rede e-Tec Brasil

Algoritmos e Programação

Adolfo José Gonçalves Stavaux Baudson
Francisco César Rodrigues de Araújo

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
MINAS GERAIS
Campus Ouro Preto

Ouro Preto - MG
2013

Presidência da República Federativa do Brasil

Ministério da Educação

Secretaria de Educação Profissional e Tecnológica

© Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais

Este caderno foi elaborado em parceria entre o Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – Campus Ouro Preto e a Universidade Federal de Santa Maria para a Rede e-Tec Brasil.

Equipe de Elaboração
Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – IFMG-Ouro Preto

Reitor
Caio Mário Bueno Silva/IFMG-Ouro Preto

Direção Geral
Arthur Versiani Machado/IFMG-Ouro Preto

Coordenação Institucional
Sebastião Nepomuceno/IFMG-Ouro Preto

Coordenação de Curso
Cristiano Lúcio Cardoso Rodrigues/IFMG-Ouro Preto

Professor-autor
Adolfo José Gonçalves Stavaux Baudson/IFMG-Ouro Preto
Francisco César Rodrigues de Araújo/IFMG-Ouro Preto

Equipe de Acompanhamento e Validação
Colégio Técnico Industrial de Santa Maria – CTISM

Coordenação Institucional
Paulo Roberto Colusso/CTISM

Coordenação Técnica
Iza Neuza Teixeira Bohrer/CTISM

Coordenação de Design
Erika Goellner/CTISM

Revisão Pedagógica
Andressa Rosemárie de Menezes Costa/CTISM
Fabiane Sarmento Oliveira Fruet/CTISM
Janaína da Silva Marinho/CTISM
Marcia Migliore Freo/CTISM

Revisão Textual
Ana Paula Cantarelli/CTISM
Tatiana Rehbein/UNOCHAPECÓ

Revisão Técnica
Guilherme Dhein/CTISM

Ilustração
Gabriel La Rocca Cóser/CTISM
Marcel Santos Jacques/CTISM
Rafael Cavalli Viapiana/CTISM
Ricardo Antunes Machado/CTISM

Diagramação
Cássio Fernandes Lemos/CTISM
Leandro Felipe Aguilar Freitas/CTISM

B342a **Baudson, Adolfo José Gonçalves Stavaux**
Algoritmos e programação / Adolfo José Gonçalves Stavaux Baudson, Francisco César Rodrigues de Araújo. – Ouro Preto: IFMG, 2013.
142 p. : il.
ISBN 978-85-86473-10-4

Caderno elaborado em parceria entre o Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais (IFMG) – Campus Ouro Preto e a Universidade Federal de Santa Maria – UFSM para o Sistema Escola Técnica Aberta do Brasil – Rede e-Tec Brasil.

1. Programação (Computadores). 2. Algoritmos. 3. Linguagem de programação (Computadores). I. Araújo, Francisco César Rodrigues de. II. Título.

CDU: 004.42

Apresentação e-Tec Brasil

Prezado estudante,
Bem-vindo a Rede e-Tec Brasil!

Você faz parte de uma rede nacional de ensino, que por sua vez constitui uma das ações do Pronatec – Programa Nacional de Acesso ao Ensino Técnico e Emprego. O Pronatec, instituído pela Lei nº 12.513/2011, tem como objetivo principal expandir, interiorizar e democratizar a oferta de cursos de Educação Profissional e Tecnológica (EPT) para a população brasileira propiciando caminho de o acesso mais rápido ao emprego.

É neste âmbito que as ações da Rede e-Tec Brasil promovem a parceria entre a Secretaria de Educação Profissional e Tecnológica (SETEC) e as instâncias promotoras de ensino técnico como os Institutos Federais, as Secretarias de Educação dos Estados, as Universidades, as Escolas e Colégios Tecnológicos e o Sistema S.

A educação a distância no nosso país, de dimensões continentais e grande diversidade regional e cultural, longe de distanciar, aproxima as pessoas ao garantir acesso à educação de qualidade, e promover o fortalecimento da formação de jovens moradores de regiões distantes, geograficamente ou economicamente, dos grandes centros.

A Rede e-Tec Brasil leva diversos cursos técnicos a todas as regiões do país, incentivando os estudantes a concluir o ensino médio e realizar uma formação e atualização contínuas. Os cursos são ofertados pelas instituições de educação profissional e o atendimento ao estudante é realizado tanto nas sedes das instituições quanto em suas unidades remotas, os polos.

Os parceiros da Rede e-Tec Brasil acreditam em uma educação profissional qualificada – integradora do ensino médio e educação técnica, – é capaz de promover o cidadão com capacidades para produzir, mas também com autonomia diante das diferentes dimensões da realidade: cultural, social, familiar, esportiva, política e ética.

Nós acreditamos em você!
Desejamos sucesso na sua formação profissional!

Ministério da Educação
Março de 2013

Nosso contato
etecbrasil@mec.gov.br

Indicação de ícones

Os ícones são elementos gráficos utilizados para ampliar as formas de linguagem e facilitar a organização e a leitura hipertextual.

Atenção: indica pontos de maior relevância no texto.

Saiba mais: oferece novas informações que enriquecem o assunto ou "curiosidades" e notícias recentes relacionadas ao tema estudado.

Glossário: indica a definição de um termo, palavra ou expressão utilizada no texto.

Mídias integradas: sempre que se desejar que os estudantes desenvolvam atividades empregando diferentes mídias: vídeos, filmes, jornais, ambiente AVEA e outras.

Atividades de aprendizagem: apresenta atividades em diferentes níveis de aprendizagem para que o estudante possa realizá-las e conferir o seu domínio do tema estudado.

Sumário

Palavra do professor-autor 9

Apresentação da disciplina 11

Projeto instrucional 13

Aula 1 – Conceitos 15
1.1 O computador 15
1.2 Algoritmo 15
1.3 Linguagem de programação 15
1.4 Conceitos básicos 16

Aula 2 – Variáveis e constantes 19
2.1 Uma pseudolinguagem 19
2.2 Elemento básico de qualquer linguagem: identificador (nome) 19
2.3 Declaração e definição de variáveis e constantes 19
2.4 Comentário 22
2.5 Comandos básicos 22

Aula 3 – Comandos de entrada e saída e estruturas condicionais 31
3.1 Comandos de entrada e saída 31
3.2 Estruturas de controle e fluxogramas 34

Aula 4 – Estruturas de repetição 45
4.1 Estrutura de repetição (enquanto ... faça) 45
4.2 Estrutura de repetição (para ... faça) 51
4.3 Estrutura de repetição (repita ... até) 53

Aula 5 – Procedimento 61
5.1 Definição 61
5.2 Características 61
5.3 Sintaxe 62
5.4 Exercícios resolvidos 63

Aula 6 – Função **69**
6.1 Definição 69
6.2 Características e diferenças em relação a procedimento 69
6.3 Sintaxe 70
6.4 Exercícios resolvidos 71

Aula 7 – Ambiente de programação Lazarus **75**
7.1 Linguagem de programação 75
7.2 Ambiente de desenvolvimento 75
7.3 Arquivos que compõem uma aplicação Lazarus 89
7.4 Elementos da linguagem Pascal/Object Pascal 89
7.5 Aplicação 92

Aula 8 – Estruturas condicionais **101**
8.1 Tomada de decisão 101

Aula 9 – Estruturas de repetição **121**
9.1 Repetição 121
9.2 Estrutura while ... do (enquanto ... faça) 121
9.3 Estrutura for ... do (para ... faça) 124
9.4 Estrutura repeat ... until (repita ... até) 127

Aula 10 – Procedimentos **131**
10.1 Estruturas modulares 131
10.2 Módulo procedimento 131

Aula 11 – Funções **137**
11.1 Módulo função 137
11.2 Exercício resolvido 137

Referências **141**

Currículo do professor-autor **142**

Palavra do professor-autor

A área de Automação Industrial, desde que foi constituída, tem sido objeto de constante evolução tecnológica no sentido de se aperfeiçoar processos em termos de qualidade e custo. Ao longo deste processo de constante procura pelo ótimo, soluções ainda não estabelecidas vão sendo propostas e efetivamente criadas.

Desta forma, o profissional de automação precisa ter, além de conhecimentos teóricos e práticos bem consolidados, grande capacidade de iniciativa e imaginação. É a união da imaginação para propor soluções aos problemas apresentados, baseado em uma consolidada fundamentação teórica, que procuramos abordar na disciplina de Algoritmos e Programação.

A primeira metade da disciplina procura desenvolver nos futuros profissionais o pensamento lógico, por meio do ensino de algoritmos estruturados. Em cada problema proposto, há uma sistemática que, unida ao raciocínio lógico, procura estabelecer uma estratégia de solução.

A proposta da segunda metade da disciplina é estabelecer o contato do aluno com uma ferramenta de programação comercial utilizada no mercado de trabalho. Ferramenta esta que será utilizada posteriormente para estabelecer a comunicação entre os diversos sistemas de automação.

Desejamos que vocês tenham sucesso, não somente na disciplina, mas como profissionais da área que escolheram para sua formação e acreditamos que o estudo e a dedicação necessários ao profissional contemporâneo esteja sempre presente na vida de todos.

Adolfo José Gonçalves Stavaux Baudson e
Francisco César Rodrigues de Araújo

Apresentação da disciplina

Olá, alunas e alunos

É com muita alegria que começamos hoje o estudo dos algoritmos e de uma linguagem de programação. Como vocês poderão perceber, trata-se de um conteúdo muito interessante, atraente, envolvente e muito utilizado nos dias de hoje. É de fácil compreensão e de uma aplicabilidade imensa, ainda mais em Automação. No mundo informatizado e automatizado de hoje há a necessidade de se elaborar programas que desenvolvam tarefas para fins diversos, como movimentar máquinas ou partes delas com um simples toque em um botão, por exemplo.

Veja: dissemos que vocês estudarão os algoritmos. Mas, o que é isso? Bom, se dissermos a vocês que algoritmo é uma receita de bolo, acreditariam? Pois é verdade. Se dissermos que o relato de sua caminhada até o trabalho ou à escola é um algoritmo, também estamos falando a verdade. Então pessoal, é isso mesmo. Os passos para se fazer um bolo ou uma caminhada até o trabalho exigem certa lógica, não é?

Por isso, pelo fato de se ter uma lógica, isto é, um conjunto de normas ou regras, estamos falando de algoritmos. É isso que iremos estudar nesta fase: lógica de programação, ou seja, os algoritmos.

Em seguida, iremos aplicar esses conhecimentos em uma linguagem de programação para que vocês possam sentir como é interessante a aplicabilidade dos conceitos adquiridos.

Desejamos a todas e a todos um ótimo curso e colocamo-nos à disposição de vocês. Com alegria, sejam bem vindos!

Professores Adolfo Baudson e Francisco César Araújo

Projeto instrucional

Disciplina: Algoritmos e Programação (carga horária: 75h).

Ementa: Conceitos. Variáveis e constantes. Comandos de entrada e saída. Estruturas condicionais. Estruturas de repetição. Procedimento e função. Ambiente de programação Lazarus. Estruturas condicionais. Estruturas de repetição. Procedimentos e funções.

AULA	OBJETIVOS DE APRENDIZAGEM	MATERIAIS	CARGA HORÁRIA (horas)
1. Conceitos	Conceituar algoritmo. Conceituar lógica de programação. Estudar outros conceitos de informática.	Ambiente virtual: plataforma Moodle. Apostila didática. Recursos de apoio: *links*, exercícios.	01
2. Variáveis e constantes	Conceituar e identificar variáveis e constantes. Resolver pequenos exercícios de lógica.	Ambiente virtual: plataforma Moodle. Apostila didática. Recursos de apoio: *links*, exercícios.	06
3. Comando de entrada, saída e estruturas condicionais	Entender as primeiras estruturas de controle para o desenvolvimento de algoritmos. Estudar as estruturas de entrada e saída de dados. Entender como funcionam as estruturas condicionais. Desenvolver os primeiros programas de computador utilizando uma pseudolinguagem.	Ambiente virtual: plataforma Moodle. Apostila didática. Recursos de apoio: *links*, exercícios.	07
4. Estruturas de repetição	Compreender as estruturas de repetição enquanto ... faça, para ... faça e repita ... até e suas diferenças. Desenvolver algoritmos utilizando cada uma das estruturas citadas.	Ambiente virtual: plataforma Moodle. Apostila didática. Recursos de apoio: *links*, exercícios.	08
5. Procedimento	Conhecer estruturas modulares. Aplicar novas maneiras para se desenvolver algoritmos, através da modularização de programas utilizando procedimento.	Ambiente virtual: plataforma Moodle. Apostila didática. Recursos de apoio: *links*, exercícios.	08
6. Função	Desenvolver algoritmos, através da modularização de programas utilizando função. Conhecer as diferenças entre procedimento e função.	Ambiente virtual: plataforma Moodle. Apostila didática. Recursos de apoio: *links*, exercícios.	07

AULA	OBJETIVOS DE APRENDIZAGEM	MATERIAIS	CARGA HORÁRIA (horas)
7. Ambiente de programação Lazarus	Conceituar a ferramenta de programação Lazarus/Free Pascal. Conceituar os principais termos associados aos objetos utilizados num ambiente gráfico. Instalar e configurar o Lazarus. Conhecer o ambiente de desenvolvimento Lazarus (IDE). Desenvolver as primeiras aplicações dentro do IDE Lazarus para o compilador Free Pascal.	Ambiente virtual: plataforma Moodle. Apostila didática. Recursos de apoio: *links*, exercícios.	08
8. Estruturas condicionais	Conhecer as estruturas condicionais para o compilador Free Pascal. Aplicar essas estruturas condicionais dentro do IDE do Lazarus. Desenvolver programas de computador na linguagem do Lazarus/Free Pascal que se utilizam dessas estruturas, para a solução de problemas que envolvam a tomada de decisões.	Ambiente virtual: plataforma Moodle. Apostila didática. Recursos de apoio: *links*, exercícios.	08
9. Estruturas de repetição	Entender como funcionam as estruturas de repetição no ambiente de programação visual (Lazarus). Elaborar programas utilizando estruturas de repetição.	Ambiente virtual: plataforma Moodle. Apostila didática. Recursos de apoio: *links*, exercícios.	07
10. Procedimentos	Desenvolver programas no IDE do Lazarus utilizando estruturas modulares, elaboradas pelo próprio programador – `procedure`.	Ambiente virtual: plataforma Moodle. Apostila didática. Recursos de apoio: *links*, exercícios.	08
11. Funções	Desenvolver programas no IDE do Lazarus utilizando estruturas modulares, elaboradas pelo próprio programador – `function`.	Ambiente virtual: plataforma Moodle. Apostila didática. Recursos de apoio: *links*, exercícios.	07

Aula 1 – Conceitos

Objetivos

Conceituar algoritmo.

Conceituar lógica de programação.

Estudar outros conceitos de informática.

1.1 O computador

Computador é uma máquina que processa informações de forma automática, sob o controle de grupos de instruções previamente definidas, com grande eficácia.

1.2 Algoritmo

1.2.1 Conceito

Um algoritmo nada mais é que um texto contendo comandos (instruções) que devem ser executados numa determinada ordem. Esse texto em si não nos interessa, mas, sim, seu significado, ou seja, aquilo que ele representa (GUIMARÃES; LAGES, 1994).

1.2.2 Definição

Algoritmo é um conjunto de instruções, como uma receita de bolo, constituído de um número finito de passos.

1.3 Linguagem de programação

Para que os dados sejam processados, há a necessidade de se utilizar uma linguagem de programação, pois os computadores não executam diretamente os algoritmos. Eles precisam ser transformados para uma linguagem de programação que, posteriormente, serão traduzidos para uma linguagem de máquina.

Através desta linguagem, os dados e programas podem ser entendidos pelo computador.

Uma das primeiras dessas linguagens de programação foi a Fortran, adequada para problemas de natureza técnica e científica. Na área comercial, surgiu a linguagem Cobol logo em seguida. Depois vieram outras linguagens como Basic, Pascal, C, as linguagens visuais e outras.

1.4 Conceitos básicos

- ***Bit*** (*binary digit*) – menor porção de informação entendível pelo computador. São os zeros e uns.

- ***Byte*** (*binary term*) – conjunto de 8 *bits*. Memórias são medidas em *bytes*. Cada caractere é um *byte*. Ex.: A

- ***Hardware*** – componentes físicos e eletrônicos do computador. É a parte física do computador. Ex.: O computador ou parte dele.

- ***Software*** – são programas de computador.

- **Compilador** – são programas capazes de transformar códigos-fonte em códigos-objeto, ou seja, capazes de transformar programas escritos em uma linguagem de programação mais acessível ao homem (PASCAL, DELPHI, etc.) em linguagem de máquina (a única entendida pelo computador). Ex.: PASCAL → compilador → linguagem de máquina

Resumo

Nessa aula, você teve o primeiro contato com alguns conceitos de informática. Dentre eles, o de algoritmo e linguagem de programação. O entendimento desses conceitos será muito bom para a sequência do aprendizado.

Esperamos tê-lo ajudado a compreender, através desses conceitos, um pouquinho daquilo que pretendemos ao longo desse curso.

Então, vamos continuar?

Atividades de aprendizagem

1. Defina algoritmo e linguagem de programação.

2. Conceitue:

a) *Bit*

b) *Byte*

c) Compilador

Aula 2 – Variáveis e constantes

Objetivos

Conceituar e identificar variáveis e constantes.

Resolver pequenos exercícios de lógica.

2.1 Uma pseudolinguagem

Conjunto de técnicas e comandos construídos na língua portuguesa com o objetivo de facilitar a programação e o entendimento dos programas principalmente daqueles que ainda não dominam nenhuma linguagem de programação.

Algoritmos desenvolvidos numa pseudolinguagem não são executados diretamente pelo computador. Necessitam ser transcritos para uma linguagem de programação (PASCAL, C++, por exemplo).

2.2 Elemento básico de qualquer linguagem: identificador (nome)

O objetivo desse elemento é identificar, na memória, variáveis e constantes.

É formado por um ou mais caracteres, sendo que o primeiro deve, obrigatoriamente, ser uma letra e os caracteres seguintes, letras e/ou dígitos, sublinhado ("_"), não sendo permitido o uso de símbolos especiais.

Identificadores válidos: A, X, SOMA, B34Y1, C4, SOMA_A.

Identificadores inválidos: 2YC, ?AB, -AYB, 55CDE, SOMA-A.

2.3 Declaração e definição de variáveis e constantes

2.3.1 Variáveis

Nos algoritmos, cada variável corresponde a uma posição na memória, cujo conteúdo pode variar ao longo do tempo durante a execução de um programa.

Embora uma variável possa assumir diferentes valores, ela só pode armazenar um único valor a cada instante. Toda variável é identificada por um nome ou identificador.

As variáveis só podem armazenar valores de um tipo, sendo quatro os tipos básicos, mostrados a seguir:

2.3.1.1 Tipos básicos

- **Inteiro** – qualquer número inteiro positivo, negativo ou nulo. Ex.: 3, -3, 0.

- **Real** – qualquer número real positivo, negativo ou nulo. Ex.: 3.4, -3.4, 0.0.

- **Caractere** – qualquer conjunto de caracteres desde que entre aspas. Nos algoritmos, não faremos distinção quanto ao tipo das variáveis que armazenam um ou mais caracteres. Ex.: "ABC", "sorriso", "abacate", "X".

- **Lógico** – conjunto de valores falso ou verdadeiro em proposições lógicas. Ex.: tem, achou (contendo falso ou verdadeiro).

Observação
Quando declaramos uma variável, significa dizer que criamos (definimos) locais na memória do computador rotulados com os nomes das variáveis (identificadores).

Exemplo
Inteiro: X, SOMA, CONT
Real: MEDIA, A, B
Caractere: NOME
Lógico: TEM

Assim, X é o nome de um local (endereço) de memória que pode conter valores do tipo inteiro.

2.3.2 Constantes

Constante, como o próprio nome sugere, é algo que não se modifica ao longo do tempo de execução de um programa. Uma constante pode ser: um valor numérico, um valor lógico ou uma sequência de caracteres quaisquer que possua alguma relevância para o problema em estudo. Conforme o seu tipo, a constante é classificada como sendo numérica, lógica ou caractere.

2.3.2.1 Constante numérica

A representação de uma constante numérica nos algoritmos é feita no sistema decimal.

Exemplo

10, -8, 5.3, 2.02×10^2

2.3.2.2 Constante lógica

É um valor lógico, isto é, só pode conter valores do tipo falso ou verdadeiro.

2.3.2.3 Constante caractere

Uma constante caractere pode ser qualquer sequência de caracteres. Toda constante do tipo caractere que aparece no algoritmo será colocada entre aspas, para que não seja confundida com outro item qualquer.

Nos algoritmos definiremos como sendo do tipo caractere qualquer constante que contenha um ou mais caracteres, desde que estejam entre aspas e não sofram modificações ao longo do algoritmo.

Exemplo

"ABCD", "RS", "IFMG", "X".

2.3.3 Exercícios resolvidos

1. Identificar o tipo de cada uma das constantes a seguir:

a) 12 ..... numérica

b) "AMOR" ..... caractere

c) "falso" ..... caractere

d) $3,81 \times 10^2$ ..... numérica

e) Verdadeiro ..... lógica

Talvez tenha surgido uma dúvida na letra c. Veja, a palavra "falso" está entre aspas. Portanto, independente do conteúdo, as aspas indicam ser um conteúdo de constante caractere.

2. Marque com um X os identificadores válidos:

a) (X) SOMA

b) ( ) X"Y

c) ( ) 3N

d) (X) N3

e) ( ) pai de santo

Veja que nas letras a e d, os identificadores possuem apenas letras e número, começando sempre por letra. Portanto, são identificadores válidos. Em relação às letras b, c e e, alguns caracteres inviabilizam o uso destes identificadores como as aspas (") na letra b, o número 3 iniciando o identificador na letra c e os espaços vazios (em branco) na letra e.

2.4 Comentário

É um instrumento utilizado para facilitar o entendimento do algoritmo. É um texto, ou simplesmente uma frase, que aparece delimitado por chaves em qualquer parte do programa. Não é interpretado pelo compilador.

2.5 Comandos básicos

2.5.1 Comando de atribuição

Este comando permite que se forneça um valor a certa variável, onde a natureza deste valor tem que ser compatível com o tipo de variável na qual está sendo armazenado.

O comando de atribuição tem a seguinte forma geral:

Identificador ← expressão

Onde: identificador – é o nome da variável
expressão – pode ser uma expressão aritmética, lógica ou do tipo caractere, de cuja avaliação é obtido o valor a ser atribuído à variável

Exemplo

NOTA ← 7,5
X ← 2
B ← X + 1
NOME ← "Ensino a distância"

2.5.2 Expressões aritméticas e operadores

As expressões aritméticas nos algoritmos são formadas com operadores e funções cujos operandos são constantes e/ou variáveis do tipo numérico (inteiro, real).

Exemplo

A ← B + 2 * C + ABS(B)

Onde: A – variável
B – variável
2 – constante
C – variável
ABS( ) – função
+ – operador de adição
* – operador de multiplicação
← – sinal de atribuição

Os operadores aritméticos são: a adição (+), a subtração (–), a multiplicação (*), a divisão (/), a potenciação (^) e a radiciação (√).

A seguir são consideradas algumas funções utilizadas nos algoritmos.

Quadro 2.1: Funções

Nome	Resultado fornecido
log (A)	Logaritmo de A na base 10
ln (A)	Logaritmo neperiano de A
Exp (A)	O número e elevado a A
ABS (A)	O valor absoluto de A
Trunca (A)	A parte inteira de um número fracionário
Arredonda (A)	Transforma por arredondamento um número fracionário em inteiro
Quociente (A, B)	Quociente inteiro da divisão de A por B
Resto (A, B)	Resto da divisão inteira de A por B

Fonte: Guimarães e Lages, 1994

2.5.3 Expressões lógicas e operadores

Nas expressões lógicas os operadores são lógicos ou relacionais.

2.5.3.1 Operadores lógicos

Os operadores lógicos considerados nos algoritmos são os conectivos de conjunção, disjunção e negação.

- **Conjunção** – e
- **Disjunção** – ou
- **Negação** – não

Quadro 2.2: Operadores lógicos

	E	
V	V	**V**
V	F	**F**
F	V	**F**
F	F	**F**
	OU	
V	V	**V**
V	F	**V**
F	V	**V**
F	F	**F**
	NÃO	
Não V		**F**
Não F		**V**

Fonte: Autores

2.5.3.2 Operadores relacionais

Os operadores relacionais indicam a comparação a ser realizada entre os termos da relação.

São eles:

> , < , >= , <= , <> (≠), =

2.5.3.3 Tabela de prioridades para operações mistas

a) Parênteses e funções

b) Potenciação (^) e radiciação (√)

c) Multiplicação (*) e divisão (/)

d) Adição (+) e subtração (–)

e) Relacionais (> , < , >= , <= , <> (≠), =)

f) não

g) e

h) ou

2.5.4 Exercícios resolvidos

1. Supondo que as variáveis X, Y, K sejam numéricas e possuam os valores 3, 6.7 e 7, respectivamente, a variável ENDER seja do tipo caractere e possua o conteúdo "Rua Alfa" e a variável Z seja do tipo lógico e possua o conteúdo falso, resolva as expressões a seguir obedecendo a tabela de prioridades para operações mistas. Indique se o resultado será falso ou verdadeiro para as alíneas "a" e "b". Para as demais, forneça o resultado numérico.

a) ENDER = "Rua Alfa" e Z

b) X + Y = 9 ou não Z

c) quociente (K, X), resto (K, X)

d) arredonda (Y – X), trunca (Y – X)

e) ABS (X – K)

Solução

a) Substituindo os valores das variáveis na expressão:

"Rua Alfa" = "Rua Alfa" e falso

Veja, entre o operador relacional "igual" (=) e o lógico (e), resolve-se primeiro o relacional. A expressão fica assim, então:

Verdadeiro e falso

Agora temos dois operandos lógicos e um operador também lógico. Como resposta, temos falso (veja Quadro 2.2).

b) Substituindo os valores das variáveis na expressão:

3 + 6.7 = 9 ou não falso

Segundo a ordem de prioridades, resolvemos primeiro a soma:

9.7 = 9 ou não falso

Em seguida, o relacional "=":

Falso ou não falso

Depois, a negação:

Falso ou verdadeiro

E, por último, o tipo lógico ou:

Verdadeiro

c) Quociente (K , X)

Substituindo os valores das variáveis na expressão:

Quociente (7, 3)

Faz-se a divisão inteira de 7 por 3 obtendo 2 como quociente.

Resto (K, X)

Substituindo os valores das variáveis na expressão:

Resto (7, 3)

Faz-se a divisão inteira de 7 por 3 obtendo 1 como resto.

d) Arredonda (Y – X)

Substituindo os valores das variáveis na expressão:

Arredonda (6.7 – 3)

Arredonda (3.7)

4

Portanto, a resposta é 4 por arredondamento.

e) Trunca (Y – X)

Substituindo os valores das variáveis na expressão:

Trunca (6.7 – 3)

Trunca (3.7)

3

Portanto, a resposta é 3. O número é truncado na sua parte decimal.

f) ABS (X – K)

Substituindo os valores das variáveis na expressão:

ABS (3 – 7)

ABS (–4)

4

O absoluto de um número é ele próprio em módulo, ou seja, sem sinal.

