaremos o símbolo de atribuição ←. Sua sintaxe é apresentada na figura a seguir.

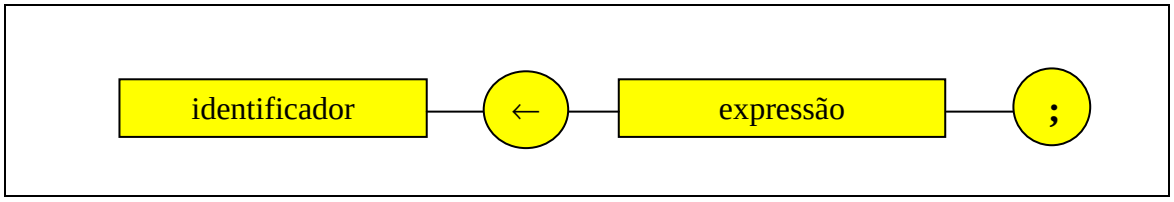

Figura 2.3 - Diagrama da atribuição de valores a identificadores.

Exemplo: VALOR ← (TOTAL1+TOTAL2) * (A/B)

Neste exemplo, a variável VALOR receberá o resultado do cálculo da expressão da direita.

2.4.2 Operadores

Os operadores são fornecidos para possibilitar a formação de vários tipos de expressões.

Os operadores possuem uma seqüência na qual as expressões serão avaliadas e resolvidas. Se dois operadores numa mesma expressão possuírem o mesmo nível de precedência, a expressão será avaliada da esquerda para a direita. As expressões contidas entre parênteses serão resolvidas em primeiro lugar, a começar pelos parênteses mais internos.

A ordem de precedência é a seguinte:

1º - Expressões dentro de parênteses e funções

2º - Operador unário menos ou negação

3º - Operadores aritméticos multiplicativos: *, /

4º - Operadores aritméticos aditivos: +, -

5º - Operadores relacionais: =, <>, <, >, <=, >=

6º - Operadores lógicos: e, ou, não

Operadores Aritméticos:

São os símbolos das quatro operações básicas: +, -, *, /. Eles representam as operações de adição, subtração e negação, multiplicação e divisão respectivamente.

Operadores Relacionais:

Nas expressões que utilizam os operadores relacionais, o resultado da relação será sempre Verdadeiro ou Falso. São eles:

=	igual
<> ou ≠	diferente
>	maior que
<	menor que
>=	maior ou igual a
<=	menor ou igual a

Operadores Lógicos

São eles: - e ou $\wedge$ para **conjunção**;

- ou ou $\vee$ para **disjunção**;

- não ou $\neg$ para a **negação**.

* A conjunção de duas proposições é verdadeira se e somente se ambas as proposições são verdadeiras. Veja tabela abaixo:

p	q	$p \wedge q$
V	V	V
V	F	F
F	V	F
F	F	F

Exemplo: p: OK, onde OK é uma variável lógica com o valor verdadeiro;

q: A = 0, onde o valor de A é 3;

r: TESTE, onde TESTE é falso.

a) $p \wedge q = F$ b) $p \wedge r = F$ c) $q \wedge r = F$

* A disjunção de duas proposições é verdadeira se e somente se, pelo menos, uma delas for verdadeira. Veja a tabela abaixo:

p	q	$p \vee q$
V	V	V
V	F	V
F	V	V
F	F	F

Para o exemplo anterior:

a) $p \vee q = V$ b) $p \vee r = V$ c) $q \vee r = F$

* A negação pode ser formada inserindo-se a palavra não antes da proposição. Veja a tabela abaixo:

p	$\neg p$
V	F
F	V

2.4.3 Expressões

As expressões são constituídas por constantes, variáveis e operadores, que definem o método para calcular um valor. O resultado das expressões pode ser armazenado em uma variável de um tipo compatível com o valor resultado.

São exemplos de expressões:

a > b

Resultado ← 3 + (x * y)

D + num / alfa

DIA ← 30

2.4.4 Comentários

Os comentários, cuja importância é indiscutível dentro de um algoritmo ou programa, são delimitados por "{ }" (chaves) e podem aparecer em qualquer lugar do algoritmo. Dentro dos delimitadores de comentário não pode haver outros delimitadores, ou seja, não são permitidos comentários aninhados. Os comentários não influenciarão no funcionamento do programa. Portanto, recomendamos a utilização constante dos comentários, pois estes não afetarão em nada no programa, exceto promover maiores esclarecimentos sobre determinadas partes do algoritmo.

Exemplos de comentários:

```
Leia a,b;   { Leitura das variáveis A e B }
Escreva 'A soma de A e B = ',soma;  {Escreve a soma dos valores}
                                    {informados pelo usuário}
```

2.4.5 Comandos de entrada e saída

Todo algoritmo requer alguma forma de entrada e saída de dados e informações. Um algoritmo não teria razão se não houvesse possibilidade de entrar com valores para processamento ou mostrar os resultados desse processamento.

Em alguns algoritmos pode ser necessário incluir alguns dados de entrada. Esses dados serão processados e os resultados serão mostrados ao usuário. Uma das maneiras de exteriorizar os resultados de determinado processamento é através da impressão destes resultados. Para que isto seja possível, temos o comando de entrada: **leia**; e o comando de saída: **escreva**.

