

José Augusto N. G. Manzano

ILA+

Interpretador de Linguagem Algorítmica

SÉRIE: PORTUGOL

Programação de Computadores em Português

GUIA DE REFERÊNCIA ESCOLAR



PROPE VIVENS

ATENÇÃO

Caso deseje obter uma cópia impressa deste material em formato livro poderá fazê-lo junto as plataformas de publicação:

Clube de autores: <https://clubedeautores.com.br/>

AgBook: <https://www.agbook.com.br>

Em ambas as plataformas efetue busca por "augusto manzano" e escolha o livro desejado.

ILA+

Interpretador de Linguagem Algorítmica

Programação de Computadores em Português

Guia de Referência Escolar

Copyright © 2017 de José Augusto Navarro Garcia Manzano.

Todos os direitos reservados. Proibida e vedada a reprodução, memorização e/ou recuperação parcial ou total, por qualquer meio ou processo existente ou que venha a existir e inclusão de qualquer parte desta obra em qualquer programa juscibernético. A violação dos direitos autorais é crime (art. 184 e parágrafos, do Código Penal, conforme Lei nº 10.695, de 07/01/2003) com pena de reclusão, de dois a quatro anos, e multa, conjuntamente com busca e apreensão e indenizações diversas (artigos 102, 103 parágrafo único, 104, 105, 106 e 107 itens 1, 2 e 3 da Lei nº 9.610, de 19/06/1998, Lei dos Direitos Autorais).

O Autor responsável pelo conteúdo, proprietário do Direito Autoral acredita que as informações aqui apresentadas são corretas e podem ser utilizadas para qualquer finalidade legal. No entanto, não há qualquer garantia, explícita ou implícita, de que o uso de tais informações conduzirá ao resultado desejado.

**Dados Internacionais de Catalogação na Publicação (CIP)
(Ficha Catalográfica Confeccionada pelo Autor)**

M296i Manzano, José Augusto Navarro Garcia.
ILA+: Programação de Computadores em Português: Guia de Referência Escolar / José Augusto Navarro Garcia Manzano. -- São Paulo: Propes Vivens (formato 14,8 x 21 cm), 2017.
104 p.

Bibliografia.
ISBN: 978-85-916492-8-0

1. Portugal (Linguagem de Programação para Computadores).

CDD-005.13

Índices para catálogo sistemático:

1. Linguagem de Programação : Computadores : Processamento de dados

005.13

José Augusto N. G. Manzano

ILA+

Interpretador de Linguagem Algorítmica

Programação de Computadores em Português

Guia de Referência Escolar

1a. Edição

Contato com o autor: certificado@mail.com
colocar em assunto o título ILA+.

Somente serão respondidos e-mails com dúvidas relacionadas, estritamente, ao que é apresentado neste trabalho. Qualquer outro tipo de dúvida ou questão fora deste contexto, não será atendido.

Capa: José Augusto N. G. Manzano.
Produção e Editoração: José Augusto N. G. Manzano.

Software

Produto: **ILA+ 1.01**
Desenvolvedor: **Prof. D.Sc. Sérgio Crespo**
Sítios: <http://www.professores.uff.br/screspo/> (versão 64 bits)
<http://professor.unisinos.br/wp/crespo/ila/> (versão 32 bits)
<http://uff.academia.edu/SergioCrespoPinto>

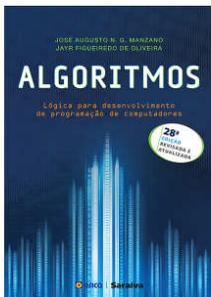
Requisitos

- ◆ Computador padrão IBM-PC padrão 64 bits;
- ◆ Sistema Operacional Microsoft Windows 8, 8.1, 10 de 64 bits;
- ◆ Memória RAM de 1GB;
- ◆ Monitor padrão VGA (operações serão executadas em modo console);
- ◆ Média de 1 MB de espaço em disco rígido;
- ◆ Modem e acesso à Internet;
- ◆ Possuir um dos livros de Algoritmos

Algoritmos: Lógica para Desenvolvimento de Programação de Computadores

Estudo Dirigido de Algoritmos

dos autores Manzano e Oliveira da editora Érica Saraiva.



Dedicatória

Ao apoio de minha família, Sandra minha esposa e Audrey minha filha, queridas da minha vida.

Aos amigos, alunos, leitores e outros cujas dúvidas, curiosidade e perguntas incentivam-me e fazem com que eu me aprimore cada vez mais, e assim possa continuar este trabalho, com um grande abraço a todos.

Agradecimentos

Ao Pai Celestial, que designou em sua infinita sabedoria a direção profissional de minha vida, fazendo-me descobrir o ser professor. Espero estar sendo digno desta missão e do propósito a mim conferido.

A você, amigo leitor, que já me conhece de outros trabalhos ou não, por estar investindo de forma legal na obtenção deste trabalho. Obrigado por não fazer fotocópias ou prática de pirataria com esta contribuição de minha pessoa a você.

Sumário

Capítulo 1 - Introdução

Interpretador de Linguagem Algorítmica	15
Notação Utilizada	18
Obtenção e Uso do ILA	18

Capítulo 2 - Programação Sequencial

Comandos e Instruções	23
Variáveis e Tipos de Dados	25
Operadores Aritméticos e suas Expressões	26
Entrada, Processamento e Saída	28

Capítulo 3 - Programação com Decisões

Condição, Decisão e Operadores Relacionais	33
Desvios Condicionais	34
Operadores Lógicos	40
Divisibilidade	44

Capítulo 4 - Programação com Laços

Laço Condicional Iterativo	49
Laço Condicional Interativo	52
Laço Incondicional	53
Considerações sobre Laços	55

Capítulo 5 - Programação com Funções

Funções Matemáticas	57
Funções Gerais	60
Constantes Matemáticas e Lógicas	62

Capítulo 6 - Programação com Matrizes

Vetores	65
Tabelas	68
Registros	71

Capítulo 7 - Programação Estruturada

Paradigma Estruturado	75
Sub-rotina sem Retorno	76
Sub-rotina com Retorno	78
Sub-rotina Recursiva	79

Capítulo 8 - Recursos Complementares

Manipulação de Cores	81
Utilização de Molduras	83
Funções Definidas pelo Programador	86

Apêndice A - Modelo ILA versus Modelo LPP

Linguagem de Projeto de Programação	93
Exemplos de Codificação em ILA de LPP	97

Prefácio

Este trabalho tem por objetivo apresentar de maneira prática e dirigida o uso da linguagem de programação algorítmica ILA. Este texto é direcionado ao aluno iniciante na prática da programação de computadores.

Não é objetivo desta obra ser um livro de lógica de programação ou substituto do mesmo, mas sim de apresentar a ferramenta que pode ser usada em complemento as aulas de lógica de programação que o leitor tem no curso que frequenta. Desta forma o texto é focado na apresentação dos recursos da linguagem e não nas explicações lógicas, de modo que deverá ser usado em conjunto do livro de lógica de programação adotado por seu professor, tomando-se os cuidados em fazer as adaptações necessárias entre a linguagem ILA e o padrão de português usado pelo seu professor de lógica de programação. É fundamental que este livro seja utilizado em conjunto com um livro de lógica de programação.

Este livro apresenta informações sobre o uso da sintaxe da linguagem “português estruturado (portugol)” ILA. A obra apresenta 100% dos detalhes da linguagem encontrados em sua documentação e programas exemplos disponibilizados na Internet. São demonstradas as técnicas de programação, normalmente, estudadas em um curso de programação de computadores, tais como: sequência; decisão; laço; funções internas, matriz (vetor, tabela e registro) e sub-rotina.

Espero que este trabalho possa ser útil ao leitor iniciante em programação de computadores e na sua aprendizagem, especialmente se for usuário do livro *Algoritmos: Lógica para Desenvolvimento de Programação de Computadores publicado pela editora Érica Saraiva, dos autores José Augusto N. G. Manzano e Jayr Figueiredo de Oliveira*, pois ao final da obra é apresentado apêndice com instruções de como adaptar o padrão do código português estruturado com o dialeto português usado pelo programa ILA.

Grande abraço.

O autor

Sobre o Autor

José Augusto Navarro Garcia Manzano é brasileiro, nascido em São Paulo, capital em 26 de abril de 1965, possui formação acadêmica em Análise e Desenvolvimento de Sistemas, Ciências Econômicas e Licenciatura em Matemática. Atua na área de Tecnologia da Informação, Computação e Informática (desenvolvimento de software, ensino e treinamento) desde 1986. Participou do desenvolvimento de aplicações computacionais para áreas de telecomunicações e comércio. Na carreira de professor iniciou sua atividade docente primeiramente em cursos livres, trabalhando posteriormente em empresas de treinamento e atuando desde então nos ensinamentos técnico e superior.

Atualmente é professor com dedicação exclusiva no IFSP (Instituto Federal de Educação, Ciência e Tecnologia de São Paulo, antiga Escola Técnica Federal). Em sua carreira desenvolveu habilidades para ministrar componentes curriculares de Lógica de Programação (Algoritmos), Estrutura de Dados, Microinformática, Informática, Linguagens de Programação Estruturada, Linguagens de Programação Orientada a Objetos, Engenharia de Software, Sistemas de Informação, Engenharia da Informação, Arquitetura de Computadores e Tecnologias Web. Possui conhecimento de uso e aplicação de diversas linguagens de programação, tais como: Classic BASIC, COMAL, Logo, Assembly, Pascal, FORTRAN, C, C++, D, Java, MODULA-2, Structured BASIC, C#, Lua, HTML, XHTML, JavaScript, VBA e ADA. Possui mais de uma centena de obras publicadas, além de artigos publicados no Brasil e no exterior.

1

Introdução

Este capítulo mostra informações sobre o programa ILA, Interpretador de Linguagem Algorítmica, sua aquisição, instalação e uso inicial, entre outras informações.

Interpretador de Linguagem Algorítmica

Uma linguagem de programação computacional é uma ferramenta que possibilita a um ser humano (chamado assim programador de computadores) definir por meio de instruções operacionais ordens a serem processadas e executadas em um computador. Formalmente, pode-se considerar que uma linguagem é um meio sistemático de comunicar ideias através de símbolos. No caso dos computadores os símbolos de uma linguagem de programação são as palavras reservadas da linguagem que dão a máquina uma ordem de execução. Desta forma, é possível especificar sobre quais dados um computador irá atuar e como será efetuada tal atuação.

As linguagens usadas para programar um computador podem ser formais ou informais. São formais as linguagens que permitem o desenvolvimento de um *software* em nível comercial, industrial ou científicos como BASIC, C, C++, Pascal, Fortran, Java, C#, Lua, COBOL, Modula-2, entre outras. Já as linguagens informais são usadas para a documentação do código de programas e não, são normalmente, exe-

cutadas em um computador, a menos que se tenha alguma ferramenta de processamento algorítmico que será utilizada em nível didático, como o programa ILA descrita neste trabalho.

O programa ILA (Interpretador de Linguagem Algorítmica) é uma ferramenta que objetiva ser usada em sala de aula para a execução de algoritmos computacionais codificados em idioma português em estilo *portugol*. Não se trata de um ambiente de programação, mas de um interpretador algorítmico executado em linha de comando, modo CLI (*Command-Line Interface*). ILA é um processador de algoritmo.

ILA, é um produto desenvolvido a partir de um projeto iniciado no ano de 1990 na Universidade do Vale do Rio dos Sinos situada na cidade de São Leopoldo, na Região Metropolitana de Porto Alegre, Rio Grande do Sul para sistema operacional Windows de 32 bits. Posteriormente o projeto passou a se chamar ILA+ disponibilizado para sistema operacional Windows de 64 bits com a promessa de ser portado para sistema operacional Linux (em desenvolvimento – primeiro semestre de 2016), agora na Universidade Federal Fluminense, Campus Rio das Ostras.

Inicialmente o projeto ILA teve a participação dos pesquisadores Ph.D. João Luis Tavares da Silva e Hamilton Freitas Coutinho com coordenação do Prof. D. Sc. Sérgio Crespo. Na sua continuidade, como ILA+, tem a participação dos pesquisadores Prof. Dr. Carlos Basílio, Prof. Dr. Eduardo Marques e o aluno João Henrique Lopes Spies, coordenado pelo Prof. D.Sc. Sérgio Crespo.

O programa ILA processa algoritmos escritos em idioma português, no estilo *português estruturado* ou *portugol*. O termo *portugol* foi cunhado no Brasil a partir da publicação do livro “Algoritmos e Estruturas de Dados” dos autores Guimarães e Lages pela editora LTC passando a ser

um termo comum na área acadêmica quando do uso da técnica para ensinar os princípios e aplicações da lógica de programação.

A técnica de escrita de códigos de programa em estilo portugol (português estruturado) não é ideia inédita, pois seus princípios de uso são baseados em uma técnica norte-americana denominada PDL (*Program Design Language*) desenvolvida no ano de 1975 por Stephen H. Caine e E. Kent Gordon para documentação de códigos de programas.

A partir do trabalho de Caine e Gordon o método passou a ser usado academicamente em salas de aula como meio de simplificação da escrita de códigos de programas facilitando a concretização das ideias abstratas tratadas pela área de desenvolvimento de *software*. Foi uma questão de tempo até a técnica ser adaptada e usada em diversos países segundo seu idioma local.

