LICENCIATURA EM CIÊNCIA
DA COMPUTAÇÃO

ALGORITMOS

PRESIDENTE DA REPÚBLICA
Dilma Roussef
MINISTRO DA EDUCAÇÃO
Aloísio Mercadante
SISTEMA UNIVERSIDADE ABERTA DO BRASIL
PRESIDENTE DA CAPES
Jorge Guimarães
DIRETOR DE EDUCAÇÃO A DISTÂNCIA DA CAPES
João Teatini

GOVERNO DO ESTADO DA BAHIA
GOVERNADOR
Jaques Wagner
VICE-GOVERNADOR
Otto Roberto Mendonça de Alencar
SECRETÁRIO DA EDUCAÇÃO
Osvaldo Barreto Filho

UNIVERSIDADE DO ESTADO DA BAHIA - UNEB
REITOR
Lourisvaldo Valentim da Silva
VICE-REITORA
Adriana do Santos Marmori Lima
PRÓ-REITOR DE ENSINO DE GRADUAÇÃO
José Bites de Carvalho
COORDENADOR UAB/UNEB
Silvar Ferreira Ribeiro
COORDENADOR UAB/UNEB ADJUNTO
André Magalhães

FABRIZIO LEANDRO FONSÊCA FISCINA
MARCIO VIEIRA BORGES

LICENCIATURA EM CIÊNCIA
DA COMPUTAÇÃO

ALGORITMOS

EDUNEB
Salvador

© UNEB 2012
Nenhuma parte deste material poderá ser reproduzida, transmitida ou gravada, por qualquer meio eletrônico, mecânico, por fotocópia e outros, sem a prévia autorização, por escrito, da Coordenação UAB/UNEB.

Depósito Legal na Biblioteca Nacional
Impresso no Brasil 2012

EDITORA DA UNIVERSIDADE DO ESTADO DA BAHIA

DIRETORA
Maria Nadja Nunes Bittencourt
COORDENADOR EDITORIAL
Ricardo Baroud

COORDENAÇÃO UAB/UNEB

COLABORADORES

SUPERVISÃO DE MATERIAL DIDÁTICO
Andréa Santos Tanure
Flávia Souza dos Santos
Naás Calasans Fagundes Pereira
Samara Kelly Rodrigues Sampaio

PROJETO GRÁFICO e CRIAÇÃO
João Victor Souza Dourado
Carla Cristiani Honorato de Souza

REVISORA
Maíta Nogueira de Andrade

PREPARAÇÃO DO ORIGINAL
Carla Cristiani Honorato de Souza

NORMALIZAÇÃO
Sheila Rangel

DIAGRAMAÇÃO
João Victor Souza Dourado

O conteúdo deste Material Didático é de inteira responsabilidade do(s)/da(s) autores (as), por cuja criação assume(m) ampla e total responsabilidade quanto a titularidade, originalidade do conteúdo intelectual produzido, uso de citações de obras consultadas, referências, imagens e outros elementos que façam parte desta publicação.

FISCINA, Fabrizio Leandro Fonseca; BORGES, Marcio.
F531 Algoritmos: licenciatura em ciência da computação. / Fabrizio Lenadro Fonseca Fiscina; Marcio Borges. Salvador: UNEB/GEAD, 2013.
96 p.

1. Algoritmos 2.Lógica de Programação 3. Linguagem Pascal I. Fabrizio Lenadro Fonseca Fiscina II. Marcio Borges III. Título. IV. Universidade Aberta do Brasil. V. UNEB / GEAD

CDD: 658.05

Editora da Universidade do Estado da Bahia - EDUNEB.
Rua Silveira Martins, 2555 - Cabula
41150-000 - Salvador - BA
www.eduneb.uneb.br
editora@listas.uneb.br Tel. + 55 71 3117-5342

Caro (a) Cursista

Estamos começando uma nova etapa de trabalho e para auxiliá-lo no desenvolvimento da sua aprendizagem estruturamos este material didático que atenderá ao Curso de Licenciatura na modalidade de Educação a Distância (EaD).

O componente curricular que agora lhe apresentamos foi preparado por profissionais habilitados, especialistas da área, pesquisadores, docentes que tiveram a preocupação em alinhar o conhecimento teórico e prático de maneira contextualizada, fazendo uso de uma linguagem motivacional, capaz de aprofundar o conhecimento prévio dos envolvidos com a disciplina em questão. Cabe salientar, porém, que esse não deve ser o único material a ser utilizado na disciplina, além dele, o Ambiente Virtual de Aprendizagem (AVA), as atividades propostas pelo Professor Formador e pelo Tutor, as atividades complementares, os horários destinados aos estudos individuais, tudo isso somado compõe os estudos relacionados à EaD.

É importante também que vocês estejam sempre atentos às caixas de diálogos e ícones específicos que aparecem durante todo o texto apresentando informações complementares ao conteúdo. A ideia é mediar junto ao leitor, uma forma de dialogar questões para o aprofundamento dos assuntos, a fim de que o mesmo se torne interlocutor ativo desse material.

São objetivos dos ícones em destaque:

VOCÊ SABIA? – convida o leitor a conhecer outros aspectos daquele tema/ conteúdo. São curiosidades ou informações relevantes que podem ser associadas à discussão proposta.

SAIBA MAIS – apresenta notas, textos para aprofundamento de assuntos diversos e desenvolvimento da argumentação, conceitos, fatos, biografias, enfim, elementos que o auxiliam a compreender melhor o conteúdo abordado.

INDICAÇÃO DE LEITURA – neste campo, você encontrará sugestões de livros, sites, vídeos. A partir deles, você poderá aprofundar seu estudo, conhecer melhor determinadas perspectivas teóricas ou outros olhares e interpretações sobre determinado tema.

SUGESTÃO DE ATIVIDADE – consiste num conjunto de atividades para você realizar autonomamente em seu processo de autoestudo. Estas atividades podem (ou não) ser aproveitadas pelo professor-formador como instrumentos de avaliação, mas o objetivo principal é o de provocá-lo, desafiá-lo em seu processo de autoaprendizagem.

Sua postura será essencial para o aproveitamento completo desta disciplina. Contamos com seu empenho e entusiasmo para juntos desenvolvermos uma prática pedagógica significativa.

Setor de Material Didático
Coordenação UAB/UNEB

ÁREA DE AMBIENTAÇÃO

Dados Pessoais

Nome:
Telefones: E-mail:
Residência:
Município/Polo do curso EaD:

Dados do Curso

·Endereço de acesso à sala de aula virtual:
·Período de execução desta disciplina: de_______ à _____/ ______/______
·Quem são os orientadores desta disciplina?

Categoria	Responsável	por Nome	E-mail
Prof. Autor	Elaborar o módulo impresso		
Prof. Formador	Planejar e organizar a sala vitual		
Tutor a distância	Mediar os estudos no ambiente		
Tutor presencial	Mediar os encontros presenciais		
Coord. de Polo	Apoiar as ações do curso no local		

Identificação da Turma

Nome	Onde encontrar	Telefones	E-mail

APRESENTAÇÃO

Estamos contentes em poder participar de sua formação e gostaríamos de dar a você boas-vindas à disciplina Algoritmos. Nosso objetivo com esta disciplina é dialogar com você sobre o estudo da lógica computacional, conceituando o algoritmo, a análise e a resolução de problemas. A noção de algoritmo é muito importante para toda a computação, sendo que para sua criação, existem linhas mestras e estruturas básicas, a partir das quais o aluno poderá cria-lo. Mas a solução completa depende em grande parte do criador. Geralmente existem diversos algoritmos para resolver o mesmo problema, cada um segundo o ponto de vista do seu criador.

Este material apoiará a disciplina, abordando os conceitos básicos de algoritmo, análise e resolução de problemas, a expressão de soluções em termos de algoritmos estruturados, a aplicação das principais estruturas para controle de fluxo da solução, a aplicação de estruturas básicas para estruturação da informação, o projeto de soluções estruturadas e modularizadas para problemas simples.

Os principais objetivos da disciplina no capitulo um é conceituar o algoritmo e suas formas de representação mais conhecidas, além de apresentar os conceitos de variáveis e atribuição e assim contribuir para desenvolver um conhecimento básico geral que proporcionará base para a resolução de problemas através de soluções estruturadas. Outro objetivo deste material é fomentar no estudante a capacidade de resolução de problemas através das técnicas de algoritmos, ampliando o conceito do funcionamento do computador.

Ao longo deste módulo, você encontrará informações que atendem a crescente demanda do mercado de trabalho, que cada vez mais precisa de profissionais que tenham desenvolvido uma visão lógica e técnica da programação, fornecendo as principais estruturas e recursos na elaboração de um Algoritmo.

Para isso, a metodologia da disciplina deve contemplar também vídeo aulas, estudo de caso e realização de exercícios que estimulam o aprendizado e contribuem para o fortalecimento do conhecimento. Além disso, é importante que o aluno utilize o ambiente virtual de aprendizagem para acompanhar as atividades e a condução da disciplina.

Gostaríamos de convidá-lo a participar deste diálogo conosco, esperando que os conhecimentos aqui mediados sejam profícuos para o seu desenvolvimento profissional.

Prof. Msc. Fabrizio Leandro Fonsêca Fiscina e Prof. Msc. Marcio Vieira Borges

Anotações

SUMARIO

1 INTRODUÇÃO À LÓGICA DE PROGRAMAÇÃO 13
1.1 Formas de representação de algoritmos 16
1.1.1 Diagrama Chapin 16
1.1.2 Fluxograma 17
1.1.3 Português Estruturado 18
1.1.4 Representações de um mesmo Algoritmo 19
2 CONSTANTES, VARIÁVEIS E ATRIBUIÇÕES 21
2.1 Constantes 23
2.2 Variáveis 23
2.3 Atribuição 23
2.4 Tipos de dados 24
2.4.1 Inteiros 24
2.4.2 Reais 24
2.4.3 Caracteres 24
2.4.4 Lógicos ou booleanos 24
2.5 Utilizando as variáveis e constantes no algoritmo 24
2.6 Definindo uma variável no algoritmo 25
2.7 Conceito e utilidade de constantes 25
3 INSTRUÇÕES BÁSICAS DO ESTUDO DO ALGORITMO 27
3.1 Instrução escrever 29
3.2 Instrução ler 30
3.3 Horizontalização 30
4 OPERADORES E EXPRESSÕES 31
4.1 Operadores 33
4.1.1 Operadores de atribuição 33
4.1.2 Operadores aritméticos 33
4.1.3 Operadores relacionais 34
4.1.4 Operadores lógicos 35
4.1.5 Operadores literais 36
4.2 Expressões 36
4.2.1 Expressões aritméticas 36
4.2.2 Expressões lógicas 36
4.2.3 Expressões literais 37
4.2.4 Avaliação de expressões 37

5 INSTRUÇÕES 39
5.1 Comandos de atribuição 41
5.2 Comandos de saída de dados 41
5.3 Comandos de entrada de dados 42
5.4 Interface com usuário 42
5.5 Funções matemáticas 43

6 ESTRUTURAS DE CONTROLE 45
6.1 Comandos compostos 47
6.2 Estrutura seqüencial 47
6.3 Estruturas de decisão 47
6.3.1 Estruturas de decisão simples (se ... Então) 48
6.3.2 Estruturas de decisão composta (se ... Então ... Senão) 48
6.3.3 Estruturas de decisão múltipla do tipo escolha (escolha...Caso...Senão) 50
6.4 Estruturas de repetição 51
6.4.1 Laços contados (para ... Faça) 51
6.4.2 Laços condicionais 52
6.4.3 Laços condicionais com teste no início (enquanto ... Faça) 52
6.4.4 Laços condicionais com teste no final (repita ... Até que) 53
6.5 Estruturas de controle encadeadas ou aninhadas 54

7 ESTRUTURAS DE DADOS HOMOGÊNEAS 55
7.1 Matrizes de uma dimensão ou vetores 57
7.1.1 Operações básicas com vetores 57
7.1.2 Atribuição de uma matriz do tipo vetor 57
7.1.3 Leitura de dados de uma matriz do tipo vetor 57
7.1.4 Escrevendo dados de um vetor 58
7.1.5 Método de bolha de classificação 59
7.2 Matrizes com mais de uma dimensão 60
7.2.1 Operações básicas com matrizes de duas dimensões 60
7.2.2 Atribuição de uma matriz de duas dimensões 61
7.2.3 Leitura de informações de uma matriz de duas dimensões 61
7.2.4 Escrita de dados de uma matriz de duas dimensões 61

8 SUBALGORITMOS 63
8.1 Funcionamento do subalgoritmo 65
8.2 Elementos dos subalgoritmos 66
8.3 Funções 66
8.4 Procedimentos 67
8.5 Variáveis 68
8.6 Parâmetros 69
8.7 Mecanismos de passagem de parâmetros 69
8.7.1 Passagem de parâmetros por valor 69
8.7.2 Passagem de parâmetros por referência 70
8.8 Refinamentos sucessivos 70

9 LINGUAGEM DE PROGRAMAÇÃO PASCAL 73
9.1 A origem do pascal 75
9.2 Elementos da linguagem pascal 75
9.2.1 Elementos definidos pela linguagem pascal 75
9.2.2 Elementos definidos pelo usuário 76
9.3 Tipos de dados no pascal 76
9.3.1 Tipos predefinidos pela linguagem na pascal 76
9.4 Constantes e variáveis do pascal 77
9.4.1 Constante 77
9.4.2 Variáveis 77
9.4.3 Operadores 78

10 ESTRUTURA DO PROGRAMA DESENVOLVIDO EM PASCAL 79
10.1 Identificação do programa 81
10.2 Bloco de declarações 81
10.3 Bloco de comandos 81
10.4 Comandos 82
10.4.1 Read e readln 82
10.4.2 Write e writeln 82
10.4.3 Uses printer 83
10.4.4 Readkey 83
10.4.5 Goto 83
10.4.6 Exit 84
10.4.7 Runerror 84
10.4.8 Clrscr 84
10.4.9 GotoXY 84
10.4.10 DELAY 85
10.4.11 CHR 85
10.4.12 ORD 85
10.4.13 UPCASE 85
10.5 Estruturas de controle 85
10.5.1 Seqüência de comandos 85
10.5.2 Comandos condicionais 86
10.5.3 Comandos de repetição 86

GLOSSÁRIO 92

REFERÊNCIAS 93

INTRODUÇÃO À LÓGICA DE PROGRAMAÇÃO

CAPÍTULO 1

Anotações

CAPÍTULO 1 - INTRODUÇÃO À LÓGICA DE PROGRAMAÇÃO

No presente capítulo, você conhecerá a Lógica de Programação e terá uma visão geral do processo de desenvolvimento de programas (softwares), visto que o objetivo final é ter um bom embasamento para a prática da programação de computadores. Serão apresentadas as etapas para o ciclo de vida do sistema, as principais formas de representação dos algoritmos e os conceitos básicos de constantes, variáveis e atribuição.

Nosso objetivo é desenvolver o que chamamos de lógica de programação, quanto falamos em lógica de programação, usamos vários termos como fluxograma, algoritmos, pseudocódigo, entre outros.

No momento em que uma determinada atividade passa a ser realizada por máquinas, inclusive computadores, ao invés de ser realizada pelo homem, estamos realizando um processo de automação.

Para que a automação de uma determinada atividade tenha êxito, é fundamental que o equipamento que a irá realizar o mesmo tenha condições de desempenhar todas as etapas que o envolve com precisão e no menor espaço de tempo possível, garantindo também a repetição do processo, e tais instruções deverão ser repassadas a esta máquina.

A esta especificação dos passos a serem seguidos e suas regras é dado o nome de algoritmo, ou seja, para que o computador possa realizar um determinado trabalho, será necessário que este seja detalhado passo a passo, através de uma forma compreensível pela máquina para ser empregado no que chamamos de programa.

Para representar um algoritmo, são utilizadas diversas técnicas e cabe ao profissional, adotar aquela que melhor se adapte as suas necessidades.

Existem algoritmos que apresentam os passos apenas a nível lógico, ou seja, sem se preocupar com detalhes de uma linguagem de programação específica, e outros que tratam os passos a serem seguidos com maior riqueza de detalhes.

Na área da informática, o algoritmo é o projeto do software, ou seja, antes de se fazer um software na linguagem de programação desejada (Pascal, C, Delphi, etc.) deve-se fazer o algoritmo do software.

Um programa, é um algoritmo escrito através de uma linguagem de programação numa forma compreensível pelo computador, onde todas as ações a serem executadas devem ser especificadas de acordo com as regras de sintaxe da linguagem escolhida.

Cada Linguagem de Programação tem um conjunto de instruções, comandos, chamado de sintaxe, que deve ser seguida corretamente para que o programa funcione. Este conjunto de palavras e regras que definem o formato das sentenças válidas chama-se de sintaxe da linguagem.

Para o desenvolvimento de qualquer programa, deve-se seguir algumas etapas, conhecidas como Ciclo de Vida do Sistema:

1) Estudo da Viabilidade (Estudos Iniciais);
2) Análise detalhada do sistema (Projeto Lógico);
3) Projeto preliminar do sistema (Projeto Físico);
4) Projeto detalhado do sistema (Algoritmos);
5) Implementação ou Codificação do sistema (na Linguagem de Programação escolhida);
6) Testes do sistema; e
7) Instalação e Manutenção do sistema.

No desenvolvimento de um sistema, quanto mais tarde um erro é detectado, mais capital e tempo se gasta para repará-lo. Assim, a responsabilidade do desenvolvedor é maior na criação dos algoritmos do que na sua própria implantação, pois quando bem projetados, não se perde tempo tendo que refazê-los, assegurando assim uma implantação eficaz e eficiente.

Na literatura em informática são encontradas várias formas de representação das etapas que compõem o ciclo de vida de um sistema. Essas formas de representação podem variar tanto na quantidade de etapas quanto nas atividades a serem realizadas em cada fase. Portanto, não existe um modelo ideal.

Um algoritmo pode ser definido como um conjunto de regras (instruções), bem especificadas, para solução de um determinado problema. Conforme o dicionário Michaelis, o conceito de algoritmo é a "utilização de regras para definir ou executar uma tarefa específica ou para resolver um problema específico." (MICHAELIS, 1998, p. 21).

A partir desses conceitos de algoritmos que é bastante amplo, pode-se perceber que a palavra algoritmo não é um termo computacional, ou seja, não se refere apenas à área de informática.

Para que um problema no computador possa ser resolvido, é necessário que seja primeiro encontrado uma maneira correta e clara de descrever este problema. É preciso que encontremos uma seqüência de passos que permitam que o problema possa ser resolvido de maneira automática e repetitiva. Esta seqüência de passos é chamada de algoritmo.

Uma das formas mais eficazes de aprender algoritmos é através de muitos exercícios, construindo e testando os algoritmos.

SUGESTÃO DE ATIVIDADE

Os exercícios são importantíssimos para o apredizado dos algoritmos. É através da prática que as soluções vão sendo assimiladas e melhor conhecidas. Portanto a cada novo conteúdo você deve realizar as atividades propostas para uma melhor fixação dos conhecimentos.

1. O que são algoritmos?
2. Quais as etapas mais conhecidas no Ciclo de Vida de um Sistema?
3. Pesquise sobre as linguagens de programação mais utilizadas no mercado atualmente.
4. O que é a sintaxe de uma linguagem de programação? Dê exemplos.

1.1 FORMAS DE REPRESENTAÇÃO DE ALGORITMOS

Os algoritmos podem ser representados de várias formas, como por exemplo:

- **Linguagem Natural:** Através de uma língua (português, inglês, etc.): forma utilizada nos manuais de instruções, nas receitas culinárias, bulas de medicamentos, manual de operação de um determinado eletrodoméstico, constantes do manual do operador, etc.
- **Linguagem de programação**: Forma utilizada por alguns programadores, que pulam a etapa do projeto do programa (algoritmo) e passam direto para a programação em si.
- **Linguagem estruturada (Portugol): F**orma de escrever algoritmos que muito se assemelha a forma na qual os programas são escritos nas linguagens de programação (Pascal, C, Java, etc.).
- **Fluxograma:** Através de representações gráficas: são bastante recomendáveis, já que um diagrama ou fluxograma muitas vezes substitui, com vantagem, várias palavras. A representação gráfica de algoritmos, ou seja, das instruções e/ou módulos do processamento, também é conhecida como diagrama de bloco.

É claro que cada uma dessas formas de representar um algoritmo, tem suas vantagens e desvantagens, cabe ao programador escolher a forma que melhor lhe convir. Neste módulo serão apresentadas três formas de representação de algoritmos, são elas:

- Diagrama de Chapin ou Diagrama de Nassi-Shneiderman;
- Fluxograma (Diagrama de Fluxo);
- Português Estruturado (Pseudocódigo, Portugol ou Pseudolinguagem).

1.1.1 Diagrama Chapin

Os Diagramas Chapin, também conhecidos como Diagramas Nassi-Shneiderman, surgiram nos anos 70 como uma maneira de ajudar nos esforços da abordagem de programação estruturada. Um típico diagrama Chapin é apresentado na Figura 1 a seguir. Como você pode ver, o diagrama é fácil de ler e de entender, cada "desenho" representa uma ação/instrução diferente.

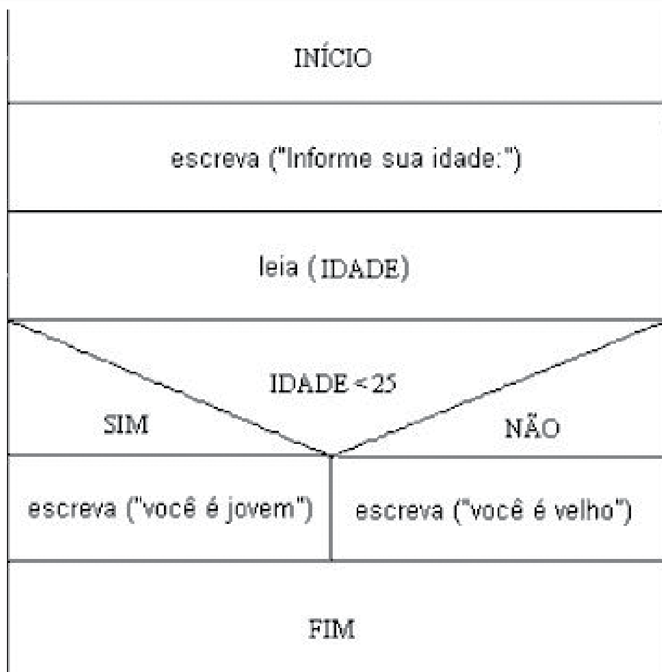


Figura 1: Diagrama de Chapin
Fonte: PREUSS (2012).