Resumo

Nessa etapa de grande importância para o aprendizado de programação, você pôde ver alguns conceitos muito importantes como identificador, variável e constante. Foram apresentados, também, alguns comandos básicos como o de atribuição de valores às variáveis. Você pôde tomar conhecimento de como resolver uma expressão aritmética. Aprendeu a utilizar os conectivos **e** (conjunção), **ou** (disjunção) e **não** (negação) numa expressão lógica e, ainda, utilizar os operadores relacionais.

Então, continue firme e vamos passar à próxima aula estudando novos conceitos.

Atividades de aprendizagem

1. Identificar o tipo de cada uma das constantes a seguir:

a) "Verdadeiro" –

b) 3.1416 –

c) falso –

d) "Maria" –

2. Marque com um X os identificadores válidos:

( ) "X

( ) X1Y

( ) N

( ) ?VB

( ) casa branca

3. Sendo as variáveis A, B e C numéricas e contendo os valores 3, 6 e 9, respectivamente, a variável caractere NOME, contendo "JOAO" e a variável lógica TEM, contendo o valor lógico falso, responda informando se o resultado da expressão será verdadeiro ou falso:

a) A + B > C e NOME = "JOAO"

b) TEM ou B > = A

c) não TEM e quociente (C, B) + 1 = A

d) NOME = "MARCOS" e TEM ou A^2 < C + 10

4. Dadas as variáveis A, B, X e Y do tipo numérico (inteiro, real) contendo os valores 10, 3, 2.5 e 1.2, quais os resultados fornecidos em cada uma das funções a seguir:

a) quociente (A, B), resto (A, B)

b) arredonda (A – X), arredonda (B + Y), arredonda (Y – X)

c) trunca (B^2 + X), trunca (A/3 + 1), trunca (X – 3.2)

d) ABS (A – B^3), ABS (A – 3)

Aula 3 – Comandos de entrada e saída e estruturas condicionais

Objetivos

Entender as primeiras estruturas de controle para o desenvolvimento de algoritmos.

Estudar as estruturas de entrada e saída de dados.

Entender como funcionam as estruturas condicionais.

Desenvolver os primeiros programas de computador utilizando uma pseudolinguagem.

3.1 Comandos de entrada e saída

Apresentaremos, a seguir, o comando para entrada de dados no algoritmo. Também, o comando para impressão de dados ou resultados dos algoritmos. Podemos imprimir na tela do computador ou na impressora.

- **Entrada de dados** (`leia`)

Entrar com um dado nesse momento significa digitar alguma informação no teclado. Esse dado será armazenado em um local na memória. Isto será efetuado por um comando escrito no algoritmo.

O comando utilizado para se entrar com dados é o leia. Portanto, sempre que se quiser entrar com uma informação, um dado no algoritmo, utiliza-se a seguinte sintaxe:

Sintaxe

`leia` (identificador(es))

Exemplo

```
leia (NOTA)
```

Neste exemplo anterior, o algoritmo espera que se digite uma nota. Assim que for digitada, ela será armazenada na variável cujo identificador é NOTA.

- **Saída de dados** (no vídeo ou na impressora)

A saída de dados é feita para que se possa visualizar o resultado do algoritmo. Deve-se imprimir na tela do computador ou na impressora o que foi solicitado pelo algoritmo. O comando utilizado no algoritmo para se fazer isto, é o `escreva` ou `imprima`. A sintaxe desse comando é a seguinte:

Sintaxe

`escreva` (identificador(es))

Exemplo

```
escreva (MEDIA)
```

ou

```
escreva ("A média é:" , MEDIA)
```

No exemplo anterior, o valor armazenado na variável MEDIA será mostrado no vídeo ou na impressora. Pode-se imprimir (escrever) apenas o conteúdo da variável MEDIA ou, então, esse conteúdo acompanhado de um texto escrito entre aspas.

3.1.1 Exercícios resolvidos

1. Fazer um algoritmo para imprimir dois nomes que deverão ser lidos através de uma unidade de entrada de dados qualquer.

Solução

```
início
   caractere: NOME1, NOME2
   escreva ("Digite um nome: ")
   leia (NOME1)
   escreva ("Digite outro nome: ")
   leia (NOME2)
   escreva (NOME1, NOME2)
fim
```

Explicando

Todo algoritmo começa com a palavra `início` e termina com a palavra `fim`. Em seguida à palavra `início`, faz-se a declaração das variáveis do programa, ou seja, do algoritmo. Neste exemplo, temos duas variáveis do tipo caractere, pois irão armazenar na memória do computador dois nomes. Declarando as variáveis, o computador reconhece a existência de tais variáveis e elas podem ser utilizadas em todo o programa a partir desse momento. Sendo assim,

pode-se entrar com os dados, isto é, as informações para NOME1 e NOME2. O comando utilizado para isto é o leia. Mas, seria interessante que aparecesse uma mensagem na tela antes de entrar com cada informação para que o usuário deste programa possa ter certeza de qual informação, qual dado ele deverá digitar. Para isto, utiliza-se o comando de saída escreva já que esse texto deverá ser impresso na tela do computador. Em seguida, faz-se a leitura da variável, ou seja, digitam-se os nomes desejados, um de cada vez, pressionando "enter" assim que digitar cada um deles. Por fim, através do comando escreva, são mostrados na tela os nomes digitados.

2. Fazer um algoritmo para calcular e imprimir a média aritmética das notas de um grupo de três alunos. As notas deverão ser lidas, uma para cada aluno, através de uma unidade de entrada qualquer, por exemplo, o teclado.

Solução

```
início
   real: NOTA1, NOTA2, NOTA3, MEDIA
   escreva ("Digite uma nota: ")
   leia (NOTA1)
   escreva ("Digite a segunda nota: ")
   leia (NOTA2)
   escreva ("Digita a terceira nota: ")
   leia (NOTA3)
   MEDIA ← (NOTA1 + NOTA2 + NOTA3) / 3
   escreva ("MÉDIA= ", MEDIA)
fim
```

Explicando

Veja que neste exercício tivemos que criar três variáveis para receber as notas e outra para fazer o cálculo da média. Todas são do mesmo tipo, real, porque são números fracionários. Seguindo o critério adotado no exercício anterior, fizemos a leitura dos dados. Em seguida, efetuamos o cálculo solicitado e a impressão do mesmo. Para facilitar ainda mais o entendimento, vamos fazer a simulação completa do exercício anterior, também conhecida como "teste de mesa", "chinês" ou "simulação". Como exemplo, vamos supor os seguintes valores para as notas que deverão ser lidas: NOTA1 = 5.0, NOTA2 = 8.5, NOTA3 = 7.5.

Simulação

NOTA1	NOTA2	NOTA3	MEDIA
5.0	8.5	7.5	7.0

Valor escrito (impresso): MÉDIA = 7.0

3.2 Estruturas de controle e fluxogramas

3.2.1 Estrutura sequencial

Essa estrutura mostra uma sequência de comandos, um por linha, como nos dois exemplos vistos anteriormente (item 3.1.1).

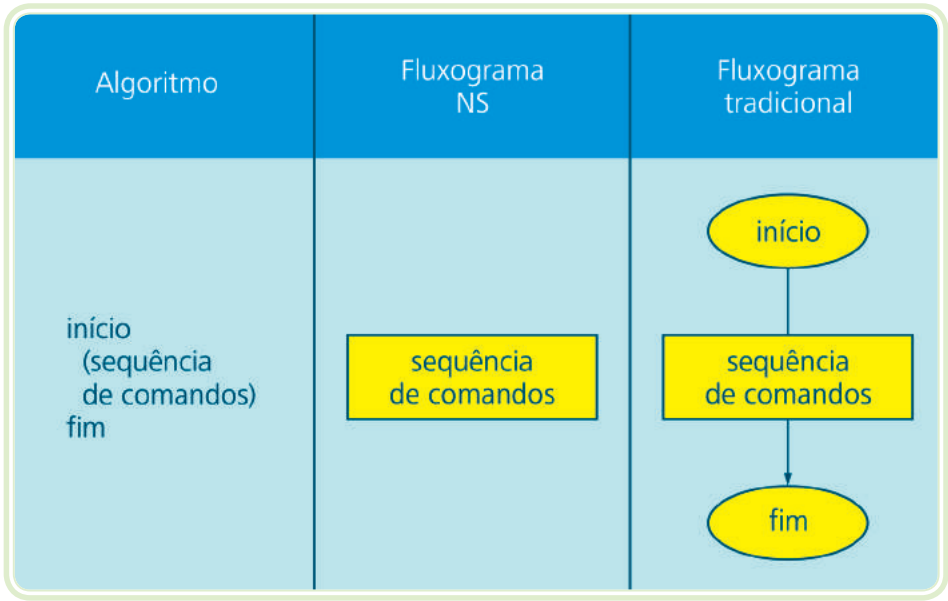

Figura 3.1: Fluxogramas NS e tradicional – estrutura sequencial
Fonte: Guimarães e Lages, 1994

Os algoritmos podem ser expressos em forma de fluxogramas também. Existem alguns tipos de fluxogramas e, de forma rápida, iremos mostrar dois tipos: o NS (Nassi-Shneiderman) e o tradicional. São utilizados principalmente para documentação de programas ou sistemas, porém, hoje em dia, são pouco utilizados devido às novas ferramentas existentes para esse fim. Vamos ver como ficaria o primeiro exercício do item 3.1.1 utilizando-se fluxogramas.

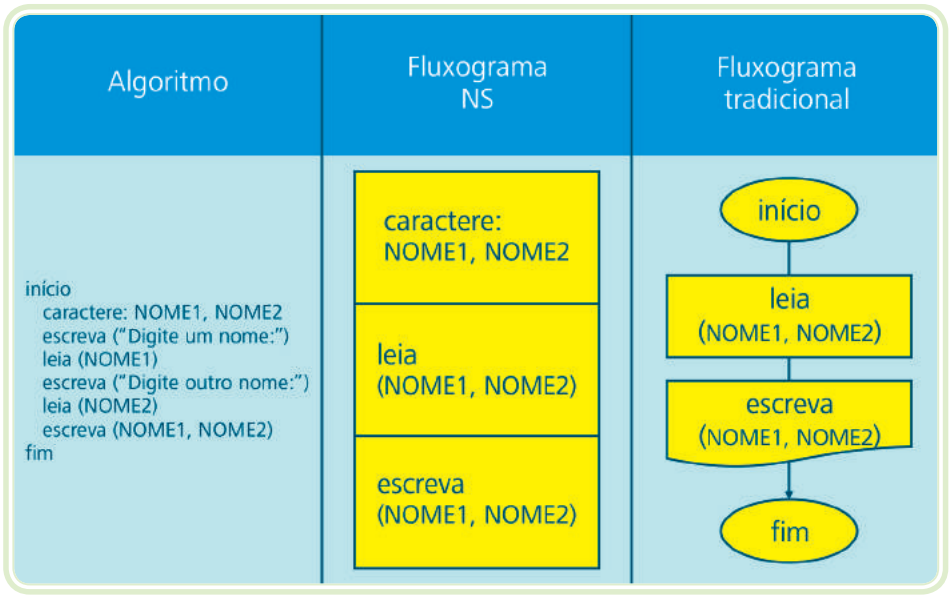

Figura 3.2 Estrutura sequencial
Fonte: Autores

3.2.2 Alternativa simples

A estrutura condicional permite a escolha do grupo de ações e estruturas a ser executado quando determinadas condições representadas por expressões lógicas são ou não satisfeitas (MANZANO; OLIVEIRA, 2005).

A estrutura alternativa simples nos permite executar ações apenas se a(s) condição(ções) for(em) verdadeira(s). Caso seja(m) falsa(s), não se executa essa estrutura.

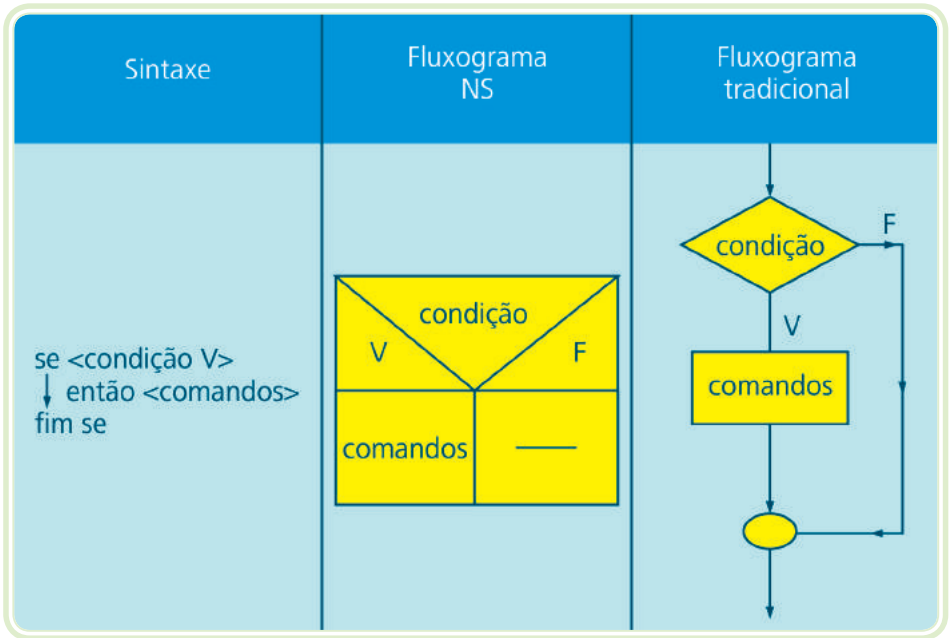

Figura 3.3: Alternativa simples
Fonte: Guimarães e Lages, 1994

Exemplo de um trecho de algoritmo:

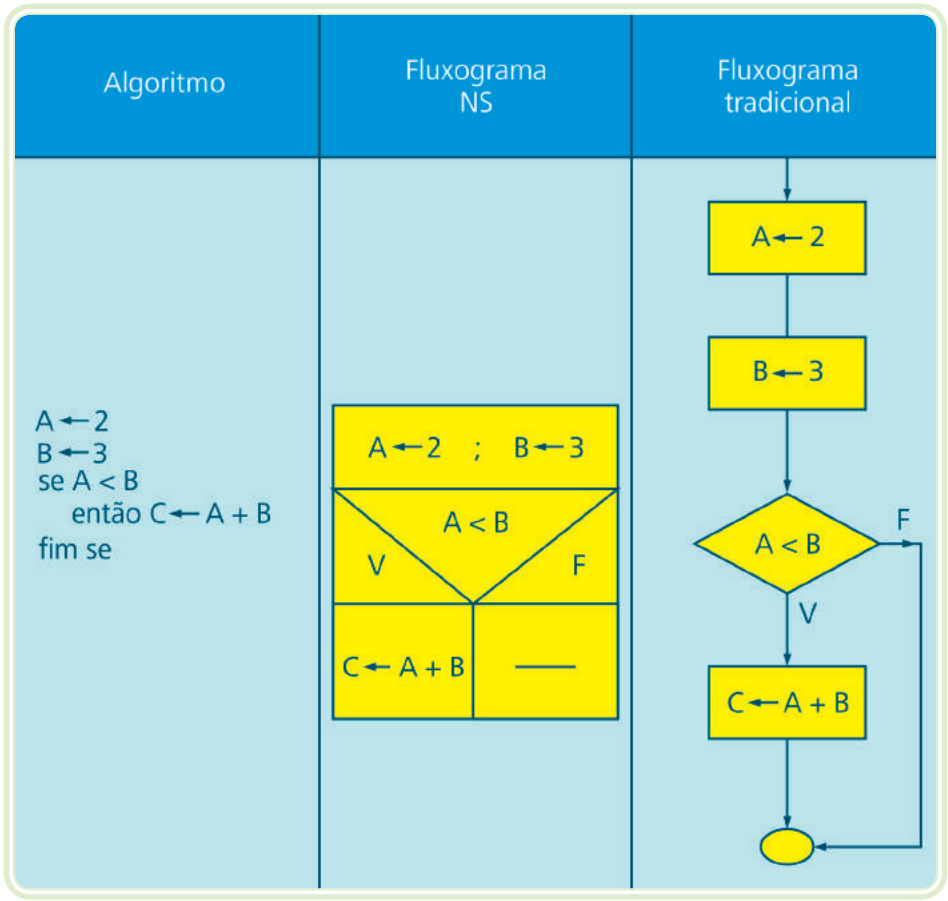

Figura 3.4: Alternativa simples
Fonte: Guimarães e Lages, 1994

3.2.2.1 Exercícios resolvidos

1. Fazer um algoritmo que leia 2 números inteiros A e B. Imprimir uma mensagem informando que A é maior que B.

Solução

```
início
   inteiro: A, B
   escreva ("Digite um número inteiro: ")
   leia (A)
   escreva ("Digite outro número inteiro: ")
   leia (B)
   se A > B
      então escreva ("A é maior que B")
   fim se
fim
```

Explicando

Inicialmente, declaramos as variáveis A e B do tipo inteiro, como foi dito no enunciado. Em seguida, fizemos a leitura dos dados. Posteriormente, utilizamos a estrutura condicional simples para verificar se o valor armazenado na variável A é maior do que o armazenado na variável B. Portanto, só será impresso a mensagem "A é maior que B" se a condição for verdadeira, ou seja, se A for maior que B.

2. Ler o sexo de uma pessoa (M, F) e imprimir a mensagem "É homem" caso seja do sexo masculino.

Solução

```
início
   caractere: SEXO
   escreva ("Digite o sexo (M=masculino, F=feminino): ")
   leia (SEXO)
   se SEXO = "M"
      então escreva ("É homem")
   fim se
fim
```

3. Ler a altura em centímetros de uma garota e imprimir uma mensagem caso esteja acima de 1,80 m.

Solução

```
início
   real: ALTURA
   escreva ("Digite a altura da garota em centímetros: ")
   leia (ALTURA)
   ALTURA ← ALTURA / 100 {para transformar a altura para metros}
   se ALTURA > 1.80
      então escreva ("Essa garota é muito alta")
   fim se
fim
```

4. Fazer um algoritmo que leia o raio de um círculo, determine e imprima a área correspondente. Imprimir a mensagem "Área pequena" se for menor que 5 cm^2. Dado: $A = \pi \times R^2$.

Solução

```
início
   real: AREA, RAIO
   escreva ("Digite o raio do círculo em centímetros: ")
   leia (RAIO)
   AREA ← 3.1416 * (RAIO * RAIO) {π = 3.1416, aproximado}
   escreva ("Área = ", AREA)
   se AREA < 5
      então escreva ("Área pequena")
   fim se
fim
```

3.2.3 Alternativa composta

A estrutura alternativa composta nos permite executar ações caso a condição seja verdadeira ou não.

Se a condição for verdadeira, executam-se os comandos que estão depois da palavra "então". Caso a condição seja falsa, executam-se os comandos que estão depois da palavra "senão".

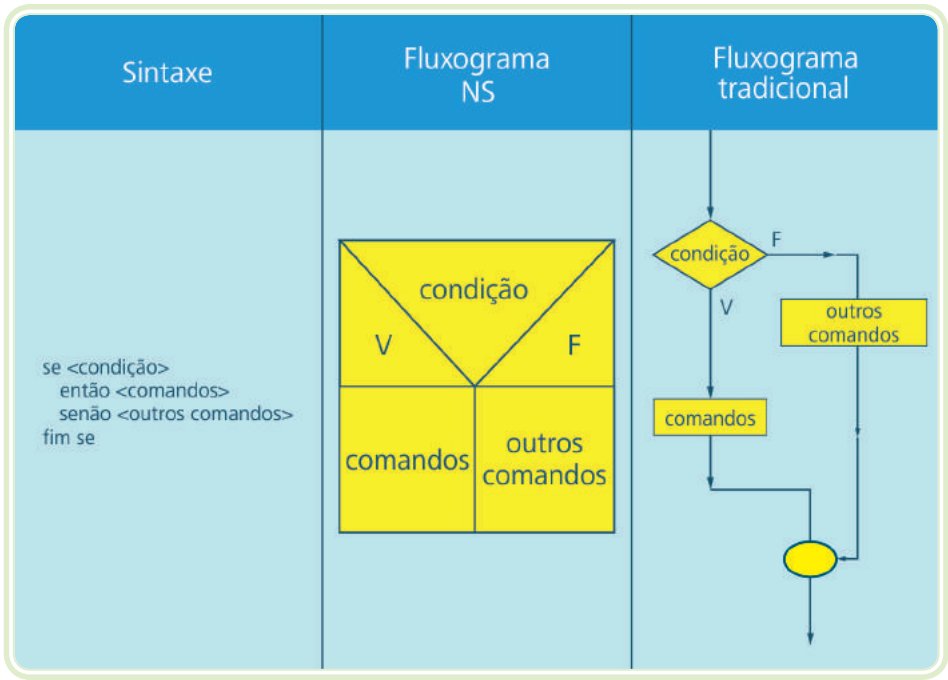

Figura 3.5: Alternativa composta
Fonte: Guimarães e Lages, 1994

Exemplo de um trecho de algoritmo:

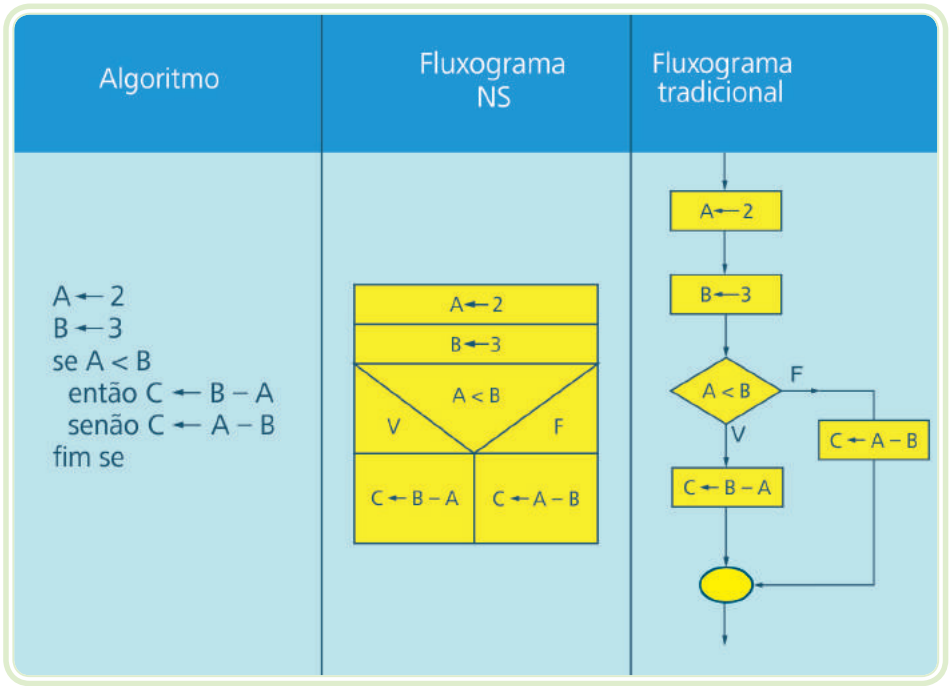

Figura 3.6: Alternativa composta
Fonte: Guimarães e Lages, 1994

3.2.3.1 Exercícios resolvidos

1. Fazer um algoritmo que leia 2 números inteiros A e B. Imprimir uma mensagem informando qual dos dois é o maior. Suponha que são números diferentes.

Solução

```
início
   inteiro: A, B
   escreva ("Digite um número inteiro: ")
   leia (A)
   escreva ("Digite outro número inteiro: ")
   leia (B)
   se A > B
      então escreva ("A é maior que B")
      senão escreva ("B é maior que A")
   fim se
fim
```

Explicando

Inicialmente, declaramos as variáveis A e B do tipo inteiro, como foi dito no enunciado. Em seguida, fizemos a leitura dos dados. Posteriormente, utilizamos a estrutura condicional composta para verificar se o valor armazenado na variável A é maior do que o armazenado na variável B ou se o valor armazenado na variável B é maior do que o armazenado na variável A. Portanto, só será impressa a mensagem "A é maior que B" se a condição for verdadeira, ou seja, se A for maior que B. Caso contrário, isto é, se a condição for falsa, será impresso a mensagem "B é maior que A".

2. Fazer um algoritmo que leia 2 números inteiros A e B. Imprimir uma mensagem informando qual dos dois é o maior ou se são iguais.

Solução

```
início
   inteiro: A, B
   escreva ("Digite um número inteiro: ")
   leia (A)
   escreva ("Digite outro número inteiro: ")
   leia (B)
   se A > B
      então escreva ("A é maior que B")
      senão se B > A
         então escreva ("B é maior que A")
         senão escreva ("Iguais")
      fim se
   fim se
fim
```

Explicando

Declaramos as variáveis A e B do tipo inteiro. Em seguida, fizemos a leitura dos dados. Posteriormente, utilizamos a estrutura condicional composta encadeada para verificar se o valor armazenado na variável A é maior do que o armazenado na variável B ou se o valor armazenado na variável B é maior do que o armazenado na variável A ou se são iguais. Portanto, sempre que se tiver mais de duas opções para serem testadas, pode-se utilizar a estrutura encadeada que nada mais é que um "se" dentro de outro "se". No exemplo anterior, o encadeamento foi feito após o comando "senão", mas, se necessário, pode ser feito também após o "então".

3. Ler o sexo de uma pessoa (M, F) e imprimir uma mensagem informando se "É homem" ou se "É mulher" .

Solução

```
início
   caractere: SEXO
   escreva ("Digite o sexo (M=masculino, F=feminino): ")
   leia (SEXO)
   se SEXO = "M"
      então escreva ("É homem")
      senão escreva ("É mulher")
   fim se
fim
```

4. Elaborar um algoritmo que leia o sexo e a altura de uma pessoa e imprima o peso ideal. (GUIMARÃES; LAGES, 1994).

Dados: Peso ideal para o sexo masculino: altura (cm) * 0.95 – 95

Peso ideal para o sexo feminino: altura (cm) * 0.85 – 85

Solução

```
início
   real: ALTURA, PESO
   caractere: SEXO
   escreva ("Digite a altura (cm): ")
   leia (ALTURA)
   escreva ("Digite o sexo (M, F): ")
   leia (SEXO)
   se SEXO = "M"
      então PESO ← ALTURA * 0.95 - 95
      senão PESO ← ALTURA * 0.85 - 85
   fim se
   escreva ("Peso Ideal: ", PESO)
fim
```

Resumo

Vimos nessa aula, como se faz para entrar com uma informação através de uma unidade de entrada qualquer (teclado, por exemplo). Utilizamos o comando `leia` para isso. Vimos também, como se retira uma informação, um resultado, do algoritmo proposto através do comando `escreva`. Também foram abordadas, através das estruturas condicionais, maneiras de se trabalhar com a informação dependendo do valor (verdadeiro ou falso) de uma condição. Você pôde perceber que é muito parecido com a forma como a gente fala, conversa, no dia-a-dia.

Nesse momento, caso necessário, refaça a leitura das aulas anteriores bem como refaça todos os exercícios propostos até então. Se necessário, procure por outras bibliografias, mídias, etc.