Além da ferramenta tratada nesta obra a outras disponíveis para uso. No entanto, cada qual faz uso de um estilo de escrita do portugol, pois não existe uma norma técnica que regente tal ocorrência. Assim é preciso tomar algum cuidado no uso dessas ferramentas e na adaptação ao modelo de portugol usado. Neste sentido, esta obra foca o modelo portugol da ferramenta tratada com as regras do português estruturado dos livros: *Algoritmos - Lógica para o Desenvolvimento de Programação de Computadores* e *Estudo Dirigido de Algoritmos* publicado pela editora Érica Saraiva dos autores Manzano e Oliveira.

O uso do programa ILA tem como objetivo ser um produto facilitador para a aprendizagem das técnicas iniciais de programação para computadores adaptadas ao idioma português permitindo o desenvolvimento do raciocínio lógico pelos iniciantes em programação.

Notação Utilizada

Neste trabalho os códigos escritos em ILA seguem a estrutura de formatação seguinte:

- ◆ Variáveis são expressas em caracteres maiúsculos e formato itálico.
- ◆ Comandos são escritos em caracteres minúsculos e formato negrito.
- ◆ Na indicação das sintaxes, o leitor encontra alguns termos escritos entre os símbolos "<" (menor que) e ">" (maior que), que se caracterizam como informações de cunho obrigatório que devem ser substituídas por algum elemento da linguagem.
- ◆ O que for mencionado entre os símbolos "[" (colchete esquerdo) e "]" (colchete direito) é um detalhe opcional, a qual pode ser omitido para a utilização do recurso indicado. Exceção quando da definição e uso de tabelas em memória para a representação da quantidade de elementos.
- ◆ Toda menção feita à utilização de condições será definida entre os símbolos de parênteses "("e")" para maior legibilidade, apesar de não ser um elemento obrigatório da sintaxe da linguagem ILA.

Obtenção e Uso do ILA

Para fazer uso do ILA, é fundamental possuir instalado em seu computador o interpretador da linguagem em um diretório de trabalho (pasta). Para tanto, acesse o sítio <http://www.professores.uff.br/screspol/>, como indicado junto a Figura 1.1 (abril de 2016).



Figura 1.1 – Sítio de obtenção do ILA+.

Role a tela do sítio para baixo até localizar o **link** identificado como **ILA+ (64 bits)** o qual deverá ser acionado com o ponteiro do *mouse* para que o arquivo **ILA32bits.zip** seja copiado para seu computador. Em seguida crie a partir da raiz de seu disco rígido o diretório (pasta) **ILA64 (C:\ILA64)**, copie para **ILA64** o arquivo **ILA32bits.zip** e em seguida descompacte-o em **ILA64**. A descompactação colocará um arquivo executável chamado **ila.exe**, o interpretador, e um conjunto de programas exemplos escritos em ILA.

Para fazer uso, o ILA deve ser executado a partir da linha de comando do local onde está armazenado ou ter seu local associado ao **path** do sistema operacional para ser executado a partir de qualquer diretório.

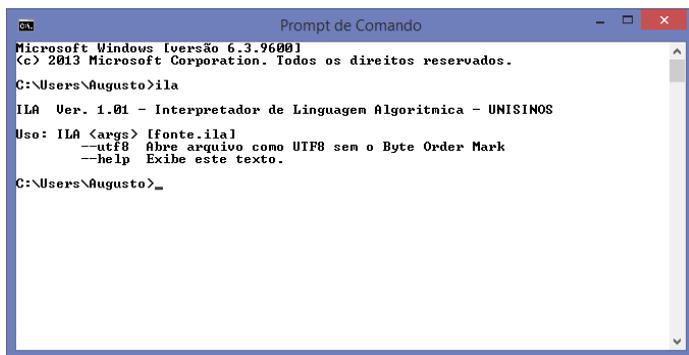
Para associar o ILA ao **path** do sistema operacional abra o programa **Explorador de arquivos (Windows Explorer)** do Windows, selecione na lateral esquerda da janela a opção **Meu computador**, dê um clique com o botão direito do *mouse* e selecione no menu de contexto apresentado a opção **Propriedades**.

Ao ser apresentada a janela **Sistema** selecione a opção **Configura-**

ções Avançadas do sistema. Na janela **Propriedades do Sistema** apresentada acione o botão **Variáveis de Ambiente** que apresentará a tela **Variáveis de Ambiente**. Em **Variáveis de sistema** selecione a variável **Path** e acione o botão **Editar** que apresentará a caixa de diálogo **Editar Variável de Sistema**.

No campo **Valor da variável** acione a tecla **<End>**, coloque o símbolo de ponto-e-vírgula (;) e acrescente **C:ILA64** e acione o botão **OK** para sair da caixa de diálogo **Editar Variável de Sistema**. Na tela **Variáveis de Ambiente** acione o botão **OK** e na janela **Propriedades do Sistema** acione o botão **OK**.

Abra uma linha de *prompt* de comando do sistema e execute **ILA** que deverá apresentar algo semelhante a tela indicada na Figura 1.2.



```
Microsoft Windows [versão 6.3.9600]
(c) 2013 Microsoft Corporation. Todos os direitos reservados.
C:\Users\Augusto>ila
ILA Ver. 1.01 - Interpretador de Linguagem Algoritmica - UNISINOS
Uso: ILA <args> [fonte.ilal]
      --utf8 Abre arquivo como UTF8 sem o Byte Order Mark
      --help Exibe este texto.
C:\Users\Augusto>_
```

Figura 1.2 – Saída da chamada de execução do programa ILA.

O uso do programa ILA deverá ser efetuado a partir da linha de comando:

```
ila <programa.ilal>
```

Lembre-se de que **programa.ila** é um arquivo de código que deverá ser escrito em um editor de textos simples, como o programa **Bloco de notas**.

Há a possibilidade de fazer uso do programa ILA a partir do ambiente de programação **Geany**, desde que seja devidamente configurado. O programa **Geany** deve ser baixado e instalado a partir do sítio de seu projeto <https://www.geany.org/>. Selecione em **Download** a partir da área **Windows Binaries** o arquivo **geany-1.27_setup.exe** e efetue *sua instalação.

Após executar o programa **Geany** acione o menu **Ferramentas** e selecione a opção **Arquivos de configuração** e em seguida selecione o arquivo **filetype_extensions.conf** e acrescente na lista apresentada a opção **ila=*.ila**; respeitando a ordem alfabética das extensões, salve e feche a janela do arquivo. Na sequência abra uma página nova de operação e acione a partir do menu o comando **Construir**, selecione a opção **Definir comandos de Construção** e na janela **Definir Comandos de Construir** apresentada, na área **Executar comandos** informe na primeira célula do campo **Executar** o comando de ação **ila "%f"** e acione **OK**. Em seguida escreva o código seguinte, que pode ser definido com caracteres em formato maiúsculo, gravando-o com o nome **alo.ila**.

```
inicio
  escrever "Alo, mundo!"
fim
```

Na barra de ferramentas do programa **Geany** acione botão **Executar**.

2

Programação Sequencial

Neste capítulo é indicado os princípios de programação que podem ser usados com o programa ILA. São apresentados detalhes iniciais de programação relacionados variáveis, operadores, expressões aritméticas e a execução das ações de entrada e saída de dados.

Comandos e Instruções

Uma linguagem de programação para computadores tem por objetivo passar ordens de execução a uma máquina. Uma ordem dada a um computador é definida a partir da composição de uma instrução. O conjunto de instruções ordenadas de forma lógica, dando um sentido operacional a certa ação é chamada de programa, sendo um programa o produto direto de um algoritmo.

Cada instrução, passada a um computador por meio de um programa, deve expressar um sentido operacional e ser estabelecida a partir do uso dos comandos e das funcionalidades existentes na linguagem de programação em uso.

Um comando oferecido por uma linguagem de programação, é normalmente, representado por: substantivo, preposição, conjunção, adjetivo, verbo (que pode estar no tempo imperativo afirmativo ou no tempo infinitivo), advérbio, interjeição. No entanto, pode acontecer de alguma lin-

guagem de programação não possuir todos esses elementos, e neste caso, a linguagem conterá no mínimo substantivo, verbo e preposição.

Uma funcionalidade ofertada por uma linguagem de programação pode ser estabelecida a partir de uma coleção de ações internas e pre-existentes, normalmente chamadas de funções. Uma função é um recurso que quando utilizado fornece uma resposta de sua execução a partir da indicação de algum conteúdo operacional fornecido a sua execução. Uma função é normalmente representada por verbo seguido dos símbolos de parênteses.

Uma instrução pode ser estabelecida a partir de um ou mais comandos e suas relações com o conjunto de funcionalidades ofertados pela linguagem. O comando que compõe uma instrução pode ser escrito de maneira isolada ou com uso de parâmetros. A definição de uma instrução pode ser representada em uma ou mais linhas de código no programa.

Ao longo da apresentação deste e dos demais capítulos além do estudo do livro **Algoritmos: Lógica para Desenvolvimento de Programação de Computadores** ou do livro **Estudo Dirigido de Algoritmos** dos autores *Manzano* e *Oliveira* da editora Érica Saraiva os conceitos de comando, funcionalidade e instrução se tornarão bem claros. Para cada exemplo indicado neste trabalho será feita a sua relação com os livros de referência aqui mencionados.

Variáveis e Tipos de Dados

Já é sabido, a partir do estudo de disciplinas como: *Algoritmos computacionais*, *Lógica de programação de computadores*, *Técnicas de programação de computadores*, entre outras que o uso de um dado em programação exige que este dado esteja associado a uma variável. Cabe lembrar que uma variável possui como característica a capacidade de armazenar apenas um valor por vez, sendo considerado valor um conteúdo numérico, alfanumérico ou lógico.

O nome de uma variável (também denominado rótulo) é utilizado para sua identificação e posterior uso dentro de um programa, portanto é necessário estabelecer algumas regras para a definição do nome de uma variável:

- ◆ Nomes de variável podem ter um ou mais caracteres;
- ◆ O nome de uma variável não pode ter espaços em branco;
- ◆ Não pode ser nome de uma variável uma palavra reservada da linguagem como o nome de uma instrução ou identificador;
- ◆ Não podem ser utilizados outros caracteres no nome de uma variável, a não ser que sejam letras e números, com exceção do caractere *underline* "_", que pode ser usado para simular a separação de palavras compostas, como: NOME_ALUNO.

Uma variável pode exercer dois papéis. O papel de ação, quando é modificada ao longo do programa para apresentar certo resultado, ou o papel o de controle, em que a variável será "vigiada" e controlada durante a execução de um programa quando do uso de tomada de decisões e laços.

A declaração de variáveis em ILA é executada com o uso da instrução:

```
variaveis
<tipo> <var1>, <var2>, ..., <varN>
```

Onde **variaveis** é o comando que estabelece a definição dos nomes de identificação das variáveis a serem utilizadas e **tipo** é a definição do tipo de dado que será associado as variáveis indicadas.

O programa ILA opera com variáveis que representam dados dos tipos: valores numéricos – inteiros e reais (comando **numerico**), valores alfanuméricos (comando **caracter**) e valores lógicos (comando **logico** com a possibilidade de uso das constantes **falso** e **verdadeiro**).

Operadores Aritméticos e suas Expressões

Operadores aritméticos são responsáveis pela elaboração e execução de cálculos matemáticos. O estabelecimento de cálculos matemáticos são definidos por meio de expressões aritméticas.

A tabela seguinte apresenta os operadores aritméticos a serem usados na elaboração de cálculos matemáticos:

Operador	Operação	Resultado
=	Atribui um valor a uma variável	Atribuição
+	Manutenção de sinal	Positivo
-	Inversão de sinal	Negativo
^	Exponenciação	Numérico
raiz(base, índice)	Raiz de índice qualquer	Numérico
resto(dividendo, divisor)	Resto de divisão	Numérico
/	Divisão com quociente real	Numérico
*	Multiplicação	Numérico
+	Adição	Numérico
-	Subtração	Numérico

Considere a fórmula $AREA = \pi \cdot RAIIO^2$ para o cálculo da área de uma circunferência, em que estão presentes as variáveis $AREA$ e $RAIO$, a constante π ($\pi = 3.14$) e os operadores aritméticos de multiplicação e também a operação de potência, elevando o valor da variável $RAIO$ ao quadrado. Assim sendo, a fórmula pode ser expressa em ILA com a expressão aritmética $AREA = \pi i * RAIIO ^ 2$.

Expressões aritméticas escritas para a execução em um computador seguem um formato um pouco diferente do conhecido na ciência matemática. Por exemplo, a expressão: $X = \{43 \cdot [55 : (30 + 2)]\}$ será normalmente escrita como $X = (43 * (55 / (30 + 2)))$. Note que as chaves e colchetes são abolidos, utilizando em seu lugar apenas parênteses.

Se a fórmula a ser usada fosse para calcular a área de um triângulo, em que é necessário efetuar a multiplicação da base pela altura e em seguida dividir este valor pela constante 2, como ficaria?

Observe a fórmula seguinte:

$$A = \frac{b \cdot h}{2}$$

Que deverá ser escrita como a expressão aritmética: $A = (B * H) / 2$.