Comando Leia

Este comando permite que o usuário informe dados de entrada para o algoritmo. A sintaxe deste comando é definida na figura a seguir:

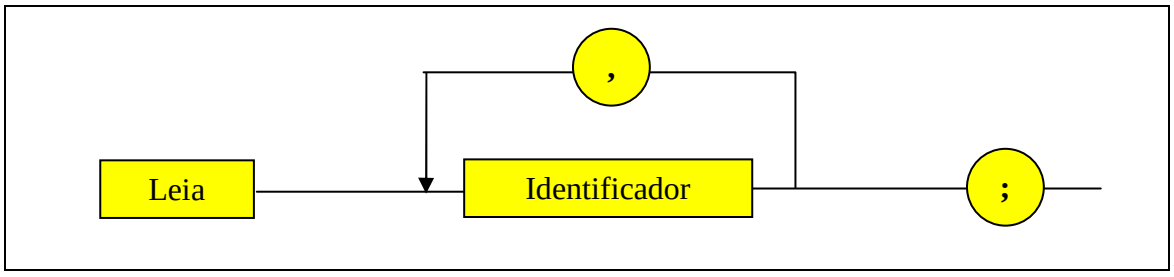

Figura 2.4 - Diagrama de fluxo do comando LEIA.

Exemplos:

Leia VALOR;

Leia A,B,CONTADOR;

Leia CODIGO, NOME;

O comando **Leia** lê as variáveis da lista e mantém o cursor preparado de tal forma que o próximo comando **Leia**, se houver, vai procurar o valor a ser lido na mesma linha.

Exemplo:

```
Algoritmo
    Declare NOME literal;
    Declare NOTA1, NOTA2 real;
    Leia NOME;
    Leia NOTA1;
    Leia NOTA2;
    Escreva NOME,' ',NOTA1,' ',NOTA2;
Fim-algoritmo.
```

Comando Escreva

Este comando permite apresentar ao usuário mensagens e dados contidos em identificadores. A sintaxe do comando de saída é apresentada na figura a seguir:

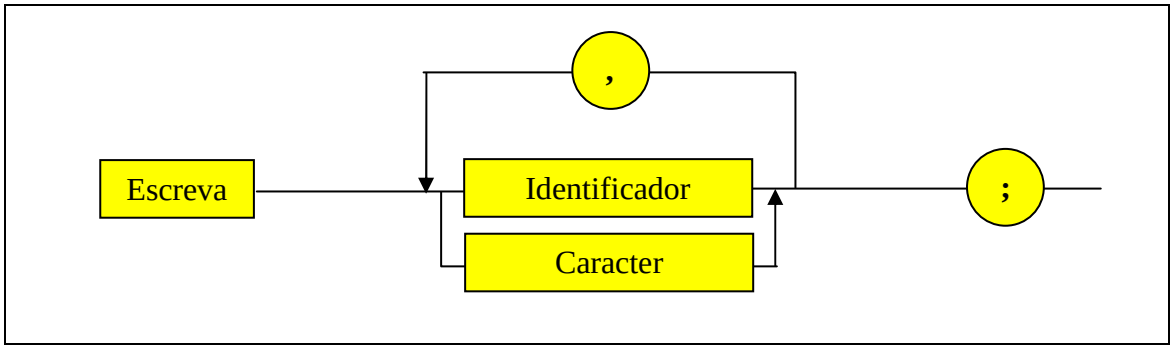

Figura 2.5 - Diagrama de fluxo do comando ESCREVA.

Exemplos:

Escreva SOMA;

Escreva ‘Media = ‘,MED;

Escreva ‘Valor total = ‘,VALOR;

Um algoritmo simples ilustrando os comandos leia e escreva é mostrado a seguir:

```
Algoritmo
    Declare A, B, SOMA inteiro;
    Leia A;
    Leia B;
    SOMA ← A + B;
    Escreva 'A soma de A e B = ',SOMA;
Fim-algoritmo.
```

ATIVIDADES

Responda às questões a seguir e envie e-mail para: marco@unigran.br ou marcoafs@unigran.br. No cabeçalho do e-mail coloque as seguintes informações: "Algoritmos I – RGM – Nome". No corpo do e-mail, informe qual a atividade ou exercício que está sendo enviado.

1) Identificar o tipo de cada uma das constantes abaixo:

a) 21;
b) "bola";
c) "VERDADEIRO";
d) 0,21;
e) falso;

2) Assinalar com um X os identificadores válidos:

( ) VALOR	( ) SALARIO_LIQUIDO	( ) B248
( ) X2	( ) NOTA*DO*ALUNO	( ) A1B2C3
( ) 3*4	( ) MARIA	( ) KM/H
( ) XYZ	( ) NOMEDAEMPRESA	( ) SALA215
( ) "NOTA"	( ) AH!	( ) M{A}

3) Quais os valores fornecidos por cada uma das expressões aritméticas abaixo?

a) 17 – 3 * 2
b) –15 * 3 – 4
c) 123 % 5
d) 123 / 5
e) 5 * 123 / 5 – 123 % 5
f) –21 * 3 * (-1)
g) 14 * (3 – 18 / 4) – 50
h) 100 – (4 * (3 – 2)) * (-2)
i) –56%3
j) 14 * 8 % 5 – 23 / (-4)
k) 3,21 – (-5,02) – 6,1
l) 6,0 / (2,0 + 3,0)

4) Considerando A, B e C variáveis numéricas, contendo os valores 1, 4 e 8, respectivamente; NOME e COR variáveis literais contendo as seqüências de caracteres "Tânia" e "branco"; e, TESTE variável lógica contendo o valor verdadeiro, determinar os resultados obtidos da avaliação das seguintes expressões lógicas:

a) A = 1 e TESTE;

b) NOME = “Pedro” ou COR ≠ “branco”;

c) não TESTE ou B/2 = 0,5;

d) C < 10 ou TESTE e COR = “preto”;

e) A^2 + C = 3 e (A * (B + C) > 13 ou NOME = “Ana”);

f) TESTE e não TESTE;