Veja que estamos avançando e estudaremos, a seguir, como fazer para tratar de uma grande variedade de dados num algoritmo. Vamos lá, então?

Atividades de aprendizagem

1. Elaborar um algoritmo que leia 3 números inteiros. Calcule e imprima a soma desses números.

2. Fazer um algoritmo que calcule e imprima a média de 4 números inteiros que deverão ser lidos através de uma unidade de entrada qualquer.

3. Fazer um algoritmo que leia os lados (A, B, C) do paralelepípedo, determine e imprima o volume (V) e a área superficial (S) e, imprima uma mensagem informando se o volume é superior a 20 cm^3.

Dados: V = A * B * C
S = 2 * (A * B + A * C + B * C)

4. Executando-se o algoritmo a seguir, que valores serão escritos:

```
início
   inteiro: A, C, I
   real: B, J, K
   A ← 32
   C ← 2
   I ← 5
   B ← A ^ (1/5) {a elevado a um quinto ou raiz quinta de a}
```

```
    J ← C * 3/4
    se (B > J)
       então K ← A + I / A - I
    fim se
    escreva (B, J, K)
fim
```

5. Sendo dadas 2 variáveis (A e B) do tipo caractere, verificar se as mesmas possuem os conteúdos "AGUA" e "TERRA", respectivamente. Se isto ocorrer, imprimir a mensagem "TUDO OK".

6. O que está errado no algoritmo a seguir?

```
início
   inteiro: N, PAR, X
   leia (N)
   X ← resto (N, 2)
   se (X = 0)
      então PAR ← verdadeiro
      senão PAR ← falso
   fim se
fim
```

7. O que será impresso depois de executado o algoritmo se NUM = -3.

```
início
   caractere: VALOR
   inteiro: NUM
   leia (NUM)
   se (NUM > 0)
      então VALOR ← "número positivo"
      senão se (NUM < 0)
         então VALOR ← "número negativo"
         senão VALOR ← "zero"
      fim se
   fim se
   escreva (VALOR)
fim
```

8. Sendo dadas 2 variáveis (A e B) do tipo caractere, verificar se as mesmas possuem os conteúdos "água" e "terra", respectivamente. Se isto ocorrer, imprimir a mensagem "tudo ok", caso contrário, imprimir a mensagem "problemas".

9. Implementar um algoritmo que leia uma letra. Se a letra for uma vogal, imprimir a mensagem "vogal", caso contrário, imprimir a mensagem "consoante".

10. Elaborar um algoritmo que leia dois valores numéricos diferentes e apresente a diferença do maior pelo menor.

Aula 4 – Estruturas de repetição

Objetivos

Compreender as estruturas de repetição enquanto ... faça, para ... faça e repita ... até e suas diferenças.

Desenvolver algoritmos utilizando cada uma das estruturas citadas.

4.1 Estrutura de repetição (enquanto ... faça)

A estrutura de repetição "enquanto ... faça" permite que uma sequência de comandos seja executada repetidamente enquanto uma determinada condição for verdadeira.

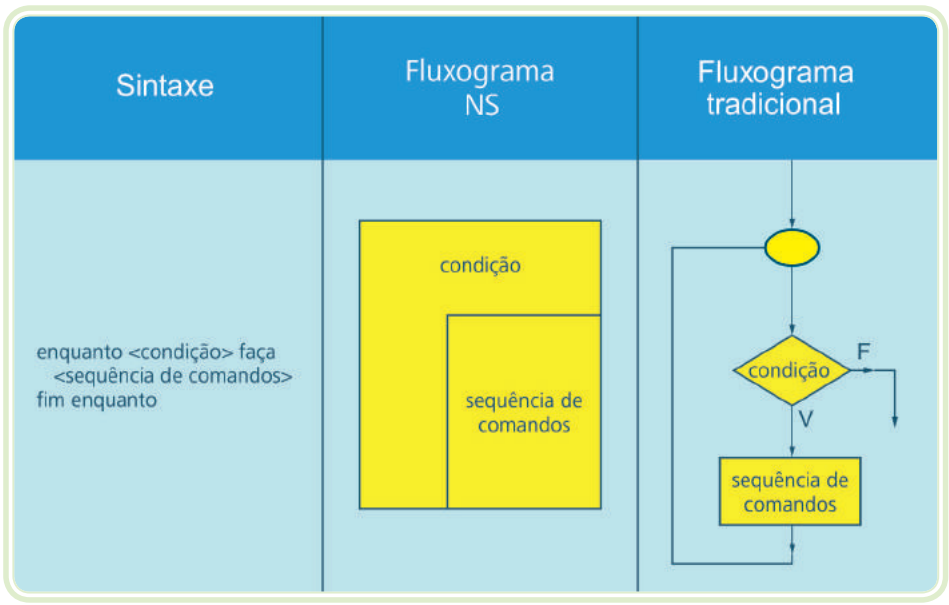

Figura 4.1: Estrutura de repetição
Fonte: Guimarães e Lages, 1994

Exemplo de um trecho de algoritmo:

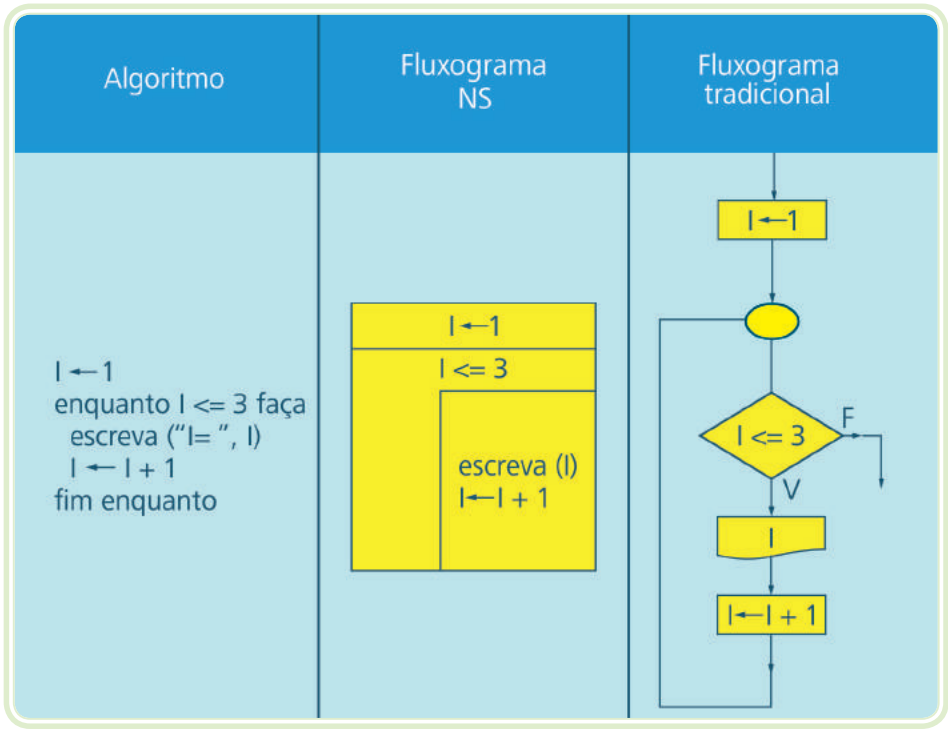

Figura 4.2: Estrutura de repetição
Fonte: Guimarães e Lages, 1994

4.1.1 Exercícios resolvidos

1. Indique os valores impressos pelo algoritmo a seguir.

Solução

```
início
   inteiro: I, S
   S ← 0
   I ← 1
   enquanto I <= 10 faça
      S ← S + I
      I ← I + 1
      escreva (I)
   fim enquanto
   escreva (S)
fim
```

Explicando

Primeiramente são declaradas as variáveis I e S que serão utilizadas no algoritmo. Em seguida, inicializamos as variáveis S e I com os valores zero e um, respectivamente. Pretendemos fazer o comando de repetição (laço de repetição)

executar dez vezes as instruções internas a ele. Por isso, ele deverá variar de 1 a 10. No final, quando a condição I <= 10 não for mais satisfeita, ou seja, quando a variável I atingir o valor 11, abandona-se o comando de repetição e passa-se para a próxima instrução (`escreva (S)`).

Como podemos perceber, este programa não imprime o primeiro conteúdo da variável I, ou seja, o número 1. Isso acontece porque esta variável é incrementada antes do comando `escreva (I)`, dentro do comando `enquanto ... faça`. Sendo assim, o primeiro valor da variável I impresso é 2. Portanto, a cada repetição será somado à variável S o conteúdo anterior desta variável com o valor atual da variável I, isto é, após as dez repetições, teremos a soma dos números inteiros de 1 a 10.

Na variável I é acrescentado o valor 1 (um) a cada repetição e impresso este valor. Esta variável, cujo valor inicial é 1 (um) variará de 1 a 10 dentro do comando de repetição e, quando atingir o valor 11 este valor será impresso, pois o comando `escreva (I)` encontra-se depois de se incrementar o valor de I e a condição I <= 10 se tornará falsa fazendo com que o comando seja abandonado.

Por último, será impresso o valor armazenado na variável S.

2. Construir um algoritmo para calcular a média de um conjunto de 100 valores inteiros fornecidos em uma unidade de entrada qualquer (GUIMARÃES; LAGES, 1994).

Solução

```
início
   inteiro: NUM, I, SOMA
   real: MEDIA
   I ← 1
   SOMA ← 0
   enquanto I <= 100 faça
      escreva (“Digite um número ”)
      leia (NUM)
      SOMA ← SOMA + NUM
      I ← I + 1
   fim enquanto
   MEDIA ← SOMA / 100
   escreva (“Média = ”, MEDIA)
fim
```

Explicando

Observe que é necessário declarar todas as variáveis que serão utilizadas no algoritmo. Em seguida faz-se a inicialização dessas variáveis. A partir daí, trabalha-se o laço de repetição. É quase sempre assim. Neste exercício, temos de somar todos os números. Por isso, essa etapa é feita dentro do comando de repetição. Então temos que ler o número, somá-lo com o que se tinha, ou seja, com aquele valor que estava armazenado na variável SOMA e incrementar o contador I somando mais 1 (um). Este contador serve para contarmos quantos números serão lidos. Quando atingir 101, a condição se tornará falsa e o comando de repetição será abandonado. Saindo do laço de repetição, calcula-se a média aritmética e a imprime em seguida.

3. Fazer um algoritmo que calcule e escreva o valor de S.

S = 1/1 + 3/2 + 5/3 + 7/4 + ... + 99/50 (50 termos)

Solução

```
início
   inteiro: NUM, I
   real: S
   I ← 1
   NUM ← 1
   S ← 0
   enquanto I <= 50 faça
      S ← S + NUM / I
      NUM ← NUM + 2
      I ← I + 1
   fim enquanto
   escreva ("Soma = ", S)
fim
```

Explicando

Neste exercício temos de somar frações. Veja que há uma lógica entre elas. O numerador é incrementado de 2 e o denominador de 1. Temos 50 frações e não temos que ler nada neste exercício. A cada iteração alteramos os valores das variáveis NUM (numerador) e I (contador de termos e denominador). Ao final, imprimimos o valor da soma (S).

4. O departamento de Educação Física deseja informatizar este setor e colocou à disposição os seguintes dados de 40 alunos:

Matrícula, sexo (M, F), altura (cm) e status físico (1 – bom, 2 – regular, 3 – ruim)

Estes dados deverão ser lidos através de uma unidade de entrada qualquer.

Calcular e imprimir:

a) A quantidade de alunos do sexo feminino com altura superior a 170 cm.

b) A % de alunos do sexo masculino (em relação ao total de alunos do sexo masculino) cujo status físico seja bom.

Solução

```
início
  inteiro: ALTURA, STATUS, CONTF170, CONTH, CONTHB, I
  caractere: SEXO, MATRÍCULA
  real: P
  CONTF170 ← 0; CONTH ← 0; CONTHB ← 0; I ← 1
  enquanto I < = 40 faça
    escreva ("Digite a matrícula neste formato (999.999-99): ")
    leia (MATRÍCULA)
    escreva ("Digite a altura (cm): ")
    leia (ALTURA)
    escreva ("Digite o status físico (1-bom, 2-regular, 3-ruim): ")
    leia (STATUS)
    escreva ("Digite o sexo (M, F): ")
    leia (SEXO)
    se (SEXO = "F") e (ALTURA > 170)
      então CONTF170 ← CONTF170 + 1
    fim se
    se SEXO = "M"
      então CONTH ← CONTH + 1
    fim se
    se (SEXO = "M") e (STATUS = 1)
      então CONTHB ← CONTHB + 1
    fim se
    I ← I + 1
  fim enquanto
  P ← (CONTHB / CONTH) * 100
  escreva ("Quantidade de alunos do sexo feminino com altura
  superior a 170 cm: ", CONTF170)
```

```
    escreva ("Porcentagem de alunos do sexo masculino com
    status físico bom: ", P)
fim
```

5. Construir um algoritmo para calcular a média de um conjunto de valores inteiros fornecidos em uma unidade de entrada qualquer.

Solução

```
início
   inteiro: NUM, SOMA, CONT
   real: MEDIA
   caractere: OP
   SOMA ← 0; CONT ← 0; OP ← "S"
   enquanto OP = "S" faça
      escreva ("Digite um número ")
      leia (NUM)
      SOMA ← SOMA + NUM
      CONT ← CONT + 1
      escreva ("Deseja continuar? (S/N): ")
      leia (OP)
   fim enquanto
   MEDIA ← SOMA / CONT
   escreva ("Média = ", MEDIA)
fim
```

Explicando

Neste exercício, não sabemos quantos elementos existem. Introduzimos, então, o conceito de *flag*. *Flag* é um artifício, uma forma de se interromper o comando de repetição. Por isso, criamos uma variável (OP) do tipo caractere (OP lembra opção) inicializada com o valor "S". A cada laço de repetição, perguntamos ao usuário se ele quer continuar entrando com dados (qual a sua opção?), ou seja, digitando novas informações. Se sim, digita-se "S", caso contrário, digita-se "N". Portanto, a condição será em função desta variável (OP). Digitando "S" o laço continua. Digitando "N" encerra-se o laço de repetição e passa-se para a próxima instrução depois do fim do laço.

Como não sabemos quantos dados existem, temos de criar uma variável contadora (CONT, por exemplo) para que possamos calcular a média. Saindo do laço de repetição, calcula-se a média aritmética e a imprime em seguida.

4.2 Estrutura de repetição (para ... faça)

Sintaxe

```
para i de a até b faça
   <comando 1>
   <comando 2>
   <comando 3>
   <comando n>
fim para
```

A semântica do comando é a seguinte: `i` é uma variável de controle (será inicializada e incrementada pelo próprio "`para ... faça`") e não deve ser modificada nos comandos `<comando 1>`, `<comando 2>`, ..., `<comando n>`; `a` e `b` podem ser constantes ou variáveis e também não devem ser modificadas nos comandos `<comando 1>`, `<comando 2>`, ..., `<comando n>`.

O comando "`para ... faça`" é, na verdade, o comando "`enquanto`" utilizando uma variável de controle, escrito numa notação compacta. Neste caso, existirá sempre uma inicialização da variável de controle, um teste para verificar se a variável atingiu o limite e um acréscimo na variável de controle.

Portanto, esta estrutura de repetição não pode ser aplicada em algoritmos que não informem quantos dados deverão ser utilizados. Neste caso, utiliza-se a estrutura "`enquanto ... faça`" ou a estrutura "`repita ... até`" que será mostrada mais adiante.

No exemplo a seguir é calculado o quadrado do conteúdo armazenado na variável "I" e o resultado é atribuído à variável "X". Em seguida, imprime-se o valor contido na variável "X". Isso será feito 5 vezes com o "I" variando de 1 a 5.

Exemplo

```
para I de 1 até 5 faça
   X ← I ^ 2
   escreva (X)
fim para
```

Observação

Neste exemplo a variável (I) é inicializada pelo "`para ... faça`" com o valor 1 (um) e toda vez que o "`fim para`" é encontrado, a variável de controle (I) é incrementada de 1 (um) automaticamente e o teste é feito.

4.2.1 Exercícios resolvidos

1. Elaborar um algoritmo que leia 40 notas e calcule a média aritmética das mesmas.

Solução

```
início
   inteiro: I
   real: SOMA, NOTA, MEDIA
   SOMA ← 0
   para I de 1 até 40 faça
      escreva ("Digite uma nota: ")
      leia (NOTA)
      SOMA ← SOMA + NOTA
   fim para
   MEDIA ← SOMA / 40
   escreva ("Média da turma: ", MEDIA)
fim
```

Explicando

Neste exercício, temos de somar as notas para depois calcularmos a média. Então temos que ler a nota, somá-la com o que se tinha, ou seja, com aquele valor que estava armazenado na variável SOMA. Veja que não é necessário inicializar a variável de controle I. Isso é feito automaticamente. Não devemos incrementar a variável de controle dentro do laço de repetição, pois isso também é automático, ou seja, essa variável (I) será incrementada de 1 (um) automaticamente. Saindo do laço de repetição, calcula-se a média aritmética e a imprime em seguida.

2. Fazer um algoritmo que calcule e escreva o valor de S.

S = 1 / 1 + 2 / 2 + 4 / 3 + 8 / 4 + ... (100 termos)

Solução

```
início
   inteiro: NUM, I
   real: S
   NUM ← 1
   S ← 0
   para I de 1 até 100 faça
      S ← S + NUM / I
      NUM ← NUM * 2
```

```
    fim para
    escreva ("Soma = ", S)
fim
```

Explicando

Neste exercício temos de somar frações. O numerador é multiplicado por 2 e o denominador é incrementado de 1. São 100 frações e não temos que ler nada neste exercício. A cada iteração alteramos o valor da variável NUM (numerador). Como a variável de controle I varia de 1 a 100, assim como o denominador da expressão proposta, utilizamos esta variável (I) como denominador. A variável I é incrementada automaticamente pelo comando `para ... faça`. Ao final, imprimimos o valor da soma (S).

4.3 Estrutura de repetição (repita ... até)

Essa estrutura nos permite fazer o teste da condição no final do comando repita.

Sintaxe

```
repita
    <sequência de comandos>
até <condição>
```

Para sair do comando de repetição, a "condição" tem de se tornar verdadeira, ou seja, os comandos pertencentes à estrutura de repetição "`repita ... até`" serão executados até que a condição se torne verdadeira.

Exemplo de um trecho de algoritmo:

```
I ← 1
repita
    escreva ("I = ", I)
    I ← I + 1
até I > 3
```

Portanto, as diferenças em relação às duas estruturas vistas anteriormente são:

- Os comandos serão executados pelo menos uma vez.
- O teste é feito no final da estrutura.
- Os comandos serão executados até que a condição se torne verdadeira.

4.3.1.1 Exercícios resolvidos

1. Construir um algoritmo para calcular a média de um conjunto de 100 valores inteiros fornecidos em uma unidade de entrada qualquer utilizando o comando `repita ... até`.

Solução

```
início
   inteiro: NUM, I, SOMA
   real: MEDIA
   I ← 1
   SOMA ← 0
   repita
      escreva ("Digite um número ")
      leia (NUM)
      SOMA ← SOMA + NUM
      I ← I + 1
   até I > 100
   MEDIA ← SOMA / 100
   escreva ("Média = ", MEDIA)
fim
```

Explicando

Veja como se escreve a estrutura de repetição. Os comandos internos à estrutura são os mesmos do comando "`enquanto ... faça`". A mudança está na condição que, no comando repita, é escrita e analisada no final. E, por causa disso, os comandos que ficam no laço de repetição são executados pelo menos uma vez, diferentemente das outras duas estruturas de repetição vistas anteriormente. Neste exercício, temos de somar todos os números. Por isso essa etapa é feita dentro do comando de repetição. Então temos que ler o número, somá-lo com o que se tinha, ou seja, com aquele valor que estava armazenado na variável SOMA e incrementar o contador I somando mais 1 (um). Este contador serve para contarmos quantos números serão lidos. Quando atingir 101, a condição se tornará verdadeira e o comando de repetição será abandonado. Saindo do laço de repetição, calcula-se a média aritmética e a imprime em seguida.

2. Utilizando o comando "`repita ... até`", fazer um algoritmo para calcular e imprimir a seguinte soma.

   S = (37 * 38) / 1 + (36 * 37) / 2 + (35 * 36) / 3 + ... + (1 * 2) / 37

Solução

```
início
   inteiro: NUM1, NUM2, I
   real: S
   I ← 1
   NUM1 ← 37
   NUM2 ← 38
   S ← 0
   repita
      S ← S + (NUM1 * NUM2) / I
      NUM1 ← NUM1 - 1
      NUM2 ← NUM2 - 1
      I ← I + 1
   até I > 37
   escreva ("Soma = ", S)
fim
```

3. Fazer um algoritmo que leia comprimento e largura de vários terrenos retangulares. Calcular e imprimir a área de cada um. Utilize o comando de repetição "`repita ... até`"

Solução

```
início
   real: COMP, LARG, AREA
   caractere: OP
   repita
      escreva ("Digite o comprimento do terreno: ")
      leia (COMP)
      escreva ("Digite a largura: ")
      leia (LARG)
      AREA ← COMP * LARG
      escreva ("Área = ", AREA)
      escreva ("Deseja continuar?(s/n): ")
      leia (OP)
   até OP = "N"
fim
```

Resumo

Vimos nessa aula o conceito de estrutura de repetição. Aprendemos como se faz para repetir uma determinada tarefa várias vezes. Apresentamos exemplos, exercícios resolvidos e sugerimos alguns outros. Se achar necessário, leia este capítulo novamente. Refaça os exercícios.

Essa estrutura (repetição) é muito importante para a elaboração de algoritmos, pois podemos executar um determinado conjunto de comandos mais de uma vez sem repetir as linhas de códigos.

Atividades de aprendizagem

1. Fazer um algoritmo, utilizando o comando `enquanto ... faça`, que: Leia um conjunto de 150 fichas contendo cada uma, a idade de um indivíduo e calcule e escreva a idade média deste grupo de indivíduos.

2. Fazer um algoritmo, utilizando o comando `enquanto ... faça`, que calcule e escreva a seguinte soma:

   S = 21 / 50 + 22 / 49 + 23 / 48 + ... + 70 / 1.

3. Fazer um algoritmo, utilizando o comando `enquanto ... faça`, que leia os seguintes dados referentes a cada um dos vários habitantes de uma determinada cidade, para serem analisados:

- Sexo ('M' ou 'F').
- Cor dos cabelos (1 = pretos, 2 = loiros, 3 = castanhos, 4 = outra cor).
- Cor dos olhos (1 = castanhos, 2 = pretos, 3 = verdes, 4 = outra cor).
- Idade (em anos).

Calcule e escreva:

a) A quantidade de pessoas do sexo feminino com idade acima de 60 anos.

b) A porcentagem de indivíduos cuja idade está entre 18 e 35 anos (inclusive) e que tenha olhos verdes e cabelos loiros.

4. Sendo H = 1 + 1/2 + 1/3 + 1/4 + ... + 1/N, fazer um algoritmo, utilizando o comando `para ... faça`, para gerar o número H. O número N é lido através de uma unidade de entrada qualquer uma única vez.

5. Fazer um algoritmo, utilizando o comando `para ... faça`, que calcule e escreva a soma dos 50 primeiros termos da seguinte série:

   S = 1000 / 1 - 997 / 2 + 994 / 3 - 991 / 4 + ...

6. Fazer um algoritmo, utilizando o comando `para ... faça`, que leia o sexo (M, F) e a resposta das pessoas (sim ou não) sobre uma pesquisa que tratava da opinião sobre um cosmético lançado no mercado. Sabendo-se que foram entrevistadas 2000 pessoas, calcule e imprima:

a) O número de pessoas que responderam sim.

b) O número de pessoas que responderam não.

c) A porcentagem de mulheres que responderam sim (em relação ao total de mulheres).

d) A porcentagem de homens que responderam não (em relação ao total de homens).

7. O sistema de avaliação da disciplina 'Introdução à Internet' obedece aos seguintes critérios:

- Durante o semestre são dadas três notas.
- A nota final é obtida pela média aritmética das notas dadas durante o curso.
- Para ser aprovado, o aluno tem que obter a nota final superior ou igual a 6,0 e ter comparecido a um mínimo de 80% das aulas. Considere aulas dadas = 100.

Fazer um algoritmo, utilizando o comando `para ... faça`, que:

a) Leia um conjunto de dados contendo o número de matrícula, as três notas e a frequência (números de aulas frequentadas) de 100 alunos.

b) Calcule e imprima a nota final de cada aluno (média do aluno).

c) Calcule e imprima a nota média da turma (média da turma).

d) Calcule e imprima o total de alunos reprovados.

e) Imprima para cada aluno o número de matrícula, a frequência, a nota final e o código (aprovado ou reprovado).

8. Fazer um algoritmo que leia os lados (A, B, C) de 30 paralelepípedos. Calcule e imprima:

a) O volume (V) e a área superficial (S) de cada paralelepípedo.

Dados: $V = A \times B \times C$

$S = 2 \times (A \times B + A \times C + B \times C)$

b) A média das áreas.

9. Fazer a simulação completa do algoritmo a seguir:

```
início
   inteiro: A, C, D, E
   real: X
   A ← 1
   C ← 4
   E ← 1
   D ← C - 2 - E
   X ← 0
   enquanto A <= 3 faça
      X ← X + (A + D) / 2
      D ← D * 2
      A ← A + 1
   fim enquanto
   C ← C * 2
   escreva ("D = ", D)
   escreva ("C = ", C)
   escreva ("X = ", X)
fim
```

Exercício 9.1: Simulação
Fonte: Autores

10. Escreva um algoritmo que leia as notas de vários alunos. Calcule e imprima a média da turma, bem como a situação (mensagem) de cada um deles, conforme critério apresentado a seguir:

- "Aprovado": caso a nota seja superior a 6,0.
- "Final": se a nota estiver compreendida entre 4,0 e 6,0.
- "Reprovado": se a nota for inferior a 4,0.

Aula 5 – Procedimento

Objetivos

Conhecer estruturas modulares.

Aplicar novas maneiras para se desenvolver algoritmos, através da modularização de programas utilizando procedimento.

5.1 Definição

Um procedimento é um bloco de programa, contendo início e fim e será identificado por um nome, através do qual será referenciado em qualquer parte do programa principal. (GUIMARÃES; LAGES, 1994).

5.2 Características

- É um subprograma dentro do programa principal.
- Pode usar as variáveis do programa – variáveis globais (*).
- Pode ter suas próprias variáveis – variáveis locais (**).
- Pode ter todas as partes de um programa inclusive outro procedimento.
- Todo procedimento possui um nome.
- Um mesmo procedimento pode ser chamado em locais diferentes no algoritmo.

(*) variável global – quando é declarada no início de um programa principal, podendo ser utilizada por qualquer sub-rotina subordinada (procedimento e função).

(**) variável local – quando é declarada dentro de uma sub-rotina (procedimento e função) e é somente válida dentro da rotina à qual está declarada.