A idéia deste diagrama é representar as ações de um algoritmo dentro de um único retângulo, subdividindo-o em retângulos menores, que representam os diferentes blocos de seqüência de ações do algoritmo.

SAIBA MAIS

O primeiro artigo elaborado pelos autores do Diagrama de Chapin, foi escrito em 1973, e pode ser acessado na rede de computadores mundial. O diagrama de Chapin foi criado por Ned Chapin a partir de trabalhos de Nassi-Shneiderman, os quais resolveram substituir o fluxograma tradicional por um diagrama que apresenta uma visão hierárquica e estruturada da lógica do programa. A grande vantagem de usar este tipo de diagrama é a representação das estruturas que tem um ponto de entrada e um ponto de saída e são compostas pelas estruturas básicas de controle de seqüência, seleção e repartição.

1.1.2 Fluxograma

Os Fluxogramas também conhecidos como Diagramas de Fluxo, são uma representação gráfica que utilizam formas geométricas padronizadas ligadas por setas de fluxo, para indicar as diversas instruções e decisões que devem ser as mais seguidas para resolver o problema abordado.

Eles permitem visualizar o fluxo e as etapas de processamento de dados possíveis e, dentro destas, os passos para a resolução do problema. Na Figura 2 é apresentado um exemplo de fluxograma:

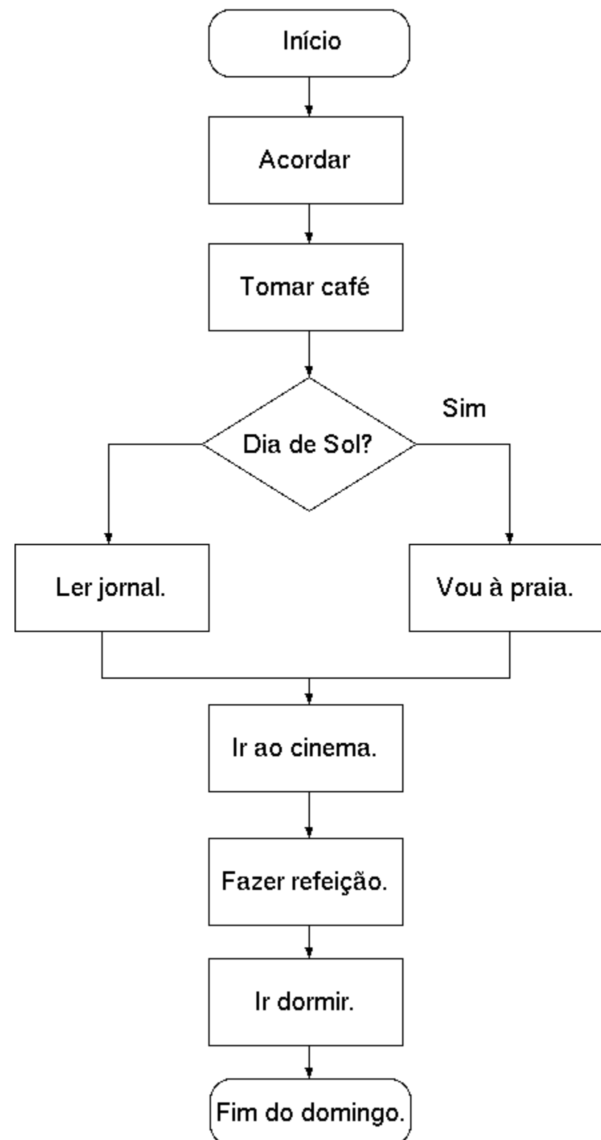


Figura 2: Diagrama de Fluxo
Fonte : MULLER(2012)

Existem atualmente vários padrões para definir as formas geométricas a serem utilizadas para as diversas instruções (passos) a serem seguidos pelo sistema. A seguir, na figura 3, é apresentado alguns símbolos que são utilizados no fluxograma:

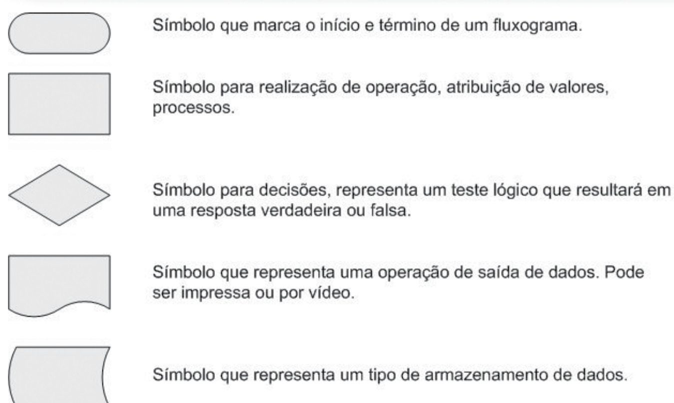


Figura 3: Símbolos utilizados no fluxograma
Fonte: MULLER (2012)

Vamos a seguir, exibir um fluxograma que teria o objetivo de calcular o valor diário de um salário recebido mensalmente. Seguindo uma lógica simples, teríamos que dividir o valor do salário, neste caso R$ 1.200,00, por 30 (número de dias de um mês).

Observe como ficaria este fluxograma:

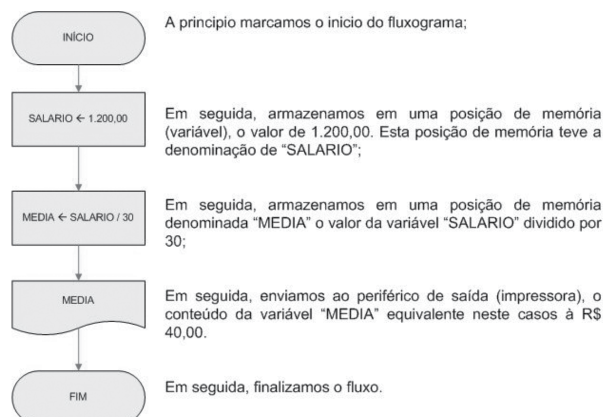


Figura 4: Fluxograma para calcular média diária de um salário
Fonte: MULLER(2012)

Observe que da forma que o mesmo foi exposto, existe um procedimento lógico para o cálculo proposto, bem como, poderemos repetir este procedimento por quantas vezes for necessário, que o resultado sempre será correto e preciso.

Posteriormente, iremos criar fluxogramas mais complexos, expressando situações reais de processamento, este teve apenas o objetivo de lhe dar uma noção simples e objetiva de fluxogramas.

1.1.3 Português Estruturado

O Português Estruturado, também conhecido como Portugol, Pseudocódigo ou Pseudolinguagem é uma forma especial de linguagem bem mais restrita que a Língua Portuguesa e com significados bem definidos para todos os termos utilizados nas instruções. O Português Estruturado é uma simplificação extrema da língua portuguesa, limitada a poucas palavras e estruturas que têm significado, pois deve-se seguir um padrão. Emprega uma linguagem intermediária entre a linguagem natural e uma linguagem de programação, para descrever os algoritmos.

A sintaxe do Português Estruturado não precisa ser seguida tão rigorosamente quanto a sintaxe de uma linguagem de programação, já que o algoritmo não será executado como um programa.

O Português Estruturado é uma linguagem bastante simplificada, entretanto, ela possui todos os elementos básicos e uma estrutura semelhante à de uma linguagem de programação de computadores. No uso do Português Estruturado para a solução de um problema é permitido uma menor rigidez que quando se utiliza uma linguagem de programação, pois a sintaxe errada não influenciará na solução. A seguir um exemplo de um programa representado através do Portugol.

```
Declarações
  inteiro IDADE;
Inicio
  escreva("Informe sua idade: ");
  leia (IDADE);
  se (IDADE < 16) então
     escreva ("Você não pode votar");
  senão
     escreva ("Você pode votar");
  fim se;
Fim
```

O Portugol é uma forma de representar um algoritmo mais detalhada que o fluxograma, se assemelhando muito às linhas de código de uma linguagem de programação, onde temos que definir as variáveis, rotinas, sub-rotinas, etc. Ao invés de símbolos gráficos, utilizamos comandos ou ordens para solicitar uma determinada tarefa/rotina. Sua sintaxe básica é:

```
Algoritmo <nome do algoritmo>
   <declaração das variáveis>
Inicio da rotina
```

<instruções a serem seguidas>
Fim da rotina

Vamos utilizar o mesmo exemplo e objetivo exposto na exemplificação do fluxograma, empregando o mesmo no Portugol. ilustração

Algoritmo VALOR_DIA
Var SALARIO, MEDIA : real
Inicio
SALARIO ← 1.200,00
MEDIA ←SALARIO/30
Envie para impressora "MEDIA"
Fim

No exemplo acima, a linha que contém a definição VAR tem o objetivo de declarar as variáveis que serão usadas e o tipo REAL representa uma variável que pode assumir valores numéricos com parte decimal. Mais adiante, iremos estudar os principais tipos de variáveis.

1.1.4 Representações de um mesmo Algoritmo

Para uma completa compreensão, seguem abaixo exemplos onde vamos mostrar a representação de um mesmo algoritmo de formas distintas.
A função do algoritmo é calcular a média de um aluno que realizou duas avaliações e que para ser aprovado deve obter a média igual ou superior a 7,0.

Exemplo 01: Representação na forma da Linguagem Natural

- Obter as notas da primeira e da segunda prova
- Calcular a média aritmética entre as duas
- Se a média for maior ou igual a 7, o aluno foi aprovado, senão ele foi reprovado.

Esta representação é pouco usada na prática porque o uso de linguagem natural muitas vezes dá oportunidade a más interpretações, ambigüidades e imprecisões.

Exemplo 02: Fluxograma Convencional

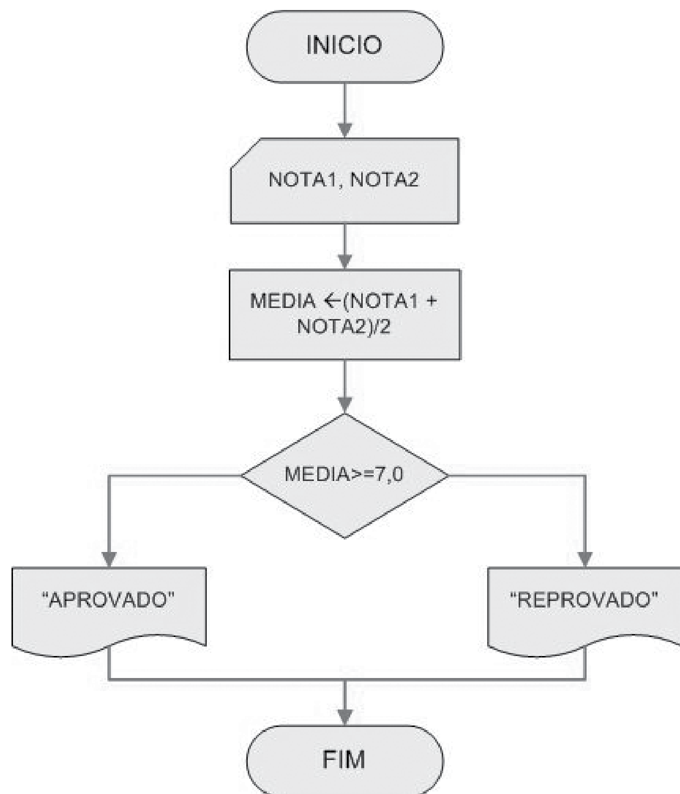


Figura 5: Representação do algoritmo de cálculo da média de um aluno sob a forma de um fluxograma.
Fonte: MULLER (2012)

Exemplo 03: Diagrama de Chapin

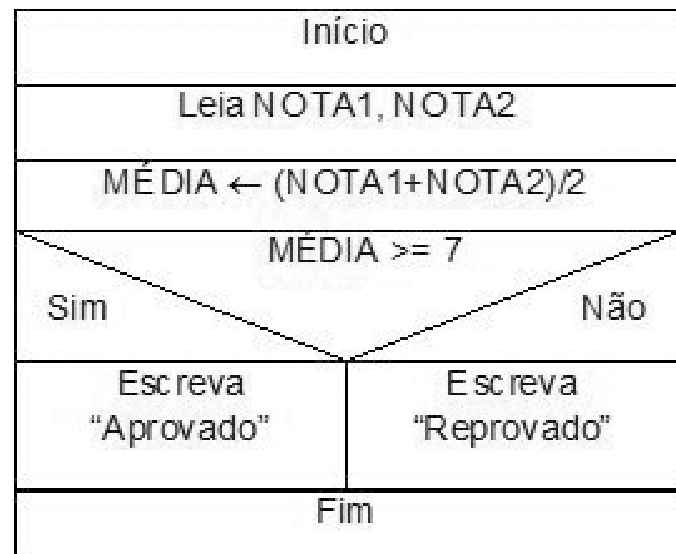


Figura 6: Diagrama de Chapin para o algoritmo do cálculo da média de um aluno.
Fonte: MULLER (2012)

Exemplo 04: Português Estruturado/ Pseudocódigo / Pseudolinguagem

A seguir, é apresentada a representação do algoritmo de cálculo da média de um aluno na forma de um pseudocódigo.

```
Algoritmo Média
Var NOTA1, NOTA2, MEDIA
Início
 Leia NOTA1, NOTA2
 MÉDIA ← (NOTA1+NOTA2)/2
 Se MEDIA >= 7 Então
         Escreva "Aprovado"
 Senão
         Escreva "Reprovado"
Fim.
```

SUGESTÃO DE ATIVIDADE

1. Mostre a Representação do algoritmo a seguir na forma da Linguagem Natural, na forma de Fluxograma Convencional, no diagrama de Chapin e no Português Estruturado. Escreva um algoritmo que leia três valores e armazene nas variáveis IDADE1, IDADE2 e IDADE3, depois calcule a média das idades e armazene na variável IDADE_MEDIA em seguida informe se a média das idades é maior ou menor que 20.

2. No algoritmo abaixo é possível perceber a representação em Portugol. Mostre como ficaria a sua representação na forma da Linguagem Natural, na forma do Fluxograma Convencional e no diagrama de Chapin.

```
Declarações
   inteiro IDADE;
Inicio
   escreva("Informe sua idade: ");
   leia(IDADE);
   se (IDADE < 16) então
      escreva ("Você não pode votar");
   senão
      escreva("Você pode votar");
   fim se;
Fim
```

INDICAÇÃO DE LEITURA

Guimarães, Angelo de Moura. Algoritmos e Estruturas de Dados. Rio de Janeiro: Ed. LTC, 1985.

CONSTANTES, VARIÁVEIS E ATRIBUIÇÕES

CAPÍTULO 2

Anotações

CAPÍTULO 2 - CONSTANTES, VARIÁVEIS E ATRIBUIÇÕES

Na construção de um algoritmo, alguns conceitos são fundamentais, pois constituem a base para todo o desenvolvimento do algoritmo e para a posterior solução do problema. Neste capítulo são apresentados e explicados três conceitos fundamentais para a construção de algoritmos, são eles: Constante, Variável e Atribuição.

2.1 CONSTANTES

As constantes são as informações que não variam com o tempo, ou seja, permanecem sempre com o mesmo conteúdo, é um valor fixo. Como exemplos de constantes temos: números, letras, palavras, etc.

2.2 VARIÁVEIS

Uma variável é uma área da memória do computador que é "reservada" para guardar informações. As variáveis podem conter valores diferentes a cada instante de tempo, ou seja, seu conteúdo pode variar de acordo com as instruções do algoritmo. Compreender o conceito de variável é fundamental para elaboração de algoritmos e conseqüentemente de programas.

As variáveis são referenciadas através de um nome criado por você durante o desenvolvimento do algoritmo. Exemplos de nomes de variáveis: nome, idade, nota, media, etc. O conteúdo de uma variável pode ser alterado, consultado ou apagado quantas vezes forem necessárias durante o algoritmo. Mas, ao alterar o conteúdo da variável, a informação armazenada anteriormente é perdida, ou seja, sempre uma informação é substituída pela informação mais recente.

Uma variável armazena um conteúdo de cada vez, assim, simplificando, podemos dizer que uma variável é como uma "caixa com um rótulo" colado nela, que em um dado momento guarda um determinado objeto. O conteúdo desta caixa não é algo permanente ele pode ser alterado diversas vezes.

No exemplo a seguir, temos uma variável chamada de PRECO, que inicialmente recebe o valor 15,00, em seguida, o valor é alterado para receber um desconto de 50%, ficando assim com um novo valor, que é informado no final.

```
Declarações
   inteiro PRECO;
Inicio
   Escreva ("Informe o preço: ");
   Leia (PRECO);
   PRECO ← PRECO*50%
 Escreva ("O preço com desconto é:",PRECO)
Fim
```

2.3 ATRIBUIÇÃO

A atribuição é uma notação utilizada para atribuir um valor a uma variável, ou seja, para armazenar um determinado conteúdo em uma variável. A operação de atribuição, normalmente, é representada por uma seta apontando para a esquerda, mas existem outros símbolos para representar a atribuição, depende da forma de representação do algoritmo.

Algumas possibilidades de atribuição são as relações:

a) Variável recebe valor armazenado em constante, ex: PRECO ← 150
b) Variável recebe valor armazenado em variável, ex: PRECO ← VALOR
c) Variável recebe valor calculado em uma expressão, ex: TOTAL← PRECO - DESC

Nas atribuições, a parte que irá receber um valor (localizada na esquerda, antes da seta) não pode haver nada além da variável, ou seja, é só a variável que "recebe" algum conteúdo, não é possível ter uma expressão, por exemplo, ou uma constante, recebendo algum valor. Veja o exemplo a seguir:

a) MEDIA ← (NOTA1 + NOTA2) /2, CORRETO
b) TOTAL ← VALOR1 + VALOR2, CORRETO
c) PRECO – CUSTO ← LUCRO, ERRADO
d) LARGURA * ALTURA ←TAXA, ERRADO

2.4 TIPOS DE DADOS

Todo o trabalho realizado por um computador é baseado na manipulação das informações contidas em sua memória. Estas informações podem ser classificadas em dois tipos:

- As instruções, que comandam o funcionamento da máquina e determinam a maneira como devem ser tratados os dados.
- Os dados propriamente ditos, que correspondem à porção das informações a serem processadas pelo computador.

A classificação apresentada a seguir não se aplica a nenhuma linguagem de programação específica; pelo contrário, ela sintetiza os padrões utilizados na maioria dos algoritmos e linguagens de programação.

2.4.1 Inteiros

Os tipos inteiros compreendem os dados numéricos positivos ou negativos. Excluindo-se destes qualquer número fracionário. Como exemplo deste tipo de dado, tem-se os valores: 46, 0, -120, 256 dentre outros.

2.4.2 Reais

Os tipos reais compreendem os dados numéricos positivos e negativos e números fracionários. Como exemplo deste tipo de dado, tem-se os valores: 46, 0, -120, 1.34, -12.345, 256 dentre outros.

2.4.3 Caracteres

São caracterizados como tipos caracteres, as seqüências contendo letras, números e símbolos especiais. Uma seqüência de caracteres deve ser indicada entre aspas (“”). Este tipo de dado também é conhecido como alfanumérico, string, literal ou cadeia. Como exemplo deste tipo de dado, tem-se os valores: “Nome”, “Praça Igaratinga, 120, Apto 1”, “Telefone 3221-1018”, “A média das notas foi”, “ ”, “150” dentre outros.

2.4.4 Lógicos ou Booleanos

São caracterizados como tipos lógicos os dados com valor verdadeiro e falso, sendo que este tipo de dado poderá representar apenas um dos dois valores. Como exemplo deste tipo, tem-se os valores: sexo, possui_filhos, aposentado, dentre outros.

2.5 UTILIZANDO AS VARIÁVEIS E CONSTANTES NO ALGORITMO

Entender como as variáveis e as constantes são aplicadas é o primeiro passo para a elaboração corretas dos algoritmos. No processo de armazenamento de dados podemos supor que a memória de um computador é um grande arquivo com várias gavetas, onde cada gaveta pode armazenar apenas um único valor (seja ele numérico, caractere ou lógico). Sabendo que o local onde a informação vai ser armazenada é um arquivo com diversas gavetas, é necessário identificar com um nome a gaveta que se pretende utilizar. Desta forma o valor armazenado pode ser utilizado a qualquer momento. Outra observação é que dependendo da informação a ser armazenada, o tipo de gaveta deve ser adequado tanto para o formato da informação quanto para o tamanho.

Como visto anteriormente, informações correspondentes a diversos tipos de dados são armazenadas nas memórias dos computadores. Para acessar individualmente cada uma destas informações, em princípio, seria necessário saber o tipo de dado desta informação (ou seja, o número de bytes de memória por ela ocupados) e a posição inicial deste conjunto de bytes na memória.

Percebe-se que esta sistemática de acesso a informações na memória é bastante ilegível e difícil de se trabalhar. Para contornar esta situação criou-se o conceito de variável, que é uma entidade destinada a guardar uma informação. Basicamente, uma variável possui três atributos:

a) Um nome;

b) Um tipo de dado associado à mesma; e

c) Uma informação por ela guardada.

Assim, toda variável possui um nome que tem a função de diferenciá-la das demais. Cada linguagem de programação estabelece suas próprias regras de formação de nomes de variáveis.

Adotaremos para os algoritmos, as seguintes regras:

a) Um nome de variável deve necessariamente começar com uma letra;

b) Um nome de variável não deve conter nenhum símbolo especial, exceto a sublinha (_) e nenhum espaço em branco;

c) Um nome de variável não poderá ser uma palavra reservada a uma instrução de programa ou co-

mando.