5) Sendo:

SOMA, NUM, X variáveis numéricas,
NOME, COR, DIA variáveis literais, e
TESTE, COD, TUDO variáveis lógicas,

assinalar os comandos de atribuição considerados inválidos:

( ) NOME ← 5;

( ) SOMA ← NUM+2*X;

( ) TESTE ← COD ou X^2 ≠ SOMA;

( ) TUDO ← SOMA;

( ) COR ← “preto” - X;

( ) X ← X + 1;

( ) NUM ← “*ABC*”;

( ) DIA ← “SEGUNDA”;

( ) SOMA + 2 ← X^2 - NUM;

( ) X ← NOME ≥ COD;

6) Supondo: N e P variáveis do tipo literal
X e A variáveis do tipo numérico,

interpretar a seqüência de comandos a seguir e preencher o formulário de impressão com os valores que serão impressos na unidade de saída.

X ← 0;
leia N, A;
X ← X + A;
P ← N;

escreva P, X;

X ← X + A;

escreva X;

A ← X;

escreva N, X, A;

7) Avalie os algoritmos a seguir e responda quais serão os valores armazenados nas variáveis ao final de sua execução:

a)

Algoritmo
Declare A, B, C inteiro;
A ← 100;
B ← 200;
C ← (B/A) +2;
B ← C*(A+3);
A ← B+C;
Escreva A, B, C;
Fim-algoritmo.

A	B	C

b)

Algoritmo
Declare X, Y, Z real;
X ← 2.5;
Y ← 3.5;
Z ← ((X + Y) - 2) * (3 + X);
Y ← Y + 1;
X ← Y + 1;
Y ← X + Y;
Z ← Z + Y + X;
Escreva X, Y, Z;
Fim-algoritmo.

X	Y	Z

c)

Algoritmo
Declare S, R, U inteiro;
Declare X real;
S ← 3;
R ← S;
U ← S * R;
X ← U / 5 + R;
Escreva S, R, U, X;
Fim-algoritmo.

S	R	U	X

d)

```
Algoritmo
Declare M, P1, P2, P3 real;
        P1 ← 10;
        P2 ← 10;
        P3 ← 10;
        M ← P1 + P2 + P3 / 3;
Escreva M, P1, P2, P3;
Fim-algoritmo.
```

M	P1	P2	P3

e)

```
Algoritmo
Declare M, P1, P2, P3 real;
        P1 ← 10;
        P2 ← 10;
        P3 ← 10;
        M ← (P1 + P2 + P3) / 3;
Escreva M, P1, P2, P3;
Fim-algoritmo.
```

M	P1	P2	P3

f)

```
Algoritmo
Declare M, P1, P2, P3 real;
        P1 ← 10;
        P2 ← 10;
        P3 ← 10;
        M ← P1 + (P2 + P3) / 3;
Escreva M, P1, P2, P3;
Fim-algoritmo.
```

M	P1	P2	P3

g)

```
Algoritmo
Declare K, Y, W inteiro;
        K ← 34;
        Y ← 4;
        W ← ((K % 4) * 2) + (10 – Y);
        K ← W * 2;
Escreva K, Y, W;
Fim-algoritmo.
```

K	Y	W

h)

```
Algoritmo
Declare NOME literal;
        NOME ← 'teresa';
        NOME ← 'iolanda';
        NOME ← 'reginaldo';
Escreva NOME;
Fim-algoritmo.
```

NOME

i)

```
Algoritmo
Declare A, B inteiro;
Declare C, D lógico;
        A ← 50;
        B ← 30;
        C ← A <> B;
        D ← (A – 15) > (B + 40);
Escreva A, B, C, D;
Fim-algoritmo.
```

A	B	C	D

j)

```
Algoritmo
Declare X, Y, Z inteiro;
Declare W, S lógico;
        X ← 10 * 2 - 2;
        Y ← 3;
        Z ← X + Y – 4;
        W ← (X >= Y);
        S ← (Z = X);
Escreva X, Y, Z, W, S;
Fim-algoritmo.
```

X	Y	Z	W	S

8) Avalie os algoritmos a seguir e verifique se os mesmo apresentam algum tipo de problema relacionado a tipos de variáveis incompatíveis. Caso exista algum problema, explique.

a)

```
Algoritmo
Declare A, B, C inteiro;
Declare D real;
        A ← 1;
        B ← 2;
        C ← 3;
        D ← C / (A + B);
Escreva A, B, C, D;
Fim-algoritmo.
```

A	B	C	D

Apresentou problema? _____
Qual? _____
_____
_____
_____
_____
_____

b)

```
Algoritmo
Declare A, B, C inteiro;
Declare D real;
        A ← 1;
        B ← 2;
        C ← 3;
        D ← C / (A + B);
        A ← B * C;
        B ← D;
Escreva A, B, C, D;
Fim-algoritmo.
```

A	B	C	D

Apresentou problema? _____
Qual? _____
_____
_____
_____
_____
_____

c)

```
Algoritmo
Declare X, Y, Z real;
        X ← 3.5;
        Y ← 2;
        Z ← 3;
        X ← (X – 0.5);
Escreva X, Y, Z;
Fim-algoritmo.
```

X	Y	Z

Apresentou problema? _____
Qual? _____
_____
_____
_____
_____
_____

d)

```
Algoritmo
Declare TESTE literal;
        TESTE ← ‘falso’;
Escreva TESTE;
Fim-algoritmo.
```

TESTE

Apresentou problema? _____
Qual? _____
_____
_____
_____
_____
_____

e)

```
Algoritmo
Declare TESTE literal;
        TESTE ← ‘8’;
Escreva TESTE;
Fim-algoritmo.
```

TESTE

Apresentou problema? _____
Qual? _____
_____
_____
_____
_____
_____

f)

Algoritmo
Declare TESTE literal;
TESTE ← falso;
Escreva TESTE;
Fim-algoritmo.