5.3 Sintaxe

```
procedimento <nome> [(parâmetros: tipos)] {cabeçalho}
[variáveis locais: tipos]
início
   <instruções>
fim {<nome>}
```

Exemplo

```
{passagem de parâmetro por valor}
procedimento SOMAR (A, B: inteiro)
R: inteiro
início
   R ← A + B
   escreva ("Resultado = ", R)
fim {SOMAR}
```

ou

```
{passagem de parâmetro por referência}
procedimento SOMAR (A, B: inteiro; var R: inteiro)
AUX: inteiro
início
   AUX ← A + B
   R ← AUX
fim {SOMAR}
```

Explicando

Veja que existem dois tipos de construção de procedimentos. O primeiro com passagem de parâmetro(s) por valor e o outro, com passagem de parâmetro(s) por referência.

Nos algoritmos em que se utiliza procedimento(s) com passagem de parâmetro(s) por valor, as variáveis globais que estão trabalhando como parâmetros de entrada do procedimento passam seus valores para os parâmetros locais, isto é, aqueles que se encontram no cabeçalho do procedimento.

Naqueles algoritmos em que se utiliza procedimento(s) com passagem de parâmetro(s) por referência, veja que tudo que acontece com os parâmetros da chamada do procedimento, acontece também com os parâmetros do cabeçalho do procedimento. Por este motivo, não há a necessidade de se imprimir o resultado dentro do procedimento. Isso pode ser feito no programa principal, ou seja, logo depois da chamada do procedimento. Para

que um parâmetro seja por referência, basta colocar a palavra var antes do parâmetro, no cabeçalho do procedimento. Veja os exercícios resolvidos a seguir. Lembre-se: um procedimento só é executado quando for chamado e ele pode ser chamado em partes diferentes do mesmo programa.

5.4 Exercícios resolvidos

1. Ler 3 pares de números inteiros, calcular e imprimir a soma de cada par separadamente. Em seguida, fazer a soma entre os números relativos à sua idade e à idade de um colega. Utilizar procedimento com passagem de parâmetros por valor para calcular a soma de 2 números.

```
início
   inteiro X, Y, I {variáveis globais}
   procedimento SOMAR (A, B: inteiro)
   R: inteiro  {variável local}
   início
      R ← A + B
      escreva ("Resultado= ", R)
   fim {SOMAR}
{Programa Principal}
   I ← 1
   enquanto I <= 3 faça
      escreva ("Digite 2 números inteiros: ")
      leia (X, Y)
      SOMAR (X, Y) {passagem de parâmetro por valor}
      I ← I + 1
   fim enquanto
   escreva ("Digite a sua idade: ")
   leia (X)
   escreva ("Digite a idade do colega: ")
   leia (Y)
   SOMAR (X, Y)
fim
```

Explicando

Neste exercício foi criado um procedimento por valor para calcularmos a soma de dois números quaisquer. Primeiramente, declaramos as variáveis globais, aquelas que podem ser utilizadas em todo o programa (X, Y, I). Em seguida, declaramos o procedimento que foi denominado de SOMAR. Ele possui dois parâmetros (A, B), já que pretendemos somar dois números inteiros. Estes parâmetros, portanto, devem ser do tipo inteiro.

No procedimento, criamos uma variável local (R) para armazenar o resultado da soma dos valores contidos nos parâmetros A e B. Após o cálculo da soma, imprimimos o resultado.

Tudo que for calculado dentro de um procedimento com passagem de parâmetro(s) por valor e que você quiser imprimir tem que ser impresso dentro do referido procedimento. Não pode ser devolvido ao programa principal. O mesmo não ocorre quando a passagem de parâmetro é por referência.

Ainda neste exercício, podemos verificar que o mesmo procedimento é chamado em partes diferentes do programa caracterizando, assim, o reaproveitamento de código que é uma das vantagens de se utilizar procedimento.

2. Resolver o exercício anterior utilizando "passagem de parâmetro por referência".

```
início
   inteiro X, Y, Z, I
   procedimento SOMAR (A, B: inteiro; var R: inteiro)
   AUX: inteiro
   início
      AUX ← A + B
      R ← AUX
   fim {SOMAR}
{Programa Principal}
   I ← 1
   enquanto I <= 3 faça
      escreva ("Digite 2 números inteiros: ")
      leia (X, Y)
      SOMAR (X, Y, Z)
      escreva ("RESULTADO = ", Z)
      I ← I + 1
   fim enquanto
   escreva ("Digite a sua idade: ")
   leia (X)
   escreva ("Digite a idade do colega: ")
   leia (Y)
   SOMAR (X, Y, Z)
   escreva ("IDADE TOTAL = ", Z)
fim
```

Explicando

Observe que, diferentemente do exercício anterior, onde o procedimento era com passagem de parâmetro(s) por valor, criamos um parâmetro com passagem por referência (`var R: inteiro`). O parâmetro `R` irá corresponder com o parâmetro `Z`, da chamada do procedimento. Os dois apontam para a mesma referência na memória. Por isso, o que acontece com um, acontece com o outro. Tudo por causa daquela palavrinha `var` que precede o parâmetro `R`.

Então, não há necessidade de se imprimir o cálculo da soma dentro do procedimento. Isso pode ser feito no corpo do programa principal através do parâmetro `Z`.

Uma vantagem de se utilizar procedimento com passagem de parâmetro(s) por referência é que se pode utilizar no programa principal o resultado de cálculos efetuados dentro do procedimento. Imagine se quiséssemos acumular o resultado de todas as somas? Isso seria perfeitamente possível neste algoritmo. Bastaria criar uma variável global inteira S, por exemplo, e, no programa principal, após a chamada do procedimento, fazer o cálculo (`S←S+ Z`). Não poderia se esquecer de zerar a variável S antes do comando de repetição "`enquanto ... faça`".

Também nessa modalidade de procedimento podemos fazer o reaproveitamento de código. Se necessário, podemos chamar o procedimento em qualquer parte do programa, como mostrado anteriormente.

Resumo

A modularização de programas através de procedimentos facilita a elaboração, o entendimento e a manutenção dos mesmos. Tratar um problema em partes quase sempre é mais fácil que tratá-lo no todo. Outra vantagem é o reaproveitamento de códigos. Pode-se chamar o procedimento várias vezes dentro do mesmo programa sem ter que escrever todos os códigos pertencentes ao procedimento novamente.

Procedimento deve ser utilizado quando não se deseja retornar valor. Quando for necessário retornar um valor é mais interessante se utilizar função que será mostrada no próximo capítulo.

Nesta aula, vimos os procedimentos com passagem de parâmetros por valor e por referência. A seguir, algumas atividades para que você possa fixar estes conhecimentos.

Atividades de aprendizagem

1. Fazer um algoritmo que leia 100 números inteiros calcule e imprima o cubo de cada um deles. Utilizar procedimento com passagem de parâmetros por valor (FARRER et al, 1999).

2. Fazer um algoritmo que leia 100 números inteiros, dois a dois (50 pares), e faça a troca do primeiro pelo segundo, em cada par, utilizando procedimento com passagem de parâmetros por referência (GUIMARÃES; LAGES, 1994).

3. Fazer a simulação completa do exercício a seguir, utilizando procedimento com passagem de parâmetros por valor:

```
início
   inteiro: X, Y, Z, I
   procedimento Menorde3 (A, B, C: inteiro)
   AUX: inteiro
   início
      se (A < B) e (A < C)
         então AUX ← A
         senão se (B < C) e (B < A)
            então AUX ← B
            senão AUX ← C
         fim se
      fim se
      escreva("Menor dos 3:", aux)
   fim
   para I de 1 até 10 faça
      escreva ("Digite um número inteiro: ")
      leia (X)
      escreva ("Digite um número inteiro: ")
      leia (Y)
      escreva ("Digite um número inteiro: ")
      leia (Z)
      menorde3 (X, Y, Z)
   fim para
   X ← 10
   Y ← 3
   Z ← 20
   menorde3 (X, Y, Z)
   X ← 12
```

```
    Y ← 13
    Z ← 14
    menorde3 (X, Y, Z)
fim
```

4. Fazer um algoritmo que leia o valor do comprimento e da largura de 20 retângulos. Calcule a área de cada um, utilizando procedimento com passagem de parâmetro(s) por referência, e imprima a média das áreas.

5. Fazer um algoritmo que leia 30 números inteiros, 3 a 3, e, utilizando procedimento com passagem de parâmetros por referência, coloque-os ordenados de forma crescente imprimindo-os em seguida. Posteriormente, leia o ano de fundação de 3 instituições de ensino de seu estado e, utilizando o mesmo procedimento, coloque-os ordenados de forma crescente imprimindo apenas o ano de fundação da instituição mais antiga.

6. Fazer um algoritmo que leia os coeficientes A, B e C de 10 equações do segundo grau. No programa principal, calcule o valor de DELTA para cada equação. Elaborar um procedimento que receba os valores de A, B e C e também o valor de DELTA e calcule as raízes da equação e imprima-as no programa principal (passagem de parâmetros por referência).

Aula 6 – Função

Objetivos

Desenvolver algoritmos, através da modularização de programas utilizando função.

Conhecer as diferenças entre procedimento e função.

6.1 Definição

Assim como procedimento, uma função é um bloco de programa, contendo início e fim e será identificado por um nome, através do qual será referenciado em qualquer parte do programa principal (GUIMARÃES; LAGES, 1994).

Difere de procedimento basicamente por poder fazer parte de um comando ou de uma expressão. Difere, também, por devolver (retornar) ao local da chamada um determinado valor.

6.2 Características e diferenças em relação a procedimento

- Uma função sempre devolverá um resultado.
- Um procedimento é usado como um comando qualquer.
- Uma função sempre fará parte de um comando ou de uma expressão.
- Uma mesma função pode ser chamada em mais de um lugar no mesmo algoritmo, assim como procedimento.

6.3 Sintaxe

```
função <nome> [(parâmetros:tipos)]: <tipo da função>
[(variáveis: tipos)]
início
   <instruções>
   <nome> ← <expressão ou resultado>
fim {<nome>}
```

Exemplo

```
função ABSOL (X: inteiro): inteiro
   AUX: inteiro
   início
      se X < 0
         então AUX ← X × (-1)
         senão AUX ← X
      fim se
      ABSOL ← AUX
   fim
```

Explicando

A função difere do procedimento principalmente na maneira como é chamada. Vimos que para se chamar um procedimento, basta colocar o seu nome e os parâmetros. Em se tratando de função, não é bem assim. A função sempre virá acompanhada de um comando ou uma expressão. Por exemplo, `escreva (SOMAR (X, Y))`. Ou ainda: `3 × SOMAR (X, Y)/2`. Isso é possível com função. No primeiro exemplo, seria impresso o valor da soma de X com Y. No segundo, veja que a função faz parte da expressão. O resultado da soma de X com Y seria multiplicado por 3 e, depois, o resultado dividido por 2.

Utilizando-se procedimento, isso não seria possível, pois o procedimento não pode fazer parte de uma expressão. Ele é chamado sem a companhia de ninguém.

E como seria com procedimento? Deveríamos criar uma terceira variável (parâmetro), por referência, chamar o procedimento e utilizar a variável global, que corresponde com este parâmetro, na expressão.

Veja também que o resultado da função é sempre atribuído ao nome da função, que trabalha como se fosse um parâmetro de saída de dados.

6.4 Exercícios resolvidos

1. Ler 3 pares de números inteiros, calcular e imprimir a soma de cada par separadamente. Utilizar função para calcular a soma de 2 números.

```
início
   inteiro: X, Y, I
   função SOMAR (A, B: inteiro): inteiro
   AUX: inteiro
   início
      AUX ← A + B
      SOMAR ← AUX
   fim {SOMAR}
{programa principal}
   I ← 1
   enquanto I <= 3 faça
      escreva ("Digite 2 números inteiros: ")
      leia (X, Y)
      escreva ("RESULTADO = ", SOMAR (X, Y))
      I ← I + 1
   fim enquanto
fim
```

Explicando

Observe que a forma de se escrever uma função é um pouco diferente de procedimento. Há a declaração do tipo da função já que o seu nome funciona como uma variável. No final da função, há a devolução do resultado para o nome da função.

Neste exercício, é possível fazer o cálculo da soma de A por B e armazenar o resultado diretamente no nome da função. Com isso, eliminaríamos a variável AUX. Assim, ficaria: `SOMAR ← A + B`.

No programa principal, faz-se a chamada da função junto com o comando "escreva".

2. Fazer um algoritmo que leia 30 números inteiros e imprimir o valor absoluto de cada um deles. Utilizar função para calcular ABSOL.

```
início
   inteiro: A, I
   função ABSOL (X: inteiro): inteiro
   AUX: inteiro
   início
```

```
        se X < 0
           então AUX ← X * (-1)
           senão AUX ← X
        fim se
        ABSOL ← AUX
     fim
     para I de 1 até 30 faça
        escreva ("Digite um número inteiro:")
        leia (A)
        escreva ("Absoluto =", ABSOL(A))
     fim para
  fim
```

Resumo

Nesta aula, estudamos as funções, ou seja, as estruturas modulares. Você pôde ver que, subdividindo o algoritmo em blocos menores, fica mais fácil a sua manutenção. O entendimento também é facilitado. Você pode perceber que uma função é um pequeno algoritmo inserido no algoritmo principal.

A modularização utilizada no algoritmo nos permite tratar apenas parte do código sem nos preocupar com o todo o que facilita muito a sua manutenção.

A partir de agora, você irá aplicar esses conceitos numa linguagem de programação tornando os estudos bem mais agradáveis. Você irá ver como é fantástico programar. Parabéns por chegar até aqui.

Atividades de aprendizagem

1. Fazer um algoritmo que leia 100 números inteiros, calcule e imprima o cubo de cada um deles. Utilizar função para calcular o cubo (FARRER et al, 1999).

2. Fazer um algoritmo que leia 50 pares de números inteiros, referentes a catetos de triângulos retângulos. Calcule e imprima a hipotenusa de cada um dos triângulos. Utilizar função para calcular hipotenusa (GUIMARÃES; LAGES, 1994).

3. Fazer um algoritmo que leia 10 pares (N e P) de números inteiros, sendo o primeiro (N) sempre maior que o segundo (P). Calcule e imprima o valor de S, dado pela fórmula a seguir:

   S = N! / (P! (N - P)!)

Utilizar função para se calcular o fatorial de um número inteiro.

Obs.: o fatorial de um número, representado pelo ponto de exclamação (!), é a multiplicação sucessiva de inteiros entre o número 1 e o próprio número. Exemplo: 5! = 1 × 2 × 3 × 4 × 5 = 120

4. Fazer um algoritmo que leia 40 notas, sendo 4 por aluno. Calcule, utilizando função, a média (aritmética) de cada aluno imprimindo-a no programa principal. Calcule e imprima, também, a média da turma.

5. Fazer um algoritmo que leia 20 notas, sendo uma por aluno. Utilizando função, verifique se o aluno foi aprovado ou não. Para ser aprovado, o aluno tem que ter nota superior ou igual a 6. Portanto, a função receberá por parâmetro uma nota e devolverá um texto ("aprovado" ou "não está aprovado") informando a situação do aluno.

6. Fazer um algoritmo que leia o valor do comprimento e da largura de 20 retângulos. Calcule a área de cada um utilizando função, imprimindo-a em seguida. Calcule e imprima também, no programa principal, a média das áreas.

Aula 7 – Ambiente de programação Lazarus

Objetivos

Conceituar a ferramenta de programação Lazarus/Free Pascal.

Conceituar os principais termos associados aos objetos utilizados num ambiente gráfico.

Instalar e configurar o Lazarus.

Conhecer o ambiente de desenvolvimento Lazarus (IDE).

Desenvolver as primeiras aplicações dentro do IDE Lazarus para o compilador Free Pascal.

7.1 Linguagem de programação

Prezado(a) aluno(a), nesta aula, iniciaremos o estudo de uma linguagem de programação, que lhe permitirá desenvolver soluções computacionais para os problemas que você encontra em seu cotidiano. Com base neste objetivo, lhe será apresentado o Lazarus, que é um IDE (*Integrated Development Environment*), ou Ambiente de Desenvolvimento Integrado, o qual permitirá desenvolver aplicações com interfaces gráficas para o compilador Free Pascal.

Dentre as várias linguagens de programação disponíveis no mercado, optou-se pelo Lazarus/Free Pascal pelo fato de os mesmos serem projetos de código aberto (*open source*), o que significa, segundo Almeida (2010), que eles não são de propriedade exclusiva de uma empresa ou de um único desenvolvedor, podendo ser utilizado por qualquer pessoa, sem a necessidade de se pagar por sua utilização.

7.2 Ambiente de desenvolvimento

7.2.1 Conceitos

7.2.1.1 Lazarus

Conforme Almeida (2010), "o Lazarus (nome do personagem bíblico que foi ressucitado por Jesus) é a continuação de um projeto que havia sido encerrado,

o projeto 'Megido'", o qual tinha por objetivo ser um clone do Delphi. O Lazarus é um ambiente de desenvolvimento integrado que utiliza o Free Pascal como compilador. Foi desenvolvido para ser uma alternativa à linguagem de programação Delphi, cujos direitos pertencem à empresa Embarcadero, e que devido a sua grande semelhança com este *software* em sua versão 7, é considerado por muitos, como um "clone" do mesmo.

7.2.1.2 Free Pascal

O compilador Free Pascal (FPC) é um compilador de código aberto de 32/64 *bits*, usado na programação de computadores através das linguagens de programação Pascal e Object Pascal, desde meados da década de 1990. Permite gerar código nativo em diferentes sistemas operacionais, tais como: Windows, Linux, MacOs, OS/2 entre outros.

7.2.1.3 Eventos, propriedades e métodos

a) **Eventos** – os programas feitos no Lazarus são desenvolvidos através de eventos, os quais podem ser entendidos como todas as maneiras que o usuário pode interagir com os objetos, durante a utilização de um programa. Alguns desses eventos, associados aos componentes, são descritos a seguir:

- `OnClick` – ocorre quando o usuário clica sobre o objeto.
- `OnMouseMove` – ocorre quando o usuário movimenta o ponteiro do *mouse*.
- `OnKeyPress` – este evento é chamado quando cada tecla é pressionada, sendo executado antes da tecla que foi pressionada chegar a seu destino.
- `OnKeyDown` – ocorre quando o usuário pressiona uma tecla.
- `OnEnter` – este evento é chamado quando o objeto recebe o foco.
- `OnExit` – este evento é chamado quando o objeto perde o foco.
- `OnChange` – este evento é chamado cada vez que o conteúdo de um determinado objeto é alterado.
- `OnMouseDown` – ocorre quando o usuário pressiona o botão do *mouse*.
- `OnMouseUp` – ocorre quando o usuário libera o botão do *mouse*.

É importante ressaltar que, no desenvolvimento de uma aplicação, os códigos são escritos através de procedimentos, os quais são gerados para cada um dos eventos utilizados na elaboração do projeto. A criação desses procedimentos será detalhada mais adiante.

b) **Propriedades** – determinam a maneira como os objetos de sua aplicação funcionam, em outras palavras, representam as características de um determinado objeto. Podemos exemplificar como propriedades de um objeto a sua altura, largura, cor, tipo de fonte usada, entre outras características. Sendo assim, para manipulá-las durante o desenvolvimento do código de um programa, deve-se, primeiramente, informar o nome do componente (objeto) seguido de um ponto e, por último, o nome da propriedade, conforme sintaxe a seguir: `<componente>.<propriedade>`.

Exemplo

Atribuir a informação "e-Tec Brasil" à propriedade `Text` do componente `Edit1`.

Resposta

`Edit1.Text`:= 'e-Tec Brasil';

c) **Métodos** – são procedimentos ou funções embutidos nos componentes e formulários, definidos pela linguagem de programação em questão.

7.2.2 *Download* do Lazarus

Neste curso, utilizaremos a ferramenta de programação Lazarus que seja compatível com o sistema operacional Windows. Sendo assim, para se acessar a versão disponível para *download*, vá ao endereço: <**www.lazarus.freepascal.org**>. Em seguida, no quadro "**Recent Announcements**", clique no *link* "**Lazarus 1.0.10 release available for download**" ou numa versão mais recente. Na janela seguinte, clique em **Downloads**, conforme apresentado na Figura 7.1.

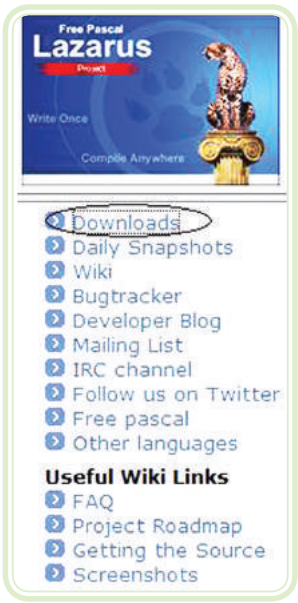

Figura 7.1: *Download* Lazarus
Fonte: http://www.lazarus.freepascal.org/

Na janela seguinte, Figura 7.2, clique no *link* ao lado da pergunta "**Looking for the latest version**?"

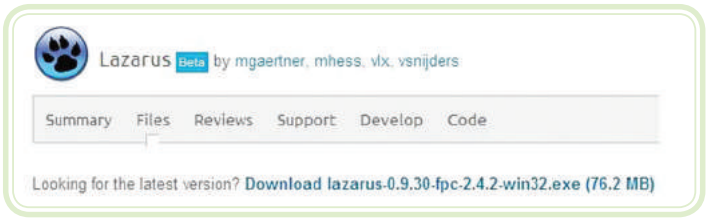

Figura 7.2: *Link download* Lazarus
Fonte: http://sourceforge.net/projects/lazarus/files/

Na sequência, na caixa de diálogo que é exibida, clique no botão **Salvar**. A Figura 7.3 apresenta essa caixa de diálogo.

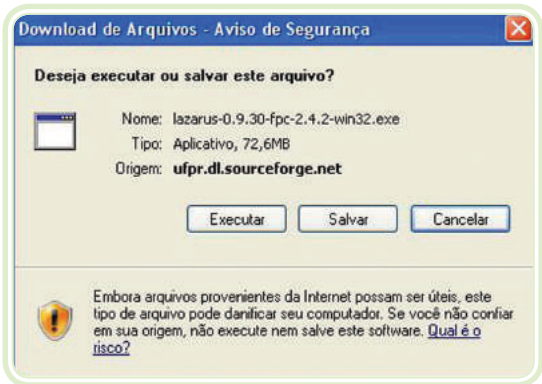

Figura 7.3: Caixa de diálogo referente ao *download* do Lazarus
Fonte: http://sourceforge.net/projects/lazarus/files/Lazarus%20Windows%2032%20bits/Lazarus%200.9.30/lazarus-0.9.30--fpc-2.4.2-win32.exe/download

Por último, escolha uma pasta para armazenar o arquivo referente ao *download*, conforme apresentado na Figura 7.4 e clique em **Salvar**.

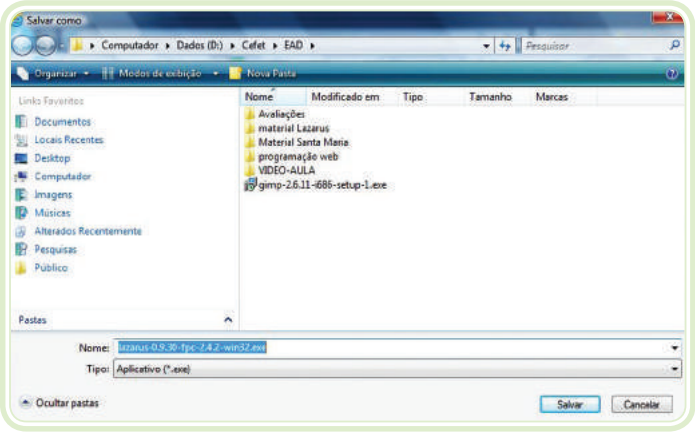

Figura 7.4: Caixa de diálogo Salvar como
Fonte: Sistema operacional Windows

7.2.3 Instalação e configuração do Lazarus

O processo de instalação do Lazarus é bastante simples. Os passos são apresentados a seguir:

a) Dê um duplo clique no arquivo lazarus-0.9.30-fpc-2.4.2-win32.exe ou na versão mais recente, que está salvo em seu computador, referente ao *download* do tópico anterior, o qual representa o programa instalador do Lazarus. Em seguida, clique no botão **Executar**.

b) Escolha o idioma a ser utilizado na instalação do programa, conforme exibido na Figura 7.5, clicando em seguida no botão **OK**.

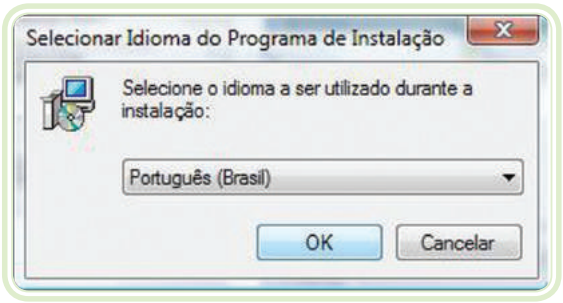

Figura 7.5: Tela para escolha do idioma do instalador
Fonte: *Software* Lazarus

c) Na janela inicial do assistente de instalação, clique em **Avançar** e, depois, escolha a pasta em seu computador no qual o programa será instalado. De preferência mantenha o local sugerido pelo instalador (C:\Lazarus) e, em seguida, clique em **Avançar**.

d) Na janela Selecionar Componentes (Figura 7.6), desmarque a opção: Associar Lazarus com a extensão de arquivo.pas, para evitar conflitos de arquivos.pas associados ao Delphi, e, logo a seguir, clique em **Avançar**.

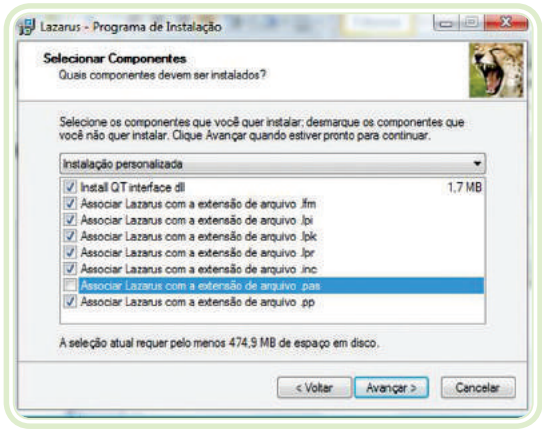

Figura 7.6: Tela instalação personalizada do Lazarus
Fonte: *Software* Lazarus

e) Nas janelas seguintes, escolha avançar, até que surja a janela Pronto para Instalar, na qual se deve clicar no botão **Instalar**.

f) Por último, será mostrada a janela para finalizar o assistente de Instalação do Lazarus, bastando clicar em **Concluir**.

7.2.4 Elementos do IDE

Após instalado o Lazarus, podemos inicializá-lo clicando no ícone correspondente que se encontra na área de trabalho (*desktop*), caso você tenha optado por criá-lo durante a instalação do *software*, ou então, acessando o menu **Iniciar > Programas > Lazarus** e escolhendo o item de programa Lazarus. A Figura 7.7 ilustra o IDE do Lazarus.