A seguir é possível perceber os exemplos de nomes de variáveis utilizados de forma correta e errada:

- Media – correto;
- 1MES – errado (não começou por uma letra)
- Ano1 – correto
- O menino – errado (contém o caractere branco)
- SAL/HORA – errado (contém o caractere "/")
- SAL_HORA – correto
- _DESCONTO – errado (não começou com uma letra)

Obviamente, é interessante adotar nomes de variáveis relacionados às funções que serão exercidas pelas mesmas dentro de um programa. Ou seja, na hora de definir o "nome" da variável, deve-se tentar escolher um que realmente esteja relacionado com o conteúdo que ficará armazenado naquela variável. Assim se facilita o entendimento e lógica deste algoritmo.

Outro atributo característico de uma variável é o tipo de dado que ela pode armazenar. Este atributo define a natureza das informações contidas na variável. Por último, há o atributo informação, que nada mais é do que a informação útil contida na variável.

É importante saber que, uma vez definidos, os atributos nome e tipo de dado de uma variável, estes não podem ser alterados e assim permanecem durante toda a sua existência, desde que o programa não seja modificado. Por outro lado, o atributo informação está constantemente sujeito a mudanças de acordo com o fluxo de execução do algoritmo.

O conceito de variável foi criado para facilitar a vida dos desenvolvedores, permitindo acessar informações na memória dos computadores por meio de um nome, em vez do endereço de uma célula de memória.

2.6 DEFININDO UMA VARIÁVEL NO ALGORITMO

Todas as variáveis utilizadas em algoritmos devem ser definidas antes de serem utilizadas. Isto se faz necessário para permitir que o compilador reserve um espaço na memória para as mesmas.

Existem algumas linguagens de programação (como BASIC e FORTRAN) que dispensam esta definição, uma vez que o espaço na memória é reservado à medida que novas variáveis são encontradas no decorrer do programa. Nos algoritmos usaremos a definição de variáveis por assemelhar-se com as principais linguagens de programação como Pascal e C.

Nos algoritmos, todas as variáveis utilizadas serão definidas no início do mesmo, por meio de um comando de uma das seguintes formas:

VAR <nome_da_variável> : <tipo_da_variável>

ou

VAR <lista_de_variáveis> : <tipo_das_variáveis>

As regras para esta definição podem ser entendidas da seguinte maneira:

a) A palavra-chave VAR deverá estar presente sempre e será utilizada uma única vez na definição de um conjunto de uma ou mais variáveis;
b) Em uma mesma linha poderão ser definidas uma ou mais variáveis do mesmo tipo; neste caso, deve-se separar os nomes das mesmas por vírgulas;
c) Variáveis de tipos diferentes devem ser declaradas em linhas diferentes.

Exemplos de definição de variáveis:

VAR
nome: caracter[40]
numero_de_filhos: inteiro
custo: real
tem_filhos: lógico

No exemplo anterior foram declaradas quatro variáveis:

- A variável nome, capaz de armazenar dados caractere de comprimento 40 (40 caracteres), a exemplo: Fabrizio Leandro Fonsêca Fiscina;
- A variável numero_de_filhos, capaz de armazenar um número inteiro, a exemplo 3;
- a variável custo, capaz de armazenar um número real, a exemplo 13.987,45;
- a variável tem_filhos, capaz de armazenar uma informação lógica, a exemplo SIM.

2.7 CONCEITO E UTILIDADE DE CONSTANTES

Uma constante por definição é tudo aquilo que é fixo ou estável. Existirão vários momentos em que este conceito deverá estar em uso, quando desenvolvermos programas.

Geralmente é bastante comum definirmos uma constante no início do algoritmo, e a utilizarmos no

decorrer do mesmo, com o objetivo de facilitar o entendimento ou então para poupar tempo no caso de ter que alterar o seu valor, de modo que alterando uma única vez a declaração da constante, todos os comandos e expressões que a utilizam são automaticamente atualizados.

Nos algoritmos, todas as constantes utilizadas serão definidas no início do mesmo, por meio de um comando da seguinte forma:

CONST <nome_da_constante> = <valor>

Exemplo de definição de constantes:

```
CONST       pi = 3.14159
            nome_da_empresa = "Edutec Consultoria Ltda"
            numero_de_times = 30
```

Assim, o conceito, definição e utilização das constantes e variáveis constituem na base inicial para o desenvolvimento de algoritmos precisos e eficientes.

SUGESTÃO DE ATIVIDADE

1. Informe se os nomes das variáveis utilizados a seguir podem ser utilizados ou não:

- Valor (   )
- 2NOTA (   )
- MES1 (   )
- CASA (   )
- TEMPO (   )
- SALADRIO (   )
- ALTURA&HOMENS (   )
- ALTURA_MULHER (   )
- 1234 (   )

2. Qual o valor de X, Y e Z, sabendo que T ←5 e A ←10.

X ← T + A + 20
Y ← A + A – 5
Z ← T + X + Y

3. Abaixo, identifique se a variável deve ser declarada como inteira, real, lógica ou caracter.

VAR
endereço:
numero_de_netos:
preço: real
tem_carro:
desconto:
media:
telefone:
cidade:

4. No exemplo a seguir temos uma variável chamada de PRECO, que inicialmente recebe o valor 30,00, em seguida o valor é alterado para receber um desconto de 50%, ficando assim com um novo valor, que é informado no final. Entretanto, existe um erro no algoritmo. Indique o erro e a solução para equacionar este problema

```
Declarações
   inteiro PRECO;
Inicio
   escreva("Informe o preço: ");
   leia(PRECO);
   PRECO ← VALOR*50%
         escreva("O preço com desconto é:",PRECO)
Fim
```

INDICAÇÃO DE LEITURA

Forbellone, André. Lógica de Programação - A Construção de Algoritmos e Estruturas de Dados. São Paulo: Ed. Makron Books, 1993.

INSTRUÇÕES BÁSICAS DO ESTUDO DO ALGORITMO

CAPÍTULO 3

Anotações

CAPÍTULO 3 - INSTRUÇÕES BÁSICAS DO ESTUDO DO ALGORITMO

Neste capitulo, para que possamos iniciar o entendimento de alguns exemplos relacionados aos algoritmos, você encontrará informações sobre as instruções básicas do estudo do algoritmo (escreva e leia), posteriormente serão apresentadas as instruções de forma mais específicas. Também será apresentada a escrita linear (horizontalização) indispensável na construção dos algoritmos.

3.1 INSTRUÇÃO ESCREVER

Conhecer as instruções é algo importante para quem deseja realizar bons algoritmos. Basicamente existem duas instruções principais em algoritmos que são: Escrever e Ler. Neste capítulo veremos como funciona a instrução *Escreva*.

Na construção de algoritmos teremos palavras que determinaremos **reservada**, ou seja, toda palavra considerada reservada não mais poderá ser utilizada como nome de variável, de modo que toda a vez que for encontrada em algoritmos será identificada como tal. Neste caso, a palavra *Escreva* daqui por diante será considerada uma palavra reservada e determinará um comando de saída de dados.

A instrução *Escreva* é utilizada quando deseja-se mostrar informações na tela do computador, sendo denominada de comando de saída de dados. Para simplificar, usa-se a instrução *Escreva*, quando necessita-se mostrar algum dado para o usuário do algoritmo e posteriormente no programa).

Tanto no Diagrama de Chapin quanto em Português Estruturado representa-se a saída de dados através da palavra *Escreva*. Já em Fluxogramas a representação da saída de dados é feita através de uma forma geométrica específica.

Alguns exemplos da instrução *Escreva*:

Escreva um algoritmo para armazenar o valor 30 em uma variável X e o valor 10 em uma variável Y. A seguir, armazenar a soma do valor de X com o de Y em uma variável Z. Escrever na tela o valor armazenado em X, em Y e em Z.

X ←30

Y ←10

Z ← X+Y

Escreva (X)

Escreva (X)

Escreva (X)

Escreva um algoritmo para armazenar o valor 12 em uma variável CAP e o valor 8 em uma variável VAL. A seguir, armazenar a soma de CAP com VAL em uma variável TOT e a subtração de CAP com VAL em uma variável DOL. Escrever o valor de CAP, VAL, TOT e DOL e também escrever a mensagem ‘Fim do Algoritmo’.

CAP ← 12

VAL ← 8

TOT ← CAP + VAL

DOL ← CAP – VAL

Escreva CAP, VAL, TOT, DOL

Escreva “Fim do Algoritmo”

É importante ressaltar que quando queremos escrever alguma mensagem na tela (letra, frase, número etc.) literalmente, devemos utilizar aspas para identificar o que será escrito, pois o que estiver entre aspas no algoritmo, será exatamente o que aparecerá na tela do computador. Diferente de quando queremos escrever o conteúdo de uma variável, pois neste caso não utiliza-se aspas.

Veja como fica o uso das aspas:

IDADE1 ← 20

IDADE2 ← 30

MEDIA ← (IDADE1+INDADE2)

Escreva “A média geral é:”,MEDIA

Neste caso, a frase que está entre as aspas “A média é:” se mantém constante e o valor armazenado na variável MÉDIA será apresentado. Assim será exibida a seguinte informação na tela do computador: A média geral é 15.

SUGESTÃO DE ATIVIDADE

1. Desenvolva um algoritmo onde a variável A receba o valor 30, a variável B receba o valor 70 e a variável C receba o valor 100. Em seguida calcule e armazene na variável D a soma das variáveis A, B, C e na variável E a multiplicação de A por B. escreva o resultado na tela.

2. Escreva um algoritmo que armazene o valor 3 na variável F e o valor 10 na variável G. A seguir, armazenar a multiplicação do valor de F com o de G em uma variável H. Escrever na tela o valor armazenado em F, em G e em H.

3.2 INSTRUÇÃO LER

Como vimos anteriormente, a Instrução Escreva, existem basicamente duas instruções principais em algoritmos (e em programação em geral) que são: Escrever e Ler. No capítulo 4, foi apresentada a instrução *Escreva*, agora, neste capítulo, veremos como funciona a instrução *Leia*.

A instrução *Leia* é utilizada quando deseja-se obter informações do teclado do computador, ou seja, é um comando de entrada de dados. Para simplificar, usa-se a instrução *Leia*, quando necessita-se que o usuário do algoritmo digite algum dado (e posteriormente do programa).

VOCÊ SABIA?

Tanto no Diagrama de Chapin quanto em Português Estruturado representa-se a entrada de dados através da palavra Leia (ou ler). Já em Fluxogramas a representação da entrada de dados é feita através de uma forma geométrica específica.

Para melhor entender a instrução ler vamos utilizar esse exemplo do algoritmo Média visto no capitulo 2.

```
Algoritmo Média
Var NOTA1, NOTA2, MEDIA
Início
 Leia NOTA1,
         Leia NOTA2,
 MÉDIA ← (NOTA1+NOTA2)/2
 Se MEDIA >= 7 Então
         Escreva “Aprovado”
 Senão
         Escreva “Reprovado”
Fim.
```

Observamos que tanto o valor informado para a NOTA1 como para a NOTA2 são valores externos, diferentes de uma constante. São valores informados pelo usuário e vão variar conforme o usuário. Ao contrário da variável MEDIA que receberá um valor calculado e que não sofre nenhuma interferência externa.

3.3 HORIZONTALIZAÇÃO

Para o desenvolvimento de algoritmos que possuam cálculos matemáticos, as expressões aritméticas devem estar horizontalizadas, ou seja, linearizadas e também não devemos esquecer de utilizar os operadores corretamente. No exemplo a seguir, é apresentado um exemplo de uma expressão aritmética na forma tradicional e como deve ser utilizada nos algoritmos e em programação em geral (linearmente).
Na matemática tradicional a representação é

$$\text{MÉDIA} = \frac{\text{NOTA1}+\text{NOTA2}}{2}$$

No algoritmo a representação é MÉDIA ← (NOTA1+NOTA2)/2

As expressões matemáticas na forma horizontalizada não são apenas utilizadas em algoritmos, mas também na maioria das linguagens de programação.

OPERADORES E EXPRESSÕES

CAPÍTULO 4

Anotações

CAPÍTULO 4 - OPERADORES E EXPRESSÕES

Neste capitulo você conhecerá os principais operadores utilizados no desenvolvimento dos algoritmos, e também será apresentado às expressões / cálculos que, relacionados aos operadores, resultam em ações fundamentais para a construção dos algoritmos.

4.1 OPERADORES

Os operadores são elementos importantíssimos que atuam sobre operandos e produzem um determinado resultado. Por exemplo, a expressão matemática 1 + 2 relaciona dois operandos (os números 1 e 2) por meio do operador (+) que representa a operação de adição/soma.

Conforme o número de operandos sobre os quais os operadores atuam, estes podem ser classificados em:

- **binários**, quando atuam sobre dois operandos. Esta operação é chamada diádica. Ex.: os operadores das operações aritméticas básicas (soma, subtração, multiplicação e divisão).
- **unários**, quando atuam sobre um único operando. Esta operação é chamada monádica. Ex.: o sinal de (-) na frente de um número, cuja função é inverter seu sinal.

Também existem outras classificações dos operadores. Observando neste caso o tipo de dado de seus operandos e o valor resultante de sua avaliação. Nesta classificação, os operandos dividem-se em aritméticos, lógicos e literais. A classificação está diretamente relacionada com o tipo de expressão onde aparecem os operadores.

Existem ainda os operadores relacionais, que permitem comparar pares de operandos de tipos de dados iguais, resultando sempre num valor lógico.

4.1.1 Operadores de Atribuição

Um operador de atribuição serve para atribuir um valor a uma variável. Na construção dos algoritmos usamos o operador de atribuição: **:=**

A sintaxe de um comando de atribuição é:

Variável **:=** expressão

A expressão localizada no lado direito do sinal de igual é avaliada e seu valor resultante é armazenado na variável à esquerda. O nome da variável aparece sempre sozinho, no lado esquerdo do sinal de igual deste comando.

4.1.2 Operadores Aritméticos

Os operadores aritméticos se relacionam às operações aritméticas básicas e possuem prioridades diferentes, conforme o quadro 01:

Quadro 01 – Prioridade dos operadores e suas relações com as operações matemáticas básicas.

Operador	Tipo	Operação	Prioridade
+	Binário	Adição	4
-	Binário	Subtração	4
*	Binário	Multiplicação	3
/	Binário	Divisão	3
MOD	Binário	Resto da Divisão	3
DIV	Binário	Divisão Inteira	3
**	Binário	Exponenciação	2
+	Unário	Manutenção do Sinal	1
-	Unário	Inversão do Sinal	1

Fonte : PREUSS (2012)

A prioridade entre operadores define a ordem em que os mesmos devem ser avaliados dentro de uma mesma expressão.

4.1.3 Operadores Relacionais

Já os operadores relacionais são operadores binários que devolvem os valores lógicos verdadeiros e falsos.

Quadro 02 – Operadores Relacionais

Operador	Comparação
=	Igual
<>	Diferente
>	maior que
<	Menor que
>=	maior ou igual
<=	Menor ou igual

Fonte: PREUSS (2012)

Os operadores relacionais são somente usados quando se deseja efetuar comparações. Desta forma as comparações só podem ser feitas entre objetos de mesma natureza, isto é, variáveis do mesmo tipo de dado. O resultado de uma comparação é sempre um valor lógico

Por exemplo, digamos que a variável inteira *marco* contenha o valor 10. A primeira das expressões a seguir fornece um valor falso, e a segunda um valor verdadeiro:

marco <= 4 (valor falso, pois o valor de marco é 10, neste caso maior que 4)

marco > 4 (valor verdadeiro, pois marco é 10, neste caso maior que 4)

No caso da utilização dos operadores relacionais com valores de string (para escrever um texto, mais de um caracter é necessário. Na programação, precisamos representar essa cadeia de caracteres de alguma forma. A palavra “cadeia” em inglês é “string”), os operadores relacionais comparam os valores levando como base a tabela ASCII dos caracteres correspondentes em cada string. Uma string é dita menor que outra se os caracteres correspondentes tiverem os números de códigos ASCII menores. Por exemplo, todas as expressões a seguir são verdadeiras:

“informatica” > “INFORMATICA” (verdadeiro)

“ABC” < “EFG” (verdadeiro)
“computador” < “computador pessoal” (verdadeiro)

Note que as letras minúsculas têm códigos ASCII maiores do que os das letras maiúsculas. Observe também que o comprimento da string se torna o fator determinante na comparação de duas ou mais strings, quando os caracteres existentes na string menor são os mesmos que os caracteres correspondentes na string maior. Neste caso, a string maior é dita maior que a menor. Como exemplo de valores falsos temos:
“casa” > “grande casa” (falso)
“HIJ” > “TUV” (falso)

4.1.4 Operadores Lógicos

Os operadores lógicos, também são chamados de booleanos e são utilizados para combinar expressões relacionais. Também devolvem como resultado valores lógicos: verdadeiro ou falso, entre estes operadores também existe a relação de grau de prioridade.

Quadro 03 - Prioridade nos Operadores lógicos

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

Fonte: PREUSS (2012)

Uma expressão relacional ou lógica retornará falso para o valor lógico falso e verdadeiro para o valor lógico verdade.

Para melhor explicar, vamos fornecer dois valores ou expressões lógicas, representadas por expressãoA e expressãoB. Podemos descrever as operações lógicas no Quadro 04.

Quadro 04 - Descrição da operações lógicas na Conjunção

expressãoA	Operador	expressãoB	resultado
verdadeira	E	verdadeira	Verdadeira
verdadeira	E	falsa	Falsa
Falsa	E	verdadeira	Falsa
Falsa	E	falsa	Falsa

Fonte: PREUSS (2012)

Quadro 05 - Descrição da operações lógicas na Disjunção

expressãoA	Operador	expressãoB	resultado
verdadeira	OU	verdadeira	Verdadeira
verdadeira	OU	falsa	Verdadeira
Falsa	OU	verdadeira	Verdadeira
Falsa	OU	falsa	Falsa

Fonte: PREUSS (2012)

O operador NÃO na *expressãoA* avalia verdadeiro se *expressãoA* for falsa; de modo contrário, a expressão NÃO resultará em falso, se *expressãoA* for verdadeira.

Quadro 06 - Descrição da operações lógicas na Negação

Operador	expressãoB	Resultado
NÃO	verdadeira	Falsa
NÃO	falsa	Verdadeira

Fonte: PREUSS (2012)

4.1.5 Operadores Literais

Dentre os operadores que atuam sobre caracteres, o mais comum e mais usado é o operador que faz a concatenação de strings: toma-se duas strings e acrescenta-se (concatena-se) a segunda ao final da primeira.

O operador que faz esta operação é representado: **+**

Por exemplo, a concatenação das strings "INFOR" e "MÁTICA" é representada por:

"INFOR" + "MÁTICA" e o resultado de sua avaliação é: "INFORMÁTICA"

4.2 EXPRESSÕES

O conceito de expressão em termos computacionais está intimamente ligado ao conceito de expressão ou fórmula matemática, onde um conjunto de variáveis e constantes numéricas relacionam-se por meio de operadores aritméticos compondo uma fórmula que, uma vez avaliada, resulta num valor.

4.2.1 Expressões Aritméticas

Expressões aritméticas são aquelas cujo resultado da avaliação é do tipo numérico, seja ele inteiro ou real. Somente o uso de operadores aritméticos, variáveis numéricas e parênteses é permitido em expressões deste tipo.

É muito comum no desenvolvimento de algoritmos o uso de expressões matemáticas para a resolução de cálculos. A seguir são apresentados os operadores aritméticos necessários para determinadas expressões. Veja o quadro 07.

Quadro 07 - Prioridades nas operações aritméticas

Operação	Símbolo	Prioridade
Multiplicação (Produto)	*	1ª
Divisão	/	1ª
Adição (Soma)	+	2ª
Subtração (Diferença)	-	2ª

Fonte: PREUSS(2012)

Nas linguagens de programação e no desenvolvimento de algoritmos, as expressões matemáticas sempre obedecem às regras matemáticas comuns, ou seja:

- As expressões dentro de parênteses são sempre resolvidas antes das expressões fora dos parênteses. Quando existem vários níveis de parênteses, ou seja, um parêntese dentro de outro, a solução sempre inicia do parêntese mais interno até o mais externo (de dentro para fora).
- Quando duas ou mais expressões tiverem a mesma prioridade, a solução é sempre iniciada da expressão mais à esquerda até a mais à direita.

Assim, é possível estabelecer uma ordem de prioridade para a resolução das expressões. Veja a seguir alguns exemplos:

Exemplo: 2 + (5 * (3 + 2)) = 27
Exemplo: 2 + 4 * (3 + 2) = 22
Exemplo: 4 + (8 – (3 + 2)) = 7

4.2.2 Expressões Lógicas

As expressões lógicas são aquelas cujo resultado da avaliação é um valor lógico verdadeiro ou falso. Nestas expressões são usados os operadores relacionais e os operadores lógicos, podendo ainda serem combinados com expressões aritméticas.

Quando forem combinadas duas ou mais expressões que utilizem operadores relacionais e lógicos, os mesmos devem utilizar os parênteses para indicar a ordem de precedência, como foi apresentado anteriormente.

4.2.3 Expressões Literais

As expressões cujo resultado da avaliação é um valor literal (caractere) são chamadas de expressões Literais. Neste tipo de expressões só é utilizado o operador de literais (+).

4.2.4 Avaliação de Expressões

Como já foi visto anteriormente, as expressões podem ser avaliadas, e aquelas que apresentam apenas um único operador podem ser avaliadas diretamente. No entanto, à medida que as mesmas vão tornando-se mais complexas com o aparecimento de mais de um operando na mesma expressão, é necessária a avaliação da mesma passo a passo, tomando um operador por vez.

A seqüência destes passos é definida de acordo com o formato geral da expressão, considerando-se a prioridade de avaliação de seus operadores e a existência ou não de parênteses na mesma.

As seguintes regras são essenciais para a correta avaliação de expressões:

1. Deve-se observar a prioridade dos operadores, conforme mostrado nas tabelas de operadores: operadores de maior prioridade devem ser avaliados primeiro. Se houver empate com relação à precedência, então a avaliação se faz da esquerda para a direita.
2. Os parênteses são usado em expressões para definir um maior grau de prioridade dos demais operadores, forçando a avaliação da subexpressão em seu interior.
3. Entre os grupos de operadores existentes, a saber, aritmético, lógico, literal e relacional, há uma prioridade de avaliação: os aritméticos e literais devem ser avaliados primeiro; a seguir, são avaliadas as subexpressões com operadores relacionais e, por último os operadores lógicos são avaliados.