TESTE

Apresentou problema? _____
Qual? _____
_____
_____
_____
_____
_____

g)

Algoritmo
Declare TESTE logico;
TESTE ← falso;
Escreva TESTE;

TESTE

Fim-algoritmo.
Apresentou problema? _____
Qual? _____
_____
_____
_____
_____
_____

h)

Algoritmo
Declare TESTE literal;
TESTE ← 'verdadeiro';
Escreva TESTE;
Fim-algoritmo.

TESTE

Apresentou problema? _____
Qual? _____
_____
_____
_____
_____
_____

i)

Algoritmo
Declare NUMERO literal;
Declare A inteiro;
NUMERO ← ‘24’;
A ← NUMERO + 10;
Escreva A, NUMERO;
Fim-algoritmo.

NUMERO	A

Apresentou problema? _____
Qual? _____
_____
_____
_____
_____
_____

j)

Algoritmo
Declare A, B, C inteiro;
A ← ‘48’ + ‘33’;
B ← A;
C← B;
Escreva A, B, C;
Fim-algoritmo.

A	B	C

Apresentou problema? _____
Qual? _____
_____
_____
_____
_____
_____

Capítulo 3

ESTRUTURAS DE CONTROLE

1 ESTRUTURAS CONDICIONAIS

1.1 SE...ENTÃO

Esta é a estrutura básica de controle em quase todas as linguagens de programação. É empregada quando a ação a ser executada depende de uma inspeção ou teste. Este comando nos fornece a habilidade de fazer uma decisão simples, se uma dada condição for verdadeira.

Sintaxe:

se <condição> então
    <comando1>
    <comando2>
        .....
    <comando n>
fim-se;

O bloco de comandos será executado se a condição for verdadeira. Caso a condição seja falsa, a execução do algoritmo não executará o bloco de comandos e passará o controle para a linha imediatamente após o “fim-se”.

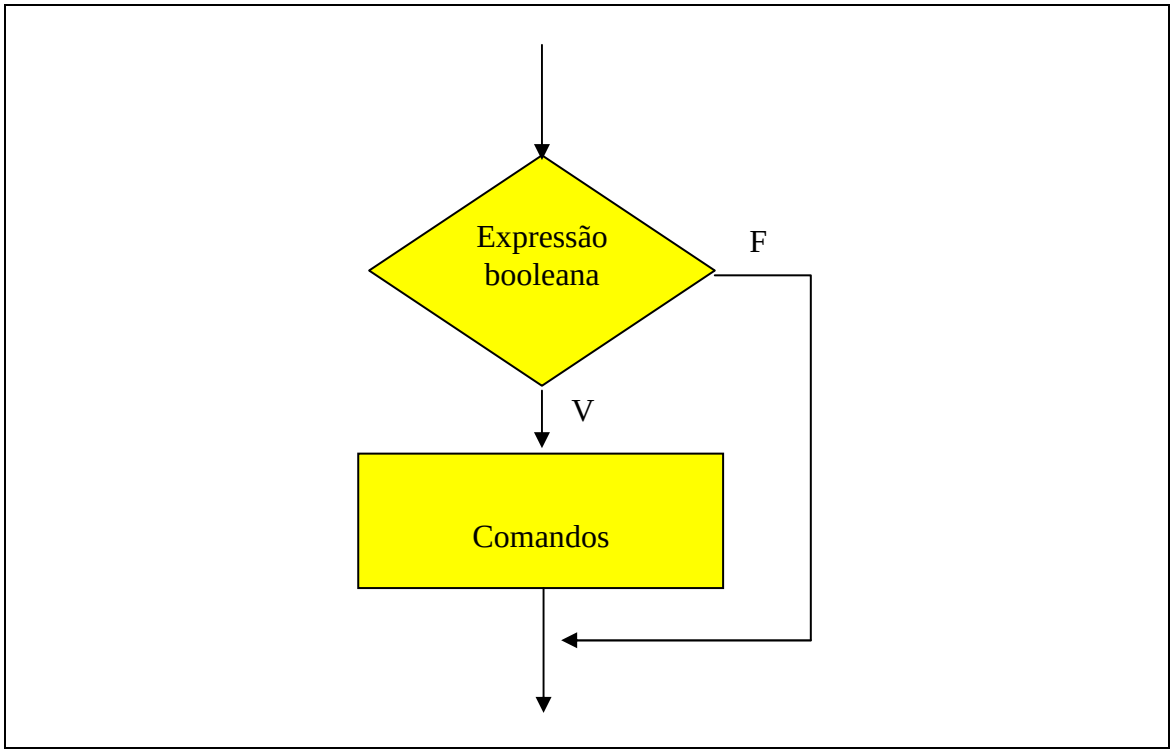

Figura 3.1 - Diagrama de fluxo da estrutura SE..ENTAO.

1.2 SE...ENTÃO...SENÃO

Esta estrutura de decisão é usada quando a ação a ser executada depende de uma inspeção ou teste. Ele nos fornece a habilidade de executar um comando composto, se determinada condição for verdadeira ou falsa.

Sintaxe:

se <condição> então
<comando1>
<comando2>
.....
<comando n>
senão
<comando1>
<comando2>
.....
<comando m>
fim-se;

A seqüência de comandos do bloco “então” será executada caso a condição seja verdadeira. Se a condição for falsa, executará a seqüência de comandos do bloco “senão”.