É oportuno levar em consideração a precedência matemática para a realização das operações aritméticas.

Entrada, Processamento e Saída

A ação de *entrada* é responsável por permitir a um determinado usuário de um programa fornecer os dados que serão armazenados na memória (nas variáveis) para posterior uso na fase de *processamento*.

A ação de *processamento* pode ocorrer sobre dois aspectos: *matemático* ou *lógico*. Um processamento é matemático quando do uso de operações matemáticas com a aplicação dos operadores aritméticos. Um processamento é lógico quando se usa controles em tomadas de decisão e execução de laços ou em qualquer operação não matemática.

A ação de *saída* ocorre normalmente após a conclusão de uma ação de processamento ou de uma ação de entrada, permitindo assim apresentar para o usuário um determinado valor como resposta de uma ação anteriormente realizada.

Uma entrada e uma saída podem ocorrer dentro de um computador de diversas formas. Por exemplo, uma entrada pode ser feita via *teclado*, *modem*, *leitores ópticos*, *disco*, *scanners* entre outros. Uma saída pode ser feita via *monitor de vídeo*, *impressora*, *disco*, entre outras formas. Devido a esta variedade, os programas aqui escritos utilizarão como ação de entrada o comando **ler** e para a ação de saída o comando **escrever** que poderá direcionar a saída de dados para o monitor de vídeo ou impressora dependendo do uso das funções **impressora()** e **vídeo()**.

Para colocar em prática o que foi exposto até este momento, considere o problema computacional:

Desenvolver um programa de computador que efetue a leitura de dois valores numéricos, efetue na sequência a adição (processamento

matemático) dos valores lidos e apresente como saída o resultado obtido da adição processada.

Observe a proposta a seguir relacionando-a com as figuras 3.3 dos capítulos 3 dos livros de algoritmos referenciados. Assim sendo, considere o seguinte código:

```
// programa ADICIONA NUMEROS
variaveis
  numerico A, B, X
inicio
  limpar
  escrever "Entre o 1o. valor:"
  ler A
  escrever "Entre o 2o. valor:"
  ler B
  X = A + B
  escrever "O resultado da soma equivale a: ", X
fim
```

Escreva o código de programa em um editor de textos, grave-o com o nome **cap0201.ila** e para executá-lo informe na linha de comando do sistema operacional a instrução **ila cap0201.ila**.

A linha com a instrução *// (comentário)* tem por finalidade indicar o uso de uma linha de comentário usada para dar nome ao programa. Linhas de comentário são usadas para definir trechos de notas explicativas em um programa.

A linha com o uso da instrução **variaveis** define o conjunto de variáveis usadas como sendo do tipo **numeric**.

A linha com a instrução com o comando **limpar** efetua a limpeza da tela.

As linhas com as instruções **ler** definem o trecho de entrada de dados usadas em conjunto com a linhas definidas com o comando **escrever** junto a apresentação das mensagens de entrada de dados.

A linha que efetiva a ação de processamento do programa contém a expressão aritmética de soma dos valores **A** e **B** que são atribuídos a variável **X** por meio da expressão $X = A + B$.

A instrução que efetiva a saída por meio do comando **escrever** apresenta o valor da variável **X** e a linha com a instrução **fim** identifica o fim do programa.

A seguir considere o problema computacional:

Desenvolver um programa de computador que calcula o salário líquido de um profissional que trabalha por hora. Para a execução deste programa será necessário saber alguns dados, como: valor da hora de trabalho, número de horas trabalhadas no mês e o percentual de desconto da assistência médica. O programa deve apresentar os valores do salário bruto, do valor descontado e o salário líquido.

Observe a proposta a seguir relacionando-a respectivamente com as figuras 3.5 ou 3.6 dos capítulos 3 dos livros de *Algoritmos: Lógica para Desenvolvimento de Programação de Computadores* ou do livro *Estudo Dirigido de Algoritmos*. Assim sendo, considere o seguinte código:

```
// programa SALARIO
variaveis
  numerico HT, VH, PD, SB, TD, SL
inicio
  limpar
  escrever "Quantas horas de trabalho?"
  ler HT
  escrever "Qual o valor da hora?"
  ler VH
  escrever "Qual o percentual de desconto?"
  ler PD
  SB = HT * VH
  TD = (PD/100) * SB
  SL = SB - TD
  escrever "Salario bruto ...: ", SB
  escrever "Desconto .....: ", TD
  escrever "Salario liquido .: ", SL
fim
```

Grave o programa com o nome **cap0202.ila** e em seguida faça sua execução.

Dentro do escopo de operação com entrada, processamento e saída pode-se efetuar além de operações aritméticas operações com concatenação de seqüências de caracteres. As concatenações são feitas exclusivamente com dados do tipo caractere na formação de cadeias. O código seguinte demonstra o uso de uma ação de concatenação:

```
// programa CONCATENA
inicio
  limpar
  escrever "Interpretador " + "ILA"
fim
```

Grave o programa com o nome **cap0203.ila** e em seguida faça sua execução.

Os comandos escritos em ILA podem ser definidos em caracteres maiúsculos ou minúsculos. Nesta obra são usados em caracteres minúsculos.

Na sequência, pegue os exercícios que seu professor de lógica de programação tenha lhe passado e os reescreva com o código ILA executando-os e prestando a atenção nas ocorrências ensinadas.

Aproveite o apêndice desta obra para ver a relação de equivalência entre os comandos usados nos livros de algoritmos com os comandos da linguagem aqui apresentados. Faça isso para cada capítulo estudado.

3

Programação com Decisões

Este capítulo mostra o processo de tomada de decisão, utilizando estruturas de condição, decisão, operadores relacionais, operadores lógicos, desvio condicional simples, desvio condicional composto e desvio condicional sequencial.

Condição, Decisão e Operadores Relacionais

Decisão é o ato de deliberar, de julgar, de tomar uma decisão, de decidir. Para que uma decisão seja tomada é preciso que esta esteja baseada sobre uma proposição, ou seja, uma condição que do ponto de vista da ciência da computação pode-se dizer que *é a relação lógica entre os elementos* que formam a proposição (condição) para a tomada de uma decisão e que a resposta da tomada de uma decisão poderá ser *verdadeira* ou *falsa*.

A relação lógica entre esses elementos ocorre sob dois aspectos:

- Variável versus variável;
- Variável versus constante¹.

A forma usada para estabelecer uma relação lógica é conseguida com o uso de *operadores relacionais*. Os operadores relacionais são fer-

¹ *Constante é qualquer valor de qualquer tipo de dado.*

ramentas que permitem a definição da relação lógica entre elementos de uma condição (proposição), a qual será utilizada para a execução do processo de tomada de uma decisão. Neste sentido os operadores relacionais sempre retornam como resultado operacional os valores **verdadeiro** ou **falso**.

A tabela seguinte apresenta os operadores relacionais que podem ser utilizados no interpretador ILA.

Símbolo	Significado
=	Igual a
<>	Diferente de
>	Maior que
<	Menor que
>=	Maior ou igual que
<=	Menor ou igual que

Os operadores relacionais possuem o mesmo nível de prioridade. Assim, não há necessidade de se preocupar em alterar sua prioridade quando de seu uso em uma expressão lógica.

Desvios Condicionais

O desvio condicional está associado à utilização de decisões em um programa. A resposta a uma decisão pode ser verdadeira ou falsa. Se verdadeira, executará certa ação; se falsa, executará outra ação. Um desvio condicional pode ser simples, composto ou sequencial.

Para se fazer uso de um desvio condicional simples é necessário fazer uso de instrução:

```
se (<condição>) entao
    ação para condição verdadeira
fim_se
```

Se a *condição* for verdadeira, será executada a ação estabelecida. Se condição for falsa o fluxo de programa é desviado para a próxima linha de código.

No sentido de exemplificar o uso de desvio condicional simples considere o seguinte problema computacional:

Desenvolver um programa de computador que efetue a leitura de dois valores numéricos e apresenta os valores lidos em ordem crescente.

Observe para este recurso as orientações do capítulo 4, tópico 4.3 ou 4.1.1, respectivamente para os livros *Algoritmos: Lógica para Desenvolvimento de Programação de Computadores* ou *Estudo Dirigido de Algoritmos*. Assim sendo, considere o seguinte código:

```
// programa CONDICA01
variaveis
    numerico A, B, X
inicio
    limpar
    posicionar 1,1
    escrever "Entre 1o. valor:"
    posicionar 1,18
    ler A
    posicionar 2,1
    escrever "Entre 2o. valor:"
    posicionar 2,18
    ler B
    se (A > B) entao
        X = A
        A = B
        B = X
    fim_se
    posicionar 4,1
    escrever "Os valores sao: ", A, " e ", B, "."
fim
```

Grave o programa com o nome **cap0301.ila** e em seguida faça sua execução.

A linha que possui a instrução formada pelo comandos **se / entao** usa entre os comandos a definição de uma condição representada entre parênteses. Se o resultado da condição definida entre os comandos **se** e **entao** for **verdadeira** ocorrerá a execução das ações indicadas entre os comandos **se / entao** e **fim_se**. Se o valor da variável **A** for maior que o valor da variável **B** ocorrerá a troca dos valores. Se a condição não for verdadeira será executada diretamente a linha com a instrução **escrever "Os valores sao: ", A, " e ", B, ". "**.

Além dos comandos já conhecidos está sendo usado o comando **posicionar** que tem por finalidade localizar o cursor em uma posição de tela. O comando **posicionar** faz uso de duas informações, sendo respectivamente as indicações da posição de linha e coluna da tela. Considere para a coordenada de linhas valores entre 1 e 24 e para a coordenada de colunas valores entre 1 e 80.

Para se fazer uso de tomada de decisão com desvio condicional composto é necessário utilizar a instrução:

```
se (<condição>) entao
    ação para condição verdadeira
senao
    ação para condição falsa
fim_se
```

Se *condição* for verdadeira, será executada a ação definida após os comandos **se** e **entao**, caso a condição seja falsa será executada a ação definida entre os comandos **senao** e **fim_se**.

No sentido de exemplificar o uso de desvio condicional composto considere o seguinte problema computacional:

Desenvolver um programa que efetua a leitura de duas notas escolares, calcule a média aritmética das notas fornecidas e apresente mensagem “Aprovado” caso a condição seja maior ou igual a 5, senão apresente a mensagem “Reprovado”. Mostre junto das mensagem o valor da média obtida.

Observe para este recurso as orientações do capítulo 4, tópico 4.4 ou 4.1.2, respectivamente para os livros *Algoritmos: Lógica para Desenvolvimento de Programação de Computadores* ou *Estudo Dirigido de Algoritmos*. Assim sendo, considere o seguinte código:

```
// programa CONDICA02
variaveis
    numerico N1, N2, MD
inicio
    limpar
    posicionar 1,1
    escrever "Entre 1a. nota:"
    posicionar 1,17
    ler N1
    posicionar 2,1
    escrever "Entre 2a. nota:"
    posicionar 2,17
    ler N2
    MD = (N1 + N2) / 2
    se (MD >= 5) entao
        posicionar 4,1
        escrever "Aprovado"
    senao
        posicionar 4,1
        escrever "Reprovado"
    fim_se
    posicionar 5,1
    escrever "com media: ", MD
fim
```

Grave o programa com o nome **cap0302.ila** e em seguida faça sua execução.

A linha de instrução com os comandos **se** e **entao** tendo a condição definida como verdadeira fará a execução da ação indicada entre os comandos **se / entao** e **senao**, neste caso, apresentando a mensagem **Aprovado**, sendo a condição falsa ocorrerá apresentação da mensagem **Reprovado** definida entre os comandos **senao** e **fim_se**.

Além dos comandos de definição de ações de decisão simples e decisão composta, há um recurso para a definição de decisão seletiva por meio de uso dos comandos **faca caso / caso / fim_caso**, o qual obedece a estrutura sintática:

```
faca caso
  caso (<expressão_lógica1>):
    bloco de ação1 executada para expressão lógica 1
  caso (<expressão_lógica2>):
    bloco de ação2 executada para expressão lógica 2
  caso (<expressão_lógicaN>):
    bloco de açãoN executada para expressão lógica N
  [
    outro_caso:
      bloco ação para nenhuma ação anterior executada
  ]
fim_caso
```

No sentido de exemplificar o uso de desvio condicional seletivo (decisão seletiva) considere o seguinte problema computacional:

Desenvolver um programa de computador que aceite a entrada de um valor numérico entre 1 e 6 que represente um dos seis primeiros meses do calendário Cristão. Fornecido o valor referente a determinado mês o programa deve apresentar o nome do mês por extenso. Qualquer valor fornecido que não seja de 1 a 6 deve ter a apresentação da mensagem “Valor invalido”.

Observe para este recurso as orientações do capítulo 4, tópico 4.5.3 ou 4.1.5, respectivamente para os livros *Algoritmos: Lógica para De-*

envolvimento de Programação de Computadores ou Estudo Dirigido de Algoritmos. Assim sendo, considere o seguinte código:

```
// programa CONDICA03
variaveis
  numerico N
inicio
  limpar
  posicionar 1,1
  escrever "Entre valor:"
  posicionar 1,14
  ler N
  posicionar 3,1
  faca caso
    caso (N = 1):
      escrever "Janeiro"
    caso (N = 2):
      escrever "Fevereiro"
    caso (N = 3):
      escrever "Marco"
    caso (N = 4):
      escrever "Abril"
    caso (N = 5):
      escrever "Maio"
    caso (N = 6):
      escrever "Junho"
    outro_caso:
      escrever "Mes invalido"
  fim_caso
fim
```

Grave o programa com o nome **cap0303.ila** e em seguida faça sua execução.