SUGESTÃO DE ATIVIDADE

1. Dados as variáveis e operações:

```
v1 := 10
v2 := 5 + v1
v1 := v2 * 2
```

Como fazer para segurar e mostrar o valor inicial da variável v1 no final das operações?

2. Como fazer para passar o valor de uma variável para outra e vice-versa?

3. Quais as operações necessárias para intercambiar os valores de 3 variáveis a, b e c de modo que a fique com o valor de b; b fique com o valor de c e c fique com o valor de a?

4. Se x possui o valor 10 e foram executadas as seguintes instruções:

```
x := X + 6
X := X - 4
X := X / 2
X := 6 * X
```

Qual será o valor armazenado em x?

INDICAÇÃO DE LEITURA

Farrer, Harry. Algoritmos Estruturados. Rio de Janeiro: Editora Guanabara Koogan S.A, 1989. 252p.
______.Programação Estruturada de Computadores. Rio de Janeiro: Ed. LTC, 1989.

CAPÍTULO 5 INSTRUÇÕES

Anotações

CAPÍTULO 5

INSTRUÇÕES

Anotações

Neste capitulo reforçaremos o conceito de instruções, e apresentaremos as estruturas de controle do fluxo de execução, mostrando como o algoritmo ganha sua forma e responde a cada instrução e estrutura de controle.

As instruções são os comandos básicos que efetuam tarefas essenciais para a operação dos computadores, como entrada e saída de dados e movimentação dos mesmos na memória. Estes tipos de instrução estão presentes na absoluta maioria das linguagens de programação.

Como já foi visto anteriormente, é necessário entender o que representa a sintaxe e a semântica para uma melhor compreensão das instruções. Assim, a sintaxe é a forma como os comandos devem ser escritos, a fim de que possam ser entendidos pelo tradutor de programas. A violação das regras sintáticas é considerada um erro sujeito à pena do não reconhecimento por parte do tradutor

Já a semântica é o significado, ou seja, o conjunto de ações que serão exercidas pelo computador durante a execução do referido comando. Deste ponto em diante, todos os comando novos serão apresentados por meio de sua sintaxe e sua semântica, isto é, a forma como devem ser escritos e as ações que executam.

5.1 COMANDOS DE ATRIBUIÇÃO

O comando de atribuição ou simplesmente atribuição, é a principal maneira de armazenar uma informação numa variável. Sua sintaxe é:

<nome_da_variável> := *<expressão>*

Exemplos:
```
Nome := "fabrizio"
preco := 25.67
quant := 10
media : idade * quant
imposto := total * 17 / 100
```

O modo de funcionamento (semântica) de uma atribuição consiste:

1. Na avaliação da expressão;
2. No armazenamento do valor resultante na variável que aparece à esquerda do comando.

Na seqüência, apresentamos um exemplo de um algoritmo utilizando o comando de atribuição:

Algoritmo exemplo_comando_de_atribuição

Var preço_unit, preço_total : **real**
    quantidade : **inteiro**

Início
    preco_unit := 10.0
    quantidade := 50
    preço_total := preço_unit * quantidade

Fim.

5.2 COMANDOS DE SAÍDA DE DADOS

Os comandos de saída de dados é a forma pela qual as informações contidas na memória dos computadores são colocadas nos dispositivos de saída, para que os usuários possam apreciá-las. Existem quatro sintaxes possíveis para esta instrução:

- ESCREVA *<variável>*
  Ex: Escreva QUANTIDADE
- ESCREVA *<lista_de_variáveis>*
  Ex: Escreva NOME, SOBRENOME, RUA, BAIRRO, CIDADE
- ESCREVA *<literal>*
  Ex: Escreva "Estou aprendendo a fazer Algoritmo!"
- ESCREVA *<literal>*, *<variável>*, ... ,*<literal>*, <variável>
  Ex: Escreva "Meu nome é:", NOME, "e meu endereço é:", ENDERECO

Uma lista_de_variáveis é um conjunto de nomes

de variáveis separados por vírgulas. Um *literal* é simplesmente um dado do tipo literal (string ou cadeia de caracteres) delimitado por aspas.

A semântica da instrução primitiva de saída de dados é muito simples: os argumentos do comando são enviados para o dispositivo de saída. No caso de uma lista de variáveis, o conteúdo de cada uma delas é pesquisado na memória e enviado para o dispositivo de saída. No caso de argumentos do tipo literal ou string, estes são enviados diretamente ao referido dispositivo.

Existe ainda a possibilidade de se combinar nomes de variáveis com literais na lista de um mesmo comando. O efeito obtido é bastante útil e interessante: a lista é lida da esquerda para a direita e cada elemento da mesma é tratado separadamente; se um nome de variável for encontrado, então a informação da mesma é colocada no dispositivo de saída; no caso de um literal, o mesmo é escrito diretamente no dispositivo de saída.

A seguir, temos um exemplo de um algoritmo utilizando o comando de saída de dados:

Algoritmo exemplo_comando_de_saída_de_dados

Var preço_unit, preço_total : **real**

quantidade : **inteiro**

Início

preco_unit := 10.0

quantidade := 50

preço_total := preço_unit * quantidade

Escreva preço_total

Fim.

5.3 COMANDOS DE ENTRADA DE DADOS

Os comandos de entrada de dados correspondem ao meio pelo qual as informações dos usuários são transferidas para a memória dos computadores, para que possam ser usadas nos programas. Já tivemos um contato inicial com este comando no capitulo 2 quando falamos da instrução *ler*.

Formalmente existem duas sintaxes possíveis para esta instrução:

- LEIA *<variável>*

  Ex: LEIA X

- LEIA *<lista_de_variáveis>*

  Ex: LEIA NOME, RUA, BAIRRO, CIDADE

Da mesma forma que *Escreva*, estaremos utilizando deste ponto em diante *Leia* como uma palavra-reservada e não mais poderá ser usada como nome de variável em algoritmos. A *lista_de_variáveis* é um conjunto de um ou mais nomes de variáveis separados por vírgulas.

A semântica da instrução de entrada ou leitura de dados é, de certa forma, inversa à da instrução de escrita: os dados são fornecidos ao computador por meio de um dispositivo de entrada e armazenados nas posições de memória das variáveis cujos nomes aparecem na *lista_de_variáveis.*

A seguir, será apresentado um exemplo de um algoritmo utilizando o comando de entrada de dados:

Algoritmo exemplo_comando_de_entrada_de_dados

Var preço_unit, preço_total : **real**

quantidade : **inteiro**

Início

Leia preco_unit, quantidade

preço_total := preço_unit * quantidade

Escreva preço_total

Fim.

5.4 INTERFACE COM USUÁRIO

A interface com o usuário basicamente é o meio pelo qual o programa e o usuário interagem. Esta interface deve ser desenvolvida com o intuito de oferecer um programa "amigável ao usuário". Em termos práticos, isto se resume à aplicação de duas regras básicas:

1. Toda vez que um programa estiver esperando que o usuário forneça a ele um determinado dado (operação de leitura), ele deve antes enviar uma mensagem dizendo ao usuário o que ele deve digitar, por meio de uma instrução de saída de dados;
2. Antes de enviar qualquer resultado ao usuário, um programa deve escrever uma men-

sagem explicando o significado do mesmo.

Estas regas tornam o diálogo entre o usuário e o desenvolvedor muito mais fácil. A seguir, temos um exemplo do algoritmo anterior, utilizando as regras de construção de uma interface amigável:

Algoritmo exemplo_interface_amigavel

Var preço_unit, preço_total : **real**

quantidade : **inteiro**

Início

Escreva "Digite o preço unitário:"

Leia preco_unit

Escreva "Digite a quantidade:"

Leia quantidade

preço_total := preço_unit * quantidade

Escreva "O Preço total da compra é: ", preço_tot

Fim.

5.5 FUNÇÕES MATEMÁTICAS

As funções servem para diminuir o trabalho do desenvolvedor e tornar a resposta a determinados problemas mais exata. O seu uso pode variar de desenvolvedor para desenvolvedor e também entre as linguagens de programação. Entre as principais funções, temos:

Quadro 08 - Principais funções matemáticas utilizadas para construção de algoritmos.

Função	Ação Prevista
ABS (x)	Retorna o valor absoluto de uma expressão
ARCTAN (x)	Retorna o arco de tangente do argumento utilizado
COS (r)	Retorna o valor do co-seno
EXP (r)	Retorna o valor exponencial
FRAC (r)	Retorna a parte fracionária
LN (r)	Retorna o logaritmo natural
SQR (r)	Retorna o parâmetro elevado ao quadrado.
SQRT (r)	Retorna a raiz quadrada.

Fonte: PREUSS (2012)

As funções muitas vezes não estão prontas e às vezes é necessário utilizar uma fórmula equivalente, ou uma combinação de mais de uma função, veja no quadro 09:

Quadro 09 - Exemplos de combinações de funções

Expressão	Fórmula prevista
Y^X	= EXP (LN (Y) * X)
$\sqrt[x]{Y}$	= EXP (LN (Y) * (1 / X))
LOG (x)	= LN (X) / LN (10)

Fonte: PREUSS (2012)

Anotações

ESTRUTURAS DE CONTROLE

CAPÍTULO 6

Anotações

CAPÍTULO 6 - ESTRUTURAS DE CONTROLE

Até agora os algoritmos estudados utilizam apenas instruções básicas de atribuição e de entrada e saída de dados. Qualquer conjunto de dados fornecido a um algoritmo destes será submetido ao mesmo conjunto de instruções, executadas sempre na mesma seqüência.

Entretanto, na prática, muitas vezes é necessário executar ações diversas em função dos dados fornecidos ao algoritmo. Ou seja, dependendo do conjunto de dados de entrada do algoritmo, deve-se executar um conjunto diferente de instruções. Além disso, pode ser necessário executar um mesmo conjunto de instruções um número repetido de vezes. Resumindo, se faz necessário controlar o fluxo de execução das instruções, ou seja, a seqüência em que as instruções são executadas em um determinado algoritmo em função dos dados fornecidos como entrada do mesmo.

As estruturas básicas de controle são classificadas de acordo com o modo como o controle do fluxo de instruções de um algoritmo é feito. Assim temos:

- Estruturas seqüenciais;
- Estruturas de decisão; e
- Estruturas de repetição.

6.1 COMANDOS COMPOSTOS

Um comando composto é um conjunto de zero ou mais comandos (ou instruções) simples, como atribuições e instruções básicas (instruções primitivas) de entrada ou saída de dados, ou alguma das construções apresentadas neste capítulo.

6.2 ESTRUTURA SEQÜENCIAL

Na estrutura seqüencial os comandos de um algoritmo são executados numa seqüência pré-estabelecida. Cada comando é executado somente após o término do comando anterior.

As palavras-reservadas **Início** e **Fim** delimitam uma estrutura seqüencial e contém basicamente comandos de atribuição, comandos de entrada e comandos de saída. Os algoritmos do capítulo anterior são algoritmos que utilizam uma única estrutura seqüencial.

Um algoritmo puramente seqüencial é aquele cuja execução é efetuada em ordem ascendente dos números que identificam cada passo. A passagem de um passo ao seguinte é natural e automática, e cada passo é executado uma única vez.

6.3 ESTRUTURAS DE DECISÃO

Na estrutura de decisão o fluxo de instruções a ser seguido é escolhido em função do resultado da avaliação de uma ou mais condições. Uma condição representa então uma expressão lógica.

Existe uma classificação para as estruturas de decisão. Assim, a classificação das estruturas de decisão é feita de acordo com o número de condições que devem ser testadas para que se decida qual o caminho a ser seguido. Segundo esta classificação, têm-se 3 tipos de estruturas de decisão:

- Estrutura de Decisão Simples (*Se ... então*)
- Estrutura de Decisão Composta (*Se ... então ... senão*)
- Estrutura de Decisão Múltipla do Tipo Escolha (*Escolha ... Caso ... Senão*)

Os algoritmos puramente seqüenciais podem ser usados na solução de um grande número de problemas, porém, existem problemas que exigem o uso de alternativas de acordo com as entradas do mesmo. Nestes algoritmos, as situações são resolvidas através de passos cuja execução é subordinada a uma condição. Assim, o algoritmo conterá passos que são executados somente se determinadas condições forem observadas.

Um algoritmo em que se tem a execução de

determinados passos subordinada a uma condição é denominado algoritmo com seleção.

6.3.1 Estruturas de Decisão Simples (Se ... então)

A estrutura de Decisão Simples (se...então) prevê que uma única condição (expressão lógica) seja avaliada. Dependendo do resultado desta avaliação, um comando ou conjunto de comandos serão executados (se a avaliação for verdadeira) ou não serão executados (se a avaliação for falsa).

Para a estrutura de decisão simples existem duas sintaxes possíveis:

- SE <*condição*> ENTÃO <*comando_único*>
  Ex: SE X > 15 ENTÃO Escreva "X é maior que 15"
- SE <*condição*> ENTÃO
  INÍCIO
  <*comando_composto*>
  FIM
  Ex: SE X > 15 ENTÃO
  INÍCIO
  cont := cont + 1
  soma := soma + x
  Escreva "X é maior que 15"
  FIM

A semântica desta construção é a seguinte: a condição é avaliada:

- Se o resultado for verdadeiro, então o *comando_único* ou o conjunto de comandos (*comando_composto*) delimitados pelas palavras-reservadas ***início*** e ***fim*** serão executados. Ao término de sua execução, o fluxo do algoritmo prossegue pela instrução seguinte à construção, ou seja, o primeiro comando após o *comando_único* ou a palavra-reservada *fim*.
- No caso da condição ser falsa, o fluxo do algoritmo prossegue pela instrução seguinte à construção, ou seja, o primeiro comando após o *comando_único* ou a palavra-reservada *fim*, sem executar o *comando_único* ou o conjunto de comandos (*comando_composto*) entre as palavras-reservadas *início* e *fim*.

Exemplo de algoritmo que lê um número e escreve se o mesmo é maior que 15:

Algoritmo exemplo_estrutura_de_decisão_simples

Var X : inteiro

Início

Leia X

Se X > 15 **Então Escreva** "X é maior que 15"

Fim.

6.3.2 Estruturas de Decisão Composta (*Se ... então ... senão*)

Nesta estrutura, uma única condição (expressão lógica) é avaliada. Se o resultado desta avaliação for verdadeiro, um comando ou conjunto de comandos serão executados. Caso contrário, ou seja, quando o resultado da avaliação for falso, um outro comando ou um outro conjunto de comandos serão executados.

Existem duas sintaxes possíveis para a estrutura de decisão composta:

- SE <*condição*> ENTÃO <*comando_único_1*>
  SENÃO <*comando_único_2*>

  Ex: SE X > 1000 ENTÃO
  Escreva "X é maior que 1000"
  SENÃO
  Escreva "X não é maior que 1000"
- SE <*condição*> ENTÃO
  INÍCIO
  <*comando_composto_1*>
  FIM
  SENÃO
  INÍCIO
  <*comando_composto_2*>
  FIM

  Ex: **SE** X > 1000 **ENTÃO**
  INÍCIO
  cont_a := cont_a + 1
  soma_a := soma_a + x
  Escreva "X é maior que 1000"

```
                    FIM
                    SENÃO
                    INÍCIO
                    cont_b := cont_b + 1
                    soma_b := soma_b +
x
                    Escreva "X não é maior
que 1000"
                    FIM
```

A semântica desta construção é a seguinte: a condição é avaliada:

- Se for verdadeiro, então o *comando_único_1* ou o conjunto de comandos (*comando_composto_1*) delimitados pelas palavras-reservadas *início* e *fim* serão executados. Ao término de sua execução, o fluxo do algoritmo prossegue pela instrução seguinte à construção, ou seja, o primeiro comando após o *comando_único_2* ou a palavra-reservada *fim* do *comando_composto_2*.
- Se a condição é avaliada como falsa, o *comando_único_2* ou o conjunto de comandos (*comando_composto_2*) delimitados pelas palavras-reservadas *início* e *fim* serão executados. Ao término de sua execução, o fluxo do algoritmo prossegue pela instrução seguinte à construção, ou seja, o primeiro comando após o *comando_único_2* ou a palavra-reservada *fim* do *comando_composto_2*.

Exemplo de algoritmo que lê um número e escreve se o mesmo é ou não maior que 1000:

Algoritmo exemplo_estrutura_de_decisão_composta

Var X : inteiro

Início

Leia X

Se X > 1000 **Então Escreva** "X é maior que 1000"

Senão Escreva "X não é maior que 1000"

Fim.

Na construção de algoritmos, a correta formulação de condições ou expressões lógicas, é de fundamental importância, visto que as estruturas de seleção são baseadas nelas. A colocação de uma expressão errada pode alterar totalmente o objetivo final do algoritmo. As diversas formulações das condições podem levar a algoritmos distintos. Considerando o problema:

"Dado um par de valores **x**, **y**, que representam as idades de duas pessoas, determinar se ambas podem votar, se elas não podem votar, ou se alguma delas apenas pode votar."

A solução do problema consiste em determinar todas as combinações de ***x*** e ***y*** para as classes de valores positivos, negativos e nulos.

Os algoritmos podem ser baseados em estruturas concatenadas uma em seqüência a outra ou em estruturas aninhadas uma dentro da outra, de acordo com a formulação da condição.

O algoritmo a seguir utiliza estruturas concatenadas.

Algoritmo estruturas_concatenadas

Var x, y : inteiro

Início

Ler x, y

Se x>15 e y>15 **Então Escrever** "Ambas podem votar"

Se x<15 e y>15 **Então Escrever** "Apenas y pode votar"

Se x>15 e y<15 **Então Escrever** "Apenas x pode votar"

Se x<15 e y<15 **Então Escrever** "Nenhuma pode votar"

Fim.

O algoritmo a seguir utiliza estruturas aninhadas ou encadeadas.

Algoritmo estruturas_aninhadas

Var x, y : inteiro

Início

Ler x, y

Se x>15

Então Se y>15

Então Escrever "Ambas podem votar"

Senão Se y<15

Então Escrever “Apenas x pode votar”

Senão Se y>15

Então Escrever “Apenas y pode votar”

Senão Escrever “Nenhuma pode votar”

Fim.

A vantagem do uso das estruturas concatenadas é tornar o algoritmo mais legível, facilitando a correção do mesmo em caso de erros. O uso das estruturas aninhadas ou encadeadas tem a vantagem de tornar o algoritmo mais rápido, pois são efetuados menos testes e menos comparações, o que resulta num menor número de passos para chegar ao final do mesmo.

VOCÊ SABIA?

Geralmente as estruturas concatenadas são mais utilizadas nos algoritmos, isso porque neste caso existe uma maior facilidade de entendimento das mesmas. Por outro lado, o uso das estruturas aninhadas ou encadeadas é aplicado em casos em que seu uso é fundamental.

6.3.3 Estruturas de Decisão Múltipla do Tipo Escolha (Escolha...caso...senão)

Neste tipo de estrutura é utilizada uma generalização da construção *Se*, onde somente uma condição era avaliada e dois caminhos podiam ser seguidos. Na utilização da estrutura de decisão do tipo *Escolha* pode haver uma ou mais condições a serem testadas e um comando diferente associado a cada uma destas.

A sintaxe da construção de *Escolha* é:

```
ESCOLHA
    CASO <condição_1>
        <comando_composto_1>
    CASO <condição_2>
        <comando_composto_2>
    ...
    CASO <condição_n>
        <comando_composto_n>
    SENÃO
        <comando_composto_s>
FIM
```

Basicamente, o funcionamento da estrutura de decisão múltipla (escolhar) é o seguinte:

Ao entrar-se numa construção do tipo *Escolha*, a *condição_1* é testada: se for verdadeira, o *comando_composto_1* é executado e após seu término, o fluxo de execução prossegue pela primeira instrução após o final da construção (*fim*).

Porém, se a *condição_1* for falsa, a *condição_2* é testada: se esta for verdadeira, o *comando_composto_2* é executado e ao seu término, a execução prossegue normalmente pela instrução seguinte ao final da construção (*fim*).

O mesmo raciocínio é estendido a todas as condições da construção. No caso em que todas as condições são avaliadas como falsas, o *comando_composto_s* (correspondente ao *Senão* da construção) é executado.

O caso do *comando_composto_s* não conter nenhuma instrução, ocorre nas situações em que não se deseja efetuar nenhuma ação quando todas as condições são falsas. Assim, pode-se dispensar o uso do *Senão* na construção *Escolha*.

Um exemplo de aplicação desta construção é o algoritmo para reajustar o salário de acordo com a titulação do professor. Se for Doutor, aumentar o salário 50%, se for Mestre, aumentar 30% e se for outra titulação, aumentar 20%.

```
Algoritmo exemplo_estrutura_do_tipo_escolha
Var salário, salário_reaj : real
    prof: caracter[20]
Início
  Leia salário, prof
  Escolha
      Caso prof = “Doutor”
              salário_reaj := 1.5 * salário
      Caso prof = “Mestre”
              salário_reaj := 1.3 * salário
      Senão
              salário_reaj := 1.2 * salário
  Fim
```

Escreva "Salário Reajustado = ", salário_reaj

Fim.

6.4 ESTRUTURAS DE REPETIÇÃO

As aplicações da estrutura de repetição são inúmeras. Toda a situação em que se deseja repetir um determinado trecho de um programa um certo número de vezes é passível do uso da estrutura de repetição. Por exemplo, pode-se citar o caso de um algoritmo que calcula a soma dos números primos entre 100 e 1000.

As estruturas de repetição são muitas vezes chamadas de *Laços* ou também de *Loops*.

Os algoritmos que possuem um ou mais de seus passos repetidos um determinado número de vezes denomina-se algoritmo com repetição.

A classificação das estruturas de repetição é feito de acordo com o conhecimento prévio do número de vezes que o conjunto de comandos será executado. Assim, os *Laços* se dividem em:

- **Laços Contados**, quando se conhece previamente quantas vezes o comando composto no interior da construção será executado;
- **Laços Condicionais**, quando não se conhece de antemão o número de vezes que o conjunto de comandos no interior do laço será repetido, pelo fato do mesmo estar amarrado a uma condição sujeita à modificação pelas instruções do interior do laço.