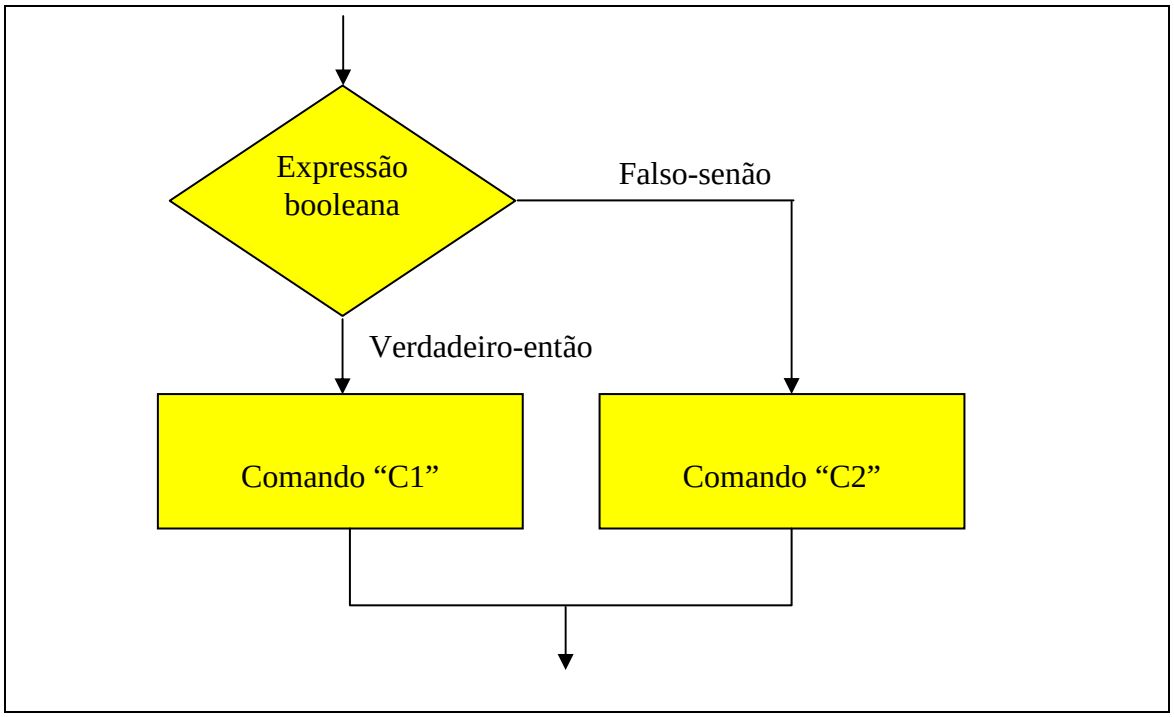

Figura 3.2 - Diagrama de fluxo da estrutura SE...ENTAO...SENAO.

Exemplos:

a) Dados dois números, determinar o maior entre eles:

```
Algoritmo
    declare A, B inteiro;
    leia A;
    leia B;
    se A > B então
        escreva ‘A é maior que B’;
    senão
        se A = B então
            escreva ‘A é igual a B’;
        senão
            escreva ‘B é maior que A’;
        fim-se;
```

```
        fim-se;
fim-algoritmo.
```

b) Dados os lados de um triângulo, determinar se é Equilátero, Isóceles ou Escaleno:

```
Algoritmo
        Declare A, B, C real;
        Leia A ;
        Leia B;
        Leia C;
        se (X = Y) e (X = Z) então
                escreva 'O triângulo é equilátero';
        senão
                se (X = Y) ou (X = Z) ou (Y = Z) então
                        escreva 'O triângulo é isóceles';
                senão
                        escreva 'O triângulo é escaleno';
                fim-se;
        fim-se;
fim-algoritmo.
```

2 ESTRUTURAS DE REPETIÇÃO

Em alguns casos é necessário repetir uma parte do algoritmo um determinado número de vezes. Para isto estão disponíveis as estruturas de repetição.

2.1 ENQUANTO...FAÇA

Esta estrutura executa uma seqüência de comandos repetidas vezes, enquanto uma determinada condição permanece válida (verdadeira). Esta estrutura faz o teste da condição antes de iniciar a repetição; se o primeiro teste falhar, o bloco de instruções dentro do *looping* não será executado nenhuma vez.

Sintaxe:

```
enquanto <condição> faça
        <comando1>
```

<comando2>

.....

<comando n>

fim-enquanto;

Exemplos:

a) Faça um algoritmo que escreva os números inteiros positivos menores que 100.

Algoritmo

Declare NUMERO inteiro;

NUMERO ← 0;

Enquanto NUMERO < 100 faça

Escreva NUMERO;

NUMERO ← NUMERO + 1;

Fim-enquanto;

Fim-algoritmo.