A estrutura condicional **faca caso / caso / fim_caso** tem por finalidade simplificar a definição de um conjunto de decisões sequenciais.

Operadores Lógicos

Quando houver a necessidade de se trabalhar com o relacionamento de duas ou mais condições ao mesmo tempo para a tomada de uma única decisão torna-se necessário o uso de operadores lógicos, sendo possível usar principalmente os operadores lógicos **e**, **ou** e **nao**.

O operador lógico de conjunção **e** é utilizado quando duas ou mais relações lógicas necessitam ser verdadeiras, caso contrário, o resultado do valor lógico retornado será falso. A tabela-verdade para o operador lógico **e** é a seguinte:

Condição 1	Condição 2	Resultado
Falsa	Falsa	Falso
Verdadeira	Falsa	Falso
Falsa	Verdadeira	Falso
Verdadeira	Verdadeira	Verdadeiro

O operador **e** faz com que o resultado lógico seja verdadeiro quando todas as condições envolvidas na decisão forem também verdadeiras, gerando assim um resultado lógico verdadeiro.

No sentido de exemplificar o uso de operador de conjunção considere o seguinte problema computacional:

Desenvolver um programa de computador que efetua a leitura de um valor numérico entre 1 e 9 e apresenta mensagem informando se o valor está na faixa de 1 a 9 ou se está fora dessa faixa de valores.

Observe para este recurso as orientações do capítulo 4, tópico 4.6.1 ou 4.2.1, respectivamente para os livros *Algoritmos: Lógica para Desenvolvimento de Programação de Computadores* ou *Estudo Dirigido de Algoritmos*. Assim sendo, considere o seguinte código:

```
// programa CONDICAO4
variaveis
  numerico N
inicio
  limpar
  posicionar 1,1
  escrever "Entre numero:"
  posicionar 1,15
  ler N
  se (N >= 1) e (N <= 9) entao
    escrever "Valor na faixa de 1 a 9."
  senao
    escrever "Valor fora da faixa."
  fim_se
fim
```

Grave o programa com o nome **cap0304.ila** e em seguida faça sua execução.

O exemplo anterior mostra, a utilização do operador de conjunção **e**, que permitirá a apresentação da mensagem: **Valor na faixa de 1 a 9**, caso o valor da variável **N** esteja entre 1 e 9. Qualquer valor fornecido fora da faixa definida apresentará a mensagem: **Valor fora da faixa**.

O operador lógico de disjunção inclusiva **ou** é utilizado quando pelo menos um dos relacionamentos lógicos de uma decisão necessita ser verdadeiro para obter-se um resultado lógico verdadeiro, caso contrário, o valor do resultado lógico retornado será falso.

A tabela-verdade para o operador lógico **ou**, considerando o uso de duas condições, pode retornar um dos seguintes resultados lógicos:

Condição 1	Condição 2	Resultado
Falsa	Falsa	Falso
Verdadeira	Falsa	Verdadeiro
Falsa	Verdadeira	Verdadeiro
Verdadeira	Verdadeira	Verdadeiro

O operador **ou** faz com que o resultado lógico seja verdadeiro quando pelo menos uma das condições envolvidas na decisão for verdadeira, fornecendo um resultado lógico verdadeiro.

No sentido de exemplificar o uso de operador de disjunção considere o seguinte problema computacional:

Desenvolver um programa de computador que efetua a leitura do sexo biológico de um ser humano e apresente mensagem informando se o sexo fornecido é ou não válido. Considere a entrada de dados de tipo não numérico identificados pelas letras M para masculino e F para feminino.

Observe para este recurso as orientações do capítulo 4, tópico 4.6.2 ou 4.2.3, respectivamente para os livros *Algoritmos: Lógica para Desenvolvimento de Programação de Computadores* ou *Estudo Dirigido de Algoritmos*. Assim sendo, considere o seguinte código:

```
// programa CONDICA05
variaveis
  caracter SEXO
inicio
  limpar
  posicionar 1,1
  escrever "Entre seu sexo:"
  posicionar 1,17
  ler SEXO
  se (SEXO = "M") ou (SEXO = "F") entao
    escrever "Sexo valido."
  senao
    escrever "Sexo invalido."
  fim_se
fim
```

Grave o programa com o nome **cap0305.ila** e em seguida faça sua execução.

O exemplo anterior mostra, utilização do operador lógico **ou**, que somente permite a apresentação da mensagem "**Sexo valido**", caso o valor fornecido para a variável **SEXO** seja **M** ou **F**. Qualquer outro valor fornecido apresentará a mensagem "**Sexo invalido**".

O operador lógico de negação é utilizado quando se necessita estabelecer a inversão do valor de um determinado resultado lógico de uma decisão. É possível obter valores: não verdadeiro e não falso. O operador lógico **nao** inverte o resultado lógico de uma condição. A tabela-verdade para o operador lógico **nao**, que é utilizado antes de uma condição, pode retornar um dos seguintes resultados lógicos:

Condição	Resultado
Verdadeira	Não Verdadeira
Falsa	Não Falsa

O operador **nao** faz com o resultado lógico de uma determinada decisão seja invertido.

No sentido de exemplificar o uso de operador de negação considere o seguinte problema computacional:

Desenvolver um programa de computador que efetue a leitura de um valor numérico e apresente o valor informado caso este valor não seja menor que 3, ou seja, apresenta somente valores que estejam acima do valor numérico 3.

Observe para este recurso as orientações do capítulo 4, tópico 4.6.3 ou 4.2.2, respectivamente para os livros *Algoritmos: Lógica para Desenvolvimento de Programação de Computadores* ou *Estudo Dirigido de Algoritmos*. Assim sendo, considere o seguinte código:

```
// programa CONDICA06
variaveis
  numerico N
inicio
  limpar
  posicionar 1,1
  escrever "Entre um numero:"
  posicionar 1,18
  ler N
  se nao (N <= 3) entao
    escrever N
  fim_se
fim
```

Grave o programa com o nome **cap0306.ila** e em seguida faça sua execução.

O exemplo anterior mostra o uso do operador lógico **nao**, que permite executar certa ação quando uma condição não é verdadeira.

Divisibilidade

Divisibilidade é a qualidade do que é divisível e duas questões devem ser consideradas, sendo os múltiplos e os divisores dos números naturais. Entende-se por número natural um valor numérico inteiro, positivo e diferente de zero.

Múltiplos são os resultados obtidos a partir da multiplicação de dois números naturais, enquanto divisores são os números que dividem outros números com o objetivo de gerar um resultado de divisão exato, ou seja, obter resto de divisão zero com quociente inteiro. Quando o resto de uma divisão de números naturais é igual a zero, ocorre o efeito divisibilidade. Valores reais não entram nesta abordagem.

O pivô das operações de divisibilidade é a obtenção do cálculo do resto (incógnita r) conseguido a partir da divisão de valores naturais forne-

cidos como dividendo (incógnita **a**) e divisor (incógnita **n**) da operação de divisão a partir da equação:

$$r = a - n \left\lfloor \frac{a}{n} \right\rfloor$$

Do ponto de vista computacional, a equação matemática para cálculo do resto de uma divisão deve ser convertida em uma expressão aritmética:

$$r = a - n * \text{inteiro}(a / n).$$

A função **inteiro()** converte um valor real em valor inteiro. Esta função despreza do valor real sua mantissa mantendo apenas o expoente do valor, ou seja, sua parte inteira. No entanto em ILA esta mesma operação pode ser representada pela expressão:

$$r = \text{resto}(a, n).$$

Para maiores detalhes sobre operações de divisibilidade consulte no capítulo 4 o tópico 4.7 ou 4.3 respectivamente indicados nos livros *Algoritmos: Lógica para Desenvolvimento de Programação de Computadores* ou *Estudo Dirigido de Algoritmos*.

Como exemplo, considere: *Desenvolver um programa de computador que efetue a leitura de um valor numérico inteiro e apresente mensagem informando se o valor fornecido é par ou ímpar.* A solução a seguir faz uso da função **inteiro()**.

```
// programa CONDICA07
variaveis
  numerico N, R
inicio
  limpar
  posicionar 1,1
  escrever "Entre um valor numerico inteiro:"
  posicionar 1,34
  ler N
  R = N - 2 * inteiro(N / 2)
  se (R = 0) entao
    escrever "Valor par."
  senao
    escrever "Valor impar."
  fim_se
fim
```

Grave o programa com o nome **cap0307.ila** e em seguida faça sua execução.

Considere como próximo exemplo: *Desenvolver um programa de computador que efetue a leitura de um valor numérico inteiro e apresente mensagem informando se o valor fornecido é par ou ímpar.* A solução a seguir faz uso da função **resto()**.

```
// programa CONDICA08
variaveis
  numerico N, R
inicio
  limpar
  posicionar 1,1
  escrever "Entre um valor numerico inteiro:"
  posicionar 1,34
  ler N
  R = resto(N, 2)
  se (R = 0) entao
    escrever "Valor par."
  senao
    escrever "Valor impar."
  fim_se
fim
```

Grave o programa com o nome **cap0308.ila** e em seguida faça sua execução.

Em relação ao dois programas nota-se no primeiro o efeito detalhado do obtenção do cálculo do valor do resto de uma divisão criado a partir de uma equação com uso da função **inteiro()** e no segundo exemplo se usa a função **resto()** que demonstra a mesma ação de forma mais resumida.

Na sequência, pegue os exercícios que seu professor de lógica de programação tenha lhe passado e os reescreva com o código ILA executando-os e prestando a atenção nas ocorrências ensinadas.

4

Programação com Laços

Este capítulo apresenta o uso da técnica de laços que possibilita definir repetições com números variados de vezes, tanto com laços iterativos (quando se sabe o número de vezes que o laço deve ser executado) até laços interativos (quando não se sabe quantas vezes o laço deve ser executado). Serão vistos os dois laços condicional e incondicional suportados pelo ILA.

Laço Condicional Iterativo

Como laço da categoria condicional o ILA oferece para execução o comando **faca enquanto / fim_enquanto** que caracteriza-se por ser um laço do tipo pré-teste com fluxo de ação para condição verdadeira. Laços do tipo pré-teste realizam teste lógico no início do laço, antes de executar alguma ação verificando se é permitido executar o trecho de instruções subordinado a condição em uso ou não.

O laço **faca enquanto / fim_enquanto** executa certo conjunto de instruções, enquanto a condição do laço for satisfeita, podendo ser utilizado para ações iterativas e interativas. No momento em que a condição torna-se falsa, o processamento é desviado para fora do laço. Se a condição não é satisfeita no início do laço, as instruções contidas dentro do laço são ignoradas.

O laço do tipo pré-teste pode ser efetuado com decisão verdadeira (forma padrão) ou decisão falsa com auxílio do operador lógico **nao**, a partir da sintaxe:

```
faca enquanto [<nao>] (<condição>)  
    ação executada enquanto condição verdadeira  
fim_enquanto
```

Onde enquanto for a condição verdadeira (ou não verdadeira com o operador lógico **nao**) serão executadas todas as instruções existentes entre os comandos **faca enquanto** e **fim_enquanto**.

No sentido de exemplificar o uso de laço pré-teste que efetue laço iterativo considere o seguinte problema computacional: *Desenvolver um programa de computador que apresente os resultados de uma tabuada de um valor numérico qualquer.* Observe para este recurso as orientações do capítulo 5, tópico 5.3 ou 5.1, respectivamente para os livros *Algoritmos: Lógica para Desenvolvimento de Programação de Computadores* ou *Estudo Dirigido de Algoritmos*. Assim sendo, considere o seguinte código:

```
// programa LAC01  
variaveis  
    numerico N, I, R  
inicio  
    limpar  
    posicionar 1,1  
    escrever "Entre um valor numerico: "  
    posicionar 1,26  
    ler N  
    posicionar 3,1  
    I = 1  
    faca enquanto (I <= 10)  
        R = N * I  
        escrever N, " X ", I, " = ", R  
        I = I + 1  
    fim_enquanto  
fim
```

Grave o programa com o nome **cap0401.ila** e em seguida faça sua execução.

Note que os comandos **faca enquanto** com a condição verdadeira faz o cálculo do processamento $R = N * I$, apresenta a tabuada e efetua o cálculo do incremento de **1** a variável **I**. Enquanto o valor da variável **I** permanecer menor ou igual a **10** o programa efetua a execução do bloco de ações entre os comandos **faca enquanto** e **fim_enquanto**.