Com a utilização de estruturas de repetição para a elaboração de algoritmos, torna-se necessário o uso de dois tipos de variáveis para a resolução de diversos tipos de problemas*: variáveis contadoras* e *variáveis acumuladoras*, conforme pode ser observado a seguir.

- **Variável contadora** é uma variável que recebe um valor inicial, geralmente 0 (zero) antes do início de uma estrutura de repetição, e é incrementada no interior da estrutura de um valor constante, geralmente 1, conforme o exemplo abaixo:

Exemplo:

...

cont := 0

<estrutura_de_repetição>

...

cont := cont + 1

...

<fim_da_estrutura_de_repetição>

...

- **Variável acumuladora** é uma variável que recebe um valor inicial, geralmente 0 (zero) antes do início de uma estrutura de repetição, e é incrementada no interior da estrutura de um valor variável, geralmente a variável usada na estrutura de controle, conforme o exemplo abaixo:

Exemplo

...

soma := 0

<estrutura_de_repetição_com_variável_x>

...

soma := soma + x

...

<fim_da_estrutura_de_repetição>

...

6.4.1 Laços Contados *(Para ... faça)*

Os laços contados são úteis quando se conhece previamente o número exato de vezes que se deseja executar um determinado conjunto de comandos. Então, este tipo de laço nada mais é que uma estrutura dotada de mecanismos para contar o número de vezes que o corpo do laço é executado.

Neste caso existem duas sintaxes possíveis usadas em algoritmos para os laços contados. Vamos utilizar um algoritmo para calcular e escrever a tabuada de 9 de 1 até o número 10, para exemplificar:

- **PARA** *<variável>* **DE** *<início>* **ATÉ** *<final>* **FAÇA** *<comando_único>*

  Ex.: **Para** i **De** 1 **Até** 10 **Faça** **Escrever** i, " x 9 = ", i * 9

- **PARA** *<variável>* **DE** *<início>* **ATÉ** *<final>* **FAÇA**

INÍCIO
<comando_composto>
FIM

Exemplo: somatorio := 0
Para i **De** 1 **Até** 10
Faça
Início
somatorio := somatorio + i
Escrever i, " x 9 = ", i * 9
Escrever "Soma acumulada = ", somatorio
Fim

A semântica do laço contado é a seguinte: no início da execução da construção o valor *início* é atribuído à variável *var*. A seguir, o valor da variável *var* é comparado com o valor *final*. Se *var* for maior que *final*, então o comando composto não é executado e a execução do algoritmo prossegue pelo primeiro comando seguinte ao *comando_único* ou à palavra-reservada *fim* que delimita o final da construção. Por outro lado, se o valor de *var* for menor ou igual a *final*, então o comando composto no interior da construção é executado e, ao final do mesmo, a variável *var* é incrementada em 1 unidade. Feito isso, retorna-se à comparação entre *var* e *final* e repete-se o processo até que *var* tenha um valor maior que *final*, quando o laço é finalizado e a execução do algoritmo prossegue pela instrução imediatamente seguinte à construção.

Existe uma condição especial em que a contagem deve ser de forma decrescente, onde o valor da variável é decrementado em uma unidade. A sintaxe deste laço é a seguinte:

PARA *<variável>* **DE** *<início>* **ATÉ** *<final>*
PASSO -1 **FAÇA**
INÍCIO
<comando_composto>
FIM

Exemplo de um algoritmo que escreve a tabuada de um número específico:

Algoritmo tabuada

Var i, tab, num : inteiro

Início

Escrever "Tabuada: "

Ler tab

Escrever "Até que número: "

Ler num

Para i **de** 1 **Até** num **Faça**

Início

Escrever i, " x ", tab, " = ", i * tab

Fim

Fim.

6.4.2 Laços Condicionais

Os Laços condicionais são aqueles cujo conjunto de comandos em seu interior é executado até que uma determinada condição seja satisfeita. Ao contrário do que acontece nos laços contados, nos laços condicionais não se sabe de antemão quantas vezes o corpo do laço será executado.

As construções que programam laços condicionais mais comuns na construção de algoritmos são:

- **Enquanto** - laço condicional com teste no início;
- **Repita** - laço condicional com teste no final.

Nos laços condicionais a variável que é testada, tanto no início quanto no final do laço, dever sempre estar associada a um comando que a atualize no interior do laço. Caso isso não ocorra, o programa ficará repetindo indefinidamente este loop, gerando uma situação conhecida como "*loop infinito*".

6.4.3 Laços Condicionais com Teste no Início (Enquanto ... faça)

Nesta estrutura é efetuado um teste lógico no início de um laço, verificando se é permitido ou não executar o conjunto de comandos no interior do laço.

A sintaxe é mostrada a seguir:

ENQUANTO *<condição>* **FAÇA**
INÍCIO
<comando_composto>
FIM

A semântica ocorre da seguinte forma: ao início da construção *Enquanto* a condição é testada. Se o resultado for falso, então o comando composto no seu

interior não é executado e a execução prossegue normalmente pela instrução seguinte à palavra-reservada *fim* que identifica o final da construção.

Se a condição for verdadeira, o comando composto é executado e ao seu término retorna-se ao teste da condição. Assim, o processo acima será repetido *enquanto* a condição testada for verdadeira. Quando esta for falsa, o fluxo de execução prossegue normalmente pela instrução seguinte à palavra-reservada fim que identifica o final da construção.

Uma vez dentro do corpo do laço, a execução somente abandonará o mesmo quando a condição for falsa. Neste tipo de construção deve-se prestar muita atenção quanto à necessidade de que em algum momento a condição deverá se comportar como falsa, pois caso contrário, o programa permanecerá indefinidamente no interior do laço (loop infinito). Neste tipo de laço condicional a variável a ser testada deve possuir um valor associado antes da construção do laço.

O algoritmo que escreve os números maiores que 0 enquanto a sua soma não ultrapasse 2000 é um exemplo deste tipo de laço:

```
Algoritmo exemplo_enquanto
Var soma, num : inteiro
Início
soma := 0
num := 1
Enquanto soma < 2000 Faça
    Início
    Escreva num
    num := num + 1
    soma := soma + num
    Fim
Fim.
```

6.4.4 Laços Condicionais com Teste no Final (Repita ... até que)

Os laços condicionais com teste no final caracterizam-se por possuir uma estrutura que efetua um teste lógico no final de um laço, verificando se é permitido ou não executar novamente o conjunto de comandos no interior do mesmo.

A sintaxe do repita...até é mostrada a seguir:

REPITA

<comando_composto>

ATÉ QUE *<condição>*

O funcionamento é bastante parecido com o da construção *Enquanto*. O comando é executado uma vez. Na sequencia, a condição é testada: se ela for falsa, o comando composto é executado novamente e este processo é repetido até que a condição seja verdadeira, quando então a execução prossegue pelo comando imediatamente seguinte ao final da construção.

Esta construção difere da construção *Enquanto* pelo fato de o comando composto ser executado *uma* ou mais vezes (pelo menos uma vez), ao passo que na construção *Enquanto* o comando composto é executado nenhuma ou mais vezes. Isto acontece porque na construção *Repita* o teste é feito no final da construção, ao contrário do que acontece na construção *Enquanto*, quando o teste da condição é efetuado no início.

A construção *Repita* também difere da construção *Enquanto* no que se refere à inicialização da variável, visto que na construção *Repita* a variável pode ser inicializada ou lida dentro do loop.

Como exemplo: construir um algoritmo que leia um número não determinado de vezes um valor do teclado e escreve o valor e o seu quadrado, até que seja digitado o valor zero para ser finalizado:

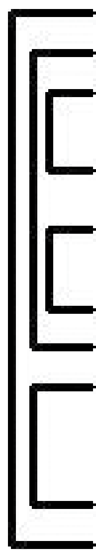
```
Algoritmo exemplo_repita
Var num : inteiro
Início
Repita
    Ler num
    Escrever num, " - ", num * num
Até que num mod 2 = 0
Fim.
```

6.5 ESTRUTURAS DE CONTROLE ENCADEADAS OU ANINHADAS

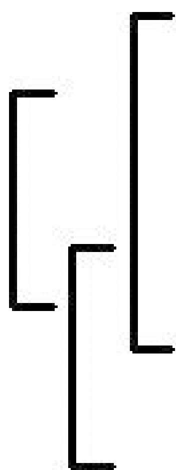
Um encadeamento é o fato de se ter qualquer um dos tipos de construção apresentados anteriormente dentro do conjunto de comandos de uma outra construção, também chamado de aninhamento.

Em qualquer tipo de aninhamento é necessário que a construção interna esteja completamente embutida na construção externa.

A seguir, são apresentados exemplos de aninhamentos válidos e inválidos.



[Figura 7: Aninhamento correto]
Fonte: PREUSS (2012)



[Figura 8: Aninhamento errado]
Fonte:PREUSS (2012)

INDICAÇÃO DE LEITURA

SALIBA, Walter Luís Caram. Técnicas de Programação: Uma Abordagem Estrutura. São Paulo: Ed. Makron Booksl, 1992.

SANTANA, João. **Algoritmos & Programação**. Disponível em: <http://www.iesam.com.br/paginas/cursos/ec/1ano/aulas/08/joao/APunidade-1.pdf>. Acesso em: 27 jul. 2010.

ESTRUTURAS DE DADOS HOMOGÊNEAS

CAPÍTULO 7

Anotações

CAPÍTULO 7 - ESTRUTURAS DE DADOS HOMOGÊNEAS

O estudo apresentado neste capitulo permitirá agrupar diversas informações dentro de uma mesma variável, permitindo assim o uso de tabelas, vetores e matrizes no algoritmo.

Como dito anteriormente, as estruturas de dados homogêneas permitem agrupar diversas informações dentro de uma mesma variável. Este tipo de estrutura de dados recebe diversos nomes, como: variáveis indexadas, variáveis compostas, variáveis subscritas, arranjos, vetores, matrizes, tabelas em memória ou arrays. Os nomes mais utilizados e que utilizaremos para estruturas homogêneas são: *matrizes* (genérico) e *vetores* (matriz de uma linha e várias colunas).

7.1 MATRIZES DE UMA DIMENSÃO OU VETORES

A utilização mais comum dos vetores está vinculada à criação de tabelas. Caracteriza-se por ser definida uma única variável vinculada dimensionada com um determinado tamanho. A dimensão de uma matriz é constituída por constantes inteiras e positivas. Os nomes dados às matrizes seguem as mesmas regras de nomes utilizados para indicar as variáveis simples.

Este tipo de estrutura em particular é também denominado por profissionais da área como matrizes unidimensionais.

A sintaxe do comando de definição de vetores é a seguinte:

Var

<nome_da_variável> **: MATRIZ [<***coluna_inicial***> .. <***coluna_final***>] DE** *<tipo_de_dado>*

Exemplo: **VAR**
M : MATRIZ [1 .. 10] DE INTEIRO

7.1.1 Operações Básicas com Vetores

Da mesma forma que acontece com as variáveis simples, também é possível operar com as matrizes. Contudo, não é possível operar diretamente com o conjunto completo, mas com cada um de seus componentes isoladamente.

O acesso individual a cada componente de um vetor é realizado pela especificação de sua posição na mesma por meio do seu índice. No exemplo anterior foi definida uma variável **M** capaz de armazenar 10 números inteiros. Para acessar um elemento deste vetor deve-se fornecer o nome do mesmo e o índice do componente desejado do vetor, neste exemplo um número de 1 a 10.

Por exemplo, **M[1]** indica o primeiro elemento do vetor, **M[2]** indica o segundo elemento do vetor e **M[10]** indica o último elemento do vetor.

Por isso, não é possível operar diretamente sobre vetores como um todo, mas apenas sobre seus componentes, um de cada vez.

7.1.2 Atribuição de Uma Matriz do Tipo Vetor

Quando falamos sobre as instruções anteriormente, o comando de atribuição foi definido como:
<nome_da_variável> := <expressão>

No caso de vetores, além do nome da variável deve-se necessariamente fornecer também o índice do componente do vetor onde será armazenado o resultado da avaliação da expressão.

Exemplo: M[1] := 25
M[3] := 100
M[7] := 12
M[10] := 30

7.1.3 Leitura de Dados de Uma Matriz do Tipo Vetor

A leitura de um vetor é feita passo a passo, um de seus componentes por vez, usando a mesma sintaxe da instrução da entrada, onde além do nome da variável, deve ser explicitada a posição do componente lido:

LEIA *<nome_da_variável> [<índice>]*

É importante observar que a utilização da construção *Para* a fim de efetuar a operação de leitura várias vezes, em cada uma delas lendo um determinado componente do vetor. Esta construção é muito comum quando se opera com vetores, devido à necessidade de se realizar uma mesma operação com os diversos componentes dos mesmos.

O algoritmo a seguir exemplifica a operação de leitura de 20 números de 1 até 20 de um vetor:

Algoritmo exemplo_leitura_de_vetor

Var

numeros : matriz[1..20] de inteiro

i : inteiro

Início

Para i **de** 1 **até** 20 **faça**

    Início

    Leia numeros[i]

    Fim

Fim.

7.1.4 Escrevendo Dados em um Vetor

Escrever em um vetor obedece à mesma sintaxe da instrução de saída de dados e também vale lembrar que, além do nome do vetor, deve-se também especificar por meio do índice o componente a ser escrito:

ESCREVA *<nome_da_variável> [<índice>]*

O algoritmo a seguir exemplifica a operação de leitura e escrita de um vetor, utilizando a construção *Para*:

Algoritmo exemplo_escrita_de_vetor

Var

numeros : matriz[1..20] de inteiro

i : inteiro

Início

Para i **de** 1 **até** 20 **faça**

    Início

    Leia numeros[i]

    Fim

Para i **de** 1 **até** 20 **faça**

    Início

    Escreva numeros[i]

    Fim

Fim.

Em outro exemplo, temos um vetor de dez números que é lido e guardado no vetor números. Paralelamente, a soma destes números é calculada e mantida na variável soma, que posteriormente é escrita.

Algoritmo exemplo_escrita_de_vetor_com_soma

Var

numeros : matriz[1..10] de inteiro

i : inteiro

soma : inteiro

Início

soma := 0

Para i **de** 1 **até** 10 **faça**

    Início

    Leia numeros[i]

    soma := soma + numeros[i]

    Fim

Para i **de** 1 **até** 10 **faça**

    Início

    Escreva numeros[i]

    Fim

Escrever “Soma = ”, soma

Fim.

A aplicação de vetores em algoritmos é muito vasta, mas normalmente os vetores são usados em duas tarefas muito importantes no processamento de dados: pesquisa e classificação.

- A **pesquisa** consiste na verificação da existência de um valor dentro de um vetor. Trocando em mi-

údos, pesquisar um vetor consiste em procurar dentre seus componentes um determinado valor.

- A **classificação** de um vetor consiste em arranjar seus componentes numa determinada ordem, segundo um critério específico. Por exemplo, este critério pode ser a ordem alfabética de um vetor de dados caracter. Há vários métodos de classificação, mas o mais conhecido é o método da bolha de classificação.

7.1.5 Método da Bolha de Classificação

Também conhecido como Bubble Sort este método não é o mais eficiente, mas é um dos mais populares devido à sua simplicidade.

Este método consiste em varrer o vetor, comparando os elementos vizinhos entre si. Caso estejam fora de ordem, os mesmos trocam de posição entre si. Procede-se assim até o final do vetor. Na primeira "varredura" verifica-se que o último elemento do vetor já está no seu devido lugar (no caso de ordenação crescente, ele é o maior de todos). A segunda "varredura" é análoga à primeira e vai até o penúltimo elemento. Este processo é repetido até que seja feito um número de varreduras igual ao número de elementos a serem ordenados menos um. Ao final do processo, o vetor está classificado segundo o critério escolhido.

O exemplo a seguir ilustra o algoritmo *bubble sort* para ordenar 30 número inteiros em ordem crescente:

Algoritmo Bubble_Sort

Var

numeros : matriz [1..30] de inteiros

aux, i, j: inteiro

Início

Para i **de** 1 **até** 30 **faça**

    Início

    Ler numeros[i]

    Fim

j := 30

Enquanto j > 1 **faça**

    Início

    Para i **de** 1 **até** j-1 **faça**

    Início

        Se numeros[i] > numeros[i+1] **Então**

            Início

        aux := numeros[i];

        numeros[i] := numeros[j];

        numeros[j] := aux;

            Fim

        Fim

    j:=j-1;

    Fim

Escreva "vetor ordenado: "

Para i **de** 1 **até 3**0 **faça**

    Início

    Escrever numeros[i]

    Fim

Fim.

Outra forma de uso deste algoritmo, com as mesmas características, utiliza duas construções *Para*, fazendo a comparação iniciando do primeiro elemento até o penúltimo, comparando com o imediatamente seguinte até o último elemento:

Algoritmo Outra_forma_do_Bubble_Sort

Var

numeros : matriz [1..30] de inteiros

aux, i, j: inteiro

Início

Para i **de** 1 **até** 30 **faça**

    Início

    Ler numeros[i]

    Fim

Para i **de** 1 **até** 29 **faça**

    Início

```
        Para j de i + 1 até 30 faça
                Início
                Se numeros[i] > numeros[j] Então
                    Início
                aux := numeros[i];
                numeros[i] := numeros[j];
                numeros[j] := aux;
                    Fim
                Fim
        Fim
Escreva "vetor ordenado: "
Para i de 1 até 30 faça
        Início
        Escrever numeros[i]
        Fim
Fim.
```

VOCÊ SABIA?

É possível observar que para ordenar o vetor em ordem decrescente basta inverter o sinal de comparação no teste da condição lógica Se numeros[i] > numeros[j], para: Se numeros[i] < numeros[j]

7.2 MATRIZES COM MAIS DE UMA DIMENSÃO

Esta estrutura, como a anterior, também tem sua principal utilização vinculada à criação de tabelas. Caracteriza-se por ser definida uma única variável vinculada dimensionada com um determinado tamanho. A dimensão de uma matriz é constituída por constantes inteiras e positivas. Os nomes dados às matrizes seguem as mesmas regras de nomes utilizados para indicar as variáveis simples.

A sintaxe do comando de definição de matrizes de duas dimensões é a seguinte:

Var

<nome_da_variável> : **MATRIZ** [*<linha_inicial>* .. *<linha_final>* , *<coluna_inicial>* .. *<coluna_final>*] **DE** *<tipo_de_dado>*

```
Exemplo:    VAR
            M : MATRIZ [1 .. 5 , 1 .. 10] DE INTEIRO
```

Também é possível definir matrizes com várias dimensões, por exemplo:

```
Exemplos:   VAR
            F : MATRIZ [1 .. 4] DE INTEIRO
            G : MATRIZ [1 .. 50 , 1 .. 4] DE INTEIRO
            H : MATRIZ [1 .. 5, 1 .. 50 , 1 .. 4] DE INTEIRO
            I : MATRIZ [1 .. 3 , 1 .. 5, 1 .. 50 , 1 .. 4] DE INTEIRO
```

A utilidade de matrizes desta forma é muito grande. No exemplo acima, cada matriz pode ser utilizada para armazenar uma quantidade maior de informações:

- A matriz F pode ser utilizada para armazenar 4 idades de pessoas.
- A matriz G pode ser utilizada para armazenar 4 idades de 50 pessoas.
- A matriz H pode ser utilizada para armazenar 4 Idades de 50 pessoas de 5 bairros.
- A matriz I pode ser utilizada para armazenar 4 Idades de 50 pessoas de 5 bairros, de 3 cidades.

7.2.1 Operações Básicas com Matrizes de Duas Dimensões

Do mesmo modo que acontece com os vetores, não é possível operar diretamente com o conjunto completo, mas com cada um de seus componentes isoladamente. Geralmente são chamadas simplesmente de matrizes.

Cada componente de uma matriz é acessado de forma individual através da especificação de sua posição na mesma por meio do seu índice. No exemplo anterior foi definida uma variável M capaz de armazenar 10 número inteiros em cada uma das 5 linhas. Para acessar um elemento desta matriz deve-se fornecer o nome da mesma e o índice da linha e

da coluna do componente desejado da matriz (um número de 1 a 5 para a linha e um número de 1 a 10 para a coluna, neste caso).

Por exemplo, M[1,1] indica o primeiro elemento da primeira linha da matriz, M[1,2] indica o segundo elemento da primeira linha da matriz, M[1,10] indica o último elemento da primeira linha da matriz e M[5,10] indica o último elemento da última linha da matriz

Da mesma forma como vetores, não é possível operar diretamente sobre matrizes como um todo, mas apenas sobre seus componentes, um por vez. Por exemplo, para somar duas matrizes é necessário somar cada um de seus componentes dois a dois. Da mesma forma, as operações de atribuição, leitura e escrita de matrizes devem ser feitas elemento a elemento.

7.2.2 Atribuição de Uma Matriz de Duas Dimensões

Da mesma forma vista anteriormente com os vetores, na atribuição de matrizes, além do nome da variável, deve-se necessariamente fornecer também o índice do componente da matriz onde será armazenado o resultado da avaliação da expressão. O índice referente ao elemento é composto por tantas informações quanto o número de dimensões da matriz.

Para matrizes com duas dimensões, o primeiro número se refere à linha e o segundo número se refere à coluna da matriz em que se encontra a informação

Ex.: M[1,1] := 10
M[1,10] := 15
M[3,5] := 25
M[5,10] := 35

7.2.3 Leitura de informações de Uma Matriz de Duas Dimensões

Neste caso, a leitura de uma matriz é feita passo a passo, um de seus componentes por vez, usando a mesma sintaxe da instrução básicas da entrada de dados, onde além do nome da variável, deve ser explicitada a posição do componente lido:

LEIA *<nome_da_variável> [<índice_1>, ... , < índice_n>]*

É importante ressaltar que é feito o uso de construções *Para* aninhadas ou encadeada a fim de efetuar a operação de leitura repetidas vezes, em cada uma delas lendo um determinado componente da matriz. Esse uso se faz necessário, devido à necessidade de se realizar uma mesma operação com os diversos componentes das mesmas.