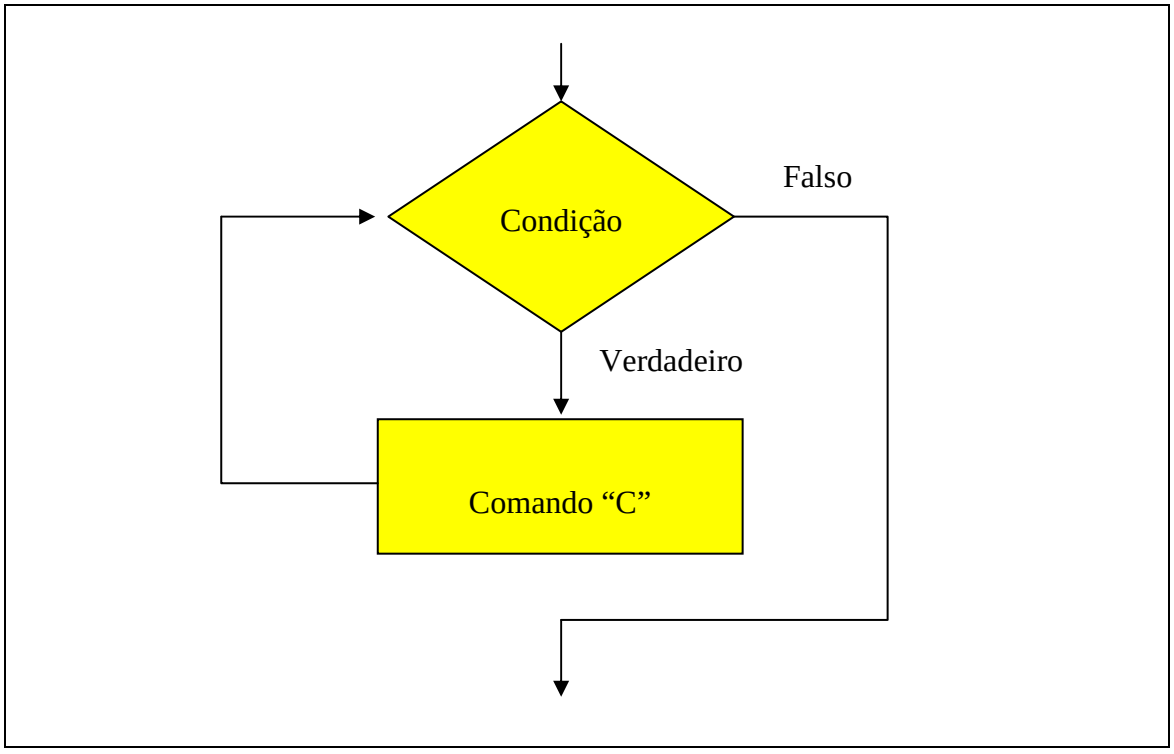

Figura 3.3 - Diagrama de fluxo da estrutura ENQUANTO...FACA.

b) Mostrar a soma dos números pares inteiros positivos menores que 100.

Algoritmo

Declare SOMA, NUMERO inteiro;

```
        NUMERO ← 2;
        SOMA ← 0;
        Enquanto NUMERO < 100 faça
                SOMA ← SOMA + NUMERO;
                NUMERO ← NUMERO + 2;
        Fim-enquanto;
        Escreva 'A soma dos números pares inteiros é: ', SOMA;
Fim-algoritmo.
```

2.2 REPITA...ATÉ-QUE

A estrutura "Repita" executa um bloco de comandos até que uma condição seja verdadeira. Isto significa dizer que ela executa os comandos enquanto a condição for falsa, quando esta condição passar a ser verdadeira, a repetição se encerrará. Os comandos dentro do bloco desta estrutura serão executados pelo menos uma vez. Quando a condição é encontrada, ela será testada. Se for verdadeira passa o controle para o comando imediatamente abaixo da instrução "Até-que". Se a condição for falsa, os comandos do bloco são novamente executados, até que se tenha uma condição verdadeira.

Sintaxe:

```
Repita
        <comando1>
        <comando2>
            .....
        <comando n>
Até-que <condição>;
```

Exemplo:

Faça um algoritmo que escreva os números inteiros positivos menores que 100.

```
Algoritmo
        Declare NUMERO inteiro;
        NUMERO ← 0;
        Repita
                Escreva NUMERO;
```

NUMERO ← NUMERO + 1;

Até-que NUMERO >= 100;

Fim-algoritmo.