A título de ilustração o próximo código mostra a ação de um programa que apresenta os valores de uma tabuada de um número qualquer a partir do uso de um laço pré-teste com fluxo de ação falso. Observe para este recurso as orientações do capítulo 5, tópico 5.3 ou 5.1, respectivamente para os livros *Algoritmos: Lógica para Desenvolvimento de Programação de Computadores* ou *Estudo Dirigido de Algoritmos*. Assim sendo, considere o seguinte código:

```
// programa LACO2
variaveis
  numerico N, I, R
inicio
  limpar
  posicionar 1,1
  escrever "Entre um valor numerico: "
  posicionar 1,26
  ler N
  posicionar 3,1
  I = 1
  faca enquanto nao (I > 10)
    R = N * I
    escrever N, " X ", I, " = ", R
    I = I + 1
  fim_enquanto
fim
```

Grave o programa com o nome **cap0402.ila** e em seguida faça sua execução.

Note que os comandos **faca enquanto** com a condição sendo falsa efetua o cálculo do processamento $R = N * I$, apresenta a tabuada e processa o incremento de 1 a variável I. Enquanto o valor da variável I permanecer maior que 10 o programa efetua a execução do bloco de ações entre os comandos **faca enquanto** e **fim_enquanto**.

Laço Condicional Interativo

Um laço condicional interativo é em ILA escrito com os mesmos comandos definidos para um laço condicional iterativo tendo como diferencial apenas os elementos de controle do laço. Como exemplo de uso de laço condicional interativo considere: *Desenvolver um programa que solicita a entrada de um valor numérico e apresenta o resultado do quadrado do valor fornecido enquanto o usuário assim o desejar.*

```
// programa LACO3
variaveis
  numerico N, R
  caracter RESP
inicio
  RESP = "S"
  faca enquanto (RESP = "S") ou (RESP = "s")
    limpar
    posicionar 1,1
    escrever "Entre um valor numerico: "
    posicionar 1,26
    ler N
    R = N ^ 2
    posicionar 3,1
    escrever "O quadrado de ", N, " equivale a ", R
    posicionar 6,1
    escrever "Deseja continuar?"
    posicionar 7,1
    escrever "[S] para sim / outra letra para nao."
    posicionar 7,38
    ler RESP
  fim_enquanto
fim
```

Grave o programa com o nome **cap0403.ila** e em seguida faça sua execução.

Note que após fornecer um valor para o cálculo do quadrado o programa pergunta se há o desejo de processar novo cálculo. Enquanto o usuário fornecer para a pergunta as respostas **S** ou **s** o programa efetuará novo cálculo. No momento que for fornecido um conteúdo de respostas diferente de **S** ou **s** o programa será encerrado.

Laço Incondicional

Os laços iterativos incondicionais podem ser construídos de maneira simplificada com o uso dos comandos **para / ate / passo / proximo**.

Os laços que possuírem um número finito de passos são efetivados por meio de uma variável de controle do tipo contador que pode ser crescente ou decrescente, o qual possui como sintaxe:

```
para <variável> = <início> ate <fim> [passo <incremento>]  
    ação executada a quantidade de vezes definida  
proximo
```

Onde a **variável** definida após o comando **para** possuirá um valor que irá variar do valor de *início* até o valor de *fim*, de acordo com o valor de **incremento** estabelecido que quando for **1** poderá ser omitido. O comando **proximo** finaliza a ação do laço.

Como exemplo considere um programa que apresente como resultado a tabuada de um número qualquer. Observe para este recurso as orientações do capítulo 5, tópico 5.6 ou 5.3, respectivamente para os livros *Algoritmos: Lógica para Desenvolvimento de Programação de Computadores* ou *Estudo Dirigido de Algoritmos*. Assim sendo, considere o seguinte código:

```
// programa LACO4
variaveis
  numerico N, I, R
inicio
  limpar
  posicionar 1,1
  escrever "Entre um valor numerico: "
  posicionar 1,26
  ler N
  posicionar 3,1
  para I = 1 ate 10 passo 1
    R = N * I
    escrever N, " X ", I, " = ", R
  proximo
fim
```

Grave o programa com o nome **cap0404.ila** e em seguida faça sua execução.

O comando **para** efetua para a variável **I** um acréscimo de **1** toda vez que o comando **proximo** é executado dentro da faixa estabelecida de valores para a variável indicada, neste caso é **I = 1 ate 10** com o passo de evolução **passo 1**. A instrução **para I = 1 ate 10 passo 1** do programa anterior poderia ter sido estabelecida como **para I = 1 ate 10** uma vez que o trecho **passo 1** é opcional quando a evolução da contagem entre o início e fim é de um em um. Para incrementos diferentes de **1** o uso do comando **pas-so** é obrigatório.

O exemplo a seguir demonstra a apresentação dos valores numéricos gerados entre a contagem de **1 a 20** de **2** em **2**.

```
// programa LACO5
variaveis
  numerico I
inicio
  para I = 1 ate 20 passo 2
    escrever I
  proximo
fim
```

Grave o programa com o nome **cap0405.ila** e em seguida faça sua execução.

Considerações sobre Laços

Por ser o programa ILA um interpretador de linguagem algorítmica simples, este possui um conjunto limitado de operações (pelo menos até a versão 1.01 usada para a produção desta obra). Assim sendo, sua parte de sua limitação está no conjunto de laços suportados não se tendo a possibilidade de definir laços do tipo pós-teste.

No entanto, a existência dos conjuntos de laços **faca enquanto / fim_ - enquanto** e **para / ate / passo / próximo** é suficiente para que um aprendiz em programação tenha nítida noção de como esses recursos operam no contexto da programação de computadores.

5

Programação com Funções

Neste capítulo são apresentadas as funções internas existentes no programa ILA indicadas junto ao comando **escrever** (para serem usadas devem ser incorporadas dentro do bloco de um programa delimitados entre os comandos **inicio** e **fim**). Funções são recursos que permitem realizar de maneira simplificada alguma ação específica operacional, como por exemplo, ações matemáticas. Uma função é um recurso que retorna resposta após seu uso.

Funções Matemáticas

As funções ILA definidas para a realização de operações matemáticas são: **acos()**, **aleatorio()**, **asen()**, **atan()**, **cos()**, **inteiro()**, **log()**, **raiz()**, **resto()**, **sen()** e **tan()** indicadas a seguir:

acos(n)

Esta função retorna o resultado do arco-cosseno do valor **n** fornecido:

```
escrever acos(-1) // mostra 3.14
escrever acos(1) // mostra 0
```

aleatório()

Esta função retorna um valor de ponto flutuante entre 0 e 1:

```
escrever aleatorio() // mostra entre 0 e 1
```

asen(n)

Esta função retorna o resultado do arco-seno do valor **n** fornecido:

```
escrever asen(-1) // mostra -1.57  
escrever asen(1) // mostra 1.57
```

atan(n)

Esta função retorna o resultado do arco-tangente do valor **n** fornecido:

```
escrever atan(-1) // mostra -0.79  
escrever atan(1) // mostra 0.79
```

cos(n)

Esta função retorna o resultado do cosseno do valor **n** fornecido:

```
escrever cos(-1) // mostra 0.54  
escrever cos(1) // mostra 0.54
```

inteiro(n)

Esta função apresenta a parte inteira de um valor numérico com ponto flutuante do valor **n** fornecido:

```
escrever inteiro(1.5) // mostra 1  
escrever inteiro(1.4) // mostra 1
```

log(n)

Esta função apresenta o logaritmo de base 10 do valor **n** fornecido:

```
escrever log(2) // mostra 0.30
escrever log(1) // mostra 0
```

raiz(b,i)

Esta função retorna o valor da raiz de uma base (**b**) de um índice (**i**):

```
escrever raiz(25,2) // mostra 5
escrever raiz(10,3) // mostra 2.15
```

resto(n,m)

Esta função retorna o resto da divisão do dividendo **n** pelo divisor **m**:

```
escrever resto(5,2) // mostra 1
escrever resto(4*5,6/2) // mostra 2
```

sen(n)

Esta função retorna o resultado do seno do valor **n** fornecido:

```
escrever sen(-1) // mostra -0.84
escrever sen(1) // mostra 0.84
```

tan(n)

Esta função retorna o resultado da tangente do valor **n** fornecido:

```
escrever tan(-1) // mostra -1.56
escrever tan(1) // mostra 1.56
```

Funções Gerais

As funções para a realização de operações gerais (genéricas) em ILA são: **comprimento()**, **esperar()**, **impressora()**, **lertecla()**, **parte()**, **teste()**, **testef()**, **valor()** e **video()**, indicadas a seguir:

comprimento(c)

Esta função retorna a quantidade de caracteres do parâmetro **c**:

```
escrever comprimento("A")      // mostra 1
escrever comprimento("Teste") // mostra 5
```

esperar(n)

Esta função estabelece o tempo em milissegundos de espera no processamento da próxima instrução. Para aguardar 3 segundo deve-se estabelecer o valor 3000 (não funcionou no ILA+, apenas no ILA para arquitetura de 32 bits):

```
escrever "Teste 1" // mostra mensagem: Teste 1
esperar(3000)     // espera 3 segundos
escrever "Teste 2" // mostra mensagem: Teste 2 após 3 seg.
```

impressora()

Esta função desvia a saída do comando **escrever** para a impressora, se esta estiver devidamente conectada e configurada. Não podendo enviar para impressora o conteúdo é direcionado para o monitor de vídeo:

```
impressora() // habilita impressora
escrever "Teste" // mostra mensagem: Teste na impressora
```

lertecla()

Esta função captura o valor ASCII da tecla acionada no teclado:

```
numerico TECLA      // define variável (comando variáveis)
TECLA = lertecla()  // captura valor da tecla acionada
escrever TECLA      // mostra valor da tecla acionada
```

parte(c,i,q)

Esta função retorna uma parte da cadeia **c** delimitadas entre o início da posição **i** e a quantidade de caracteres a ser retirado **q**:

```
escrever parte("comparador", 1, 3) // mostra: com
escrever parte("comparador", 4, 4) // mostra: para
escrever parte("comparador", 8, 3) // mostra: dor
```

teste()

Esta função habilita a execução de um depurador durante a interpretação de um programa mostrando uma caixa na tela com o nome da variável corrente e seu conteúdo, devendo ser escrita após o comando **inicio**.

O efeito dessa funcionalidade é melhor percebido no ILA para 32 bits. No ILA+ o efeito não ocorre de forma satisfatória na versão 1.01.

É pertinente salientar que independentemente do uso da função **teste()** o ILA gera um arquivo chamado MESA.MAP que armazena os valores das variáveis e sua evolução durante a execução de um programa.

testef()

Esta função desabilita a execução simultânea do depurador.

valor(c)

Esta função converte o conteúdo numérico de seu parâmetro em sua forma numérica:

```
escrever valor("123")           // mostra 123
escrever valor("1") + valor("1") // mostra 2
```

video()

Esta função desvia a saída do comando **escrever** para o monitor de vídeo, sendo esta a forma padrão de operação do interpretador ILA:

```
video() // definido após comando: inicio
```

Constantes Matemáticas e Lógicas

Para auxiliar algumas operações de processamento matemático ou lógico o ILA oferece as constantes **pi** e **np** para processamento matemático e **falso** e **verdadeiro** para processamento lógico, os quais são comentados a seguir.

pi

A constante numérica **pi** estabelece para uso o valor **3.14**:

```
escrever pi // mostra o valor 3.14
```

np

A constante numérica **np** estabelece para uso o valor **2.72**:

```
escrever np           // mostra o valor 2.72
escrever log(np)      // mostra o resultado de log(e) = 0.43
```

verdadeiro

A constante lógica com valor **verdadeiro**:

```
logico VLR      // define variável (comando variáveis)
VLR = verdadeiro // atribui valor verdadeiro em valor
escrever VLR    // mostra verdadeiro
```

falso

A constante lógica com valor **falso**:

```
logico VLR // define variável (comando variáveis)
VLR = falso // atribui valor verdadeiro em valor
escrever VLR // mostra verdadeiro
```


6

Programação com Matrizes

Este capítulo mostra a técnica de programação que permite trabalhar com o agrupamento de dados em uma mesma variável na forma de uma coleção de dados. Este tipo de agrupamento faz uso do mesmo tipo de dado dando origem a estrutura denominada tabelas de vetores e tabelas de matrizes. Também são chamados de arranjos, variáveis indexadas, variáveis compostas ou listas.

Vetores

Um vetor (tabela de uma dimensão, também conhecido como tabela unidimensional ou variável composta ou ainda variável indexada) é representado também com o uso do comando **variaveis** usado para definir variáveis simples, o nome da variável e a indicação da dimensão de seu tamanho, sendo com a sintaxe:

```
variaveis
  matriz <tipo> <var1[tamanho]>
  matriz <tipo> <var2[tamanho]>
  matriz <tipo> <varN[tamanho]>
```

Onde **var1[]**, **var2[]** e **varN[]** representam os nomes das matrizes, seus **tamanhos** sendo a definição da quantidade de elementos a serem armazenados e o comando **matriz <tipo>** a definição da matriz com seu respectivo tipo de dados.

Para fazer uso de vetores em ILA é necessário declarar o vetor no início do código de programa antes de usá-lo.