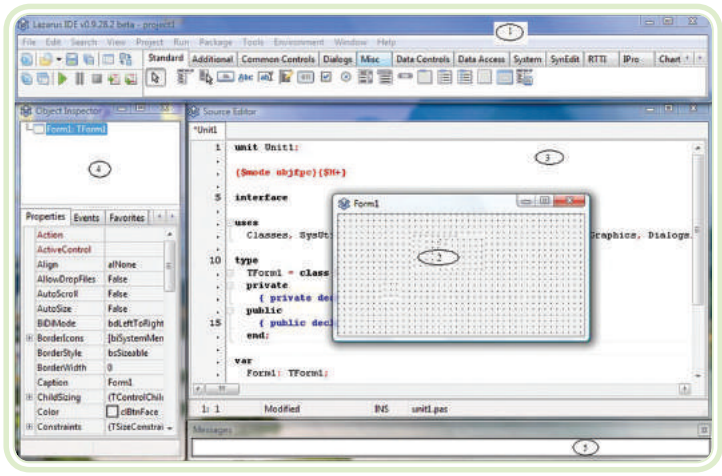

Figura 7.7: IDE do Lazarus
Fonte: *Software* Lazarus

A seguir, são apresentados os principais elementos que compõem o IDE, conforme numeração apresentada na Figura 7.7:

1. **Janela principal** – é constituída basicamente de três seções: barra de menu, barra de botões e paleta de componentes.

- Barra de menu (Figura 7.8) – fornece as ferramentas necessárias ao desenvolvimento da aplicação, bem como os recursos para se configurar o IDE do Lazarus.

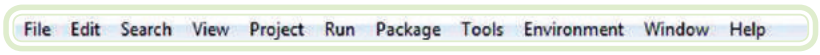

Figura 7.8: Barra de menu
Fonte: *Software* Lazarus

- Barra de botões (Figura 7.9) – fornece um atalho para os comandos frequentemente utilizados no desenvolvimento de uma aplicação.

Figura 7.9: Barra de botões
Fonte: *Software* Lazarus

- Paleta de componentes (Figura 7.10) – contém o conjunto de componentes existentes na biblioteca de componentes, os quais possibilitam a implementação de diversas funcionalidades no desenvolvimento de um projeto. É constituída por uma série de abas, que dividem e agrupam diversos componentes de acordo com suas funções.

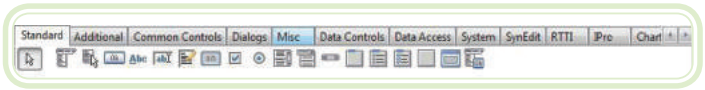

Figura 7.10: Paleta de componentes
Fonte: *Software* Lazarus

2. **Form Designer** – a parte principal do IDE do Lazarus é a área de trabalho. Essa área exibe o form designer, o qual permite a criação dos formulários de uma aplicação. Esses formulários podem representar a janela principal do projeto, uma caixa de diálogo ou qualquer outro tipo de janela.

3. **Editor de código** – área do IDE onde você irá desenvolver o código (lógica) de sua aplicação.

4. **Object Inspector** – painel que apresenta as propriedades (aba Properties) e os códigos (aba Events) associados ao componente atualmente selecionado no formulário.

5. **Janela de mensagens** – apresenta informações referentes à compilação do projeto, tais como: erros do código (sintaxe), informações do compilador FPC, entre outros.

A partir de agora, tendo em vista que você já foi apresentado ao IDE do Lazarus, retomaremos um ponto citado anteriormente, que é sobre a sua configuração. Para termos um ambiente adequado (produtivo) ao desenvolvimento de nossos projetos, podemos ajustar algumas opções do IDE. O Lazarus apresenta vários parâmetros de configuração desse ambiente, dentre os quais podemos destacar alguns bem interessantes:

- Caso queira-se trabalhar com o IDE em Português, deve-se configurá-lo da seguinte maneira: acesse o menu **Tools > Options > Environment Desktop > Language** e escolha **<Brazilian Portuguese [pt_BR]>**. Em seguida, feche o Lazarus e o inicialize novamente, para que as configurações sejam aplicadas. A partir deste momento, o IDE se apresenta em português.

Observação

Caro estudante, apesar de ser mais cômodo usarmos o IDE em português, neste material trabalharemos no idioma original, ou seja, o inglês, tendo em vista que a maioria das linguagens de programação se encontra em inglês e, além disso, permitirá a você uma maior afinidade com esse idioma.

- Ao usarmos o formulário do Lazarus para elaboração da interface gráfica de um determinado aplicativo e, ao clicarmos na janela de mensagens por algum motivo, o foco da operação será levado ao editor de código do IDE, o que ocasiona uma perda de produtividade no desenvolvimento da operação em questão. Sendo assim, podemos ajustar o IDE de maneira a não redirecionar o foco da operação apenas com um clique mas, quando realmente se fizer necessário, a partir de um duplo clique, da seguinte maneira: Em **Tools > Options > Environment > Desktop**, acesse o quadro **Misc Options** e marque a primeira opção **Jump from message to source line on double click**, conforme Figura 7.11.

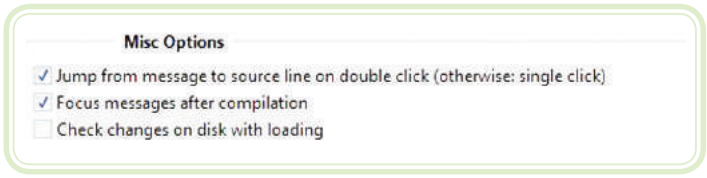

Figura 7.11: Opções diversas
Fonte: *Software* Lazarus

- Uma opção importante a ser configurada no IDE é quanto à remoção de métodos vazios existentes no editor de códigos. Os métodos representam os procedimentos ou funções criados para a realização de tarefas específicas na solução de um problema, mas quando se encontram vazios, não fazem sentido. Sendo assim, podemos configurar o IDE para eliminarmos tais métodos do sistema, da seguinte forma: **Tools > Options > Editor > Completion and Hints** e marque a opção **Auto remove empty methods**.

- Caso queira que na inicialização do Lazarus seja apresentado um novo projeto, e não o projeto manipulado na última vez em que você usou o IDE, utilize o seguinte procedimento: acesse o menu **Tools > Options > Environment** e desmarque a opção **Open last Project at start**.

A partir dos ajustes (configurações) do IDE já apresentados, constata-se que existe uma variedade enorme de outros parâmetros que podem ser configurados. Entretanto, você perceberá a necessidade de realizá-los, quando você se sentir mais familiarizado com o ambiente de programação que estamos estudando, bem como quando no desenvolvimento das soluções para os problemas que lhe serão apresentados ao longo deste curso.

7.2.5 Biblioteca de componentes Lazarus

O Lazarus permite o desenvolvimento rápido de aplicações gráficas (RAD) de maneira semelhante ao Delphi, o que significa ter uma biblioteca de componentes (LCL) muito parecida com a do Delphi (VCL). Essa biblioteca de componentes representa um conjunto de **classes** que seguem uma determinada hierarquia. Entretanto, não quer dizer que essas bibliotecas sejam idênticas, pois, muitos dos controles existentes na VCL podem não existir na LCL ou vice-versa. Para ilustrar esse fato, tem-se que a VCL contém componentes visuais e não visuais, já a LCL apresenta apenas componentes visuais, tendo em vista que grande parte dos componentes não visuais é provida por outra biblioteca, a FLC (*Free Component Library*), a qual é mantida pela equipe do Free Pascal. Adicionalmente, até mesmo para controles que sejam comuns às duas bibliotecas, deverão ser feitas alterações nas aplicações, componentes e controles para migrá-los de um ambiente para o outro (LAZARUS WIKI).

A-Z

classes
É a definição de um conjunto de características e comportamentos que os elementos (objetos) deverão seguir.

A-Z

tempo de projeto
Etapa em que o programador desenvolve o projeto (aplicação) no IDE, tal como: elaboração da interface gráfica/codificação do programa.

tempo de execução
Etapa no qual o programa encontra-se em execução.

Conceitos – componentes visuais representam os componentes utilizados no desenvolvimento de uma aplicação, ou seja, em **tempo de projeto**, e que possibilitam modificarem seus comportamentos visuais nesse estado. Podemos citar, por exemplo, a utilização de uma caixa de texto, na qual alterando o tipo de fonte ou seu tamanho, refletirá instantaneamente no projeto. Já os componentes não visuais caracterizam-se como elementos de apoio ao desenvolvimento do projeto, mas que não aparecem em **tempo de execução** do mesmo. Eles poderão, portanto, serem vistos, apenas em tempo de projeto, através dos ícones que os representam. Como exemplo, podemos citar o uso do componente `MainMenu`, o qual é utilizado para a criação de menus dentro de uma aplicação, mas que não é exibido quando o projeto é executado.

7.2.6 Trabalhando com componentes

Na montagem da interface gráfica de um aplicativo, devemos utilizar os componentes existentes na paleta de componentes, já citada anteriormente, os quais se encontram organizados, funcionalmente, por meio de abas. Você verá ao longo deste curso, que é bastante simples e intuitivo trabalhar com esses elementos. A seguir, são apresentadas as abas mais comuns desta paleta, juntamente com os componentes mais usuais, utilizados na criação de um programa.

7.2.6.1 Paleta de componentes (abas)

- **Standard** (Figura 7.12) – fornece os componentes mais comuns em nível de interface gráfica.

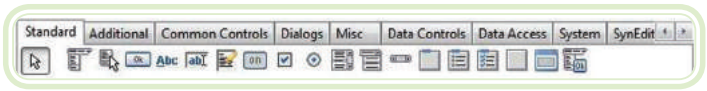

Figura 7.12: Aba Standard
Fonte: *Software* Lazarus

Quadro 7.1: Componentes mais usuais da aba Standard

Ícone	Nome	Finalidade
	`TMainMenu`	Permite a criação de menus suspensos dentro de uma aplicação.
	`TButton`	Usado na criação de botões que permitem ao usuário interagir com a aplicação por meio de ações específicas, através do acionamento (clique) desses botões.
	`TLabel`	Permite a inserção de textos no formulário, os quais não podem ser alterados pelo usuário.
	`TEdit`	Componente usado para a entrada/saída de dados num programa.
	`TMemo`	Usado para a inserção de texto por meio de várias linhas numa caixa de edição.

	TCheckBox	Possibilita por meio do clique do mouse habilitar/desabilitar uma determinada opção de sua aplicação.
	TRadioButton	Controle que permite ao usuário escolher apenas uma opção de um conjunto de opções disponíveis.
	TListBox	Usado para exibir informações dentro de uma caixa de listagem, bem como permite que o usuário possa selecionar um ou mais itens desta caixa de listagem.
	TComboBox	É uma combinação de uma caixa de listagem e uma caixa de texto que permite ao usuário digitar a informação no próprio controle ou escolhê-la da caixa de listagem.
	TGroupBox	Componente que se apresenta na forma de uma moldura, com o objetivo de agrupar um conjunto de controles, tais como um conjunto de opções como RadioButtons.
	TRadioGroup	Representa um *container* de botões de rádio, ou seja, combina as características do GroupBox com o RadioButton.
	TPanel	É um *container* genérico para agrupamento de controles.

Fonte: Adaptado de Squadra Tecnologia em Software, 2000

- **Additional** (Figura 7.13) – nesta aba encontramos alguns componentes de interface um pouco mais elaborados e com alguns recursos a mais.

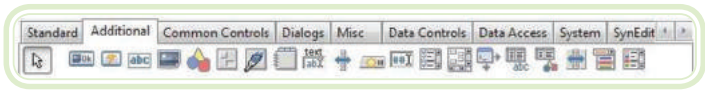

Figura 7.13: Aba Additional
Fonte: *Software* Lazarus

Quadro 7.2: Componentes mais usuais da aba Additional

Ícone	Nome	Finalidade
	TBitBtn	Semelhante ao Button da aba Standard, mas com a possibilidade de se adicionar uma imagem bitmap.
	TImage	Controle usado para adicionar imagens ao programa.
	TShape	Usado para se desenhar alguns tipos de figuras geométricas (quadrado, círculo, etc.).
	TMaskEdit	Controle usado para formatação dos dados em diferentes formatos.
	TStringGrid	Apresenta/edita dados do tipo string através de linhas e colunas.

Fonte: Adaptado de Squadra Tecnologia em Software, 2000

- **Common Controls** (Figura 7.14) – apresenta componentes que permitem acompanhar a execução de rotinas (ações) específicas do programa, através do controle do tempo, bem como definir atalhos para recursos do aplicativo ou sistema.

Figura 7.14: Aba Common Controls
Fonte: *Software* Lazarus

Quadro 7.3: Componentes mais usuais da aba Common Controls

Ícone	Nome	Finalidade
	TTrackBar	Barra de controle "deslizante" que fornece uma guia a partir da qual o usuário pode movimentá-la por toda sua região.
	TProgressBar	Controle usado para representar uma operação em andamento como, por exemplo, o *download* ou transferência de um arquivo.
	TStatusBar	Usado para exibir informações referentes a uma interface gráfica e que permite ser dividida em várias seções ou painéis.
	TToolBar	Utilizado na criação de uma barra de ferramentas dentro da aplicação.
	TPageControl	Controle usado para se criar guias que exibem páginas, as quais podem conter outros componentes.
	TImageList	Permite o agrupamento de imagens que serão utilizadas numa determinada aplicação.

Fonte: Adaptado de Squadra Tecnologia em Software, 2000

- **Dialogs** (Figura 7.15) – contém componentes que possibilitam a criação de caixas de diálogo em uma aplicação.

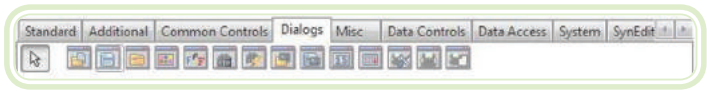

Figura 7.15: Aba Dialogs
Fonte: *Software* Lazarus

Quadro 7.4: Componentes mais usuais da aba Dialogs

Ícone	Nome	Finalidade
	TOpenDialog	Controle usado para se gerar caixa de diálogo para a seleção do arquivo que será aberto.
	TSaveDialog	Fornece caixa de diálogo para que se informe o nome e o destino do arquivo a salvar.
	TFontDialog	Controle usado para fornecer caixa de diálogo para a seleção de fontes.
	TPrintDialog	Fornece a caixa de diálogo para se configurar opções de impressão.

Fonte: Adaptado de Squadra Tecnologia em Software, 2000

- **System** (Figura 7.16) – fornece componentes para se configurar HTML, XML e o tempo necessário à execução de rotinas específicas de uma aplicação.

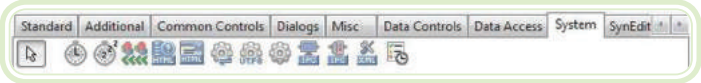

Figura 7.16: Aba System
Fonte: *Software* Lazarus

Quadro 7.5: Componentes mais usuais da aba System

Ícone	Nome	Finalidade
	`TTimer`	Controle usado para se executar códigos específicos de uma aplicação, de tempo em tempo.

Fonte: Autores

Um primeiro exemplo

Agora, será mostrado como usar o IDE do Lazarus através de um exemplo bem simples. Os programas aqui desenvolvidos serão chamados de projetos. Sendo assim, este nosso primeiro projeto terá por objetivo exibir a mensagem: Bem-vindo ao e-Tec Brasil!

Abra o Lazarus e observe que se apresenta um projeto vazio juntamente com seu formulário (pressione **F12** se o mesmo não estiver visível). Caso você já tenha um projeto aberto e deseja iniciar um novo projeto, basta escolher a seguinte opção: **Project > New Project > Application**.

Antes de iniciar a elaboração da interface gráfica do nosso projeto, é importante observar que, para cada projeto desenvolvido no sistema, faz-se necessário armazená-lo numa pasta específica. Portanto, minimize o IDE e crie, em seu sistema, essa pasta. Logo em seguida, restaure o IDE e torne ativa a aba Standard clicando sobre a mesma. Neste momento, vamos inserir um botão no formulário: clique no ícone `TButton` e depois clique na posição do formulário onde se deseja inserir esse componente. Neste botão, altere o seu rótulo para EAD, ou seja, altere a propriedade `Caption`. Para isso, faça o seguinte: selecione o botão clicando uma vez sobre ele, e no **Object Inspector**, que se encontra à esquerda do formulário, ative a aba Properties (provavelmente ela já se encontre ativa) clicando sobre a mesma. Por último, substitua o valor da propriedade `Caption` de `Button1` para EAD. A Figura 7.17 ilustra como ficou o formulário.

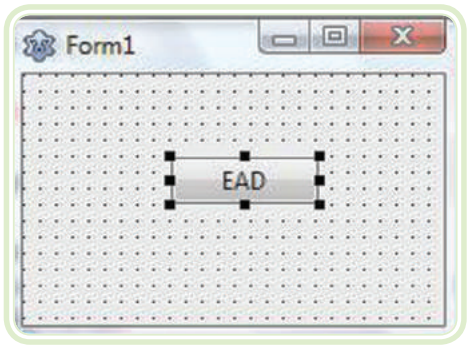

Figura 7.17: Formulário primeiro exemplo
Fonte: *Software* Lazarus

Após elaborar a parte gráfica da aplicação, que por sinal ficou bastante simples, salve o projeto da seguinte maneira: clique no menu **File > Save All**. Na caixa de diálogo, selecione a pasta que você criou anteriormente para salvar o projeto. Em seguida, na caixa de texto **Nome**, mantenha a sugestão informada (project1.lpi) e clique em **Salvar**. Na sequência, surge outra caixa de diálogo na qual deverá ser informado o nome da Unit. Da mesma forma, mantenha o nome sugerido (Unit1.pas) e clique em **Salvar**, e assim, concluímos a operação de gravação do projeto. O passo seguinte é montarmos o código correspondente à solução do problema.

Dê um duplo clique no botão para acionarmos o método padrão deste componente, o qual está associado ao evento `OnClick`. Tenha em mente que, o método pode ser entendido como um conjunto de instruções que, por enquanto, você pode considerar como sendo uma função ou procedimento, os quais já foram estudados em algoritmos. Em seguida, a declaração do método é criada dentro do editor de códigos do Lazarus e o cursor posicionado dentro do bloco do procedimento. Para finalizarmos nosso programa, digite o seguinte comando: `Show` e aperte "**Ctrl + Space**", para que o IDE apresente todos os itens iniciados por esse comando. Complete a operação escolhendo no menu suspenso pelo comando `ShowMessage` e aperte **Enter** em seguida. Por fim, o comando deverá ficar da seguinte forma: `ShowMessage` ('Bem-vindo ao e-tec Brasil!'). Para executar a aplicação, clique no botão **Run** da barra de botões ou pressione **F9**. O resultado é apresentado na Figura 7.18.

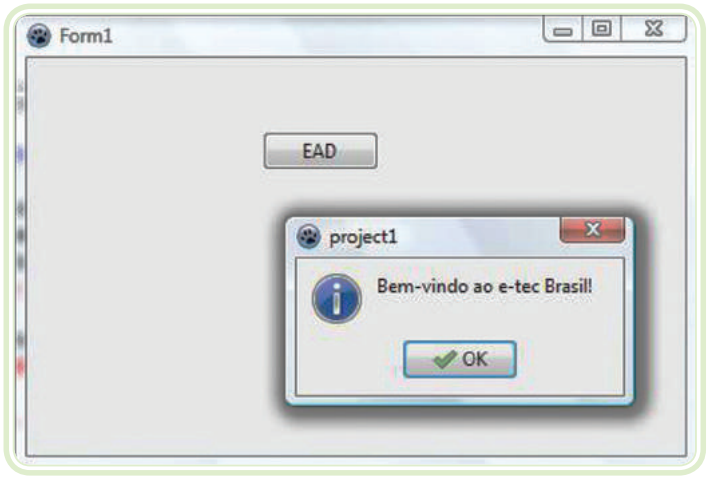

Figura 7.18: Execução primeiro exemplo
Fonte: *Software* Lazarus

7.3 Arquivos que compõem uma aplicação Lazarus

Um projeto desenvolvido através do Lazarus gera arquivos de diferentes tipos (extensões) quando o mesmo é gravado numa pasta. Segundo Almeida (2010), os arquivos mais comuns de um projeto são:

Quadro 7.6: Tipos de extensões do Lazarus

Extensão Arquivo	Finalidade
*.LPI	Arquivo que contém as informações (configurações específicas) do projeto.
*.LPR	Arquivo de projeto do Lazarus, ou seja, contém o código fonte do programa principal do projeto.
*.LFM	Contém informações da configuração de todos os objetos do formulário. Está sempre associado a um arquivo de extensão PAS.
*.PAS *PP	Arquivo que possui o código da Unit, em Pascal, normalmente associado a um determinado formulário da aplicação.
*.LRS	Arquivo de resource (recursos) do Lazarus, o qual geralmente é gerado durante a compilação do projeto.
*.PPU	Para cada Unit do projeto, durante o processo de compilação, é gerado esse tipo de arquivo (Unit compilada) através do FPC.
*.LPK	Arquivo que contém informações específicas do pacote (Package).

Fonte: Adaptado de Almeida, 2010

7.4 Elementos da linguagem Pascal/Object Pascal

Os programas em Lazarus são desenvolvidos para o Free Pascal através das linguagens Pascal e Object Pascal, originária do Pascal, sendo que esta foi desenvolvida por volta de 1968 a 1970 por Niklaus Wirth, professor da Universidade de Zurique (Suíça). O nome Pascal foi dado em homenagem ao filósofo e matemático Blaise Pascal, que viveu entre os anos de 1623 e 1662.

Nesse tópico, serão apresentadas algumas características dessa linguagem, bem como seus tipos básicos de dados e principais operadores.

7.4.1 Características da linguagem

a) As instruções em Pascal são *case insensitive*, ou seja, não há diferenciação entre maiúsculas e minúsculas.

b) As instruções devem ser seguidas por ponto-e-vírgula.

c) Utiliza-se ":=" para se atribuir um valor a uma variável.

d) Os comentários realizados dentro de um programa, com o objetivo de facilitar o entendimento de trechos específicos do código, podem ser realizados da seguinte maneira:

- Utilize os delimitadores de comentários "(*...*)" ou entre chaves "{...}", de forma a envolver o texto explicativo. O exemplo a seguir demonstra o uso desses delimitadores.

  Exemplo

  (*Programa para o cálculo da média das idades de um um grupo de indivíduos*)

  ou

  {Programa para o cálculo da média das idades de um grupo de indivíduos}

- Utilize o delimitador "//" quando se desejar fazer comentários para uma única linha.

e) O Free Pascal reconhece um grupo de palavras como sendo específicas (reservadas) da linguagem, ou seja, têm um significado para o compilador e que, portanto, não podem ser redefinidas pelo programador, ou seja, não podem ser nomes de variáveis e/ou constantes. Maiores informações sobre essas palavras reservadas são apresentadas por Canneyt (2011), em seu guia de referência para Free Pascal, que poderá ser acessado em: <**ftp.freepascal.org/pub/fpc/docs-pdf/ref.pdf**>.

f) Os identificadores são nomes atribuídos pelo programador às variáveis, constantes, procedimentos, funções e tipos definidos. Portanto, todos os nomes definidos no código fonte do programa, exceto palavras reservadas, caracterizam identificadores. As regras de construção dos identificadores seguem as apresentadas em algoritmos.

g) **END** seguido de ponto indica ao compilador o final de uma `Unit`.

7.4.2 Tipos de dados

Os dados a serem processados por um programa devem ser armazenados por meio de variáveis. O Free Pascal disponibiliza para o programador um conjunto de tipos de dados predefinidos, dentre os quais serão apresentados os tipos inteiro, real, caractere e lógico.

7.4.2.1 Tipos de dados inteiros

Quadro 7.7: Tipos inteiros

Tipo inteiro	Intervalo
Byte	0 .. 255
Shortint	-128 .. 127
Smallint	-32768 .. 32767
Word	0 .. 65.535
Integer	Mapeia para Longint em Object Free Pascal
Cardinal	Mapeia para Longword
LongInt	-2147483648 .. 2147483647
Longword	0 .. 4294967295
Int64	-9223372036854775808 .. 9223372036854775807
QWord	0 .. 18446744073709551615

Fonte: Santos, 2011

7.4.2.2 Tipos de dados reais

Quadro 7.8: Tipos reais

Tipo real	Intervalo
Real	Depende da plataforma operacional
Single	1.5^{-45} .. 3.4^{38}
Double	5.0^{-324} .. 1.7^{308}
Extended	1.9^{-4932} .. 1.1^{4932}
Comp	$-2^{64}+1$ .. $2^{63}-1$
Currency	-922337203685477.5808 .. 922337203685477.5807

Fonte: Santos, 2011

7.4.2.3 Tipos de dados caractere

O Free Pascal suporta o tipo `Char`, o qual ocupa 1 *byte* em tamanho e armazena um único caractere. Caso seja necessário armazenar um conjunto de caracteres, faz-se o uso do tipo string.

7.4.2.4 Tipos de dados lógicos

Pode-se trabalhar no Free Pascal com o tipo Boolean, que pode armazenar um dos dois possíveis valores pré-definidos: `True` ou `False`.

7.4.3 Operadores

Quadro 7.9: Operadores aritméticos/relacionais/lógicos

Tipo de operadores	Símbolo
Aritméticos	+ (adição) – (subtração) * (multiplicação) / (divisão de pontos flutuantes) Div (divisão Inteira) Mod (resto da divisão de inteiros)
Relacionais	= (igual) < > (diferente) > (maior) < (menor) >= (maior ou igual) <= (menor ou igual)
Lógicos	Not (negação) And (E lógico) Or (OU lógico)

Fonte: Autores

7.5 Aplicação

Neste tópico será desenvolvida uma aplicação, na qual faremos apenas o uso de comandos de entrada e saída para a solução do problema a ser apresentado. Entretanto, antes de passarmos ao enunciado do problema, é importante ressaltar algumas etapas necessárias ao desenvolvimento da solução do mesmo.

Segundo Manzano e Mendes (1999), a primeira etapa compreende o entendimento por completo do problema em questão, no qual devemos identificar os dados fornecidos, bem como o que se deseja calcular, de maneira a se chegar à solução final. A segunda etapa caracteriza-se pela elaboração do algoritmo. Com o algoritmo pronto, passa-se à montagem da interface gráfica do programa; e, por último, desenvolve-se o conjunto de instruções associado aos eventos (procedimentos) para cada um dos componentes existentes em seu formulário que irão sofrer uma ação direta do usuário. Diante dessas observações, passamos, agora, ao desenvolvimento do programa.

Problema

Um funcionário trabalha 5 dias por semana, 8 horas por dia. Fazer uma aplicação para calcular o salário semanal desse funcionário sendo que o valor da hora deve ser digitado via teclado. Apresentar o resultado em um componente `Edit` (caixa de texto).

Solução

1ª etapa – após entendido o problema exposto, devemos observar quais são os dados fornecidos no enunciado, bem como o que se pede para calcular. De acordo com uma primeira análise do problema, constatamos que nos é fornecido o valor de cada hora trabalhada pelo funcionário e que o mesmo trabalha 8 horas por dia durante os cinco dias da semana. Observamos, também, que, com base nessas informações, devemos calcular o salário desse funcionário. Sendo assim, podemos apresentar um pequeno resumo com essas informações:

Dados: Valor da hora trabalhada.
Trabalha 5 dias por semana sendo 8 horas por dia.

Pede-se: Salário que o funcionário irá receber, ao final de uma semana de trabalho.