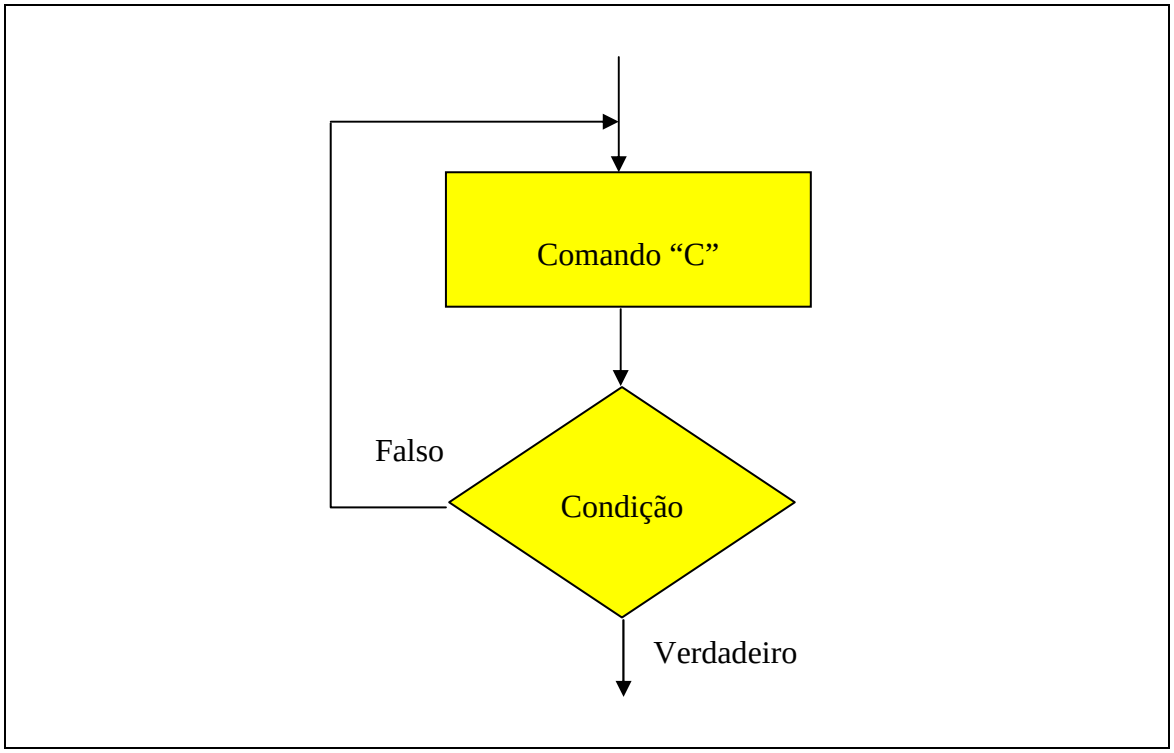

Figura 3.4 - Diagrama de fluxo da estrutura REPITA..ATE-QUE.

2.3 PARA...ATÉ...FAÇA

Esta estrutura permite executar um bloco de comandos um número específico de vezes.

Sintaxe:

Para <variável> de <valor inicial> até <valor final> passo <incremento> faça

<comando1>

<comando2>

.....

<comando n>

Fim-para;

Esta estrutura de repetição utiliza uma variável a ser incrementada de um valor inicial para um valor final. Os comandos serão executados tantas vezes quantas forem o incrementos da variável. Este incremento do valor da variável é definido pelo comando passo, que indica qual o valor a ser acrescentado ou subtraído da variável. Quando o incremento for de 1, não é necessário informar o passo.

O valor da variável de controle não deve ser alterado dentro do bloco de comandos da estrutura "Para". É recomendável o uso desta instrução sempre que se souber o valor inicial e o valor final da variável de controle.

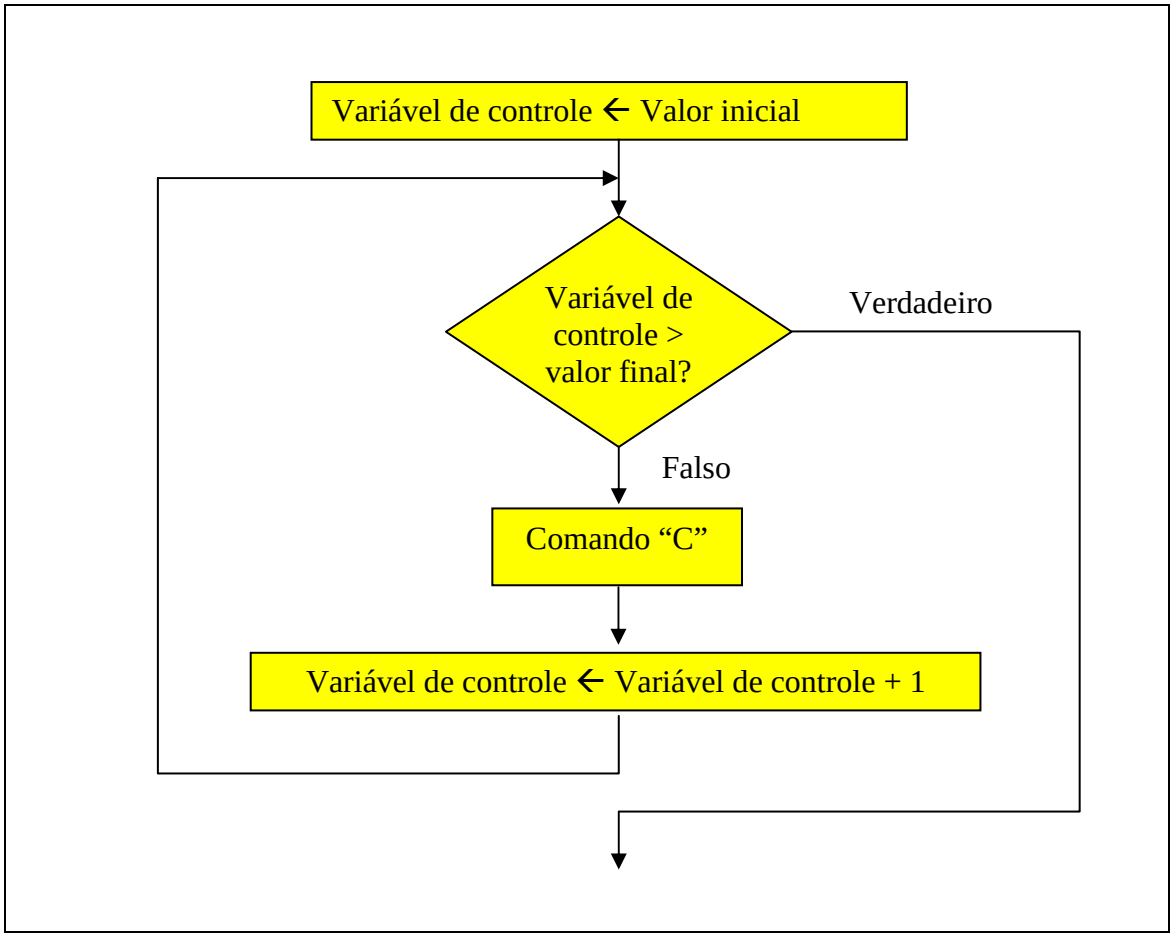

Figura 3.5 - Diagrama de fluxo da estrutura PARA..ATE..FACA.