A título de ilustração considere: *Desenvolver um programa que efetue a leitura de 10 elementos numéricos em uma matriz tipo vetor chamada **A**. Em seguida, construa um vetor chamado **B**, em que o vetor **B** seja formado de acordo com a seguinte lei de formação: se o valor do índice do vetor **A** for par, o valor deve ser multiplicado por 5; sendo o índice do vetor **B** for ímpar, deve ser somado com 5. Ao final, mostrar o conteúdo do vetor **B**.* Observe para este recurso as orientações do capítulo 6, tópico 6.2 ou 6.1, respectivamente para os livros *Algoritmos: Lógica para Desenvolvimento de Programação de Computadores* ou *Estudo Dirigido de Algoritmos*. Assim sendo, considere o seguinte código:

```
// programa TABELA1
variaveis
    numerico I, R
    matriz numerico A[10]
    matriz numerico B[10]
inicio
    limpar
    posicionar 1,1
    escrever "Índice de Tabela"
    posicionar 2,1
    para I = 1 ate 10
        posicionar I + 2,1
        escrever "Informe o ", I, "o. valor:"
        posicionar I + 2,23
        ler A[I]
    proximo
    para I = 1 ate 10
        R = resto(I, 2)
        se (R = 0) entao
            B[I] = A[I] * 5
        senao
            B[I] = A[I] + 5
        fim_se
    proximo
    posicionar 14,1
```

```
para I = 1 ate 10
  escrever "B[", I, "] = ", B[I]
proximo
fim
```

Grave o programa com o nome **cap0601.ila** e em seguida faça sua execução.

No programa são usados três laços de repetição iterativos **para / ate / proximo**; o primeiro laço controla a entrada dos dados numéricos, o segundo laço verifica se cada elemento de **A** é par ou se é ímpar processando o cálculo desejado, implicando os elementos calculados em **B**, o terceiro laço é utilizado para apresentar a matriz **B**.

No laço de processamento é utilizada a instrução **se (R = 0) entao**, onde a variável **R** é atribuída com o resultado da expressão **resto(I, 2)** que possibilita obter o resultado do resto da divisão de valores. Qualquer valor numérico dividido por **2** com quociente inteiro que resultar zero é par; se o resto for diferente de zero, o valor é ímpar.

Na sequência considere: *Desenvolver um programa que efetue a leitura de cinco elementos numérico em uma tabela **A**. No final, apresente o total da soma de todos os elementos que sejam ímpares. Em relação ao primeiro exemplo este apresenta uma diferença: o primeiro pedia para verificar se o índice era par ou ímpar. Neste exemplo, é solicitado que seja feita uma análise da condição do elemento e não do índice em si. Já foi alertado anteriormente para tomar cuidado e não confundir elemento com índice. Veja o programa.*

```
// programa TABELA2
variaveis
  numerico I, R, SOMA
  matriz numerico A[5]
inicio
  limpar
  posicionar 1,1
  escrever "Somatorio de elementos impares"
  posicionar 2,1
  para I = 1 ate 5
    posicionar I + 2,1
    escrever "Informe o ", I, "o. valor:"
    posicionar I + 2,23
    ler A[I]
  proximo
  SOMA = 0
  para I = 1 ate 5
    R = resto(A[I], 2)
    se (R <> 0) entao
      SOMA = SOMA + A[I]
    fim_se
  proximo
  escrever " " // pula linha em branco
  escrever "Soma = ", SOMA
fim
```

Grave o programa com o nome **cap0602.ila** e em seguida faça sua execução.

No trecho de processamento, usa-se a instrução **se (R <> 0) entao** que quando verdadeira executa a linha de código **SOMA = SOMA + A[I]** que ao final conterà o valor do somatório dos elementos ímpares informados durante a fase de entrada na execução do programa.

Tabelas

Com os detalhes apresentados anteriormente é possível a seguir criar um programa que efetue a leitura das notas escolares de oito alunos e as apresente, utilizando apenas tabelas unidimensionais. Porém, há de se considerar que este tipo de trabalho é grande, uma vez que é ne-

cessário manter um controle de cada índice em cada uma das tabelas para cada um dos alunos. Para facilitar o trabalho com estruturas deste porte, basta utilizar tabelas com mais de uma dimensão. A forma mais comum é usar tabela de duas dimensões. Tabelas com mais de duas dimensões são de uso menos frequente, mas são fáceis de serem implementadas quando já se conhece bem a utilização de tabelas com duas dimensões.

Uma tabela de duas dimensões é aquela que possui um determinado número de colunas com um determinado número de linhas. Por exemplo, uma tabela com 5 linhas e 3 colunas, ou seja, uma tabela de 5x3 seria definida em ILA como:

```
variaveis
  matriz <tipo> <var1[linhas, colunas]>
  matriz <tipo> <var2[linhas, colunas]>
  matriz <tipo> <varN[linhas, colunas]>
```

As tabelas de duas necessitam da definição de uma valor para linhas e para colunas.

Considere para uso de tabelas: *Desenvolver um programa que leia e apresente as quatro notas escolares de oito alunos*. Observe para este recurso as orientações do capítulo 8, tópico 8.2 ou 8.1, respectivamente para os livros *Algoritmos: Lógica para Desenvolvimento de Programação de Computadores* ou *Estudo Dirigido de Algoritmos*. Assim sendo, considere o seguinte código:

```
// programa TABELA3
variaveis
  numerico I, J
  matriz numerico NOTA[8, 4]
inicio
  limpar
  posicionar 1,1
  escrever "Leitura das Notas"
  posicionar 2,1
  para I = 1 ate 8
    posicionar I + 2,1
    escrever "Notas do ", I, "o. aluno:"
    para J = 1 ate 4
      posicionar I + 2, (J * 6) + 16
      ler NOTA[I,J]
    proximo
  escrever " "
  proximo
  posicionar 12,1
  escrever "Apresentacao das Notas"
  para I = 1 ate 8
    posicionar I + 13,1
    escrever I, "o. aluno:"
    para J = 1 ate 4
      posicionar I + 13, (J * 6) + 16
      escrever NOTA[I,J]
    proximo
  proximo
fim
```

Grave o programa com o nome **cap0603.ila** e em seguida faça sua execução.

Entre as notas solicitadas e veja a apresentação das mesmas. Note os laços **I** e **J** que controlam os limites das dimensões da tabela **NOTA** em uso.

Registros

Registros caracterizam-se por ser uma coleção de dados de tipos diferentes agrupados em uma mesma variável. Assim sendo, é possível combinar vários dados de tipos diferentes (os quais serão chamados de campos) em uma mesma estrutura de dados. Por esta razão, esse tipo de estrutura de dados é considerado heterogêneo.

No programa ILA registros são definidos por uma estrutura definida a partir do uso dos comandos **matriz composta** e **fim_composta**, que é escrita de acordo com a seguinte estrutura sintática:

```
matriz composta <registro>  
  <tipo> <campo1>  
  <tipo> <campo2>  
  <tipo> <campo3>  
  <tipo> <campoN>  
fim_composta
```

Para exemplificar o uso de registros considere: *Desenvolver um programa que receba a entrada de dois alunos em uma matriz de dados heterogêneos e suas respectivas notas bimestrais. Ao final o programa apresenta o nome as notas e as médias de cada aluno, além da média da sala.* Observe para este recurso as orientações do capítulo 9, tópico 9.2 ou 9.1, respectivamente para os livros *Algoritmos: Lógica para Desenvolvimento de Programação de Computadores* ou *Estudo Dirigido de Algoritmos*. Assim sendo, considere o seguinte código:

```

// programa TABELA4

variaveis

matriz composta ALUNO[2]
  caracter NOME
  caracter TURMA
  numerico SALA
  matriz numerico NOTA[4]
fim_composta

numerico I, J

inicio
para I = 1 ate 2
  limpar
  posicionar 1,1
  escrever "Leitura das notas do aluno ", I
  posicionar 3,1
  escrever "Entre com o nome ...:"
  posicionar 3,23
  ler ALUNO[I]:NOME
  posicionar 4,1
  escrever "Entre com a turma ..:"
  posicionar 4,23
  ler ALUNO[I]:TURMA
  posicionar 5,1
  escrever "Entre com a sala ...:"
  posicionar 5,23
  ler ALUNO[I]:SALA
  posicionar 7,1
  escrever "Notas do aluno"
  para J = 1 ate 4
    posicionar 8 + J,1
    escrever "Nota:", J
    posicionar 8 + J,8
    ler ALUNO[I]:NOTA[J]
  proximo
proximo
limpar
posicionar 1,1
escrever "Apresentacao das Notas"
escrever " "
para I = 1 ate 2
  escrever I, "o. aluno .: ", ALUNO[I]:NOME
  escrever "Turma .....: ", ALUNO[I]:TURMA

```

```
escrever "Sala .....: ", ALUNO[I]:SALA
para J = 1 ate 4
    escrever "Nota .....: ", J, " = ", ALUNO[I]:NOTA[J]
    proximo
    escrever " "
    proximo
fim
```

Grave o programa com o nome **cap0604.ila** e em seguida faça sua execução.

Observe que o uso de matrizes compostas como registros em ILA exige que se utilize de um operador de escopo (:), representado por dois pontos para efetuar a separação do nome do campo do registro em relação a variável heterogênea associada ao registro definido.

7

Programação Estruturada

Este capítulo apresenta as bases básicas da técnica de programação estruturada com o programa ILA. Apresenta informações sobre o uso de sub-rotinas definidas pelo programador.

Paradigma Estruturado

O programa ILA fornece recursos para a criação de sub-rotinas. Neste sentido é oferecido o comando **funcao** para a criação de funções sem retorno (estilo procedimento) e funções com retorno (estilo função pura com auxílio do comando **retornar**).

Uma sub-rotina seja, procedimento ou função, é um bloco de programa que pode ser chamado e executado a qualquer momento. No caso de um procedimento este quando chamado devolve sua execução à primeira linha de código após sua chamada. No entanto, uma função devolve sua execução exatamente no local da sua chamada por meio do retorno de um valor como resposta, por esta razão uma função poderá ser usada em atribuições, decisões ou diretamente em ações de exibição.

O código de uma sub-rotina deve ser escrito antes de sua chamada no bloco principal do programa e deverá seguir a sintaxe:

```
função <nome> ([parâmetros])
inicio
    instruções para ação da sub-rotina
    [retornar valor]
fim
```

Se omitido o comando **retornar** de uma função ILA esta estrutura será equivalente a uma sub-rotina do tipo procedimento que pode ser usada para efetuar entrada, saída e processamento no corpo de uma sub-rotina. Mas se usado o comando **retornar** a sub-rotina será uma função pura e neste caso não deverá efetuar ações de entrada/saída em seu corpo, mantendo-se apenas as ações de processamento.

Uma sub-rotina como estrutura, independentemente, de seu foco (procedimento ou função) poderá opcionalmente receber parâmetros que são formas de enviar valores dentro da sub-rotina.

Devido a característica operacional da ILA o interpretador não opera o escopo local de variáveis. Todas as variáveis em uso são sempre de escopo global.

A seguir são apresentadas as formas de uso das sub-rotinas em ILA que são assuntos constantes do capítulo 10 do livro de Algoritmos: Lógica para Desenvolvimento de Programação de Computadores e dos capítulos 10, 11 e 12 do livro Estudo Dirigido de Algoritmos.

Sub-rotina sem Retorno

Para demonstrar o uso de sub-rotinas tipo procedimento considere como exemplo um programa que efetua a apresentação do resultado de uma fatorial de um valor numérico qualquer calculado dentro de uma sub-rotina.

Ao ser executado o programa forneça um valor pelo teclado e observe a apresentação do resultado do cálculo solicitado.

```
// programa FATORIAL1

variaveis
  numerico N, F, I

funcao FATORIAL()
  inicio
    F = 1
    para I = 1 ate N
      F = F * I
    proximo
  escrever F
  fim

inicio
  ler N
  fatorial()
  fim
```

Grave o programa com o nome **cap0701.ila** e em seguida faça sua execução.

Note que no trecho principal ocorre após o uso do comando **ler** a chamada da sub-rotina **fatorial()** que utilizando-se do valor da variável **N** efetua no laço **para I = 1 ate N** o cálculo da fatorial pela execução da instrução **F = F * I** e assim efetua a apresentação do resultado com a instrução **escrever F**.

A mesma aplicação de cálculo de fatorial pode fazer uso de uma estrutura de sub-rotina com passagem de parâmetro

```
// programa FATORIAL2

variaveis
  numerico ENTRA, N, F, I

funcao FATORIAL(N)
inicio
  F = 1
  para I = 1 ate N
    F = F * I
  proximo
  escrever F
fim

inicio
  ler ENTRA
  fatorial(ENTRA)
fim
```

Grave o programa com o nome **cap0702.ila** e em seguida faça sua execução.

No trecho principal ocorre a entrada da variável **ENTRA** com o comando **ler** que é passada como parâmetro à sub-rotina **fatorial(N)** que utilizando-se do valor recebido efetua no laço **para I = 1 ate N** o cálculo da fatorial pela execução da instrução **F = F * I** e assim efetua a apresentação do resultado com a instrução **escrever F**.

Sub-rotina com Retorno

Para demonstrar o uso de sub-rotinas tipo função pura considere como exemplo um programa que efetua a apresentação do resultado de uma fatorial de um valor numérico qualquer calculado dentro de uma sub-rotina.

Ao ser executado o programa forneça um valor pelo teclado e observe a apresentação do resultado do cálculo solicitado.

```
// programa FATORIAL3

variaveis
  numerico ENTRADA, N, F, I

funcao FATORIAL(N)
inicio
  F = 1
  para I = 1 ate N
    F = F * I
  proximo
  retornar F
fim

inicio
  ler ENTRADA
  escrever fatorial(ENTRADA)
fim
```

Grave o programa com o nome **cap0703.ila** e em seguida faça sua execução.