O algoritmo a seguir exemplifica a operação de leitura de uma matriz:

Algoritmo exemplo_leitura_de_matriz

Var

numeros : matriz[1..5, 1..10] de inteiro

i, j : inteiro

Início

Para i **de** 1 **até** 5 **faça**

  Início

  Para j **de** 1 **até** 10 **faça**

    Início

    Leia numeros[i,j]

    Fim

  Fim

Fim.

7.2.4 Escrita de Dados de Uma Matriz de Duas Dimensões

A escrita de uma matriz obedece à mesma sintaxe da instrução de saída de dados e também vale lembrar que, da mesma forma que com vetores, além do nome da matriz, deve-se também especificar por meio do índice o componente a ser escrito:

ESCREVA *<nome_da_variável>* [*<índice_1>* , ... , *<índice_n>*]

O algoritmo a seguir exemplifica a operação de leitura e escrita de uma matriz, utilizando as construções *Para* aninhadas ou encadeadas:

Algoritmo exemplo_escrita_de_matriz

Var

numeros : matriz[1..5,1..10] de inteiro

i, j : inteiro

Início

```
Para i de 1 até 5 faça
    Início
    Para i de 1 até 10 faça
        Início
        Leia numeros[i,j]
        Fim
    Fim
Para i de 1 até 5 faça
    Início
    Para j de 1 até 10 faça
        Início
        Escreva numeros[i,j]
        Fim
    Fim
Fim.
```

A exemplo, temos uma matriz de 5 linhas por 10 colunas que é lida e guardada na matriz *numeros*. A seguir é efetuada e escrita a soma dos elementos da 2ª linha e também a soma dos elementos da 3ª coluna

```
Algoritmo exemplo_escrita_de_matriz_com_soma
Var
numeros : matriz[1..5,1..10] de inteiro
i, j : inteiro
somal2, somac3 : inteiro
Início
Para i de 1 até 5 faça
    Início
    Para i de 1 até 10 faça
        Início
        Leia numeros[i,j]
        Fim
    Fim
Para i de 1 até 5 faça
    Início
    Para i de 1 até 10 faça
        Início
        Escreva numeros[i,j]
        Fim
    Fim
somal2 := 0
somac3 := 0
Para j de 1 até 10 faça
    Início
    somal2 := somal2 + numeros[2,j]
    Fim
Para i de 1 até 5 faça
    Início
    somac3 := somac3 + numeros[i,3]
    Fim
Escrever "Soma Linha 2 = ", somal2
Escrever "Soma Coluna 3 = ", somac3
Fim.
```

INDICAÇÃO DE LEITURA

TERADA, Routo; SETZER, Valdemar W. Introdução à Computação e à Construção de Algoritmos. São Paulo: Ed. Makron Books, 1992.

SUBALGORITMOS

CAPÍTULO 8

Anotações

CAPÍTULO 8 - SUBALGORITMOS

Neste capitulo será apresentado o conceito de subalgoritmos, que na verdade, contempla a divisão do algoritmo em partes menores para facilitar e melhorar o controle do mesmo.

Quando um algoritmo se apresenta de forma extensa e complexa, existe a possibilidade de dividir problemas grandes e complicados em problemas menores e de solução mais simples. Assim, pode-se solucionar cada um destes pequenos problemas separadamente, criando algoritmos para tal (subalgoritmos). Posteriormente, pela justaposição destes subalgoritmos, elabora-se “automaticamente” um algoritmo mais complexo e que soluciona o problema original. Esta metodologia de trabalho é conhecida como Método de Refinamentos Sucessivos, cujo estudo é assunto de cursos avançados sobre técnicas de programação.

O conceito de subalgoritmo contempla um trecho de um algoritmo mais complexo e que, em geral, encerra em si próprio um pedaço da solução de um problema maior.

Em resumo, os subalgoritmos são importantes na:

- Subdivisão de algoritmos complexos, facilitando o seu entendimento;
- Estruturação de algoritmos, facilitando principalmente a detecção de erros e a documentação de sistemas; e
- Modularização de sistemas, que facilita a manutenção de softwares e a reutilização de subalgoritmos já implementados.

A reutilização de software tem sido adotada por muitos grupos de desenvolvimento de sistemas de computador, devido à economia de tempo e trabalho que proporcionam. Seu princípio é o seguinte: um conjunto de algoritmos destinado a solucionar uma série de tarefas bastante corriqueiras é desenvolvido e vai sendo aumentado com o passar do tempo, com o acréscimo de novos algoritmos. A este conjunto dá-se o nome de *biblioteca*.

Ao desenvolver sistemas, a equipe deve estar atenta a utilizar a concepção de subalgoritmos já existentes na biblioteca, de modo que a quantidade de software realmente novo que deve ser desenvolvido seja minimizada.

Geralmente, os subalgoritmos podem ser úteis para encerrar em si uma certa seqüência de comandos que é repetida várias vezes num algoritmo. Nestes casos, os subalgoritmos proporcionam uma diminuição do tamanho de algoritmos maiores. Antigamente, esta propriedade era tida como a principal utilidade dos subalgoritmos.

8.1 FUNCIONAMENTO DO SUBALGORITMO

Um algoritmo completo é dividido num algoritmo principal e diversos subalgoritmos (tantos quantos forem necessários e/ou convenientes). O algoritmo principal é aquele por onde a execução do algoritmo sempre se inicia. Este pode eventualmente invocar os demais subalgoritmos.

O algoritmo principal é sempre o último trecho do pseudocódigo de um algoritmo. As definições dos subalgoritmos estão sempre colocadas no trecho após a definição das variáveis globais e antes do corpo do algoritmo principal:

ALGORITMO *<nome do algoritmo>*

Var *<definição das variáveis globais>*

<definições dos subalgoritmos>

Inicio

<corpo do algoritmo principal>

Fim.

Assim, durante a execução do algoritmo princi-

pal, quando se encontra um comando de invocação de um subalgoritmo, a execução do mesmo é interrompida. A seguir, passa-se à execução dos comandos do corpo do subalgoritmo. Ao seu término, retoma-se a execução do algoritmo principal no ponto onde foi interrompida (comando de chamada do subalgoritmo) e prossegue-se pela instrução imediatamente seguinte. Ainda existe a possibilidade de que um subalgoritmo chame outro através do mesmo mecanismo.

8.2 ELEMENTOS DOS SUBALGORITMOS

Nos elementos que compõem um subalgoritmo constam:

- Cabeçalho, onde estão definidos o nome e o tipo do subalgoritmo, bem como os seus parâmetros e variáveis locais.
  - O nome de um subalgoritmo é o nome simbólico pelo qual ele é chamado por outro algoritmo. No corpo do subalgoritmo se encontram as instruções que são executadas cada vez que ele é invocado.
  - Variáveis locais são aquelas definidas dentro do próprio subalgoritmo e só podem ser utilizadas pelo mesmo.
  - Parâmetros são canais por onde os dados são transferidos pelo algoritmo chamador a um subalgoritmo, e vice-versa. Para que possa iniciar a execução das instruções em seu corpo, um subalgoritmo às vezes precisa receber dados do algoritmo que o chamou e, ao terminar sua tarefa, o subalgoritmo deve fornecer ao algoritmo chamador os resultados da mesma. Esta comunicação bidirecional pode ser feita de dois modos que serão estudados mais à frente: por meio de variáveis globais ou por meio da passagem de parâmetros.
- Corpo, onde se encontram as instruções (comandos) do subalgoritmo.

O tipo de um subalgoritmo é definido em função do número de valores que o subalgoritmo retorna ao algoritmo que o chamou. Segundo esta classificação, os algoritmos podem ser de dois tipos:

- Funções: que retornam um, e somente um, valor ao algoritmo chamador;
- Procedimentos: que retornam zero ou mais valores ao algoritmo chamador.

8.3 FUNÇÕES

O conceito de *Função* é originário da idéia de função matemática (por exemplo, raiz quadrada, seno, cosseno, tangente, logaritmo, entre outras), onde um valor é calculado a partir de outro(s) fornecido(s) à função.

A sintaxe da definição de uma função é dada a seguir:

FUNÇÃO *<nome>* (*<parâmetros>*) *<tipo_de_dado>*

VAR *<variáveis locais>*

INÍCIO

<comando composto>

FIM

Assim, para uma construção de um subalgoritmo com funções temos:

- *<nome>* é o nome simbólico pelo qual a função é invocada por outros algoritmos;
- *<parâmetros>* são os parâmetros da função;
- *<tipo de dado>* é o tipo de dado da informação retornado pela função ao algoritmo chamador;
- *<variáveis locais>* consiste na definição das variáveis locais à função. Sua forma é análoga à da definição de variáveis num algoritmo;
- *<comando composto>* é o conjunto de instruções do corpo da função.

No algoritmo, o comando de invocação de um subalgoritmo do tipo função sempre aparece dentro de uma expressão do mesmo tipo que o do valor retornado pela função.

A chamada de uma função é feita pelo simples aparição do nome da mesma, seguido pelos respectivos parâmetros entre parênteses, dentro de uma expressão. A função é executada e, ao seu fim, o trecho do comando que a chamou é substituído pelo valor retornado pela mesma dentro da expressão em que se encontra, e a avaliação desta prossegue.

O comando *Retorne <expressão>, quando dentro de uma função,* é usado para retornar o valor calculado pela mesma. Ao encontrar este comando, a expressão entre parênteses é avaliada, a execução da função é concluida neste ponto e o valor da expressão é retornado ao algoritmo chamador.

O algoritmo a seguir é um exemplo do emprego de função para calcular o valor de um número elevado ao quadrado.

Algoritmo Exemplo_de_função

Var X, Y : real

Função Quad(w : real) : real

Var Z : real

Inicio

Z := w * w

Retorne Z

Fim

Início

Escreva "Digite um número"

Leia X

Y := Quad(X)

Escreva X, " elevado ao quadrado é = ", Y

Fim.

No exemplo anterior é importante ressaltar que:

- A função Quad toma W como parâmetro do tipo real, retorna um valor do tipo real e possui Z como uma variável local real;
- O comando de invocação da função Quad aparece no meio de uma expressão, no comando de atribuição dentro do algoritmo principal.

8.4 PROCEDIMENTOS

Já vimos o conceito de função, agora, iremos entender sobre um procedimento. Na verdade, um procedimento é um subalgoritmo que retorna zero ou mais valores ao subalgoritmo ou algoritmo chamador. Estes valores são sempre retornados por meio dos parâmetros ou de variáveis globais, mas nunca explicitamente, como no caso de funções. Portanto, a chamada de um procedimento nunca surge no meio de expressões, como no caso de funções. Pelo contrário, a chamada de procedimentos só é feita em comandos isolados dentro de um algoritmo, como as instruções de leitura e escrita de dados.

A sintaxe da definição de um procedimento é:

PROCEDIMENTO *<nome>* (*<parâmetros>*)

Var *<variáveis locais>*

Inicio

<comando composto>

Fim.

Assim para uma construção de um subalgoritmo com procedimentos temos:

- ***<nome>*** é o nome simbólico pelo qual o procedimento é invocado por outros algoritmos;
- ***<parâmetros>*** são os parâmetros do procedimento;
- ***<variáveis locais>*** são as definições das variáveis locais ao procedimento. Sua forma é análoga à da definição de variáveis num algoritmo;
- ***<comando composto>*** é o conjunto de instruções do corpo do procedimento, que é executado toda vez que o mesmo é invocado.

Temos a seguir um exemplo simples, onde um procedimento é usado para escrever o valor das componentes de um vetor.

Algoritmo Exemplo_procedimento

Var vet : matriz[1..50] de real

Procedimento ESC_VETOR()

Var i : inteiro

Inicio

Para i **de** 1 **até** 50 **faça**

Início

Escreva vet[i]

Fim

Fim

Procedimento LER_VETOR()

Var i : inteiro

Inicio

Para i **de** 1 **até** 50 **faça**

Início

Leia vet[i]

Fim

Fim

Inicio

LER_VETOR()

ESC_VETOR()

Fim.

No exemplo vale ressaltar que:

- A forma de definição dos procedimentos LER_VETOR( ) e ESC_VETOR( ), que não possuem nenhum parâmetro, e usam a variável local *i* para "varrer" os componentes do vetor, lendo e escrevendo um valor por vez; e
- A forma de invocação dos procedimentos: por meio do seu nome, seguido de seus eventuais parâmetros, num comando isolado dentro do algoritmo principal.

8.5 VARIÁVEIS

O termo variável já foi visto anteriormente, entretanto, nesta parte, será abordado o conceito de variáveis reforçando o aprendizado e ampliando o entendimento sobre este componente do algoritmo. Existem dois tipos de variáveis: as globais e as locais, conforme podemos entender a seguir:

- **Variáveis globais** são aquelas declaradas no início de um algoritmo. Estas variáveis são visíveis (isto é, podem ser usadas) no algoritmo principal e por todos os demais subalgoritmos.
- **Variáveis locais** são aquelas definidas dentro de um subalgoritmo e, portanto, somente visíveis (utilizáveis) dentro do mesmo. Outros subalgoritmos, ou mesmo o algoritmo principal, não podem utilizá-las.

No exemplo a seguir são aplicados os conceitos de variáveis globais e locais:

Algoritmo Exemplo_variáveis_locais_e_globais

Var X : real

I : inteiro

Função FUNC() : real

Var X : matriz[1..5] de inteiro

Y : caracter[10]

Inicio

...

Fim

Procedimento PROC

Var Y : 1ógico

Inicio

...

X := 4 * X

I := I + 1

...

Fim

Início

...

X := 10.5

...

Fim.

É importante notar no exemplo anterior que:

- As variáveis X e I são globais e visíveis a todos os subalgoritmos, à exceção da função FUNC,

que redefine a variável X localmente;

- As variáveis X e Y locais ao procedimento FUNC não são visíveis ao algoritmo principal ou ao procedimento PROC. A redefinição local do nome simbólico X como uma matriz[5] de inteiro sobrepõe (somente dentro da função FUNC) a definição global de X como uma variável do tipo real;
- A variável Y dentro do procedimento PROC, que é diferente daquela definida dentro da função FUNC, é invisível fora deste procedimento;
- A instrução X := 10.5 no algoritmo principal, bem como as instruções X := 4 * X e I := I + 1 dentro do procedimento PROC, atuam sobre as variáveis globais X e I.

8.6 PARÂMETROS

Os parâmetros são canais pelos quais se estabelece uma comunicação bidirecional entre um subalgoritmo e o algoritmo chamador. Dados são passados pelo algoritmo chamador ao subalgoritmo, ou retornados por este ao primeiro por meio de parâmetros. Os parâmetros podem ser formais ou reais.

Parâmetros formais são os nomes simbólicos introduzidos no cabeçalho de subalgoritmos, usados na definição dos parâmetros do mesmo. Dentro de um subalgoritmo trabalha-se com estes nomes da mesma forma como se trabalha com variáveis locais ou globais.

Função Média(X, Y : real) : real

Inicio

Retorne (X + Y) / 2

Fim

No exemplo anterior, X e Y são parâmetros formais da função Média.

Já os parâmetros reais são aqueles que substituem os parâmetros formais quando da chamada de um subalgoritmo. Por exemplo, o trecho seguinte de um algoritmo invoca a função Média com os parâmetros reais 8 e 7 substituindo os parâmetros formais X e Y.

Z := Média(8, 7)

Assim, os parâmetros formais são úteis somente na formalização do subalgoritmo, ao passo que os parâmetros reais substituem-nos a cada chamada do subalgoritmo.

8.7 MECANISMOS DE PASSAGEM DE PARÂMETROS

Como foi visto anteriormente, os parâmetros reais substituem os formais no ato da invocação de um subalgoritmo. Esta substituição é denominada passagem de parâmetros e pode se dar segundo dois mecanismos distintos: passagem por valor ou passagem por referência.

8.7.1 Passagem de Parâmetros por Valor

Na passagem de parâmetros por valor, o parâmetro real é avaliado e uma cópia de seu valor é fornecida ao parâmetro formal, no ato da invocação do subalgoritmo. A execução do subalgoritmo prossegue normalmente e todas as modificações feitas no parâmetro formal não afetam o parâmetro real, pois trabalha-se apenas com uma cópia do mesmo.

Algoritmo Exemplo_parametro_por_valor

Var X : inteiro

Procedimento PROC(Y : inteiro)

Início

Y := Y + 1

Escreva "Durante Y = ", Y

Fim

Início

X := 1

Escreva "Antes X = ", X

PROC(X)

Escreva "Depois X = ", X

Fim.

No algoritmo anterior os seguintes resultados são encontrados:

Antes X = 1
Durante Y = 2
Depois X = 1

Uma observação importante é que o procedimento não alterou o valor do parâmetro real X durante sua execução.

8.7.2 Passagem de Parâmetros por Referência

No exemplo anterior, na passagem de parâmetros por valor não existe a alteração do valor do parâmetro real X, este tipo de ação é possível porque, neste mecanismo de passagem de parâmetros, é feita uma reserva de espaço em memória para os parâmetros formais, para que neles seja armazenada uma cópia dos parâmetros reais.

Neste mecanismo de passagem de parâmetros não é feita uma reserva de espaço em memória para os parâmetros formais. Quando um subalgoritmo com parâmetros passados por referência é chamado, o espaço de memória ocupado pelos parâmetros reais é compartilhado pelos parâmetros formais correspondentes. Assim, as eventuais modificações feitas nos parâmetros formais também afetam os parâmetros reais correspondentes.

Um mesmo subalgoritmo pode utilizar diferentes mecanismos de passagem de parâmetros, para parâmetros distintos. Para diferenciar uns dos outros, convencionou-se colocar o prefixo VAR antes da definição dos parâmetros formais passados por referência. Se, por exemplo, um procedimento tem o seguinte cabeçalho:

Procedimento PROC(X, Y : inteiro; **Var** Z : real; J: real)

Então:

- **X** e **Y** são parâmetros formais do tipo inteiro e são passados por valor;
- **Z** é um parâmetro formal do tipo real passado por referência;
- **J** é um parâmetro formal do tipo real passado por valor.

O exemplo do item anterior, alterado para que o parâmetro **Y** do procedimento seja passado por referência, torna-se:

Algoritmo Exemplo_parametro_por_referencia

Var X : inteiro

Procedimento PROC(Y : inteiro)

Início

Y := Y + 1

Escreva "Durante Y = ", Y

Fim

Início

X := 1

Escreva "Antes X = ", X

PROC(X)

Escreva "Depois X = ", X

Fim.

O resultado do algoritmo modificado é:

Antes X = 1
Durante Y = 2
Depois X = 2

Como podemos observar, depois de chamar o procedimento com o parâmetro por referência, o valor original da variável foi alterado.

8.8 REFINAMENTOS SUCESSIVOS

Chegamos a parte final do nosso estudo. No estudo do ciclo de vida de um software se encontram das seguintes fases:

- Análise de requisitos,
- Arquitetura geral,
- Projeto detalhado,
- Programação,
- Integração e testes, e
- Manutenção.

Posteriormente em outras disciplinas e estudos cada fase da vida de um software será explicada detalhadamente.

O conceito é parecido com o de subalgoritmo, porém mais amplo. O Método de Refinamentos Sucessivos para a elaboração de algoritmos dita que um dado problema deve ser subdividido em problemas menores repetidamente, até que seja possível encon-

trar um algoritmo (subalgoritmo ou comando composto) para resolver cada um destes subproblemas. Assim, o algoritmo para resolver o problema original será composto pela composição destes algoritmos.

O método apresentado pode ser subdividido na fase de Análise Top-Down, que procura resolver o problema a nível de subalgoritmos e comandos compostos, e na fase de Programação Estruturada, que detalha os subalgoritmos e comandos compostos até o nível das instruções primitivas e construções de controle de fluxo.

Para concluir, o uso do método é vantajoso devido ao seu estímulo à modularização de sistemas, que proporciona a criação de programas claros, simples, fáceis de entender e, portanto, de manutenção mais barata.

INDICAÇÃO DE LEITURA

SANTANA, João. Algoritmos & Programação. Disponível em: <http://www.iesam.com.br/paginas/cursos/ec/1ano/aulas/08/joao/APunidade-1.pdf>. Acesso em: 12 jul. 2010.
TONET, Bruno; KOLIVER, Cristian. Introdução aos Algoritmos. Disponível em: <http://dein.ucs.br/napro/Algoritmo/manuais/Manual20Visualg.pdf>. Acesso em: 21 mar. 2010

Anotações

LINGUAGEM DE
PROGRAMAÇÃO
PASCAL

CAPÍTULO 9

Anotações

CAPÍTULO 9 - LINGUAGEM DE PROGRAMAÇÃO PASCAL

Neste capitulo todos os conhecimentos estudados anteriormente poderão ser utilizados, muitos dos conceitos são similares na linguagem de programação Pascal. Assim, será apresentada uma breve introdução a essa linguagem que, apesar de não ser comercial, é muito importante para o entendimento da programação.

9.1 A ORIGEM DO PASCAL

A linguagem de programação Pascal é classificada como de alto nível e de fins genéricos, derivada do Algol-60.

VOCÊ SABIA?

A definição do ALGOL 60 foi um marco na história das linguagens de programação, pois foi a primeira linguagem de Programação estruturada.

As suas instruções são formadas por expressões do tipo algébrico e por algumas palavras inglesas, tais como BEGIN, END, READ, WRITE, IF THEN, REPEAT, WHILE, DO, etc. Neste aspecto o Pascal assemelha-se a muitas outras linguagens de alto nível. Contudo, o Pascal contém ainda alguns aspectos únicos, que foram especificamente concebidos para estimular o uso de uma programação estruturada - um método ordenado e disciplinado, do qual resultam programas claros, eficientes e sem erros.

VOCÊ SABIA?