Exemplos:

a) Faça um algoritmo que escreva os números inteiros positivos menores que 100.

```
Algoritmo
    Declare NUMERO inteiro;
    Para NUMERO de 0 até 99 faça
        Escreva NUMERO;
```

```
        Fim-para;
Fim-algoritmo.
```

b) Faça um algoritmo que escreva os números pares inteiros positivos menores que 100.

```
Algoritmo
        Declare NUMERO inteiro;
        Para NUMERO de 2 até 98 passo 2 faça
                Escreva NUMERO;
        Fim-para;
Fim-algoritmo.
```

ATIVIDADES

Responda às questões a seguir e envie e-mail para: marco@unigran.br ou marcoafs@unigran.br. No cabeçalho do e-mail coloque as seguintes informações: "Algoritmos I – RGM – Nome". No corpo do e-mail, informe qual a atividade ou exercício que está sendo enviado.

1) Faça um algoritmo para ler 100 números inteiros e mostrar qual é o maior entre eles.

2) Faça um algoritmo para ler 100 números inteiros e mostrar qual é o maior e qual é o menor.

3) Analise o trecho do algoritmo a seguir:

```
...
se B1 então
        C1;
senão
        se B2 então
                se B3 então
                        C2;
                senão
                        C3;
                        C4;
                Fim-se;
        Fim-se;
Fim-se;
...
```

No algoritmo acima,

a) Se B1 for verdadeiro, B2 for verdadeiro e B3 for falso, que comandos serão executados?

b) Se B1 for falso, B2 for verdadeiro e B3 for falso, que comandos serão executados?

c) Se B1 for falso, B2 for verdadeiro e B3 for verdadeiro, que comandos serão executados?

4) Fazer um algoritmo para ler um conjunto de valores inteiros positivos. Terminar a leitura quando o número lido for 0 e, no final, mostrar a média dos números lidos.

5) Dados os valores de A, B e C, faça um algoritmo que forneça, se existirem, as raízes de uma equação do 2º grau.

6) Escreva um algoritmo que leia valores para A, B e C e calcule a seguinte fórmula:

$$RESULTADO = \frac{(R+S)}{2}$$

Onde: $R = (A+B)^2$ e $S = (B+C)^2$

7) A conversão de graus Farenheit para Celsius (centígrados) é obtida por:

$$C = \frac{5\,(F - 32)}{9}$$

Fazer um programa que calcule e escreva uma tabela de graus centígrados em função de graus farenheit que variam de 50 a 60, de 1 em 1 grau.

8) Escreva um algoritmo que leia a idade de 20 alunos e, para cada aluno, escreva sua classificação de acordo com a tabela abaixo:

Classificação	Faixa etária
Infantil A	de 5 a 7 anos
Infantil B	de 8 a 10 anos
Juvenil A	de 11 a 13 anos
Juvenil B	de 14 a 17 anos
Adulto	de 18 acima

9) Faça um algoritmo para calcular os 30 primeiros termos da série:

$$SOMA = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots + \frac{1}{N}$$

10) Faça um algoritmo para calcular o fatorial de um número informado.

11) Dado um valor real de X, faça um algoritmo para calcular os 20 primeiros termos da série:

$$S = X - \frac{X}{1!} + \frac{X}{2!} - \frac{X}{3!} + \ldots$$

12) Faça um algoritmo para ler o salário a ser pago para N funcionários de uma empresa. Parar de ler salários quando for informado o valor 0 (zero). No final, mostrar o total de funcionários e a média dos salários.

13) Um rei requisitou os serviços de um sábio e disse-lhe que pagaria qualquer preço. O sábio, necessitando de alimentos, indagou o rei se o pagamento poderia ser feito com grãos de trigo dispostos em um tabuleiro de xadrez, de tal forma que o primeiro quadro deveria conter apenas um grão e nos quadros subseqüentes, o dobro do quadro anterior. O rei achou o trabalho barato, pediu que o serviço fosse executado sem se dar conta de que seria impossível efetuar o pagamento. Faça um algoritmo para calcular o número de grãos que o sábio esperava receber.
Obs.: Sabe-se que um tabuleiro de xadrez tem tamanho 8x8.

14) Ler um conjunto de dados de N pessoas contendo a altura em centímetros e o sexo (M ou F). O programa deverá parar de ler os dados quando for informada uma altura igual a zero. No final, informar:
    - ✓ A maior e a menor altura do grupo;
    - ✓ O total de homens e mulheres; e
    - ✓ A altura média das mulheres.

15) Uma certa firma fez uma pesquisa de mercado para saber se as pessoas gostaram ou não de um produto lançado no mercado. Para isso, foram entrevistadas 20 pessoas, onde cada uma informou o sexo e a resposta da aprovação do produto (sim ou não). Fazer um algoritmo que calcule e escreva:
    - ✓ Número de pessoas que responderam sim;
    - ✓ Número de pessoas que responderam não;

- ✓ Porcentagem de pessoas do sexo feminino que responderam sim;
- ✓ Porcentagem de pessoas do sexo masculino que responderam sim.

16) Em uma agência de empregos, muitos candidatos procuraram vagas para os cargos de Secretária, Cartógrafo, Geógrafo e Programador. Cada candidato deverá preencher uma ficha de inscrição contendo: a idade em número de anos, o sexo, a cor dos olhos e o total de anos de experiência. Cada cargo possui algumas restrições para as pessoas a serem contratadas:

- ✓ Secretária: olhos azuis, maior de idade e do sexo feminino;
- ✓ Cartógrafo: sexo masculino e experiência mínima de 2 anos;
- ✓ Geógrafo: experiência mínima de 2 anos;
- ✓ Programador: experiência mínima de 3 anos, sexo masculino.

Depois de ler um número indeterminado de fichas de inscrição, informar o total e o percentual de candidatos aptos para cada um dos cargos oferecidos, sabendo-se que deverão ser lidas fichas até que se informe uma idade igual a zero.