No trecho principal ocorre a entrada da variável **ENTRADA** com o comando **ler** que é passada como parâmetro à sub-rotina **fatorial(N)** que utilizando-se do valor recebido no laço **para I = 1 ate N** efetua o cálculo da fatorial pela execução da instrução **F = F * I** e faz o retorno do valor da variável **F** com o comando **retornar** ao ponto de chamada da sub-rotina no programa principal executado pela instrução **escrever fatorial(ENTRADA)**.

Sub-rotina Recursiva

Devido à característica operacional e de retorno de valor que uma sub-rotina do tipo função pura proporciona é possível desenvolver funções que fazem chamadas a si mesmas. Esse efeito denomina-se recursividade e permite executar a ação de um processamento sem uso de laço.

O uso de recursividade proporciona a escrita de um código mais elegante com alto grau de abstração. Como exemplo de função recursiva, considere o resultado do cálculo da fatorial de um número.

```
// programa FATORIAL4

variaveis
  numerico ENTRADA, N, I

funcao FATORIAL(N)
inicio
  se (N < 0) entao
    retornar 0
  fim_se
  se (N = 0) entao
    retornar 1
  senao
    retornar N * fatorial(N - 1)
  fim_se
fim

inicio
  ler ENTRADA
  escrever fatorial(ENTRADA)
fim
```

Grave o programa com o nome **cap0704.ila** e em seguida faça sua execução.

Ao ser executado o programa o código definido dentro junto a função **fatorial(N)** verifica se o valor fornecido é menor que zero e retorna para tal o resultado **0** (zero), se fornecido o valor **0** (zero) será retornado **1**. Qualquer valor diferente de **0** e **1** retornará o resultado respectivo da fatorial desejada com a instrução **retornar N * fatorial(N - 1)**. O parâmetro **N** determina o número de vezes que a operação deve ser efetuada e **fatorial(N - 1)** define a próxima instância de chamada da função a si mesma com o valor do parâmetro **N** menos **1**.

8

Recursos Complementares

Este capítulo mostra alguns recursos complementares existentes no ILA que não foram abordados nos demais capítulos. São assim apresentados recursos com definição de cores e criação de janelas.

Manipulação de Cores

Entre os diversos recursos oferecidos pelo programa ILA há um em especial que possibilita estabelecer a mudança das cores da apresentação de mensagens na tela. É possível alterar as cores de fundo e de texto.

A definição de cores ocorre com o uso do comando **COR** que possui a sintaxe:

```
cor [<cortexto,>,<corfundo>]
```

O comando **cor** sem parâmetros faz a tela ser apresentada com fundo preto e cor de texto branco. Quando usado com os parâmetros **cor-texto** e **corfundo** poderá ser definido com o código numérico da cor ou com a descrição textual do nome da cor entre aspas inglesas.

As cores de texto e de fundo são operacionalizadas a partir de dezesseis cores de acordo com a tabela de cores que apresenta os valores numéricos e os nomes das cores a serem usadas no comando **COR**:

Valor Numérico	Constante de Cor	Cor Real
0	preto	preto
1	azul	azul escuro
2	verde	verde escuro
3	ciano	ciano escuro
4	vermelho	vermelho escuro
5	magenta	magenta escuro
6	marron	amarelo escuro
7	cinza	cinza escuro
8	preto_intenso	cinza claro
9	azul_intenso	azul claro
10	verde_intenso	verde claro
11	ciano_intenso	ciano claro
12	vermelho_intenso	vermelho claro
13	magenta_intenso	magenta claro
14	amarelo	amarelo claro
15	branco	branco

Uma sub-rotina seja, procedimento ou função, é um bloco de programa que pode ser chamado e executado a qualquer momento. No caso de um procedimento este quando chamado devolve sua execução à primeira linha de código após sua chamada. No entanto, uma função devolve sua execução exatamente no local da sua chamada por meio do retorno de um valor como resposta, por esta razão uma função poderá ser usada em atribuições, decisões ou diretamente em ações de exibição.

O código a seguir apresenta a combinação de cores possíveis de uso com o programa ILA:

```
// programa COMPLEMENTAR1

variaveis
  numerico CORT, CORF

inicio
  para CORT = 0 ate 15
    para CORF = 0 ate 15
      posicionar CORT + 1, 4 * (CORF + 1)
      cor CORT,CORF
      escrever " * "
    proximo
  proximo
  cor 7,0
fim
```

Grave o programa com o nome **cap0801.ila** e em seguida faça sua execução.

Devido a característica operacional da ILA a última cor apresentada que deveria ser fundo e texto branco são apresentados com fundo amarelo e texto branco.

Utilização de Molduras

Além da manipulação de cores o programa ILA oferece um recurso para desenhar molduras simples na tela a fim de estabelecer caixas de texto e cercaduras para programa com o comando **janela** que possui a sintaxe:

```
janela <linhasup>, <colunasup>, <linhainf>, <colunainf>
```

Os parâmetros **linhasup** e **colunasup** são usados para definir as coordenadas do canto superior esquerdo e os parâmetros **linhainf** e **co-**

lunainf são usados para definir as coordenadas do canto inferior direito.

O código a seguir apresenta a definição de uma moldura com fundo ciano, borda branca e um texto escrito internamente na cor amarela com fundo ciano:

```
// programa COMPLEMENTAR2

inicio
  limpar
  cor "branco", "ciano"
  janela 3,3,7,22
  posicionar 5,5
  cor "amarelo", "ciano"
  escrever "TESTE DE MOLDURA"
  cor
fim
```

Grave o programa com o nome **cap0802.ila** e em seguida faça sua execução.

Para fazer uso de molduras com textos é fundamental primeiro definir a moldura primeiro para depois definir o texto que se irá inserir na moldura. A sequência em contrário não surtirá o efeito de apresentação do texto.

A definição de coordenadas de janela necessitam as vezes serem limpas e para tal ação o comando **limpar** pode ser usado com a sintaxe:

```
limpar <linhasup>, <colunasup>, <linhainf>, <colunainf>
```

O comando **limpar** com uso de parâmetros opera com as mesmas regras dos parâmetros do comando **janela**.

O código a seguir apresenta o efeito de limpeza implementado sobre a área de uma janela:

```
// programa COMPLEMENTAR3

variaveis
  numerico VALOR

funcao TECLE()
  inicio
    posicionar 24,5
    escrever "Tecla <barra de espaco>:"
    TECLA = 0
    faca enquanto (TECLA <> 32)
      limpar 24,30,24,31
      posicionar 24,30
      TECLA = lertecla()
    fim_enquanto
  fim

inicio
  limpar
  cor "branco", "vermelho"
  janela 3,3,7,22
  posicionar 5,5
  cor "amarelo", "vermelho"
  escrever "TESTE DE MOLDURA"
  cor "verde_intenso", "preto"
  tecla()
  limpar 5,5,5,20
  posicionar 5,5
  cor "ciano_intenso", "vermelho"
  escrever " TEXTO ALTERADO "
  cor
fim
```

Grave o programa com o nome **cap0803.ila** e em seguida faça sua execução.

Ao ser executado o programa são apresentados dentro da janela demarcada dois textos. O primeiro texto é apresentado e aguarda o acionamento da *tecla de espaço* para que seja o primeiro texto limpadado da

janela e um segundo texto seja apresentado. Observe detalhadamente no programa o uso do comando **limpar** dentro do programa principal e dentro da função que detecta o acionamento da tecla de espaço.

Funções Definidas pelo Programador

Uma das tarefas de qualquer programador é construir rotinas de programa que venham a executar certas tarefas, as quais muitas vezes não fazem parte da ferramenta de programação que se está utilizando.

Neste contexto a ferramenta ILA fornece algumas funções que foram usadas em diversos exemplos deste livro e foram relacionadas no capítulo 5. No entanto, seu conjunto não é completo faltando algumas ações, muitas vezes necessárias a produção de programas de computadores. Por exemplo, faltam algumas funções típicas como operação de módulo (valor negativo como positivo), teto de um número (arredondamento para cima), centralização de caracteres e conversão de valor numérico em formato caractere.

A falta desses e de outros recursos é propício para o desenvolvimento de um conjunto de funções e constantes próprias, como segue.

O exemplo a seguir desenvolve e demonstra o uso da função definida pelo programador **modulo()**.

```
// programa COMPLEMENTAR4

variaveis
  numerico N, ENTRA

funcao MODULO(N)
inicio
  se (N < 0) entao
    retornar N * -1
  senao
    retornar N
  fim_se
fim

inicio
  limpar
  posicionar 1,1
  escrever "Entre um valor numerico:"
  posicionar 1,26
  ler ENTRA
  escrever "Resultado = ", modulo(ENTRA)
fim
```

Grave o programa com o nome **cap0804.ila** e em seguida faça sua execução.

Ao ser executado o programa este apresenta sempre o valor fornecido como positivo, mesmo que a entrada seja de um valor negativo.

O próximo exemplo demonstra o uso de uma função que calcula o teto de um valor numérico fornecido.

```
// programa COMPLEMENTAR5

variaveis
  numerico N, ENTRA

  funcao TETO(N)
  inicio
    retornar inteiro(N) + 1
  fim

inicio
  limpar
  posicionar 1,1
  escrever "Entre um valor numerico:"
  posicionar 1,26
  ler ENTRA
  escrever "Resultado = ", teto(ENTRA)
fim
```

Grave o programa com o nome **cap0805.ila** e em seguida faça sua execução.

Ao ser executado o programa este apresenta sempre o maior valor em relação ao valor fornecido. Por exemplo, se fornecido o valor **3.1** será apresentado como resposta o valor **4**.

O próximo exemplo demonstra o uso de uma função que centraliza no monitor de vídeo um texto fornecido pelo teclado.

```
// programa COMPLEMENTAR6

variaveis
  numerico L
  caracter TXT, ENTRA

funcao CENTRATEXTO(L, TXT)
inicio
  posicionar L, inteiro((80 - comprimento(TXT) ) / 2)
  escrever TXT
fim

inicio
  limpar
  posicionar 1,1
  escrever "Entre um texto:"
  posicionar 1,17
  ler ENTRA
  cor "amarelo", 0
  centratexto(3, ENTRA)
  cor 7, 0
fim
```

Grave o programa com o nome **cap0806.ila** e em seguida faça sua execução.

Ao ser executado o programa o texto fornecido é apresentado de forma centralizada. Para este efeito fez-se o cálculo do comprimento do texto fornecido que é subtraído de **80** que é quantidade de colunas da tela. O valor obtido é dividido por **2** e o resultado inteiro corresponde o ponto de centralização para a mensagem fornecida.

O programa seguinte efetua a entrada de um valor numérico na faixa de **-999.999.99** até **999.999.999** e o transforma em uma sequência alfanumérica. Qualquer valor fora da faixa permitida retornará a mensagem **Erro**. O programa solicita a entrada de dois valores numéricos e mostra o resultado da soma entre esses valores e apresenta em seguida uma concatenação dos valores fornecidos após serem convertidos para o formato caractere.

```
// programa COMPLEMENTAR7

variaveis
  numerico POSITIVO, UNIDADE, N, A, B, RN
  caracter SAIDA, RC

funcao MODULO(N)
inicio
  se (N < 0) entao
    retornar N * -1
  senao
    retornar N
  fim_se
fim

funcao CARACTERE(N)
inicio
  se (N >= -999999999) e (N <= 999999999) entao
    SAIDA = " "
  se (N = 0) entao
    SAIDA = "0"
  fim_se
  POSITIVO = modulo(N)
  faca enquanto (POSITIVO > 0)
    UNIDADE = resto(POSITIVO, 10)
    faca caso
      caso UNIDADE = 0:
        SAIDA = "0" + SAIDA
      caso UNIDADE = 1:
        SAIDA = "1" + SAIDA
      caso UNIDADE = 2:
        SAIDA = "2" + SAIDA
      caso UNIDADE = 3:
        SAIDA = "3" + SAIDA
      caso UNIDADE = 4:
        SAIDA = "4" + SAIDA
      caso UNIDADE = 5:
        SAIDA = "5" + SAIDA
      caso UNIDADE = 6:
        SAIDA = "6" + SAIDA
      caso UNIDADE = 7:
        SAIDA = "7" + SAIDA
      caso UNIDADE = 8:
        SAIDA = "8" + SAIDA
      caso UNIDADE = 9:
        SAIDA = "9" + SAIDA
```

```
        fim_caso
        POSITIVO = inteiro(POSITIVO / 10)
    fim_enquanto
    se (N < 0) entao
        SAIDA = "-" + SAIDA
    fim_se
    retornar SAIDA
senao
    escrever "Erro"
fim_se
fim

inicio
    ler A
    ler B
    RN = A + B
    escrever RN
    RC = caractere(A) + caractere(B)
    escrever RC
fim
```

Grave o programa com o nome **cap0807.ila** e em seguida faça sua execução.

Ao ser executado o programa entre dois valores numéricos. Por exemplo, se fornecido os valores **3** e **-7** serão apresentados os valores **-4** na variável **RN** e **3-7** para a variável **RC**. Veja que o valor **3-7** se caracteriza por seu uma concatenação.