VOCÊ SABIA? O termo Pascal foi atribuído em homenagem ao cientista e matemático francês, Blaise Pascal (1623-1662), que se destacou com a invenção da primeira calculadora mecânica.
A linguagem Pascal foi originalmente desenvolvida no início dos anos 70 por Nicklaus Wirth, na Universidade Técnica de Zurique, Suíça. O objetivo original foi o de desenvolver uma linguagem disciplinada de alto nível para ensinar programação estruturada. Esta linguagem desenvolvida por Wirth é conhecida como Pascal Padrão. Nos Estados Unidos, foi definido conjuntamente um padrão oficial para a linguagem, pelo American National Standarts Institute (ANSI) e pelo Institute of Electrical and Electronics Engineers (IEEE). Este padrão oficial é conhecido como ANSI Pascal. Em 1983 a Borland International começou a comercializar um compilador Pascal relativamente barato, designado por Turbo Pascal, para ser usado em computadores pessoais.

9.2 ELEMENTOS DA LINGUAGEM PASCAL

Geralmente uma linguagem de programação possui dois tipos de elementos: os elementos definidos pela linguagem e os elementos definidos pelo próprio usuário:

9.2.1 Elementos definidos pela linguagem Pascal

- Letras (alfanuméricas): A até Z
- Dígitos (numéricos): 0 até 9
- Símbolos Especiais + - * / := ; > <
- Palavras Reservadas ou Palavras Chave: array, goto, until
- Delimitadores: branco, final de linha ou comentário.

A linguagem Pascal tem ainda alguns identificadores predefinidos pela linguagem conhecidos como Identificadores Padrão. Podem ser constantes, tipos, variáveis, procedimentos ou funções e podem ser escritos tanto em minúsculo como em maiúsculo. Seguem abaixo algumas funções:

Abs	arqtan	boolean	Char	chr	Cos
Eof	eoln	Exp	False	input	Integer

9.2.2 Elementos definidos pelo usuário

- **Identificadores:** Um identificador é um símbolo definido pelo usuário que pode ser um rótulo, uma constante, um tipo, uma variável, um nome de programa ou subprograma (procedimento ou função). No PASCAL padrão somente os 8 primeiros caracteres são válidos; no TURBO PASCAL pode-se usar identificadores de até 127 caracteres sendo todos significativos e não há distinção entre maiúsculas e minúsculas.

- **Comentários:** comentários não tem função nenhuma para o compilador e serve apenas para aumentar a legibilidade e clareza do programa. Um comentário é iniciado por { ou (* e é encerrado por } ou *).

- **Endentação:** A endentação também não tem nenhuma função para o compilador e serve para tornar a listagem do programa mais clara dando hierarquia e estrutura ao programa.

9.3 TIPOS DE DADOS NO PASCAL

Uma variável pode assumir um tipo de dado específico, assim, o tipo de dado, além de esta relacionado a variável ele define as operações que podem ser feitas sobre esta variável.

Toda variável em um programa deve ser associada a um e somente um tipo de dado. Esta associação é feita quando a variável é declarada na parte de declaração de variáveis do programa. Uma vez declarada uma variável de qualquer tipo, poderemos utilizar um *comando de atribuição,* também chamado de *operador de atribuição*, para armazenar um valor na variável. São classificadas de duas formas com mais outros subgrupos: Tipos definidos pela máquina e Tipos definidos pelo usuário.

9.3.1 Tipos predefinidos pela linguagem na Pascal

O Turbo Pascal tem diversas categorias de tipos de dados padrão, dos quais os mais comumente usados são:

- Tipos Inteiros: A linguagem Pascal oferece cinco tipos inteiros: INTEGER, WORD, LONGINT, SHORTINT e BYTE. Estes tipos fornecem faixas amplas de valores de números inteiros. Quanto mais ampla a faixa, maior a memória necessária para os valores de um determinado tipo**.**
- Tipos de Números Reais: Um tipo de número real pode conter dígitos tanto antes como depois do ponto decimal, e pode estar sujeito aos limites da precisão limitada. Este tipo também é conhecido como ponto flutuante, pois o Turbo Pascal armazena os números reais em duas partes **distintas: os** dígitos significativos (chamados de mantissa) e o expoente, que indica a posição correta do ponto decimal.
- Tipos de caractere: O identificador padrão CHAR define no Turbo Pascal as variáveis do tipo caractere. Esta variável pode armazenar exatamente um caractere por vez.
- Tipo de string: Uma string é uma seqüência de caracteres. O comprimento de uma string é igual ao número de caracteres que ela contém. No Turbo Pascal, o comprimento máximo de uma string é de 255 caracteres; o menor comprimento de string possível é de 0 caracteres. O comprimento máximo de uma variável declarada simplesmente como STRING é de 255 caracteres. Entretanto, você pode especificar um comprimento máximo menor para uma variável string

VOCÊ SABIA?

O computador utiliza-se exclusivamente de estados lógicos na representação binária. Portanto, é necessário um código para estabelecer uma comunicação entre as solicitações do usuário e o computador. Esta necessidade acabou criando uma linguagem universal (código padrão). A princípio não existia um consenso e cada fabricante de computador procurava definir seu próprio código de comunicação, tornando a comunicação difícil e complexa. Assim o Instituto Americano de Normas (ANSI) estabeleceu um código denominado ASCII "American Standard Code For Information Interchange" que se tornou um padrão aos microcomputadores pessoais.
A tabela ASCII fornece um padrão, que pode conter pequenas alterações de um país para outro, permitindo que o microcomputador possa editar textos nas mais diversas línguas. O código ASCII original tem 128 caracteres

- **Tipos Booleanos:** Uma variável do tipo Booleano pode ser tanto TRUE (verdadeiro) como FALSE (falso). O Turbo Pascal fornece um conjunto de operações lógicas e relacionais, que produzem expressões que resultam em valores de TRUE ou FALSE.

- **Tipos definidos pelo usuário:** Os tipos definidos pelo usuário são aqueles que usam um grupo de tipos predefinidos ou um subgrupo de algum tipo. Este tipo é chamado de tipo enumerado de dados e representa uma escolha dentre um pequeno número de alternativas. Pode ser discreto ou contínuo.

- **Tipo enumerado discreto:** Na linguagem Turbo Pascal temos o comando TYPE para definir o tipo de dados que desejamos.

- **Tipo enumerado contínuo:** Já o tipo enumerado contínuo pode ser definido como um intervalo de um tipo enumerado discreto já definido ou de um tipo padrão.

9.4 CONSTANTES E VARIÁVEIS DO PASCAL

9.4.1 Constante

Como já vimos anteriormente, as constantes são valores declarados no início do programa e que não se alteram na execução do programa.

No Pascal uma constante deve sempre ser definida antes de aparecer. Esta definição tem duas finalidades:

- Estabelece que o identificador é uma constante;
- Associa um elemento de informação à constante.

O tipo da constante será implicitamente determinado pelo elemento de informação.

Exemplo:

CONST

nome = 'Edutec Consultoria';

media = 7;

9.4.2 Variáveis

Como também já foi visto anteriormente, um identificador cujo valor pode ser alterado durante a execução do programa é denominado *variável*.

Sempre no Pascal todas as variáveis devem ser declaradas antes de serem usadas. As variáveis devem ser declaradas no início de cada função, procedimento ou início do programa. Não podem ocorrer declarações de variáveis após a primeira sentença executável de uma rotina.

Uma declaração de variável consiste do nome da variável seguido do nome do tipo.

Como também já foi visto anteriormente, as variáveis podem ser globais ou locais no Pascal.

- Variáveis globais são declaradas fora de qualquer função, valem em qualquer ponto do programa, são inicializadas com zero automaticamente e uma única vez e são armazenadas na memória.
- Variáveis locais são declaradas dentro das funções e existem apenas enquanto a função na qual foi declarada está ativa. Dentro de funções variáveis locais com mesmo nome de variáveis globais tem preferência, não são inicializadas automaticamente, ou seja, deve-se informar um valor inicial, são alocadas na pilha ("stack").

Os parâmetros das funções são tratados como se fossem variáveis locais, são inicializados com o valor passado na chamada, são declarados na lista de parâmetros, são passados por valor e, somente tipos escalares de dados podem ser parâmetros.

VAR *nome1, nome2, ... , nomen : tipo* ;

Ex:

Var

nome: **string[50];**

salario: **real;**

filhos: **integer;**

sexo: **char;**

9.4.3 Operadores

Os Operadores vistos no Pascal seguem o conceito de operadores já visto anteriormente. Os principais operadores são:

- Operadores aritméticos: são utilizados para compor expressões, que podem ser formadas por números, constantes, variáveis, etc.
- Operadores de atribuição: serve para atribuir um valor a uma variável (**:=)**
- Operadores relacionais; são operadores binários que devolvem os valores lógicos: verdadeiro e falso (>, <, >=, <=, = <>).
- Operadores lógicos ou booleanos são usados para combinar expressões relacionais. Também devolvem como resultado valores lógicos verdadeiro ou falso. (and, or, not, xor, shl, shr).
- Operadores sobre strings: também chamado de c*oncatenação* é uma operação de string que combina duas strings. O símbolo para concatenação é o sinal de mais (+).
- Operadores sobre conjuntos: No Pascal existe o operador **IN** que avalia verdadeiro se um operando pertencer a um conjunto.

ESTRUTURA DO PROGRAMA DESENVOLVIDO EM PASCAL

CAPÍTULO 10

Anotações

CAPÍTULO 10 - ESTRUTURA DO PROGRAMA DESENVOLVIDO EM PASCAL

Neste capítulo trataremos os programas em pascal, seguindo os comandos e regras estudades anteriormente.

Basicamente um programa desenvolvido em Pascal possui três partes distintas:

- Identificação do programa;
- Bloco de declarações; e
- Bloco de comandos.

10.1 IDENTIFICAÇÃO DO PROGRAMA

A identificação , também chamada de cabeçalho do programa, em Pascal, dá um nome ao programa. É a primeira linha do programa.

PROGRAM <identificação>;

Todo identificador definido pelo usuário em um programa deve ser declarado antes de ser usado, caso contrário, o compilador acusará um erro na fase de compilação por desconhecer o identificador.

10.2 BLOCO DE DECLARAÇÕES

O Bloco de Declarações define todos os identificadores utilizados pelo Bloco de Comandos, sendo todos opcionais. Quando o Bloco de Declarações existir sempre estará antes do Bloco de Comandos. No Pascal, o Bloco de Declarações é formado por cinco partes:

Parte de Declarações de Rótulos: Rótulos (labels) existem para possibilitar o uso do comando de desvio incondicional GOTO. Este comando comando gera a desestruturação do programa e não é aconselhado seu uso.

LABEL <rotulo1>, ... , <rotulo*n*>;

Parte de Declarações de Constantes: Define os identificadores que terão valores constantes durante toda a execução do programa, podendo ser números, seqüências de caracteres (strings) ou mesmo outras constantes.

A declaração de constantes inicia pela palavra reservada CONST, seguida po uma seqüência de: identificador, um sinal de igual, o valor da constante e um ponto e vírgula:

CONST <identificador> = <valor>

Parte de Declarações de Tipos: Serve para que o usuário possa criar seus próprios tipos de dados. A declaração dos tipos é iniciadas pela palavra reservada TYPE, seguida de um ou mais identificadores separados por vírgula, um sinal de igual, um tipo e um ponto e vírgula:

TYPE <tipoident1> , ... , <tipoident*n*> = <tipo> ;

Parte de Declarações de Variáveis: A declaração das variáveis é iniciada pela palavra reservada VAR, seguida de um ou mais identificadores, separados por vírgula, um tipo um ponto e vírgula:

VAR < ident1> , ... , <ident*n*> : <tipo> ;

Parte de Declarações de Subprogramas: Nesta parte são declarados e implementados os subprogramas (funções e procedimentos) e devem estar declarados e implementados antes da sua chamada em qualquer parte do programa principal ou de subprogramas.

10.3 BLOCO DE COMANDOS

Um Bloco de Comandos é a última parte que compõe um programa em Pascal e especifica as ações a serem executadas sobre os objetos definidos no Bloco de Declarações. O Bloco de Comandos é também conhecido como programa principal e é iniciado pela palavra reservada **BEGIN** seguida de por uma seqüência de comandos e finalizada pela palavra reservada **END** seguida um ponto.

10.4 COMANDOS

Os comandos simples ou não estruturados, caracterizam-se por não possuírem outros comandos relacionados. Entre os principais comandos do Turbo Pascal temos:

10.4.1 Read e Readln

Na sua forma mais simples, a procedure embutida READLN aceita um valor de entrada a partir do teclado:

READLN *(NomedaVariável)*

Este comando espera que o usuário entre com o dado, e depois armazena a resposta do usuário na variável especificada. A entrada do usuário deve se corresponder com o tipo de dado da variável identificada no comando READLN. Quando o usuário terminar de digitar o valor de entrada e pressionar a tecla Enter, o valor será armazenado na variável, e a tarefa do comando READLN se completa.

Em um programa interativo, um comando READLN é normalmente usado precedido por um comando WRITE ou WRITELN, que coloca na tela uma mensagem apropriada. A mensagem geralmente informa ao usuário o tipo de informação que o programa espera como entrada. Por exemplo:

WRITE ('Nome do usuário: ');

READLN (Nome);

Esta seqüência coloca uma mensagem na tela e espera que o usuário entre com o nome do usuário.

READLN também lhe permite aceitar mais de um valor de entrada por vez. Podemos incluir uma lista de variáveis como argumento do comando READLN:

***READLN** (NomedaVariável1, Nomeda Variável2, NomedaVariável3,...)*

Esta forma do comando READLN não é, geralmente, muito útil para entradas pelo teclado. Um programa exerce um controle mais cuidadoso e harmonioso sobre o processo de entrada, se cada valor de entrada for tomado e aceito por comandos READLN individuais.

O comando READ lê um valor de dado de uma linha de entrada e mantém a posição do cursor ou ponteiro de arquivo, de modo que um comando READ ou READLN subseqüente possa ler dados adicionais da mesma linha. Este comando é, geralmente, mais importante em programas de manipulação de arquivos do que em programas que lêem dados do teclado.

O comando READLN provoca problemas inoportunos em alguns programas interativos. O problema surge quando um programa espera por um valor numérico e o usuário entra inadvertidamente com uma sequência de caracteres que o Turbo Pascal não pode ler como um número. Quando isto acontece, a execução do programa termina com um erro em tempo de execução.

10.4.2 Write e Writeln

O comando WRITELN exibe uma linha de informação na tela. O WRITE também envia dados para a tela, mas mantém a posição final do cursor, permitindo que um comando WRITE ou WRITELN subseqüente possa exibir informações adicionais na mesma linha da tela.

WRITE e WRITELN, tomam uma lista de argumentos de dados na forma de constantes, variáveis, ou expressões, separados por vírgulas.

Exemplo

```
PROGRAM ExemploWrite
USES Crt
VAR
quant, Preco_unitario: INTEGER;


BEGIN
quant:= 100;
Preco_unitario:=34;
WRITE ('O preço de ',quant);
WRITELN (' unidades é ',itens * Preco_unitario);
END.
```

Este programa em Pascal exibe uma linha com quatro valores de dados na tela: um valor literal string, o valor de uma variável numérica, um segundo valor string e o resultado de uma expressáo aritmética. A tela de saída resultante é:

O preço de 100 unidades é 3400

Neste exemplo, os comandos WRITE e WRITELN exibem todos os quatro valores de dados numa única linha de saída - as duas constantes string, o valor armazenado na variável *itens* e o resultado da expressão *itens* Precounit.*

O dispositivo default de saída para os comandos WRITE e WRITELN é a tela. Quando queremos enviar a saída para algum outro dispositivo, devemos especificar um nome de arquivo como primeiro argumento do comando WRITE ou WRITELN. Por exemplo, a unidade PRINTER define um arquivo chamado LST e associa este arquivo ao dispositivo LPT1 do DOS. Portanto, podemos utilizar LST para enviar dados para impressora a partir dos comandos WRITE e WRITELN. Para fazer isso, devemos incluir o comando USES, a seguir, no topo do programa:

10.4.3 Uses Printer

Quando especificamos subsequentemente LST como primeiro argumento de uma chamada à procedure WRITE ou WRITELN, a saída será enviada para a impressora, ao invés da tela:

```
Program ExemploPrinter;
USES CRT, PRINTER;
VAR
quant, Preco_unitario: INTEGER;
BEGIN
quant:= 100;
Preco_unitario:=34;
WRITE (LST,'O preço de ',quant);
WRITELN (LST,' unidades é ',itens * Preco_unitario);
END.
```

10.4.4 Readkey

O comando READKEY lê um caracter do teclado e associa a uma variável do tipo char: ***variável_char* := readkey;**

10.4.5 Goto

Este Comando de desvio incondicional altera a seqüência normal de execução em um bloco de comandos, transferindo o processamento para um ponto no programa fonte marcado com o rótulo especificado no comando GOTO. Uma dica é que devemos evitar, sempre que possível, o uso de comandos desse tipo, pois pode acarretar problemas quando mal utilizado.

Exemplo:

```
Program exemplo_label;
uses crt;
label ini, fim, gre, int;
var opcao: char;

begin
ini:
clrscr;
writeln('Escolha o seu time: [G] Gremio [I] Inter');
opcao:=readkey;
if upcase(opcao)='G' then goto gre;
if upcase(opcao)='I' then goto int;
goto ini;
gre:
   writeln('Ok, Gremista!');
   goto fim;
int:
   writeln('Muito bem, Colorado!');
   goto fim;
fim:
readkey;
end.
```

10.4.6 Exit

O comando exit faz com que o programa termine o bloco que está sendo executado. Se esse bloco for o programa principal, Exit o faz terminar. Se o bloco corrente é aninhado, Exit faz com que o próximo bloco mais externo continue na instrução imediatamente após a que passou o controle ao bloco aninhado.

QUADRO 10 - Tipos de interrupção com o comando exit

Sintaxe	Resposta
EXIT BREAK	Provoca a chamada do bloco para continuar na instrução após o ponto em que ele foi chamado.
BREAK CONTI-NUE	O comando exit faz com que o programa termine o bloco que está sendo executado. Somente pode ser usado no interior de laços **for**, **while** ou **repeat**..
CON-TINUE	Este procedimento faz com que o processamento de uma iteração num loop **for**, **while** ou **repeat** seja interrompido e continue com a próxima iteração.

Fonte: PREUSS (2012)

Exemplo:

```
Program exemplos;
uses crt;

var
i: integer;

begin
clrscr;
writeln('testando o break');
for i:= 1 to 10 do
   begin
   if i=5 then break;
   writeln(i);
   end;
readkey;
writeln('testando o continue');
for i:= 1 to 10 do
   begin
   if i=5 then continue;
   writeln(i);
   end;
readkey;
writeln('testando o exit');
for i:= 1 to 10 do
   begin
   if i=5 then exit;
   writeln(i);
   end;
writeln('Esta frase nao sera exibida!');
readkey;
end.
```

10.4.7 Runerror

O comando RUNNERROR determina o término do programa no local onde estiver sendo executado, gerando um erro de execução com o mesmo valor passado como parâmetro. Caso o parâmetro seja omitido, o valor de erro de execução será zero.

Sua sintaxe é: **Runerror (*[<valor>]*:** byte**);**

10.4.8 Clrscr

Este procedimento limpa a tela e coloca o cursor no canto superior esquerdo da tela. É a contração das palavras CLeaR SCReen. Geralmente é inserido no inicio do programa.

10.4.9 GotoXY

Permite posicionar o cursor em um ponto qualquer da tela, referenciado pelos eixos X e Y (colu-

na e linha), antes de um comando de entrada ou saída.

Sua sintaxe é: **GotoXY (*<coluna>, <linha>*:** byte**)**

10.4.10 Delay

O DELAY permite fazer uma pausa programada por um determinado intervalo de tempo, antes de ser executado o próximo comando.

Sua sintaxe é: **Delay (*<milissegundos>***: byte**)**

10.4.11 CHR

Esta função retorna o caractere ASCII correspondente ao valor do parâmetro passado.

Sua sintaxe é: **CHR(*<n>***: byte**)**:char**;**

10.4.12 ORD

Esta função retorna a posição ordinal do parâmetro passado. Sua sintaxe é: **ORD(*<x>*)**: longint;

10.4.13 UPCASE

O comando UPCASE retorna o caractere contido no parâmetro em maiúsculo.

Sua sintaxe é: **Upcase(*<ch>*:**char**)**: char**;**

Exemplo

program exemplos;

uses crt;

var x: char;

begin

clrscr;

writeln('Digite uma letra minuscula: ');

x:=readkey;

writeln('A letra digitada é: ',x);

writeln('O seu valor na tabela ASCII é: ', **ord**(x));

writeln('A letra maiuscula é: ',**upcase**(x));

writeln('O seu valor na tabela ASCII é: ', **ord(upcase(x))**);

writeln('O caracter 65 da tabela ASCII é: ',**chr(65)**);

writeln('O caracter 75 da tabela ASCII é: ',#75);

readkey;

end.

10.5 ESTRUTURAS DE CONTROLE

As estruturas de controle vistas anteriormente possuem aplicações similares no Pascal, a maioria possuem sintexa exatamente igual ao uso do algoritmo.

10.5.1 Seqüência de comandos

No Pascal a seqüência é delimitada pelas palavras reservadas **BEGIN** no início e **END** no final e seus comando são separados pelo delimitador ";" (ponto e vírgula).

10.5.2 Comandos Condicionais

- **Comando IF (condição) THEN (....) ELSE (...)**

O comando **if** é usado para executar um segmento de código condicionalmente.

Sua sintaxe é:

IF *condição* THEN

{ *comandos que* serão executados se a condição for TRUE}

ELSE

{ *comandos que* serão executados se a condição for FALSE}

No Pascal, durante uma execução, se inicia uma avaliação da a condição da cláusula IF. Esta condição deve resultar num valor do tipo Booleano, embora possa tomar uma variedade de formas - ela pode ser uma expressão de qualquer combinação de operadores lógicos ou relacionais, ou simplesmente uma variável BOOLEAN.

Caso a condição for TRUE, o comando ou comandos localizados entre o IF e ELSE são executados e a execução pula a cláusula ELSE. Alternativamente, se a expressão for FALSE, a execução é desviada diretamente para o comando ou comandos localizados após a cláusula ELSE.

É importante observar que a estrutura do IF e ELSE são sempre seguidas de blocos BEGIN/END, que delimitam grupos de comandos. O formato geral desta

estrutura é:

IF *condição* THEN

BEGIN

{*comandos da cláusula* IF}

END

ELSE

BEGIN

{*comandos da cláusula* ELSE}

END;

VOCÊ SABIA?