2ª etapa – não nos preocuparemos em apresentar o algoritmo para essa aplicação, tendo em vista que esse assunto já foi bastante abordado em aulas anteriores, além do que, dedicaremos uma atenção especial para o código que será desenvolvido no IDE do Lazarus. Contudo, prezado aluno, é importante que nessa fase inicial da elaboração dos nossos primeiros programas em Lazarus, você realize essa atividade antes de passar à etapa seguinte.

3ª etapa – essa etapa consiste em se elaborar a interface gráfica do aplicativo. Sendo assim, abra o Lazarus e inicie um novo projeto (**Project > New Project > Application**). No formulário da aplicação iremos inserir alguns componentes que se encontram na aba Standard. Portanto, selecione essa aba e adicione ao formulário dois componentes do tipo `Label`, um abaixo do outro; e dois componentes do tipo `Edit`, sendo que cada um deles deverá ficar à direita de cada um dos `labels`. Em seguida, insira três `Buttons` no formulário. A Figura 7.19 ilustra essa operação.

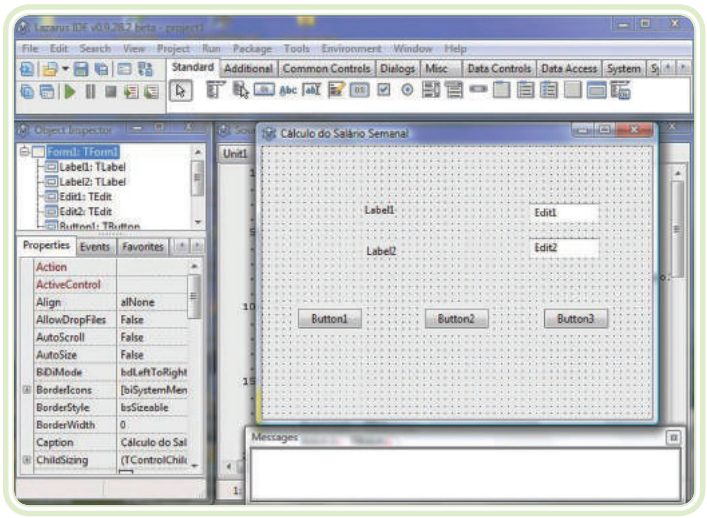

Figura 7.19: Interface inicial cálculo salário semanal
Fonte: *Software* Lazarus

O passo seguinte é alterar algumas das propriedades desses componentes através do **Object Inspector**. Os valores a serem informados às propriedades são inseridos à direita de cada propriedade, na coluna de valores. Veja o exemplo aplicado à propriedade `Caption` de um componente `TButton`.

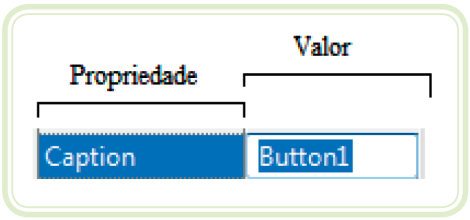

Figura 7.20: Propriedade Caption
Fonte: *Software* Lazarus

Para inserir outro valor nessa propriedade, exclua o valor existente, que no caso é `Button1`, e digite, por exemplo, "Calcular". O resultado é apresentado a seguir.

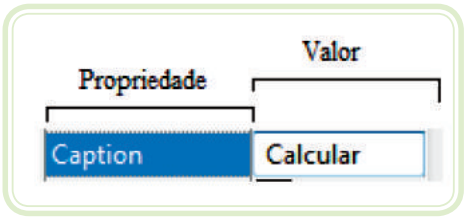

Figura 7.21: Alteração do valor da propriedade Caption
Fonte: *Software* Lazarus

Propriedades

`Form1`

Propriedade `Caption` – altere para "Cálculo do Salário Semanal"

`Label1`

Propriedade `Caption` – altere para "Digite o valor da hora trabalhada:"

`Label2`
Propriedade `Caption` – altere para "Salário = "
`Edit1`
Propriedade `Text` – deixe em branco
`Edit2`
Propriedade `Text` – deixe em branco
`Button1`
Propriedade `Caption` – altere para "Calcular"
`Button2`
Propriedade `Caption` – altere para "Limpar"
`Button3`
Propriedade `Caption` – altere para "Sair"

Observe como ficou a interface final (Figura 7.22).

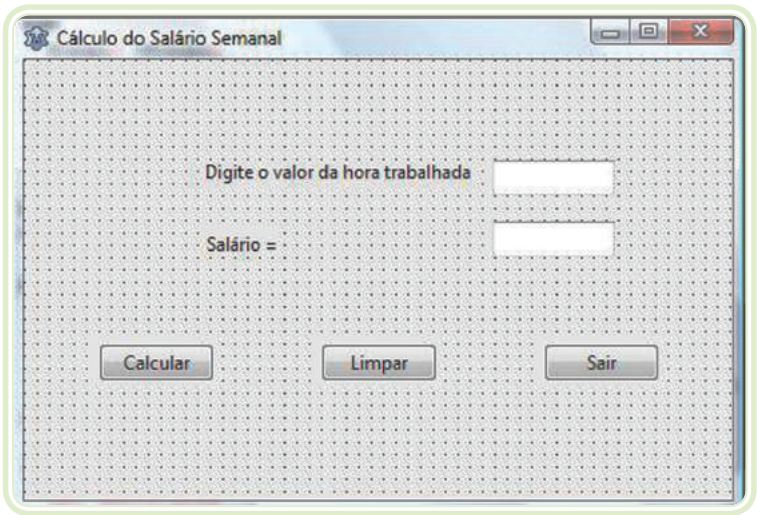

Figura 7.22: Interface final cálculo salário semanal
Fonte: *Software* Lazarus

Neste momento, salve o seu projeto numa nova pasta (**File > Save All**).

4ª etapa – nesta etapa, será feita a codificação do programa. O código é desenvolvido de acordo com as ações que o usuário poderá realizar quando estiver usando o programa, tais como: clicar sobre um objeto, mover o *mouse*, alterar o conteúdo de um componente da aplicação, etc. Dessa forma, toda vez que quisermos escrever um código em nosso aplicativo, associado a alguma ação realizada pelo usuário, utilizaremos o evento correspondente daquele componente com qual o usuário interagirá.

Portanto, na solução deste problema, vamos, inicialmente, escolher o evento `OnClick` do botão Calcular, pois, é a partir desse evento que serão executadas

as instruções para o cálculo do salário do funcionário. Sendo assim, selecione no **Object Inspector** a aba Events, localize o evento `OnClick` e, em seguida, dê um duplo clique na coluna à direita desse nome. Ao realizarmos essa operação, o Lazarus gera automaticamente duas ações. A primeira delas é a de criar a declaração de um procedimento dentro da `Unit`, a qual está associada o formulário, conforme mostrado a seguir:

```
{TForm1}
  TForm1 = class(TForm)
    Button1: TButton;
    Edit1: TEdit;
    Edit2: TEdit;
    Label1: TLabel;
    Label2: TLabel;
  procedure Button1Click(Sender: TObject);
  private
    {private declarations}
  public
    {public declarations}
  end;
var
  Form1: TForm1;
```

A segunda ação é a de inserir dentro da `Unit1`, após a seção `Implementation`, a estrutura básica (corpo) do procedimento `Button1Click`, de forma que possamos escrever as instruções necessárias à realização da operação desejada que, no caso, é a de calcular o salário. A Figura 7.23 ilustra essa ação.

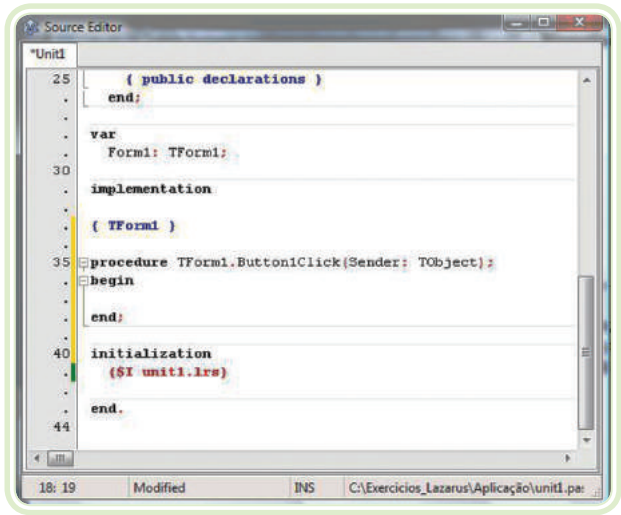

Figura 7.23: Estrutura básica do procedimento Button1Click
Fonte: *Software* Lazarus

A seguir é apresentado o procedimento, com o código completo, para se calcular o salário semanal do funcionário:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  VH, SALARIO: real;
begin
  VH:= StrToFloat(Edit1.Text);
  SALARIO:= VH * 8 * 5;
  Edit2.Text:= FloatToStr(SALARIO);
  Edit1.SetFocus;
end;
```

Segundo Santos (1998), são apresentadas algumas funções de conversão de tipos diferentes de dados:

StrToFloat (Valor-Texto): devolve o valor em real, o qual foi passado como texto através de parâmetro.

FloatToStr (Valor-Real): retorna o valor em string (texto) correspondente ao valor real passado como parâmetro.

Explicando

Inicialmente, declaramos as variáveis VH e SALARIO como sendo do tipo real, pois, as mesmas poderão receber valores fracionários. Como exemplo, podemos citar o caso do usuário informar o valor da hora igual a 12,50 e, aí, o programa calculará R$ 500,00 para o salário do funcionário. Como a fórmula para cálculo do salário contém o valor da hora que é real, assim, também, a variável SALARIO o será, apesar de o resultado encontrado ser exato. Observe que, antes da declaração dessas variáveis, existe o identificador `var`, que tem por objetivo indicar que logo após a mesma, serão criadas as variáveis do procedimento. Em seguida, fizemos a conversão do dado de entrada informado pelo usuário no `Edit1` que, neste caso, é o valor da hora e o atribuímos à variável VH. Essa conversão foi necessária, pois, qualquer dado contido num componente `Edit` é do tipo texto e, como o valor da hora é numérico devemos, dessa forma, convertê-lo de texto para real. Ao lado, são apresentadas algumas funções de conversão para números. Retomando nossa explicação, posteriormente, realizamos o cálculo do salário do funcionário e, logo após, o apresentamos no formulário por meio do `Edit2`. Veja que, novamente, foi necessária a conversão de tipos, só que agora, a mesma é feita de maneira contrária, ou seja, de real para texto, pois, o resultado deverá ser mostrado num `Edit`. Por último, foi usado o método `SetFocus` que posicionará o cursor no `Edit1` do formulário.

O próximo passo na solução deste problema, é programarmos os eventos `OnClick` dos botões Limpar e Sair, respectivamente. Para isso, criamos os procedimentos para esses botões de forma análoga ao que foi feito para o botão Calcular. Outra maneira de gerarmos esses procedimentos padrões é dando um duplo clique em cada um dos botões. O código para cada um desses procedimentos é mostrado a seguir.

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit1.Text:=' ';
  Edit2.Text:=' ';
  Edit1.SetFocus;
end;
procedure TForm1.Button3Click(Sender: TObject);
begin
  close;
end;
```

Explicando

As instruções relativas ao `OnClick` do botão Limpar, têm por objetivo apagar os conteúdos dos componentes `Edit1` e `Edit2` respectivamente; já o método `SetFocus` reposiciona o cursor no `Edit1` do formulário. Quanto ao código relativo ao procedimento do botão Sair, tem-se que, o mesmo finaliza a execução da aplicação.

Agora, salve novamente a aplicação e, em seguida, execute-a (tecle **F9**).

Resumo

Nesta aula, você teve a oportunidade de conhecer essa poderosa ferramenta de programação que é o Lazarus/Free Pascal, inicialmente através de alguns conceitos. Posteriormente, aprendendo a fazer a instalação, bem como algumas configurações básicas do Lazarus e, em seguida, foi levado a familiarizar-se com o seu IDE. Conheceu, também, os elementos básicos da linguagem Pascal/Object Pascal suportados pelo compilador Free Pascal. Por último, você executou o seu primeiro programa usando apenas estruturas sequenciais, dentro de um ambiente voltado para o desenvolvimento de aplicações com interfaces gráficas.

Encerramos assim, nossa sétima aula, e, esperamos que o assunto apresentado, tenha lhe estimulado bastante para que você possa criar, a partir de agora, programas por meio do uso dessa ferramenta com bastante criatividade e qualidade. Para auxiliá-lo nesse processo, a seguir, daremos início ao estudo das outras estruturas de controle para o desenvolvimento de soluções mais apuradas. Vamos em frente, então?

Atividades de aprendizagem

1. Elaborar um projeto para calcular o salário líquido de um funcionário. Para isto, são fornecidos o valor da hora trabalhada, o número de horas trabalhadas no mês e o percentual de desconto a ser aplicado sobre o salário base desse funcionário. Sendo assim, o programa deverá apresentar o salário líquido. Implementar, também, as rotinas de limpar os componentes do formulário e de sair da aplicação, ou seja, inserir no formulário mais dois botões e programar o evento `OnClick` dos mesmos (adaptado de Manzano; Mendes, 1999).

2. Fazer um projeto em Lazarus que leia os lados (A, B, C) do paralelepípedo, determine e imprima o volume (V) e a área superficial (S).

   Dados: $V = A \times B \times C$

   $S = 2 \times (A \times B + A \times C + B \times C)$

3. Fazer um projeto em Lazarus que leia o raio de um círculo, determine e imprima a área correspondente.

   Dado: $A = \pi \times R^2$

   Onde: $\pi = 3.1416$

Aula 8 – Estruturas condicionais

Objetivos

Conhecer as estruturas condicionais para o compilador Free Pascal.

Aplicar essas estruturas condicionais dentro do IDE do Lazarus.

Desenvolver programas de computador na linguagem do Lazarus/FreePascal que se utilizam dessas estruturas, para a solução de problemas que envolvam a tomada de decisões.

8.1 Tomada de decisão

A linguagem Pascal/Object Pascal permite o uso de duas estruturas para a tomada de decisão. Essas estruturas são:

`IF ... then ... else`	(Se ... então ... senão)
`Case ... of ... else ... end`	(Caso ... seja ... senão ... fim do comando)

8.1.1 Estrutura condicional simples

Conforme visto em algoritmos, essa estrutura tem por finalidade analisar o valor de uma expressão lógica e, caso o mesmo seja verdadeiro, executar a instrução correspondente; caso contrário, essa estrutura condicional é abandonada e a execução do programa passa para o comando seguinte. A sintaxe (conjunto de regras formais, usadas pela linguagem, para a definição dos comandos) é apresentada a seguir.

Sintaxe

```
if <expressão lógica> then
  <instrução>;
```

Exemplo

```
if A > B then
  ShowMessage ('O valor de A é maior que B!');
```

Caso ocorra a necessidade de se usar mais de uma instrução para quando a condição for verdadeira, devemos colocá-las num bloco de comandos, o qual é delimitado pelas palavras `begin ... end`; veja como fica a nova sintaxe:

Sintaxe

```
if <expressão lógica> then
  begin
    <instrução1>;
    <instrução2>;
         :
    <instruçãoN>;
  end;
```

No bloco de comandos anterior, foram inseridas várias instruções, sendo essas representadas pelas instrução1, instrução2 até instruçãoN. Essas instruções podem ser entendidas como quaisquer dos comandos já estudados até o momento, sendo inclusive outro comando `if`.

Segundo Santos (1998), são apresentadas algumas funções de conversão de tipos diferentes de dados:

IntToStr (Valor-Inteiro): retorna o valor em string (texto) correspondente ao valor inteiro passado como parâmetro.

Exemplo

```
if A > B then
  begin
    A:= B + 2;
    B:= B - 1;
    Edit1.Text := IntToStr(A);
    Edit2.Text := IntToStr(B);
  end;
```

Com o objetivo de aplicarmos o uso dessa estrutura condicional na solução de um problema, passemos então ao seu enunciado.

8.1.1.1 Aplicação

Problema

Fazer um projeto em Lazarus que leia os lados A e B de um terreno retangular. Em seguida, calcule e imprima a área desse terreno, apresentando-a através de um componente `TLabel`. Apresentar, também, a mensagem: "Terreno espaçoso", caso a área calculada seja superior a 360 m^2.

Observação

A área de um terreno retangular é dada pela fórmula: Área = A × B

Solução

a) Dados do problema:

- Valores dos lados do terreno (A e B).

b) O que se pede:

- Calcular a área do terreno retangular e exibir uma mensagem.

c) Elaboração do algoritmo ou do fluxograma.

d) Elaboração da interface gráfica.

Abra um novo projeto e insira no formulário, dois componentes do tipo `Label`, um embaixo do outro; dois componentes do tipo `Edit`, um ao lado de cada componente `Label`; abaixo do segundo componente `Label`, insira um componente `Button`; e, por último, abaixo deste componente, insira dois outros componentes `Label`, um ao lado do outro. O último `Label` (`Label4`) será usado para exibir o valor da área calculado. A Figura 8.1 ilustra essa interface.

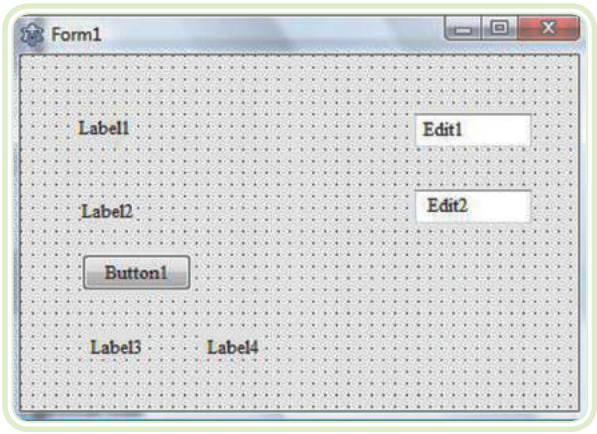

Figura 8.1: Formulário inicial do cálculo da área de um terreno
Fonte: *Software* Lazarus

A partir daí, altere as propriedades desses componentes conforme apresentado no Quadro 8.1.

Quadro 8.1: Propriedade modificadas dos componentes do formulário inicial

Componente	Propriedade	Valor
`Form1`	`Caption`	"Cálculo da Área de um Terreno"
`Label1`	`Caption`	"Informe um lado do terreno: "
`Label2`	`Caption`	"Informe o outro lado do terreno: "
`Edit1`	`Text`	Obs.: apagar o conteúdo desta propriedade.
`Edit2`	`Text`	Obs.: apagar o conteúdo desta propriedade.
`Button1`	`Caption`	"Calcular"
`Label3`	`Caption`	"Área = "

Fonte: Autores

A Figura 8.2 ilustra como ficou a interface gráfica final do projeto.

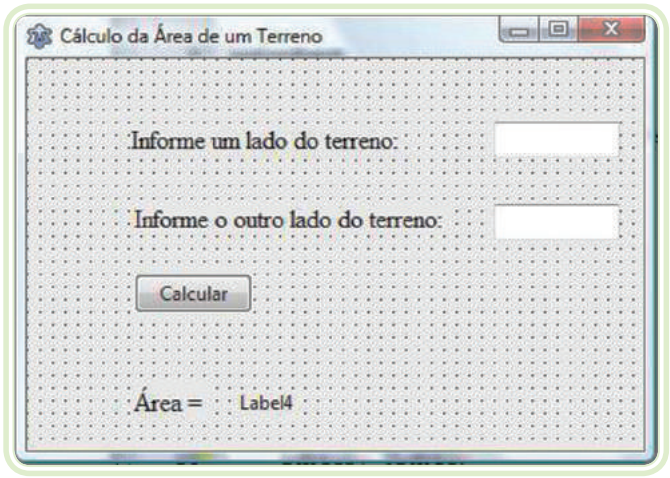

Figura 8.2: Formulário final do cálculo da área de um terreno
Fonte: *Software* Lazarus

Agora, selecione o `Label4` e apague o valor da propriedade `Caption`. Em seguida, salve o projeto numa nova pasta.

e) Código do projeto.

Agora, vamos programar no evento `OnClick` do botão Calcular. Para isso, dê um duplo clique no botão Calcular e digite dentro do procedimento, o código a seguir:

```
procedure TForm1.Button1Click(Sender: TObject);
var
   A, B, AREA: real;
begin
   A:= StrToFloat(Edit1.Text);
   B:= StrToFloat(Edit2.Text);
   AREA:= A * B;
   Label4.Caption:= FloatToStr(AREA);
   if AREA > 360 then
      ShowMessage('Terreno espaçoso');
end;
```

Explicando

Inicialmente, declaramos as variáveis A, B e AREA como sendo do tipo real, pois, as mesmas poderão receber valores fracionários. Em seguida, é solicitada a leitura dos dois lados do terreno, os quais foram atribuídos às variáveis A e B, respectivamente. Conforme explicado anteriormente, observe que é feita a conversão de tipos de dados. Posteriormente, realizamos o cálculo da área

do terreno o qual é atribuído à variável AREA e, logo após, o apresentamos no formulário por meio de um componente do tipo `TLabel`, sendo este, o `Label4`. Veja que, novamente, foi necessária a conversão de tipos, só que agora, a mesma é feita de maneira contrária à inicial, ou seja, de real para texto, pois, o resultado deverá ser guardado na propriedade `Caption`. Por último, através da estrutura condicional simples, verificamos se é possível exibir a mensagem desejada, caso a área seja maior que 360. Portanto, só será impresso a mensagem "Terreno espaçoso" se a condição for verdadeira, ou seja, se AREA > 360.

Para concluir esse programa, vamos inserir no formulário dois novos componentes do tipo `TButton`, colocando-os ao lado do botão Calcular. Em seguida, altere a propriedade `Caption` desses botões para Limpar e Sair, respectivamente, e no evento `OnClick` de cada um deles, insira os seguintes códigos: (Dê um duplo clique no botão Limpar).

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  Edit1.Text:= ' ';
  Edit2.Text:= ' ';
  Label4.Caption:= ' ';
  Edit1.SetFocus;
end;
```

Agora, dê um duplo clique no botão Sair:

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  close;
end;
```

Grave novamente o seu projeto (**File > Save All**) e, em seguida, execute (tecla **F9**) o programa para ver seu funcionamento.

8.1.2 Estrutura condicional composta

Esta estrutura tem por objetivo analisar o resultado de uma expressão lógica, de tal forma que, caso o valor seja verdadeiro, será executado a instrução que se encontra após a opção `then` e antes do `else`; caso contrário, executa-se a outra instrução, a qual se situa após a opção `else`. A sintaxe desta estrutura é apresentada a seguir.

Sintaxe

```
if <expressão lógica>
  then <instrução>
  else <outra instrução>;
```

Num comando `if`, a instrução que precede o `else` não possui ponto-e-vírgula ao final, pois, o `else` faz parte do comando `if`, sendo assim, este comando só poderá ser finalizado após a execução da instrução `else`.

Caso seja necessário utilizar mais de uma instrução quando o resultado da expressão lógica for verdadeiro ou, então, falso, devemos colocá-las num bloco de comandos, o qual é delimitado pelas palavras `begin ... end`.

Veja como fica a nova sintaxe:

Sintaxe

```
if <expressão lógica> then
  begin
    <instrução1>;
    <instrução2>;
         :
    <instruçãoN>;
  end
else
  begin
    <instrução1>;
    <instrução2>;
         :
    <instruçãoN>;
  end;
```

8.1.2.1 Aplicação

Fazer um projeto em Lazarus que leia a altura (em metros) e o sexo (masculino ou feminino) de uma pessoa. Com base nesses dados, calcular e imprimir o peso ideal desse indivíduo, de acordo com a seguinte fórmula:

- Para homens o peso ideal é: 72 × ALTURA – 60;
- Para mulheres o peso ideal é: 62 × ALTURA – 50.

Solução

a) Dados do problema:

- Valor da altura (em metros).
- Sexo (Masculino ou Feminino).

b) O que se pede:

- Peso ideal de uma pessoa.

c) Elaboração do algoritmo ou do fluxograma.

d) Elaboração da interface gráfica.

Os componentes que formam essa interface são apresentados na Figura 8.3.

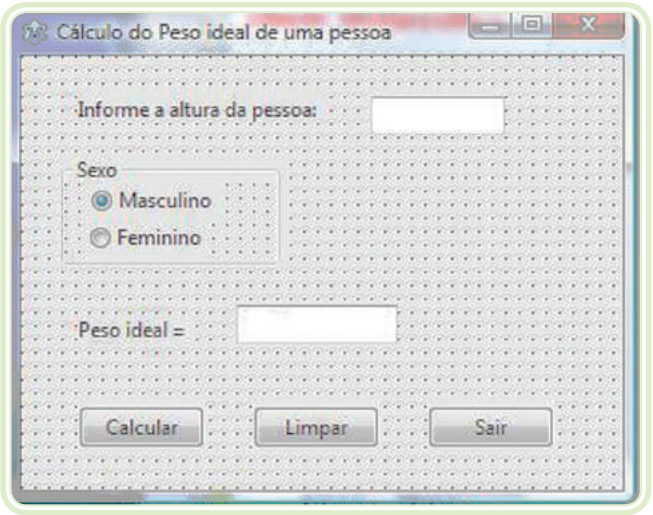

Figura 8.3: Formulário do cálculo do peso ideal
Fonte: *Software* Lazarus

Veja que nesta aplicação são usados dois novos tipos de componentes: `TGroupBox` e `TRadioButton`, ambos da aba Standard. A utilização desses componentes na interface é feita da seguinte maneira: insira o componente do tipo `TGroupBox` no formulário e altere o seu `Caption` para "Sexo"; em seguida, insira dentro deste componente `GroupBox1`, dois componentes do tipo `TRadioButton`, sendo estes representados por `RadioButton1` e `RadioButton2`, respectivamente. Altere a propriedade `Caption` do `RadioButton1` para "Masculino" e do `RadioButton2` para "Feminino". Selecione novamente, o componente `RadioButton1` e altere a propriedade `Checked` para `True`. Esta alteração tem por objetivo deixar previamente marcada a opção "Masculino" para o sexo de uma pessoa, quando a aplicação for executada. Os demais componentes do formulário deverão ser configurados da maneira como já foi apresentada em exercícios anteriores.

Depois de implementada a interface, podemos ainda, assim, alterar a propriedade `ReadOnly` do componente `Edit2`, para o valor `True`, de maneira que este componente só exibirá o resultado, bloqueando qualquer modificação no mesmo.

Por fim, grave o projeto numa nova pasta (**File > Save All**).

e) Código do projeto.