A

Modelo ILA versus Modelo LPP

Este apêndice faz um paralelo entre o código português estrutura do livro *Algoritmos: Lógica para Desenvolvimento de Programação de Computadores* publicado pela editora Érica Saraiva, dos autores José Augusto N. G. Manzano e Jayr Figueiredo de Oliveira, aqui denominado LPP (Linguagem de Projeto de Programação Manzano e Oliveira) e o português do programa ILA. Mostra quais comandos são compatíveis e apresenta alguns exemplos de código do livro de Manzano e Oliveira codificados no padrão ILA.

Linguagem de Projeto de Programação

No processo de desenvolvimento de programas de computadores, pode-se utilizar a forma de representação textual denominada pseudocódigo (metalinguagem ou linguagem de especificação). O estilo pseudocódigo é uma linguagem de projeto de programação e não uma linguagem de programação real. Assim sendo, uma linguagem de projeto de programação não deve ter o mesmo rigor sintático que possui uma linguagem de programação formal (linguagem de programação real).

Assim sendo, apresentam-se relação de comandos LPP e sua relação com os comandos ILA. Na lista apresentada não são mostradas as todas funções do programa ILA apenas **inteiro()**, **resto()** e **raiz()**.

LPP	ILA
amigo / amiga	n/c
até	ate
até_que	n/c
até_seja	n/c
cadeia	caracter
caractere	caracter
caso	caso
classe	n/c
conjunto	matriz
const	n/c
execute	n/c
de	=
efetue	n/c
enquanto	faca enquanto
enquanto_seja	n/c
então	entao
escreva	escrever
faça	n/c
fim	fim
fim_até_seja	n/c
fim_caso	fim_caso
fim_classe	n/c
fim_enquanto	fim_enquanto
fim_faça	n/c
fim_laço	n/c
fim_para	próximo

LPP	ILA
fim_registro	fim_composta
fim_se	fim_se
função	funcao
herança	n/c
início	inicio
inteiro	numerico
laço	n/c
leia	ler
lógico	logico
objeto	n/c
para	para
passo	passo
privada	n/c
procedimento	funcao
programa	n/c
protegida	n/c
pública	n/c
real	numerico
registro	matriz composta
repita	n/c
saia_caso	n/c
se	se
seção_privada	n/c
seção_protegida	n/c
seção_pública	n/c
seja	n/c

LPP	ILA
senão	senao / outro_caso
tipo	n/c
var	variaveis
virtual	n/c
.e.	e
.ou.	ou
.não.	nao
.xou.	n/c
=	=
>	>
<	<
>=	>=
<=	<=
<>	<>
+	+
-	-
*	*
/	/
←	=
↑	^
a div b	inteiro(a, b)
a - n * (a div n)	resto(a, n)
b ↑ (1/i)	raiz(b, i)

Os comandos não existentes em ILA estão grafados com a sigla **n/c** (não consta), uma vez que se referem a comandos de representação de estrutura de orientação a objetos não suportados pelo ILA.

Exemplos de Codificação em ILA de LPP

No sentido em demonstrar o uso do processador de algoritmos apresentado neste texto e sua relação com a Linguagem de Projeto de Programação proposta no livro *Algoritmos: Lógica para Desenvolvimento de Programação de Computadores* da editora Érica Saraiva dos autores Manzano e Oliveira são indicados dois dos programas mais sofisticados do livro, sendo um programa com uso da técnica de pesquisa binária que exige prévia ordenação dos elementos a serem pesquisados e outro programa (calculadora) que faz uso de sub-rotinas para o gerenciamento das rotinas do programa.

Desta forma, mostra-se que é possível usar o processador de algoritmos apresentado neste texto com boa parte dos programas propostos no livro, mesmo existindo diferenças entre as estruturas sintáticas das linguagens do processador usado e do livro indicado.

Pesquisa binária em LPP padrão Manzano e Oliveira

```
programa PESQUISA_BINÁRIA

var
  VLR : conjunto[1..10] de inteiro
  I, J, COMEÇO, FINAL, MEIO, PESQ, X : inteiro
  RESP : caractere
  ACHA : lógico
início

  para I de 1 até 10 passo 1 faça
    leia VLR[I]
  fim_para

  {*** início trecho de ordenação ***}

  para I de 1 até 9 passo 1 faça
    para J de I + 1 até 10 passo 1 faça
      se (VLR[I] > VLR[J]) então
        X ← VLR[I]
        VLR[I] ← VLR[J]
        VLR[J] ← X
    fim_se
```

```

    fim_para
fim_para

{*** fim trecho de ordenação ***}

{*** início trecho de pesquisa binária ***}

RESP ← "S"
enquanto (RESP = "S") .ou. (RESP = "s") faça
    escreva "Entre valor a ser pesquisado: "
    leia PESQ
    COMEÇO ← 1
    FINAL ← 10
    ACHA ← .Falso.
    enquanto (COMEÇO ≤ FINAL) .e. (ACHA = .Falso.) faça
        MEIO ← (COMEÇO + FINAL) div 2
        se (PESQ = VLR[MEIO]) então
            ACHA ← .Verdadeiro.
        senão
            se (PESQ < VLR[MEIO]) então
                FINAL ← MEIO - 1
            senão
                COMEÇO ← MEIO + 1
        fim_se
    fim_se
    fim_enquanto
    se (ACHA = .Verdadeiro.) então
        escreva PESQ, " foi localizado na posição ", MEIO
    senão
        escreva PESQ, " não foi localizado"
    fim_se
    escreva "Deseja continuar? (S)IM/(N)ÃO: "
    leia RESP
fim_enquanto

{*** fim trecho de pesquisa binária ***}

fim

```

Pesquisa binária em ILA

```

// programa PESQUISA_BINÁRIA (cap0901.ila)

variaveis
    matric numerico VLR[10]
    numerico I, J, COMECO, FINAL, MEIO, PESQ, X
    caracter RESP
    logico ACHA
inicio

para I = 1 ate 10 passo 1
    ler VLR[I]

```

```

proximo

// {*** início trecho de ordenação ***}

para I = 1 ate 9 passo 1
  para J = I + 1 ate 10 passo 1
    se (VLR[I] > VLR[J]) entao
      X = VLR[I]
      VLR[I] = VLR[J]
      VLR[J] = X
    fim_se
  proximo
proximo

// {*** fim trecho de ordenação ***}

// {*** início trecho de pesquisa binária ***}

RESP = "S"
faça enquanto (RESP = "S") ou (RESP = "s")
  escrever "Entre valor a ser pesquisado:"
  ler PESQ
  COMECO = 1
  FINAL = 10
  ACHA = falso
  faça enquanto (COMECO <= FINAL) e (ACHA = falso)
    MEIO = inteiro((COMECO + FINAL) / 2)
    se (PESQ = VLR[MEIO]) entao
      ACHA = verdadeiro
    senao
      se (PESQ < VLR[MEIO]) entao
        FINAL = MEIO - 1
      senao
        COMECO = MEIO + 1
    fim_se
  fim_se
fim enquanto
se (ACHA = verdadeiro) entao
  escrever PESQ, " foi localizado na posicao ", MEIO
senao
  escrever PESQ, " não foi localizado"
fim_se
escrever "Deseja continuar? (S)/(N)AO:"
ler RESP
fim enquanto

// {*** fim trecho de pesquisa binária ***}

fim

```

Calculadora em LPP padrão Manzano e Oliveira

```
programa CALCULADORA
```

```
var
```

```
  R, A, B : real
  OPCA0 : inteiro
```

```
{Trecho de sub-rotinas de entrada e saída}
```

```
procedimento ENTRADA
```

```
início
```

```
  escreva "Entre o 1o. valor: "
  leia A
  escreva "Entre o 2o. valor: "
  leia B
```

```
fim
```

```
procedimento SAÍDA
```

```
início
```

```
  escreva "O resultado da operação equivale a: ", R, "."
```

```
fim
```

```
{Trecho com função para o cálculo das operações}
```

```
função CALCULO(X, Y : real, OPERADOR : caractere) : real
```

```
início
```

```
  caso OPERADOR
```

```
    seja "+" faça CALCULO ← X + Y
    seja "-" faça CALCULO ← X - Y
    seja "*" faça CALCULO ← X * Y
    seja "/" faça CALCULO ← X / Y
```

```
  fim_caso
```

```
fim
```

```
{Trecho com sub-rotina geral}
```

```
procedimento ROTCALC(OPERAÇÃO : caractere)
```

```
início
```

```
  caso OPERAÇÃO
```

```
    seja "+" faça escreva "Rotina de Adição"
    seja "-" faça escreva "Rotina de Subtração"
    seja "*" faça escreva "Rotina de Multiplicação"
    seja "/" faça escreva "Rotina de Divisão"
```

```
  fim_caso
```

```
  ENTRADA
```

```
  se (OPERAÇÃO = "/") então
```

```

    se (B = 0) então
        escreva "O resultado da operacao equivale a: ERRO."
    senão
        R ← CALCULO(A, B, "/")
        SAÍDA
    fim_se
fim_se
se .não. (OPERAÇÃO = "/") então
    R ← CALCULO(A, B, OPERAÇÃO)
    SAÍDA
fim_se
fim

```

{Trecho principal do programa}

```

início
    OPÇÃO ← 0
    enquanto (OPÇÃO <> 5) faça
        escreva "1 - Adição"
        escreva "2 - Subtração"
        escreva "3 - Multiplicação"
        escreva "4 - Divisão"
        escreva "5 - Fim de Programa"
        escreva "Escolha uma opção: "
        leia OPÇÃO
        se (OPÇÃO <> 5) então
            caso OPÇÃO
                seja 1 faça ROTCALC("+")
                seja 2 faça ROTCALC(" - ")
                seja 3 faça ROTCALC(" * ")
                seja 4 faça ROTCALC("/")
            senão
                escreva "Opção inválida - Tente novamente."
            fim_caso
        fim_se
    fim_enquanto
fim

```

Calculadora em ILA

```
// programa CALCULADORA (cap0902.ila)
```

```

variaveis
    numerico R, A, B
    numerico OPCAO
    numerico X, Y
    caracter OPERADOR

```

```

character OPERACAO
character TECLA

// {Trecho de sub-rotinas de entrada e saída}

funcao ENTRADA()
inicio
    escrever "Entre o 1o. valor: "
    ler A
    escrever "Entre o 2o. valor: "
    ler B
fim

funcao SAIDA()
inicio
    escrever "O resultado da operacao equivale a: ", R, "."
    TECLA = lertecla()
fim

// {Trecho com função para o cálculo das operações}

funcao CALCULO(X, Y, OPERADOR)
inicio
    faca caso
        caso (OPERADOR = "+"):
            retornar X + Y
        caso (OPERADOR = "-"):
            retornar X - Y
        caso (OPERADOR = "*"):
            retornar X * Y
        caso (OPERADOR = "/"):
            retornar X / Y
    fim_caso
fim

// {Trecho com sub-rotina geral}

funcao ROTCALC(OPERACAO)
inicio
    faca caso
        caso (OPERACAO = "+"):
            escrever "Rotina de Adicao"
        caso (OPERACAO = "-"):
            escrever "Rotina de Subtracao"
        caso (OPERACAO = "*"):
            escrever "Rotina de Multiplicacao"
        caso (OPERACAO = "/"):
            escrever "Rotina de Divisao"
    fim_caso
fim

```

```

fim_caso
ENTRADA()
se (OPERACAO = "/") entao
  se (B = 0) entao
    escrever "O resultado da operacao equivale a: ERRO."
    TECLA = lerteccla()
  senao
    R = CALCULO(A, B, "/")
    SAIDA()
  fim_se
fim_se
se nao (OPERACAO = "/") entao
  R = CALCULO(A, B, OPERACAO)
  SAIDA()
fim_se
fim

// {Trecho principal do programa}

inicio
OPCAO = 0
faca enquanto (OPCAO <> 5)
  limpar
  escrever "1 - Adicao"
  escrever "2 - Subtracao"
  escrever "3 - Multiplicacao"
  escrever "4 - Divisao"
  escrever "5 - Fim de Programa"
  escrever "Escolha uma opcao:"
  ler OPCAO
  se (OPCAO <> 5) entao
    faca caso
      caso (OPCAO = 1):
        ROTCALC("+")
      caso (OPCAO = 2):
        ROTCALC("-")
      caso (OPCAO = 3):
        ROTCALC("*")
      caso (OPCAO = 4):
        ROTCALC("/")
      outro_caso:
        escrever "Opcao invalida - Tente novamente."
    fim_caso
  fim_se
fim_enquanto
fim

```

Os programas apresentados em LPP são escritos com os recursos relatados no livro *Algoritmos: Lógica para Desenvolvimento de Programação de Computadores*. São mostrados os mesmos programas codificados de acordo com a sintaxe do processador de algoritmo demonstrado neste texto.

A partir desses exemplos é possível fazer uso da ferramenta ILA para processar quase que a totalidade dos exemplos usados no livro *Algoritmos: Lógica para Desenvolvimento de Programação de Computadores*, exceto alguns programas, por não ter a ferramenta ILA suporte a execução de alguns recursos relatados.