Agora, vamos programar no evento `OnClick` do botão Calcular. Para isso, dê um duplo clique no botão Calcular e digite dentro do procedimento o código a seguir:

```
procedure TForm1.Button1Click(Sender: TObject);
var
   ALTURA, PESOID: real;
begin
   ALTURA:= StrToFloat(Edit1.Text);
   if RadioButton1.Checked then
      PESOID:= 72 * ALTURA - 60
   else PESOID:= 62 * ALTURA - 50;
   Edit2.Text:= FloatToStr(PESOID);
end;
```

Explicando

Inicialmente, declaramos as variáveis ALTURA e PESOID como sendo do tipo real. Em seguida, é solicitada a leitura da altura da pessoa, sendo seu valor atribuído à variável ALTURA. Observe que, nesta aplicação não foi necessária a criação de uma variável para se guardar o sexo da pessoa, pois, a atribuição desta informação será feita mediante escolha de uma opção dentro do componente `GroupBox1`, ou seja, quando o usuário clicar no valor desejado para o sexo, será atribuído o valor `True` à propriedade `Checked` do `RadioButton` escolhido. Baseado nesta escolha, através de uma estrutura condicional composta, verificamos se o valor contido na propriedade `Checked` do `RadioButton1` é verdadeiro (`True`). Caso seja, aplicamos a fórmula para o peso ideal masculino; caso contrário, calculamos o peso ideal feminino. Em seguida, exibimos através do componente `Edit2`, o resultado encontrado.

Outra forma de se usar a estrutura condicional neste projeto é apresentada a seguir. Entretanto, observe que se trata apenas de um exemplo para ilustrar o uso do bloco de comandos delimitado por "`begin ... end`" tanto no then quanto no else, já que a inserção do mesmo comando ("`Edit2.Text:= FloatToStr(PESOID);`") em dois blocos mutuamente excludentes é um contrassenso, ou seja, não é uma boa prática.

```
procedure TForm1.Button1Click(Sender: TObject);
var
   ALTURA, PESOID: real;
begin
   ALTURA:= StrToFloat(Edit1.Text);
   if RadioButton1.Checked then
      begin
         PESOID:= 72 * ALTURA - 60;
         Edit2.Text:= FloatToStr(PESOID);
      end
   else
      begin
         PESOID:= 62 * ALTURA - 50;
         Edit2.Text:= FloatToStr(PESOID);
      end;
end;
```

Como atividade complementar, programe os códigos dos botões Limpar e Sair. Por último, grave o projeto novamente e execute a aplicação para testar seu funcionamento.

8.1.3 Estrutura condicional encadeada

Esse tipo de estrutura é necessário quando se deseja tomar uma decisão (caminho), mediante uma série de possibilidades apresentadas num problema. É representada através de uma estrutura condicional que se encontra contida dentro de outra. Essa estrutura pode estar localizada internamente ao `then` (então) ou ao `else` (senão). Por existirem várias maneiras de se aplicar esta estrutura, faremos a seguir uma aplicação para uma melhor compreensão da mesma.

8.1.3.1 Aplicação

Fazer um projeto em Lazarus para ler o valor total que um cliente pagou no abastecimento de seu veículo e a quantidade de litros abastecida. Em seguida, calcular e imprimir o valor do litro da gasolina. Caso o valor seja inferior a R$ 2,00, escrever a seguinte mensagem: "Combustível muito barato". Se o valor estiver compreendido entre R$ 2,00 e R$ 2,40, escrever "Promoção" e caso seja maior que R$ 2,40, escrever "Sem Promoção". A Figura 8.4 ilustra a interface gráfica desse projeto.

Observação

a) Utilizar a propriedade `ReadOnly` para proteger os conteúdos de `Edit3` e `Edit4`.

b) Neste projeto será utilizado um novo tipo de componente – `TlistBox` que se encontra na aba Standard. Esse objeto permite apresentar informações dentro de uma caixa de listagem. Para este projeto, o componente será usado para exibir o valor do litro de gasolina calculado em cada operação realizada pelo usuário, desde que este clique no botão ADICIONAR.

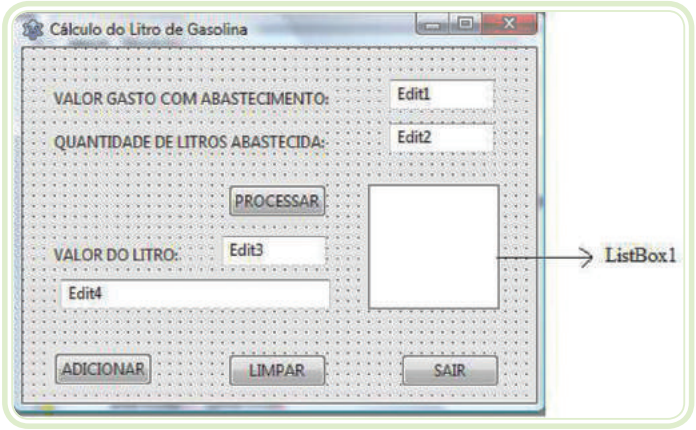

Figura 8.4: Formulário cálculo do valor do litro de gasolina
Fonte: *Software* Lazarus

Solução

a) Dados do problema:

- Valor pago no abastecimento de um veículo (em R$).
- Quantidade de litros abastecida.

b) O que se pede:

- Valor do litro da gasolina.
- Mensagem, de acordo com os seguintes critérios:
  - Valor do litro < 2,00: "Combustível muito barato".
  - 2,00 ≤ valor do litro ≤ 2,40: "Promoção".
  - Valor do litro > 2,40: "Sem Promoção".

c) Elaboração do algoritmo ou do fluxograma.

d) Elaboração da interface gráfica.

Para introduzir o componente `Listbox1` dentro do formulário, basta localizá-lo na aba Standard, em seguida selecioná-lo e depois inseri-lo no local desejado dentro do formulário. Os demais componentes deverão ser configurados da maneira como já foi apresentada em exercícios anteriores.

e) Código do projeto.

Após a elaboração do formulário do projeto, salve a aplicação numa nova pasta. Em seguida, dê um duplo clique no botão PROCESSAR e digite o código a seguir, dentro do procedimento.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  VP, QL, PL: real;
begin
  VP:= StrToFloat(Edit1.Text);
  QL:= StrToFloat(Edit2.Text);
  PL:= VP/QL;
  Edit3.Text:= FloatToStr(PL);
  if PL < 2.00 then
    Edit4.Text:= 'Combustível muito barato'
  else if (PL >= 2.00) and (PL <= 2.40) then
         Edit4.Text:= 'Promoção'
       else Edit4.Text:= 'Sem Promoção';
end;
```

Explicando

Inicialmente, declaramos as variáveis de entrada VP e QL, bem como a variável de saída PL, todas como sendo do tipo real. Em seguida, é solicitada a leitura do valor gasto no abastecimento do veículo e da quantidade de litros abastecida no mesmo, sendo os valores atribuídos às variáveis VP e QL, respectivamente. Logo após, é feito o cálculo do preço do litro de gasolina, sendo seu valor atribuído à variável PL. Na sequência, o resultado é exibido no formulário através do componente `Edit3`. Posteriormente, utilizamos a estrutura condicional encadeada para verificar dentre os diferentes critérios apresentados, qual deles torna o resultado da expressão lógica verdadeiro. A partir do momento no qual esse resultado é encontrado, o comando correspondente a essa condição é executado, ou seja, a mensagem é atribuída ao componente `Edit4` e a estrutura condicional encadeada é finalizada.

Veja que, nesta aplicação, o uso do operador `and` na expressão lógica:

`(PL>= 2.00) and (PL <= 2.40)`, faz com que o resultado da mesma somente seja verdadeiro, se ambos os testes forem verdadeiros.

- Código do botão ADICIONAR (evento `OnClick`):

```
procedure TForm1.Button2Click(Sender: TObject);
begin
   ListBox1.Items.Add(Edit3.Text);
end;
```

Explicando

O comando desta procedure tem por finalidade adicionar no `ListBox1` uma string (texto), através do método `Add` da propriedade `Items`.

A seguir, são apresentadas as instruções do botão LIMPAR associadas ao evento `OnClick`.

```
procedure TForm1.Button3Click(Sender: TObject);
begin
   Edit1.Clear;
   Edit2.Clear;
   Edit3.Clear;
   Edit4.Clear;
   ListBox1.Clear;
   Edit1.SetFocus;
end;
```

Explicando

O método `Clear` do componente do tipo `TEdit`, tem a finalidade de limpar o valor contido na propriedade `Text` deste objeto, de maneira semelhante ao comando `Editx.Text :=` ' '; que nós já utilizamos em exercícios anteriores. O uso do método `Clear` junto ao `ListBox1`, também, limpa o conteúdo da caixa de listagem.

Observação

O x indica o número de um determinado componente `Edit`.

Agora, programe o código do botão SAIR. Em seguida, grave o projeto (**File > Save All**), execute a aplicação e faça uma simulação de todos os recursos disponíveis na mesma.

8.1.4 Estrutura seletiva case ... of

Esta estrutura nos permite selecionar uma opção dentre as várias que se apresentam, baseado no valor de uma expressão ou variável, ou seja, caso o valor da expressão ou variável corresponda a algum dos valores disponíveis dentro da estrutura, o bloco de comandos referente a esse valor (opção)

será executado e, logo em seguida, o comando finalizado; caso contrário, se nenhuma das opções disponíveis for igual ao valor da expressão ou variável do `case` e, caso a estrutura possua a opção `else`, pois, esta é opcional, a mesma será executada.

Sintaxe

```
case <expressão ou variável> of
  <opção 1>: <instrução1>;
  <opção 2>: <instrução2>;
  <opção 3>: <instrução3>;
       :            :
  <opção N>: <instruçãoN>;
  else
    <outra instrução>;
end;
```

Explicando

- A expressão ou variável do comando `case`, tem de resultar, obrigatoriamente, num **valor ordinal**, normalmente do tipo Char ou Integer. O resultado jamais poderá ser do tipo real.

- Caso uma determinada opção do comando `case` possua mais de uma instrução, estas deverão ser colocadas dentro de um bloco: `begin ... end`.

- O comando `case` se assemelha à estrutura condicional encadeada, porém, se apresenta de uma maneira mais simples e eficiente.

- As opções (opção 1, opção 2, ..., opção N) contidas na sintaxe, não necessariamente devem ser definidas como um valor, mas podem especificar intervalos. Veja o exemplo:

```
case key of
  '0'..'9': s:= 'dígito';
  else s:= 'outro caractere qualquer';
end;
```

A-Z

valor ordinal
Os tipos ordinais são representados por conjuntos ordenados de valores ou que estejam em sequência.

8.1.4.1 Aplicação

Com base no Quadro 8.2, escreva um projeto em Lazarus que leia o código de um sanduíche e a quantidade comprada do mesmo. Em seguida, calcule e mostre o valor a pagar. Apresentar, também, num componente do tipo `TListBox`, o nome do sanduíche e o valor a pagar.

Quadro 8.2: Dados referentes à aplicação da estrutura case ... of		
Código	**Nome**	**Preço unitário (R$)**
1	Hambúrguer	5.00
2	X-Picanha	9.90
3	X-Salada	8.00
4	Outros	7.50

Fonte: Autores

Solução

a) Dados do problema:

- Código do sanduíche e quantidade comprada que deverão ser lidos.
- Tabela contendo nome e preço unitário do sanduíche que deverá ser utilizada no programa.

b) O que se pede:

- Valor a pagar na compra de um determinado tipo de sanduíche.

c) Elaboração do algoritmo ou do fluxograma.

d) Elaboração da interface gráfica.

Os componentes do formulário desta aplicação deverão ser configurados da maneira como já foi apresentada em exercícios anteriores. A Figura 8.5 ilustra esse formulário.

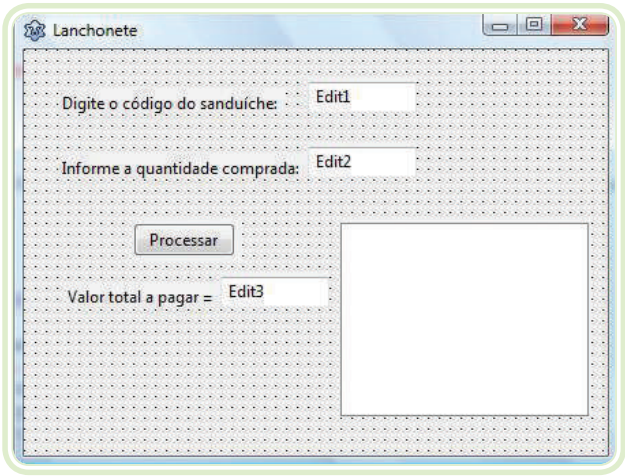

Figura 8.5: Formulário aplicação lanchonete
Fonte: *Software* Lazarus

Após a elaboração do formulário, salve a aplicação numa nova pasta.

e) Código do projeto.

Explicando

Na resolução desta aplicação, devemos verificar qual é o código do sanduíche informado pelo usuário para que possamos calcular o preço a pagar, tendo em vista que cada sanduíche tem um preço diferente. Como existem vários códigos, temos que fazer várias comparações, de tal forma que para cada um deles seja realizado um cálculo. Dessa forma, podemos utilizar uma estrutura condicional encadeada ou aplicarmos o comando `case ... of`, pois, o valor que representa o código do sanduíche é do tipo inteiro.

Para que você possa comparar essas duas estruturas, serão apresentadas duas soluções, sendo a primeira com o uso da estrutura condicional encadeada e a segunda com o comando `case ... of`. Entretanto, no seu projeto, desenvolva apenas o código referente à segunda solução, pois contém a estrutura que estamos aprendendo no momento.

1ª solução

```
procedure TForm1.Button1Click(Sender: TObject);
var
  COD,QUANT: byte;
  VP: real;
  NOME: string;
begin
  COD:= StrToInt(Edit1.Text);
  QUANT:= StrToInt(Edit2.Text);
  if COD = 1 then
    begin
      VP:= QUANT * 5.00;
      NOME:= 'Hambúrguer';
    end
  else
    if COD = 2 then
      begin
        VP:= QUANT * 9.90;
        NOME:= 'X-Picanha';
      end
    else
      if COD = 3 then
        begin
          VP:= QUANT * 8.00;
          NOME:= 'X-Salada';
        end
```

Segundo Santos (1998), são apresentadas algumas funções de conversão de tipos diferentes de dados:

StrToInt (Valor-Texto): devolve o valor em inteiro, correspondente ao texto passado como parâmetro.

```
        else
          if COD = 4 then
            begin
              VP:= QUANT * 7.50;
              NOME:= 'Outros';
            end
          else
            begin
              ShowMessage('Código do sanduíche inválido');
              Edit1.SetFocus;
              Exit;
            end;
    Edit3.Text:= FloatToStr(VP);
    ListBox1.Items.Add(NOME + ' R$ ' + Edit3.Text);
    Edit1.SetFocus;
end;
```

2ª solução

```
procedure TForm1.Button1Click(Sender: TObject);
var
  COD,QUANT: byte;
  VP: real;
  NOME: string;
begin
  COD:= StrToInt(Edit1.Text);
  QUANT:= StrToInt(Edit2.Text);
  case COD of
    1: begin
         VP:= QUANT * 5.00;
         NOME:= 'Hambúrguer';
       end;
    2: begin
         VP:= QUANT * 9.90;
         NOME:= 'X-Picanha';
       end;
    3: begin
         VP:= QUANT * 8.00;
         NOME:= 'X-Salada';
       end;
```

```
        4: begin
             VP:= QUANT * 7.50;
             NOME:= 'Outros';
           end;
        else
           begin
             ShowMessage('Código do sanduíche inválido');
             Edit1.SetFocus;
             Exit;
           end;
      end;
      Edit3.Text:= FloatToStr(VP);
      ListBox1.Items.Add(NOME + ' R$ ' + Edit3.Text);
      Edit1.SetFocus;
  end;
```

Explicando

Nesta solução não se criou uma variável para se guardar o preço de determinado sanduíche tendo em vista que ele se encontra fixo para cada código apresentado no Quadro 8.2 no enunciado do problema.

Foi necessária a criação da variável nome para que se pudesse exibi-lo no componente `Listbox1`. Observe, também, que o uso excessivo de instruções encadeadas na primeira solução torna o entendimento do problema um pouco confuso.

Tendo em vista que ambas as soluções buscam resolver o mesmo problema, a segunda solução, com o uso do comando `case`, ficou mais simples de se entender, pois as várias comparações existentes na primeira solução foram substituídas neste comando pelos possíveis valores que o código de um sanduíche pode ter. Desta forma, o valor informado pelo usuário referente ao código, que é armazenado na variável `COD`, é comparado com os diversos valores discriminados na estrutura, de tal forma que se houver algum valor que satisfaça a igualdade, as instruções referentes à mesma serão executadas e o comando finalizado.

Outro ponto a ser observado é quanto ao uso da opção `else`. Assim sendo, caso não seja informado um código válido para o sanduíche, as instruções desse bloco serão executadas. O uso do `exit` ao final dessa cláusula tem por objetivo abandonar a execução do procedimento naquele ponto.

Ainda sobre a cláusula `else`, veja que é permitido o uso do ponto e vírgula (;) antes do mesmo, no comando `case ... of`, diferentemente do que ocorre no comando `if ... then ... else`.

Por último, veja que a instrução que faz uso do componente `ListBox1` visa concatenar (agrupar) dentro do componente três strings que são: o nome do sanduíche, o símbolo monetário R$ e o valor armazenado na propriedade `Text` do `Edit3` que é o valor a pagar. Portanto, lembre-se que para concatenar strings no Free Pascal, deve-se usar o sinal de "+".

Resumo

Nesta aula, você aprendeu como usar as estruturas condicionais dentro do IDE do Lazarus. Agora, veja que, com o uso dessas estruturas, conseguimos fazer programas mais interessantes, não é mesmo? Entretanto, não iremos parar por aqui, pois ainda teremos que trabalhar com as demais estruturas de controle abordadas em algoritmos. Dessa forma, caso você tenha alguma dúvida nessa lição, refaça-a, pois precisaremos muito desses comandos nas próximas lições, certo? Então, vamos adiante?

Atividades de aprendizagem

1. Fazer um projeto em Lazarus para ler o peso (kg) e a altura (m) de uma pessoa e determinar se o indivíduo está com um peso favorável (situação). Essa situação é determinada através do cálculo do IMC (Índice de Massa Corpórea), que é definida como sendo a relação entre o peso e o quadrado da altura do indivíduo. Ou seja,

$$IMC = \frac{PESO}{ALTURA^2}$$

A situação do peso é determinada a seguir:

Condição	Situação
IMC abaixo de 20	Abaixo do peso
IMC de 20 até 25	Peso Normal
IMC acima de 25	Acima do peso

Observação

Desenvolver o *layout* do formulário usando sua criatividade.

2. Fazer um projeto em Lazarus para ler o ano e o mês de nascimento (1 a 12) de uma pessoa, bem como o mês atual. Caso o mês atual seja igual ao mês de nascimento, escrever a mensagem: "mês referente a seu aniversário", caso contrário, escrever a mensagem "não é o mês do seu aniversário". Calcular e mostrar a idade dessa pessoa levando-se em conta apenas o mês e o ano.

3. Fazer um projeto em Lazarus que leia um valor numérico inteiro, o qual representa um determinado dia da semana (1 – domingo, 2 – segunda-feira, ... , 7 – sábado). Baseado neste valor, apresentar por extenso, qual é o nome correspondente a esse dia. Caso o valor informado não represente nenhum dos dias da semana, apresentar a mensagem: "valor inválido".

Observação

No exercício 3, utilizar o comando `case ... of` usando sua criatividade.

Aula 9 – Estruturas de repetição

Objetivos

Entender como funcionam as estruturas de repetição no ambiente de programação visual (Lazarus).

Elaborar programas utilizando estruturas de repetição.

9.1 Repetição

Nesta aula serão apresentadas as estruturas de repetição abordadas em algoritmos e que são utilizadas no IDE do Lazarus, ou seja, suportadas pelo compilador Free Pascal. É importante observar que, o entendimento do funcionamento das estruturas apresentadas nesta aula é o mesmo do que foi aprendido em algoritmos. Sendo assim, o que se diferencia é apenas a sintaxe, pois essas estruturas devem ser escritas da maneira que o compilador as reconheça.

Segundo Santos (1998), são apresentadas algumas funções de conversão de tipos diferentes de dados:

DateToStr (Valor-Data): devolve a string correspondente ao valor da data passada como parâmetro.

StrToDate (Valor-Texto): retorna uma data correspondente ao texto passado como parâmetro.

9.2 Estrutura while ... do (enquanto ... faça)

Nesta estrutura, os comandos internos à mesma serão executados repetidamente, enquanto a condição permanecer verdadeira, sendo que esses comandos devem ser colocadas num bloco definido com as instruções `begin ... end`. Veja como fica a sintaxe dessa estrutura.

Sintaxe

```
while <condição> do
  begin
    <instrução1>;
    <instrução2>;
    <instruçãoN>;
 end;
```

Explicando

Nesta estrutura de repetição, o uso do "`begin ... end`" será necessário apenas, quando houver mais de um comando interno ao laço.

9.2.1 Aplicação

Desenvolver um projeto em Lazarus que apresente num componente do tipo `TListBox`, os números inteiros que sejam pares e que estejam compreendidos entre 0 e 30. Calcular e imprimir, também, a soma desses números pares.

Observação

Para que um número inteiro seja par, o resto da divisão inteira desse número por dois, tem de resultar zero.

Solução

a) Dados do problema:

- Valores inteiros situados na faixa de 0 a 30.

b) O que se pede:

- Exibir cada um dos valores pares contidos no intervalo fornecido.
- Calcular o somatório dos números pares desse intervalo.

c) Elaboração do algoritmo ou do fluxograma.

d) Elaboração da interface gráfica.

Na elaboração do formulário desta aplicação não é necessário usar componentes para receber os dados de entrada, pois esses valores já foram definidos a partir do enunciado do problema, ou seja, são valores numéricos inteiros que vão de zero a trinta. Sendo assim, logo após iniciar um novo projeto (**Project > New Project > Application**), mude a propriedade `Caption` do formulário para: "Números Pares". Adicione na parte superior do formulário três componentes do tipo `TButton`, sendo um ao lado do outro. Em seguida, altere a propriedade `Caption` de cada um deles para: "Processar", "Limpar" e "Sair", respectivamente. Abaixo desses `Buttons`, insira um componente do tipo `TGroupBox`, o qual é representado por `GroupBox1` e altere sua propriedade `Caption` para: "Respostas". A seguir, insira dentro deste componente: um `Label1`, de forma a ter sua propriedade `Caption` alterada para: "Números Pares na faixa de 0 a 30:"; abaixo deste, um `ListBox1`; e, abaixo do `ListBox1`, um `Label2`, o qual terá sua propriedade `Caption` alterada para: "Somatório dos números pares no intervalo de 0 a 30 = " e, por último, à direita deste, um `Edit1`. A Figura 9.1 ilustra esse formulário.

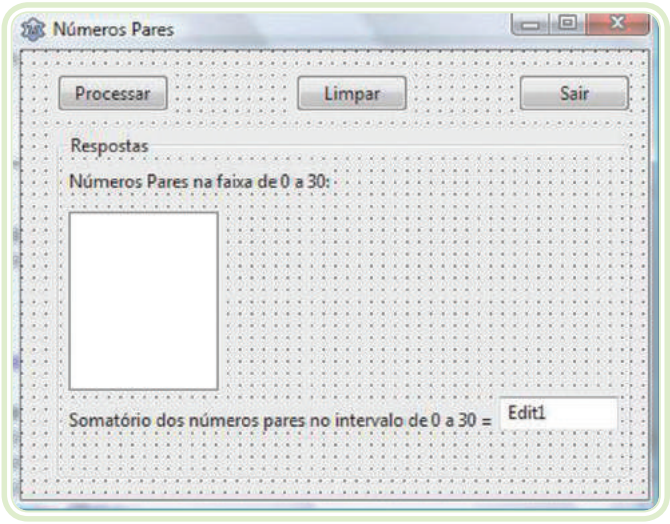

Figura 9.1: Formulário números pares
Fonte: *Software* Lazarus

e) Código do projeto.

Após a elaboração do formulário do projeto, salve a aplicação numa nova pasta. Em seguida, dê um duplo clique no botão Processar e digite o código dentro do procedimento.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I: integer;
  SOMA: integer;
begin
  I:= 0;
  SOMA:= 0;
  while I <= 30 do
    begin
      if (I MOD 2) = 0 then
        begin
            ListBox1.Items.Add(IntToStr(I));
            SOMA:= SOMA + I;
        end;
      I:= I + 1;
    end;
  Edit1.Text:= IntToStr(SOMA);
end;
```

Explicando

Primeiramente são declaradas as variáveis I e SOMA que serão utilizadas no programa. Em seguida, inicializamos cada uma dessas variáveis com o valor

zero. Nesta solução é necessário percorrer o intervalo fornecido de maneira a verificar se cada número contido nesse intervalo é par. Para isso fazemos uso do comando de repetição (`while ... do`), o qual irá executar trinta e uma vezes as instruções internas a ele, pois o intervalo deverá variar de 0 a 30. No final, quando a condição I <= 30 não for mais satisfeita, ou seja, quando a variável I atingir o valor 31, abandona-se o comando de repetição e passa-se para a próxima instrução que é a de exibir através de um componente `Edit1` o valor da soma de todos os números pares desse intervalo.

Sendo assim, este programa irá a cada laço de repetição, por meio de uma estrutura condicional simples, verificar se o valor da variável I, que representa um número desse intervalo, é par. Essa verificação é feita com o uso do operador MOD, o qual devolve o resultado do resto da divisão inteira do valor da variável I por 2. Dessa forma, caso a condição seja verdadeira, ou seja, o resultado do resto seja igual a zero, o valor de I é exibido no `ListBox1`, pois o mesmo é par. Na sequência, o valor de I é somado ao valor que já se encontrava em SOMA, resultando num novo valor para SOMA, de maneira que, ao final da estrutura de repetição, teremos o resultado acumulado de todos os números pares nesta variável.

Por fim, programe os códigos dos botões Limpar e Sair e, em seguida, salve o projeto novamente.

9.3 Estrutura for ... do (para ... faça)

Esta estrutura faz uso de uma variável de controle, de tal forma a executar um número determinado de repetições (iterações).

Sendo assim, este comando só poderá ser utilizado quando se conhece o valor inicial e final da variável de controle. Esta variável de controle será incrementada ou decrementada de um em um, automaticamente. A sintaxe dessa estrutura é apresentada a seguir.

Sintaxe

```
for <variável>:= <valor inicial> to (downto) <valor final> do
  begin
    <instrução1>;
    <instrução2>;
    <instruçãoN>;
  end;
```

Explicando

É bom observar que o comando `for ... do` pode ser utilizado tanto com o valor inicial da variável de controle menor que o valor final como o contrário, ou seja, com o valor inicial da variável de controle maior que o valor final desta variável. Para a primeira situação a sintaxe é a seguinte: `for ... to ... do`. Para a segunda, a sintaxe é: `for ... downto ... do`.

De maneira semelhante à estrutura de repetição anterior (`while ... do`), o uso do "`begin ... end`" será necessário apenas, quando houver mais de um comando interno ao laço.

9.3.1 Aplicação

Elaborar um projeto em Lazarus que calcule e imprima o resultado do fatorial de um número inteiro positivo. O fatorial de um número N pode ser entendido como sendo multiplicações sucessivas a partir do número 1 até o número para o qual se deseja calcular seu fatorial, sendo o fatorial desse número expresso por N!.

Exemplo

6! = 1 × 2 × 3 × 4 × 5 × 6 = N!

Observação

a) Não se calcula fatorial para números negativos.

b) 0! = 1.

Solução

a) Dados do problema:

- Número inteiro positivo.

b) O que se pede:

- Calcular o fatorial do número fornecido.

c) Elaboração do algoritmo ou do fluxograma.

d) Elaboração da interface gráfica.

Abra um novo projeto (**Project > New Project > Application**), mude a propriedade `Caption` do formulário para: "Fatorial de um número". Adicione na parte superior do formulário um `Label1` e altere sua propriedade `Caption` para: "Fatorial de qual número?". À direita deste `Label`, insira um

componente `Edit1`. Abaixo desses dois componentes insira um `Button1` e altere sua propriedade `Caption` para: "Processar". A seguir, insira abaixo desse `Button` um `Label2`, de forma a ter sua propriedade `Caption` alterada para: "Fatorial ="; e, por último, à direita deste `Label`, insira um `Edit2`. A Figura 9.2 ilustra esse formulário.

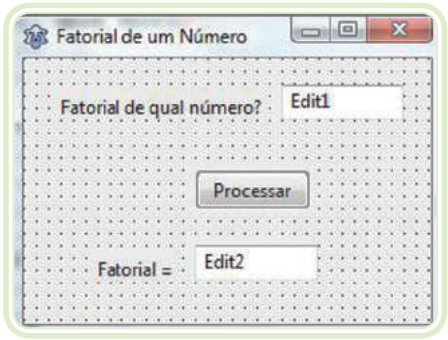

Figura 9.2: Formulário fatorial de um número
Fonte: *Software* Lazarus

e) Código do projeto.

Após a elaboração da interface do projeto, salve a aplicação numa nova pasta. Em seguida, dê um duplo clique no botão Processar e digite o código dentro do procedimento.

```
procedure TForm1.Button1Click(Sender: TObject);
var
   I, N, FAT: integer;
begin
   N:= StrToInt(Edit1.Text);
   FAT:= 1;
   for I:= 1 to N do
      begin
         FAT:= FAT * I;
      end;
   Edit2.Text:= IntToStr(FAT);
   Edit1.SetFocus;
end;
```

Explicando

Primeiramente são declaradas as variáveis I, N e FAT que serão utilizadas no programa. Em seguida, é solicitado o dado de entrada para o qual se deseja calcular seu fatorial. A partir daí, inicializamos a variável FAT com o valor um, pois a mesma terá por objetivo, acumular as multiplicações sucessivas de 1 até o valor de N e sendo assim, não poderá ser inicializada com zero.

Na sequência, fazemos uso da estrutura de repetição (`for ... do`) que terá a quantidade de iterações de acordo com o valor informado para o fatorial desejado, ou seja, se o número para o qual se deseja calcular o fatorial for cinco, a estrutura de repetição executará os comandos internos, à mesma, cinco vezes; se for dez, os comandos serão executados dez vezes e assim por diante. Dessa forma, este programa irá a cada laço de repetição, acumular na variável FAT o produto da variável I pelo valor já armazenado anteriormente em FAT, resultando num novo valor para FAT e, assim sucessivamente, até que o valor da variável de controle I exceda o valor da variável N. Por fim, apresentamos o valor encontrado para o fatorial através do componente `Edit2`.

9.4 Estrutura repeat ... until (repita ... até)

Diferentemente da estrutura de repetição `while ... do`, o teste da condição na estrutura `repeat ... until` é realizada apenas ao final da repetição, o que garante que o bloco de comandos internos a essa estrutura seja executado ao menos uma vez. Outro ponto a ser lembrado é que os comandos contidos nessa estrutura são executados enquanto a condição permanecer "falsa". Veja a seguir como se apresenta a sintaxe para essa estrutura.

Sintaxe

```
repeat
  <instrução1>;
  <instrução2>;
  <instruçãoN>;
until <condição>;
```

9.4.1 Aplicação

Fazer um projeto em Lazarus que mostre a série de Fibonacci até o vigésimo termo da série. A sequência desta série é formada da seguinte maneira: 1, 1, 2, 3, 5, 8, 13, 21, ... Por definição, essa série apresenta os dois primeiros termos com valor igual a 1 (um) e os demais são calculados como a soma dos seus dois antecessores (adaptado de GUIMARÃES; LAGES, 1994).

Solução

a) Dados do problema: ————————.

b) O que se pede:

- Gerar a série de Fibonacci até o vigésimo termo.

c) Elaboração do algoritmo ou do fluxograma.

d) Elaboração da interface gráfica.

Abra um novo projeto (**Project > New Project > Application**), mude a propriedade `Caption` do formulário para: "Série de Fibonacci". Adicione na parte superior do formulário um `Label1` e altere sua propriedade `Caption` para: "Fibonacci até o vigésimo termo:". Abaixo deste `Label`, insira um componente `ListBox1`. Em seguida, adicione dois componentes do tipo `TButton`, identificados por `Button1` e `Button2`, respectivamente. A partir daí, altere a propriedade `Caption` do primeiro `Button` para: "Processar" e do segundo para: "Limpar". A Figura 9.3 ilustra esse formulário.

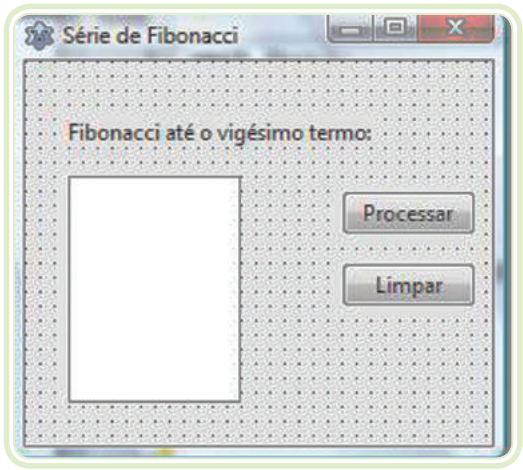

Figura 9.3: Formulário série de Fibonacci
Fonte: *Software* Lazarus

e) Código do projeto.

Após a elaboração da interface do projeto, salve a aplicação numa nova pasta. Em seguida, dê um duplo clique no botão Processar e digite o código dentro do procedimento.

```
procedure TForm1.Button1Click(Sender: TObject);
var
   I, T1, T2, ATUAL: integer;
begin
   ListBox1.clear;
   T1:= 1;
   T2:= 1;
   ListBox1.Items.Add(IntToStr(T1));
   ListBox1.Items.Add(IntToStr(T2));
   I:= 3;
```

```
    repeat
      ATUAL:= T1 + T2;
      ListBox1.Items.Add(IntToStr(ATUAL));
      T1:= T2;
      T2:= ATUAL;
      I:= I + 1;
    until I > 20;
end;
```

Explicando

A lógica para se gerar os termos da série de Fibonacci consiste no fato de que, um determinado termo a partir do terceiro (inclusive) é obtido através da soma dos seus dois antecessores. Por isso, após declararmos as variáveis do programa, inicializamos as variáveis T1 e T2 com os dois primeiros termos da série e os exibimos no `ListBox1`. Em seguida, tendo em vista que esses dois primeiros termos já foram mostrados, inicializamos a variável I com o valor três para que, a partir daí, seja feito o controle do próximo termo da série, pois, neste problema queremos gerar a série até o vigésimo termo. Na sequência, fazemos uso da estrutura de repetição (`repeat ... until`) de maneira a guardarmos na variável ATUAL a soma dos dois termos anteriores e o exibimos, em seguida, no `ListBox1`. Ainda dentro dessa estrutura, atualizamos as variáveis da seguinte maneira: T1 recebendo o valor de T2 e T2 recebendo o valor da variável ATUAL, de forma que na sequência esses comandos sejam repetidos novamente, e a geração do próximo termo se dê corretamente. Por fim, a estrutura será abandonada quando o valor da condição I > 20 se tornar "verdadeiro".

Resumo

Vimos, nesta aula, como desenvolver aplicações gráficas no IDE do Lazarus, utilizando diferentes estruturas de repetição. De maneira que, a escolha de qual estrutura utilizar dependerá bastante do problema que se apresenta e da forma como se deseja apresentar sua solução.

Sendo assim, é muito importante que você tenha o completo domínio dessas estruturas na solução de um problema, para que possa aplicá-las da melhor maneira possível. Para auxiliá-lo neste objetivo, foram desenvolvidos nas aulas de algoritmos, bem como nesta aula, várias atividades e sugeridas outras. Portanto, caso seja necessário, leia novamente este capítulo e refaça as atividades.

Não podemos deixar de parabenizá-lo neste momento, pois, avançamos bastante em nossas atividades e você continua firme conosco! Assim, vamos em frente!

Atividades de aprendizagem

1. Fazer um projeto em Lazarus para ler um número inteiro e apresentar o resultado de sua tabuada.

2. Elaborar um projeto em Lazarus para calcular a soma dos números inteiros de 1 a 50.

3. Fazer um projeto em Lazarus que leia um número inteiro positivo e apresente as potências desse número, variando de 0 a 10.

Aula 10 – Procedimentos

Objetivos

Desenvolver programas no IDE do Lazarus utilizando estruturas modulares, elaboradas pelo próprio programador – `procedure`.

10.1 Estruturas modulares

Prezado estudante, nesta aula e na próxima iremos, através do IDE do Lazarus, desenvolver programas que se utilizam de procedimentos e funções que serão elaborados por nós mesmos (programadores), pois os procedimentos utilizados até aqui, no IDE do Lazarus, foram gerados por meio de eventos decorrentes do uso do próprio ambiente de programação, de maneira a apresentar rotinas específicas como parte da solução de um problema maior.

Tendo em vista que esse assunto já lhe foi apresentado em algoritmos e que, com certeza, encontra-se bem consolidado no seu aprendizado, buscaremos aqui, apresentar-lhe as principais sintaxes dessas rotinas suportadas pelo compilador Free Pascal, bem como resolver algumas atividades e propor outras, ao longo deste estudo.

10.2 Módulo procedimento

Conforme visto anteriormente, uma `procedure` é um bloco de programa, contendo início e fim e que é identificada por um nome, podendo ser chamada (acionada) a partir da `Unit` na qual foi declarada. Veja a sintaxe dessa estrutura.

Sintaxe

```
procedure <nome> [(parâmetros: tipos);]
[var variáveis locais: tipos;]
begin
   <instruções>;
end;
```

Na sintaxe apresentada, o colchete ([) informa que tanto os parâmetros quanto as variáveis locais não são obrigatórios, contudo, você verá no dia-a-dia que o uso de parâmetros é muito comum em procedimentos e funções.

Outro ponto a ser destacado é quanto à construção das `procedures`. No IDE do Lazarus, poderemos utilizar `procedures` com passagem de parâmetros por valor e por referência.

10.2.1 Exercícios resolvidos

1. Faça um projeto em Lazarus que leia dois valores inteiros, referentes aos catetos de um triângulo retângulo, calcule e imprima a hipotenusa desse triângulo. Utilizar passagem de parâmetros por valor para o cálculo da hipotenusa.

   Fórmula da hipotenusa: $hip = \sqrt{(b^2+c^2)}$

   Onde: b e c representam os catetos do triângulo retângulo.

Solução

a) Dados do problema:

- Os valores dos dois catetos (b e c).

b) O que se pede:

- Valor da hipotenusa (procedimento por valor).

c) Elaboração do algoritmo ou do fluxograma.

d) Elaboração da interface gráfica.

Abra um novo projeto (**Project > New Project > Application**) e desenvolva o formulário, conforme apresentado na Figura 10.1.

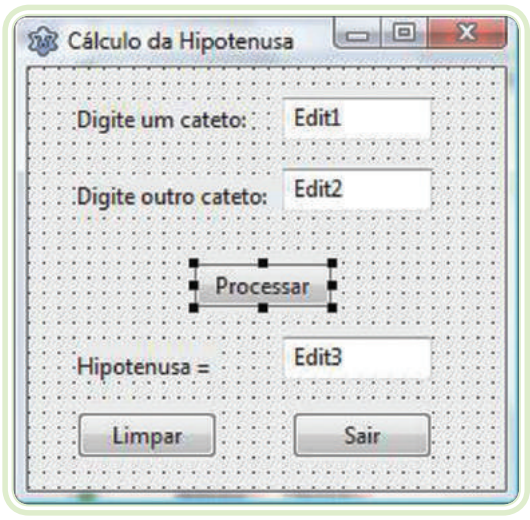

Figura 10.1: Formulário cálculo da hipotenusa
Fonte: *Software* Lazarus

e) Código do projeto.

Após a elaboração da interface do projeto, salve a aplicação numa nova pasta. Em seguida, dê um duplo clique no botão Processar e digite o código dentro da `procedure`.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  CAT1, CAT2: integer;
begin
  CAT1:= StrToInt(Edit1.Text);
  CAT2:= StrToInt(Edit2.Text);
  HIPOT(CAT1,CAT2);
  Edit1.SetFocus;
end;
```

Explicando

Nesta `procedure` foi inserida uma instrução que faz a chamada da sub-rotina HIPOT na qual são passados, através de parâmetros, os dois valores dos catetos informados pelo usuário. Assim, ao ser feito sua chamada, a mesma é executada e, logo após sua execução, o fluxo de processamento retorna para a linha subsequente à chamada dessa sub-rotina e o cursor será posicionado no primeiro `Edit` (`Edit1`) do formulário em função da execução do comando `Edit1.SetFocus`.

A seguir, é exibido o código completo da `procedure` referente à sub-rotina HIPOT. Observe que, a implementação dessa rotina deverá ser realizada antes da `procedure TForm1.Button1Click`; caso contrário o programa não a reconhecerá.

```
procedure HIPOT(B,C: integer);
var HIP: real;
begin
  HIP:= SQRT(SQR(B) + SQR(C));
  Form1.Edit3.Text:= FloatToStr(HIP);
end;
```

Explicando

Inicialmente, antes de passarmos à explicação desse tópico, observe o uso das funções: SQRT() e SQR(). A primeira tem por finalidade extrair a raiz quadrada do argumento da função, o qual é representado pela expressão ou valor que se encontra dentro dos parênteses; a segunda eleva ao quadrado o valor do argumento desta função. Como exemplo, veja: SQRT(16) = 4; SQR(2) = 4.

Voltando a nossa `procedure`, perceba que a mesma recebe por meio dos parâmetros locais B e C, os valores referentes às variáveis CAT1 e CAT2, os quais foram passados como parâmetros por valor, quando da chamada da rotina HIPOT.

Na `procedure`, criamos uma variável local para armazenar o cálculo da hipotenusa (HIP). Em seguida, fizemos esse cálculo e exibimos o resultado, através do componente `Edit3`. Veja que antes de usarmos o `Edit3`, foi preciso indicar o objeto `Form1`, no qual se encontra o referido componente. Isto se fez necessário, pelo fato da rotina HIPOT se encontrar fora da rotina chamadora (`TForm1.Button1Click`).

Por último, implemente os códigos dos botões Limpar e Sair, grave o projeto novamente e execute a aplicação.

2. Refaça o exercício anterior, de maneira que o cálculo da hipotenusa se dê pela passagem de parâmetros por referência.

Solução

a) Código do projeto.

Abra um novo projeto (**Project > New Project > Application**) e desenvolva o formulário, conforme apresentado no exercício anterior.

Após a elaboração da interface do projeto, salve a aplicação numa nova pasta. Em seguida, dê um duplo clique no botão Processar e digite o código dentro da `procedure` (rotina chamadora).

```
procedure TForm1.Button1Click(Sender: TObject);
var
   CAT1, CAT2: integer;
   H1: real;
begin
   CAT1:= StrToInt(Edit1.Text);
   CAT2:= StrToInt(Edit2.Text);
   HIPOT(CAT1,CAT2,H1);
   Edit3.Text:= FloatToStr(H1);
   Edit1.SetFocus;
end;
```

Em seguida, é apresentado o código da `procedure` HIPOT, o qual deverá ser digitado antes da `procedure TForm1.Button1Click`, conforme já explicado anteriormente.

```
procedure HIPOT(B,C: integer; var H: real);
  var HIP: real;
begin
  HIP:= SQRT(SQR(B) + SQR(C));
  H:= HIP;
end;
```

Explicando

Observe que, diferentemente do exercício anterior, onde a `procedure` era com passagem de parâmetros por valor, criamos um parâmetro com passagem por referência (`var H: real`). O parâmetro H irá corresponder com o parâmetro H1, da rotina chamadora. Os dois apontam para a mesma referência na memória. Por isso, o que acontece com um, acontece com o outro. Tudo por causa daquela palavrinha `var` que precede o parâmetro H.

Então, não há necessidade de se exibir o valor da hipotenusa dentro desta `procedure`. Isso pode ser feito no corpo da rotina chamadora através do parâmetro H1.

Resumo

Nesta aula, você aprendeu a desenvolver suas próprias rotinas dentro do IDE do Lazarus. Veja que, até então, estávamos usando `procedures` geradas pelo próprio ambiente de programação. Portanto, como nossa programação dentro desse ambiente se deu a partir da geração de eventos, constatamos que o IDE do Lazarus é totalmente modularizado, pois para cada evento teremos um procedimento.

Tenha sempre em mente que a modularização dos programas, nos permite que a partir de um problema complexo, apresentemos soluções para partes específicas do mesmo, de tal forma que ao final consigamos resolver o problema em seu todo.

Assim, terminamos aqui a aula sobre procedimentos e, a seguir, apresentaremos o uso de funções na modularização dos programas. Portanto, vamos em frente!

Atividades de aprendizagem

1. Fazer um projeto em Lazarus que leia um número inteiro, calcule e imprima o cubo desse número. Utilizar procedimento com passagem de parâmetros por valor.

2. Fazer um projeto em Lazarus que leia dois números inteiros e faça a troca do primeiro pelo segundo, utilizando procedimento com passagem de parâmetros por referência.

3. Fazer um projeto em Lazarus para ler a base e a altura de um triângulo retângulo e, através de um procedimento, calcular a área do triângulo.

   Dado: Área = (base × altura) / 2

4. Fazer um projeto em Lazarus que leia os coeficientes A, B e C de uma equação do segundo grau. No programa principal, calcule o valor de DELTA. Elaborar um procedimento que receba os valores de A, B e C e também o valor de DELTA e calcule as raízes da equação e imprima-as no programa principal (passagem de parâmetros por referência).

Aula 11 – Funções

Objetivos

Desenvolver programas no IDE do Lazarus utilizando estruturas modulares, elaboradas pelo próprio programador – `function`.

11.1 Módulo função

Uma função sempre devolverá um resultado através de seu nome. O tipo do valor devolvido é definido na estrutura da função. É importante lembrar, também, que a função faz parte de um comando (expressão), sendo que sua chamada dentro deste se dará pelo nome da função, seguido dos respectivos parâmetros (maiores explicações, ver Aula 6). A sintaxe dessa estrutura é apresentada a seguir.

Sintaxe

```
function <nome> [(parâmetros: tipos)]: <tipo da função>;
[var variáveis locais : tipos;]
begin
   <instruções>;
   <nome>: = <expressão ou resultado>;
end;
```

11.2 Exercício resolvido

1. Faça um projeto em Lazarus para ler o raio de uma esfera. O programa principal deverá fazer uso de uma função que recebe por parâmetro o raio de uma esfera, calcula o seu volume ($v = 4/3 \times \pi \times R^3$) e o devolve ao local da chamada da função (rotina chamadora), o qual deverá ser exibido.

Solução

a) Dados do problema:
   - Valor do raio de uma esfera.

b) O que se pede:
   - Volume da esfera (através de uma função).

c) Elaboração do algoritmo ou do fluxograma.

d) Elaboração da interface gráfica.

Abra um novo projeto (**Project > New Project > Application**) e desenvolva o formulário, conforme apresentado na Figura 11.1.

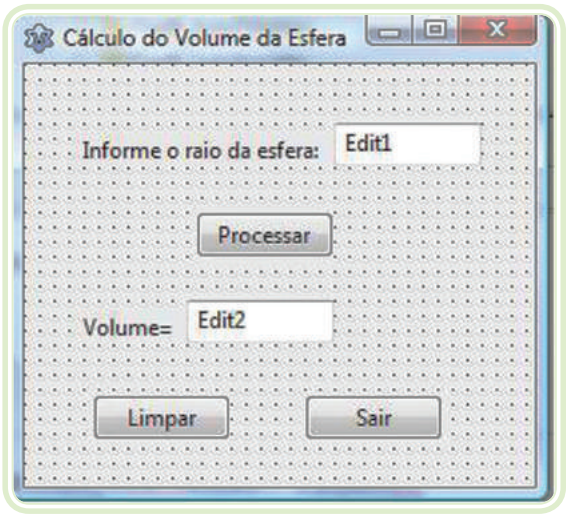

Figura 11.1: Cálculo do volume da esfera
Fonte: *Software* Lazarus

e) Código do projeto.

Após a elaboração da interface do projeto, salve a aplicação numa nova pasta. Em seguida, dê um duplo clique no botão Processar e digite o código dentro da `procedure`.

```
procedure TForm1.Button1Click(Sender: TObject);
var RAIO: real;
begin
   RAIO:= StrToFloat(Edit1.Text);
   Edit2.Text:= FloatToStr(VOLUME(RAIO));
   Edit1.SetFocus;
end;
```

Explicando

Nesta `procedure` é feita a chamada da função VOLUME que recebe através de parâ metro o valor do raio. Observe que a função foi acionada a partir de um comando que faz uso de outra função, no caso, de conversão de tipos (real para texto), sendo assim, um argumento da mesma.

Após a chamada desta sub-rotina, ela é executada e, ao término, o resultado devolvido na própria função, ou seja, ao local de onde foi feito seu acionamento. Em seguida, o valor é convertido para string e apresentado através do componente `Edit2`.

A seguir, é exibido o código completo da função referente à sub-rotina volume, o qual deverá ser inserido por nós, antes da rotina principal do programa, ou seja, antes da `procedure TForm1.Button1Click`; caso contrário o programa não a reconhecerá.

```
function VOLUME(R: real): real;
var V: real;
begin
  V:= 4/3 * 3.1416 * (R * R * R);
  VOLUME:= V;
end;
```

Explicando

Veja que ao final da `function`, o resultado encontrado é atribuído ao nome da função que o devolverá à rotina principal do programa.

Resumo

Concluímos aqui o estudo da aplicação de procedimentos e funções através de uma linguagem de programação. Dessa forma, é importante que você saiba usar corretamente, durante a elaboração da solução de um problema, essas sub-rotinas, lembrando que as principais diferenças entre elas é quanto ao retorno de valores e a forma como são chamadas.

Assim, encerramos nossos primeiros passos no uso de uma linguagem de programação, certos de que, o que lhe foi apresentado, será de grande utilidade para que você possa caminhar a passos largos no aprofundamento desse tema, daqui pra frente!

Parabéns por chegar até aqui!

Atividades de aprendizagem

1. Fazer um projeto em Lazarus para ler a altura (em metros) e o sexo (1 – masculino, 2 – feminino) de uma pessoa e, através de uma função, calcular e escrever o peso ideal. Portanto, a função deverá receber através de parâmetros a altura e o sexo de uma pessoa, calcular o peso ideal e devolvê-lo ao local da chamada da função, o qual será impresso. Para os homens a fórmula do peso ideal é a seguinte: 72 × altura – 60. Para as mulheres: 62 × altura – 50.

2. Fazer um projeto em Lazarus para ler um número inteiro e, através de uma função, verificar se o número lido encontra-se no intervalo: $10 \leq NUM \leq 150$; retornando ao local da chamada da função a mensagem "No intervalo" e "Fora do intervalo" em caso contrário (adaptado de Manzano; Mendes, 1999).

3. Fazer um projeto em Lazarus para ler 3 números inteiros e, através de uma função, calcular e escrever o menor dos 3 números lidos.

4. Fazer um projeto em Lazarus que leia a temperatura em graus Fahrenheit de uma cidade e, utilizando função, faça a conversão para graus Celsius. A temperatura em Fahrenheit é o dado de entrada e a temperatura em Celsius é o valor a ser calculado pela função. Utilize a fórmula $C = (F - 32) \times 5/9$, onde F é a temperatura em Fahrenheit e C é a temperatura em Celsius.

Referências

ALMEIDA, Daniel Simões. Lazarus e Free Pascal. **ClubeDelphi**, Rio de Janeiro, ano 9, n. 122, 123, 2010.

CANNEYT, Michaël Van. **Reference guide for Free Pascal, version 2.6.0**. Disponível em: <ftp://ftp.freepascal.org/pub/fpc/docs-pdf/ref.pdf>. Acesso em: dez. 2011.

FARRER, Harry e outros. **Programação estruturada de computadores**: algoritmos estruturados. Rio de Janeiro: Guanabara Dois, 1999.

GUIMARÃES, Ângelo de Moura; LAGES, Alberto de Castilho. **Algoritmos e estruturas de dados**. Rio de Janeiro : LTC Editora S.A., 1994.

LAZARUS WIKI. **Open Source GUI RAD IDE for Free Pascal**. Disponível em: <http://wiki.lazarus.freepascal.org/Lazarus_For_Delphi_Users>. Acesso em: 23 nov. 2010.

MANZANO, José Augusto N. G.; MENDES, Sandro S. V. **Estudo dirigido**: Delphi 4. São Paulo: Érica, 1999.

MANZANO, José Augusto N. G.; OLIVEIRA, Jayr Figueredo de. **Algoritmos**: lógica para desenvolvimento de programação. São Paulo: Érica, 2005.

SANTOS, Jean Patrick Figueiredo dos. **Desenvolvendo aplicativos com Lazarus**. 1. ed. [S.l.: s.n.], 2011.

SANTOS, Luciano André dos. **O caminho das pedras para o Delphi 4**. Rio de Janeiro: Book Express, 1998.

Squadra Tecnologia em Software. **Delphi 5**: módulo básico. Belo Horizonte, 2000. Apostila.

Currículo do professor-autor

Adolfo José Gonçalves Stavaux Baudson possui graduação em Engenharia de Minas pela Universidade Federal de Ouro Preto (1988), especialização em Administração de Sistemas de Informação pela Faculdade de Ciências Gerenciais da UNA (1993) e mestrado em Engenharia Mineral pela Universidade Federal de Ouro Preto (2008).

É professor concursado na área de Informática no Instituto Federal de Minas Gerais (IFMG), campus Ouro Preto, desde o ano de 1991, lecionando as disciplinas: Sistemas Operacionais, Banco de Dados, Algoritmos e Linguagem de Programação.

Francisco César Rodrigues de Araújo possui graduação em Engenharia de Minas pela Universidade Federal de Ouro Preto (1990), especialização em Informática Educativa – Formar III pela Escola Técnica Federal de Ouro Preto (1992) em convênio com a Escola Técnica Federal de Goiás e mestrado em Engenharia Mineral pela Universidade Federal de Ouro Preto (2008).

Publicou na revista REM da Universidade Federal de Ouro Preto (UFOP), juntamente com seu orientador, professor Marcone J. F. Souza, em março de 2011, artigo denominado "Uma heurística para o planejamento de lavra com alocação dinâmica de caminhões".

Participou do IV e do V Congresso Brasileiro de Mina a Céu Aberto e Mina Subterrânea apresentando 2 artigos (2006 e 2008).

Atuou como programador de computador na UFOP no período de 1982 a 1991.

É professor concursado na área de Informática no Instituto Federal de Minas Gerais (IFMG), campus Ouro Preto, desde o ano de 1991, lecionando, atualmente, as disciplinas Algoritmos e Linguagem de Programação.