Atenção deve ser dada para não esquecer dos marcadores BEGIN e END nos comandos compostos de várias estruturas IF. Além disso, não devemos colocar um ponto-e-virgula entre o END e o ELSE numa estrutura IF. O Pascal interpretará essa marcação incorreta como o fim do comando IF e a cláusula ELSE, a seguir, resultará num erro em tempo de compilação.

Considerando que a utilização do ELSE é opcional, a forma mais simples da estrutura IF é a seguinte:

IF *condição* THEN

{*comando ou comandos executados* se (if) a condição *for* TRUE *(verdadeira)*}

Desta forma , o programa executa os comandos localizados após a cláusula IF somente se a condição for verdadeira. Se a condição for falsa, o comando IF não terá nenhuma ação.

As estruturas IF e ELSE podem ser alinhadas, aparecer dentro de uma cláusula IF ou ELSE de uma outra estrutura. O aninhamento de estruturas IF pode resultar em seqüências de decisão complexas e poderosas.

- **Comando de Seleção múltipla: CASE**

O comando CASE (comando de seleção múltipla) é utilizando quando o desenvolvedor deseja testar uma variável ou uma expressão em relação a diversos valores.

O comando CASE divide uma sequência de possiveis ações em seções de código individuais. Para a execução de um determinado comando CASE, somente uma dessas seções será selecionada para execução. A seleção está baseada numa série de testes de comparação, sendo todos executados sobre um valor desejado.

A forma geral da estrutura CASE é a seguinte:

CASE *seleção* OF

ValordeCaso1:

{*comando ou comandos executados se a* seleção *se corresponder com* ValordeCaso1 }

ValordeCaso2:

{*comando ou comandos executados se a* seleção se *corresponder com* ValordeCaso2 }

. . .

ELSE

{*comando ou comandos executados se nenhum dos casos anteriores produzir uma correspondência* }

END;

Uma estrutura de decisão CASE consiste nos seguintes elementos:

- Uma cláusula CASE OF, que fornece a expressão de *seleção* desejada
- Uma seqüência de *ValoresdeCaso* a ser comparada com a seleção, cada qual seguida de um ou mais comandos, que serão executados se a seleção se corresponder com o *ValordeCaso*
- Um comando ELSE opcional, que fornece um comando ou grupo de comandos a serem executados somente se não tiver ocorrido nenhuma correspondência com os *ValoresdeCaso* anteriores
- Um marcador END, que identifica o fim da estrutura CASE.

10.5.3 Comandos de Repetição

Os comandos de repetição são caracterizados por permitir que uma seqüência de comandos seja executada um número repetido de vezes.

- **Comando For**

O laço FOR é, talvez, a estrutura de repetição mais familiar e comumente usada, sendo aplicada para executar uma seqüência de comandos repetida-

mente e com um número conhecido de vezes.
A representação geral da sintaxe da estrutra é:
FOR *VardeControle := ValorInicial* TO *ValorFinal* DO *comando;*
Se o bloco do loop possui comandos múltiplos, o formato geral será o seguinte:
FOR *VardeControle := ValorInicial* TO *ValorFinal* DO

BEGIN

(comandos executados para cada iteração do loop}

END;

O comando FOR identifica uma variável de controle *(VardeControle)* para o loop, e indica a faixa de valores *(ValorInicial* TO [até] *ValorFinal)* que a variável receberá e o *ValorInicial* e o *ValorFinal* devem ser compatíveis com esse tipo. Aqui está um esboço da execução do loop:

1. No início da execução, a variável de controle recebe o valor do *ValorInicial,* e a primeira iteração é executada.
2. Antes de cada iteração subseqüente, a variável de controle recebe o próximo valor da faixa de *ValorInicial .. ValorFinal.*
3. O loop termina após a iteração correspondente ao *ValorFinal.*

Exemplo:

```
Program ExemploFor;
Uses Crt;
Var i, j: integer;
begin
for i:= 1 to 10 do
    begin
    writeln('Tabuada do ',i);
    for j:= 1 to 10 do
        begin
        writeln(i, ' x ', j , ' = ', i*j);
        end;
    end;
end.
```

- **While e Repeat Until**

Diferentemente do loop FOR, os loops WHILE e REPEAT UNTIL dependem de uma condição expressa para determinar a duração do processo de repetição. Existem também algumas diferenças importantes entre estes dois tipos de loops:

Num loop WHILE, colocamos a condição no início do loop, e as iterações continuam **enquanto** a condição seja TRUE (verdadeira). Tendo em vista que a condição é avaliada sempre antes de cada iteração, um loop WHILE resulta em nenhuma iteração se a condição for FALSE (falsa) logo no principio.

No REPEAT UNTIL, a condição vai para o fim do loop, e o processo de repetição continua **até que** a condição se torne TRUE (verdadeira). Tendo em vista que a condição é avaliada após cada iteração, um loop REPEAT UNTIL sempre executa pelo menos uma iteração.

A sintaxe do loop WHILE é:
WHILE *condição* DO *comando;*
Se o bloco do loop possuir comandos múltiplos, a forma geral é a seguinte:
WHILE *condição* DO

BEGIN

{*comandos que serão executados uma vez em cada iteração do loop*}

END;

A *condição* é uma expressão que o Turbo Pascal pode avaliar como TRUE (verdadeira) ou FALSE (falsa). A repetição continua enquanto a condição for TRUE. Normalmente, em algum ponto, a ação dentro do loop altera a condição para FALSE e o loop termina.
A Estrutura do Loop REPEAT UNTIL é:
REPEAT

{*comandos que serão executados uma vez* a cada iteração do loop}

UNTIL *condição*

Esta estrutura de loop sempre realiza uma iteração antes da condição ser avaliada pela primeira vez, e continua as suas iterações até que a condição se torne TRUE; a repetição termina quando a condição se tornar TRUE. Os comandos múltiplos dentro de um loop REPEAT UNTIL não necessitam de marcadores BEGIN e END. As palavras reservadas REPEAT e UNTIL servem como delimitadores para os comandos

dentro do loop.

Exemplo:

```
Program ExemploWhile
uses crt;
Var op: char;
procedure imprimir;
begin
...
end;
procedure excluir;
begin
...
end;
Begin
op := 0;
while op <> '9' do
     begin
     clrscr;
     write('1 - Imprimir');
     write('2 - Excluir');
     write('9 - Sair');
     case op of
          '1': imprimir;
          '2': excluir;
          end;
      end;
end.
```

```
Program ExemploRepeat
uses crt;
Var op: char;
procedure imprimir;
begin
...
end;

procedure excluir;
begin
...
end;

Begin
repeat
clrscr;
write('1 - Imprimir');
write('2 - Excluir');
write('9 - Sair');
case op of
     '1': imprimir;
     '2': excluir;
      end;
until op = '9'
end.
```

SUGESTÃO DE ATIVIDADE

1. Escreva um algoritmo/programa em Pascal para ler, calcular e escrever a média aritmética entre dois números.

2. Escreva um algoritmo/programa em Pascal para ler um número positivo qualquer, calcular e escrever o quadrado e a raiz quadrada do mesmo.

3. Escrever um algoritmo/programa em Pascal que lê a hora de início de um jogo e a hora de término do jogo, ambas subdivididas em 2 valores distintos, a saber: horas e minutos. Calcular e escrever a duração do jogo, também em horas e minutos, considerando que o tempo máximo de duração de um jogo é de 12 horas e que o jogo pode iniciar em um dia e terminar no dia seguinte.

4. Escrever um algoritmo/programa em Pascal que lê 5 valores para a, um de cada vez, e conta quantos destes valores são negativos, escrevendo esta informação.

5. Escrever um algoritmo/programa em Pascal que escreve os números pares entre 100 e 200.

6. Escrever um algoritmo/programa em Pascal que escreve a soma dos números entre 0 e 100.

7. Escrever um algoritmo/programa em Pascal que escreve a soma dos números pares entre 0 e 100.

8. Escrever um algoritmo/programa em Pascal que escreve a soma dos números múltiplos de 7 entre 100 e 200.

9. Escrever um algoritmo/programa em Pascal que lê o número de um funcionário, seu número de horas trabalhadas, o valor que recebe por hora, e o número de filhos com idade menor do que 14 anos e calcula o salário deste funcionário.

10. Escrever um algoritmo/programa em Pascal que lê o número de um vendedor, o seu salário-fixo, o tota1 de vendas por ele efetuadas e o percentual que ganha sobre o total de vendas. Calcular o salário total do vendedor. Escrever número do vendedor e o salário total.

11. Uma revendedora de carros usados paga a seus funcionários vendedores, um salário fixo por mês, mais uma comissão também fixa para cada carro vendido e mais 6% do valor das vendas por ele efetuadas. Escrever um algoritmo/programa em Pascal que lê o número do vendedor, o número de carros por ele vendidos, o valor tota1 de suas vendas, o salário fixo e o valor que recebe por carro vendido e calcula o salário mensal do vendedor, escrevendo-o juntamente com o seu número de identificação.

12. Escrever um algoritmo/programa em Pascal que lê 3 valores a, b, c, e escreva os 3 valores em ordem crescente.

13. Escrever um algoritmo/programa em Pascal que lê um conjunto de 4 valores i, a, b, c, onde i é um valor inteiro e positivo e a, b, c, são quaisquer valores reais e os escreva. A seguir:
    Se i = 1 escrever os 3 valores a, b, c em ordem crescente.
    Se i = 2 escrever os 3 valores a, b, c em ordem decrescente.
    Se i = 3 escrever os 3 valores de forma que o maior valor entre a, b, c fica entre os outros 2.
14. Escrever um algoritmo/programa em Pascal que lê o número de um vendedor de uma empresa, seu salário fixo e o total de vendas por ele efetuadas. Cada vendedor recebe um salário fixo, mais uma comissão proporcional às vendas por ele efetuadas. A comissão é de 5% sobre o total de vendas até $ 1.000,00 e 10% sobre o que ultrapassa este valor. Escrever o número do vendedor, o total de suas vendas, seu salário fixo e seu salário total.

15. Escrever um algoritmo/programa em Pascal que lê 3 comprimentos de lados a, b, c e os ordena em ordem decrescente, de modo que o a represente o maior dos 3 lados lidos. Determine, a seguir, o tipo de triângulo que estes 3 lados formam, com base nos seguintes casos escrevendo sempre os valores lidos e uma mensagem adequada:

    Se $a > b + c$ não formam triângulo algum.
    Se $a^2 = b^2 + c^2$ formam um triângulo retângulo.
    Se $a^2 > b^2 + c^2$ formam um triângulo obtusângulo.
    Se $a^2 < b^2 + c^2$ formam um triângulo acutângulo.
    Se forem todos iguais formam um triângulo equilátero.
    Se a = b ou b = c ou a = c então formam um triângulo isósceles

16. Escrever um algoritmo/programa em Pascal que lê o número de um funcionário, o número de horas por ele trabalhadas, o valor que recebe por hora, o número de filhos com idade inferior a 14 anos, a idade, o tempo de serviço do funcionário e o valor do salário família por filho. Calcular o salário bruto, o desconto do INSS (8,5% do salário bruto) e o salário família.

    Calcular o IR (Imposto de Renda) como segue:
    Se Salário Bruto > 1.500,00 então IR = 15% do SB
    Se Salário Bruto > 500,00 e SB <= 1.500,00 então IR = 8% do SB
    Se salário Bruto <= 500,00 então IR = 0

Calcular o adicional conforme especificado:
Se idade superior a 40 anos ADIC = 2% do SB
Se tempo de serviço superior a 15 anos ADIC = 3.5% do SB
Se tempo de serviço < 15 anos mas superior a 5 anos e idade maior do que 30 anos então ADIC = 1,5% do SB.
Calcular o salário líquido. Escrever o número do funcionário, salário bruta, total dos descontos, adicional e salário líquido.

17. Escrever um algoritmo/programa em Pascal que escreve a soma dos números que não são múltiplos de 13 entre 100 e 200.

18. Escrever um algoritmo/programa em Pascal que lê 20 valores, um de cada vez, e conta quantos deles estão em cada um dos intervalos [0, 25], (25, 50], (50, 75], (75, 100], escrevendo esta informação

19. Escrever um algoritmo/programa em Pascal que lê um número não determinados de valores a, todos inteiros e positivos, um de cada vez, e calcule e escreva a média aritmética dos valores lidos, a quantidade de valores pares, a quantidade de valores impares, a percentagem de valores pares e a percentagem de valores ímpares.

20. Escrever um algoritmo/programa em Pascal semelhante ao anterior que calcula as médias aritméticas de cada intervalo e as escreve, juntamente com o número de valores de cada intervalo.

21. Escrever um algoritmo/programa em Pascal que lê um número e calcula e escreve o seu fatorial.

22. Escrever um algoritmo/programa em Pascal que lê um número e escreva se ele "*é primo*" ou "*não é primo*"

23. Escrever um algoritmo/programa em Pascal que escreve os números múltiplos de 7 entre 100 e 200, bem como a soma destes números.

24. Escrever um algoritmo/programa em Pascal que lê um número não conhecido de valores, um de cada vez, e conta quantos deles estão em cada um dos intervalos [0, 50], (50, 100], (100,200]. O programa deve encerrar quando for informado um valor fora dos intervalos.

25. Escrever um algoritmo/programa em Pascal que escreve os números primos entre 100 e 200, bem como a soma destes números.

26. Escrever um algoritmo/programa em Pascal que lê 5 conjuntos de 4 valores a1, a2, a3, a4, um conjunto por vez e os escreve assim como foram lidos. Em seguida, ordene-os em ordem decrescente e escreva-os novamente.

27. Escrever um algoritmo/programa em Pascal que lê 5 pares de valores a, b, todos inteiros e positivos, um par de cada vez, e com a < b, e escreve os inteiros pares de a até b, incluindo o a e b se forem pares.

28. Escrever um algoritmo/programa em Pascal que lê 5 conjuntos de 2 valores, o primeiro representando o número de um aluno e o segundo representando a sua altura em centímetros. Encontrar o aluno mais alto e o mais baixo e escrever seus números, suas alturas e uma mensagem dizendo se é o mais alto ou o mais baixo.

29. Escrever um algoritmo/programa em Pascal que lê 50 valores, um de cada vez, e encontra e escreve o maior deles.

30. Escrever um algoritmo/programa em Pascal que gera os números de 1000 a 1999 e escreve aqueles que divididos por 11 dão um resto igual a 5.

31. Escrever um algoritmo/programa em Pascal que lê um vetor V[6] e o escreve. Conte, a seguir quantos valores de V são negativos e escreva esta informação.

32. Escrever um algoritmo/programa em Pascal que lê um vetor X(100) e o escreve. Substitua, a seguir, todos os valores nulos de X por 1 e escreva novamente o vetor x

33. Escrever um algoritmo/programa em Pascal que lê um vetor C[50] e o escreve. Encontre, a seguir, o maior elemento de C e o escreva.

34. Escrever um a1goritmo que lê um vetor A[15] e o escreve. Ordene, a seguir os elementos de A em ordem crescente e escreva novamente A.

35. Escrever um algoritmo/programa em Pascal que lê um vetor X[20] e o escreve. Troque, a seguir, o 1º elemento com o último, o 2º com o penúltimo, etc., até o 10º com o 11º e escreva o vetor X assim modificado.

36. Escrever um algoritmo/programa em Pascal que lê um vetor T(20) e o escreve. Troque, a seguir, os elementos de ordem ímpar com os de ordem par imediatamente seguintes e escreva o vetor k modificado.

37. Escrever um algoritmo/programa em Pascal que lê um vetor H[20] e o escreve. Troque, a seguir, o 1º elemento com 11º, o 2º com o 12º, etc., até o 10º com o 20º e escreva o vetor assim modificado.

38. Escrever um algoritmo/programa em Pascal que lê 2 vetores K[10] e N[10] e os escreve. Crie, a seguir, um vetor C que seja a diferença entre K e N (C = K - N) e escreva o vetor C.

39. Escrever um algoritmo/programa em Pascal que lê um vetor A[13] que é o Gabarito de um teste da loteria esportiva, contendo os valores 1 (coluna 1), 2(coluna 2), e 3(coluna do meio). Ler, a seguir, para cada apostador, o nº de seu cartão e um vetor Resposta B[13]. Verificar para cada apostador o nº de acertos e escrever o nº do apostador e seu número de acertos. Se tiver 13 acertos, acrescentar a mensagem: "*ganhador, parabéns!*".

40. Escrever um algoritmo/programa em Pascal que lê um vetor V(20) e o escreve. Retire, a seguir, os elementos em duplicata, compactando o vetor Y, e escrevendo o vetor compactado.

41. Escreva um algoritmo/programa em Pascal que lê um conjunto de 30 valores e os coloca em 2 vetores conforme forem pares ou impares. O tamanho dos vetores é de 5 posições. Se algum vetor estiver cheio, escreve-lo. Terminada a leitura escrever o conteúdo dos 2 vetores. Cada vetor pode ser preenchido tantas vezes quantas for necessário.

42. Escrever um algoritmo/programa em Pascal que gera os 10 primeiros números primos acima de 100 e os armazena em um vetor A(10) escrevendo, no final, o vetor A.

43. Escrever um algoritmo/programa em Pascal que lê uma matriz H(10,10) e a escreve. Troque, a seguir:
    - a linha 2 com a linha 8.
    - a coluna 4 com a coluna 10
    - a diagonal principal com a secundária
    - a linha 5 com a coluna 10

    Escreva a matriz assim modificada.

44. Escrever um algoritmo/programa em Pascal que lê uma matriz C(6,6) e um valor A e multiplica a matriz C pelo valor A e coloca os valores da matriz multiplicados por A em um vetor de V(36) e escreve no final o vetor V.

45. Escrever um algoritmo/programa em Pascal que lê uma matriz A(12,13) e divida todos os 13 elementos de cada uma das 12 linhas de A pelo valor do maior elemento em módulo daquela linha. Escrever a matriz A lida e a matriz A modificada.

INDICAÇÃO DE LEITURA

GOTTFRIED, Byron S. Programação em Pascal. Lisboa: McGraw Hill, 1994. 567p.
MECLER, Ian & MAIA, Luiz Paulo. Programação e Lógica com Turbo Pascal. Rio de Janeiro: Campus, 1989. 223p.
PALMER, Scott D. Guia do Programador Turbo Pascal for Windows. Rio de Janeiro: Ed. Ciência Moderna Ltda, 1992. 470p.
OSIER, Dan. Aprenda em 21 dias Delphi 2. Rio de Janeiro: Campus, 1997. 840p.

Anotações

GLOSSÁRIO

ASCII: (American Standard Code for Information Interchange) Padrão muito usado em todo o mundo, no qual números, letras maiúsculas e minúsculas, alguns sinais de pontuação, alguns símbolos e códigos de controle correspondem a números de 0 a 127. Com o ASCII, os documentos criados são facilmente transferidos através da Internet.

BIT: Dígito binário, um único 0 ou 1, ativado ou desativado, armazenado no seu computador. Quatro bits formam um nibble (termo raramente usado), e 8 bits formam um byte, o equivalente a um único caracter. As CPUs possuem 8, 16 ou 32 bits. Isso se refere à quantidade de informações que podem processar de cada vez.

BYTE: conjunto de 8 bits.

Binário: Sistema de numeração composto por dois dígitos (0 e 1) usado para representação interna de informação nos computadores. Se refere também a qualquer formato de arquivo cuja informação é codificada em algum formato que não o padrão character encoding scheme (método de codificação de caracteres). Um arquivo escrito em formato binário contém um tipo de informação que não é mostrada como caracteres. Um software capaz de entender o método de codificação de formato binário é necessário para interpretar a informação em um arquivo binário. O formato binário normalmente é utilizado para armazenar mais informação em menos espaço.

C: Tipo de linguagem de programação.

Código Fonte: Durante o desenvolvimento de um programa, ele é inicialmente escrito em uma linguagem de programação (chamada neste caso de linguagem de alto nível) e depois traduzido, com o auxílio de um programa especial chamado compilador, para uma forma que pode ser entendida pelo computador. O código fonte é a versão do programa na linguagem na qual ele foi escrito. A disponibilidade do código fonte permite que um programador modifique o programa.

Dados: Qualquer tipo de informação (em um processador de texto, programa de imagem, etc.) processada pelo computador.

Delphi: Tipo de linguagem de programação.

Inteiro: Números que não possuem casas decimais, apenas a parte inteira e podem ser positivos ou negativos.

Linguagem de Programação: conjunto de instruções, comandos, chamado de sintaxe, que deve ser seguida corretamente para que o programa funcione.

Pascal: Tipo de linguagem de programação.

Real: Números que possuem casas decimais e também podem ser positivos, negativos e inteiros - para usar casas decimais é necessário usar o ponto ao ivés de vírgula.

String: Para escrever um texto, mais de um caracter é necessário. Na programação, precisamos representar essa cadeia de caracteres de alguma forma. A palavra “cadeia” em inglês é “string”).

Anotações

REFERÊNCIAS

BUFFONI, Salete. **Apostila de Algoritmo Estruturado**. 4. ed. Disponível em:< http://www.saletebuffoni.hpg.ig.com.br/algoritmos/Algoritmos.pdf>. Acesso em: 27 jul. 2010.

CHAPIN, Ned. **Flowcharting with the ANSI Standard:** A Tutorial. ACM Computing Surveys, v.2, n. 2, p. 119-146, jun.1979.

CHAPIN, Ned. **Maintenance of Information Systems**. Disponível em:< http://www.iceis.org/iceis2002/tutorials.htm>. Acesso em: 21 jun. 2010.

CHAPIN, Ned. **New Format for Flowcharts, Software—Practice and Experience**. v. 4, n. 4, p. 341-357, oct.-dec. 1974.

COLLINS, Willian J. **Programação Estruturada com estudo de casos em Pascal**. São Paulo: Ed. Mc Graw-Hill do Brasil, 1988.

CORMEN, Thomas H.; LEISERSON, Charles E. & RIVEST, Ronald L. **Introduction to Algorithms**. New York:McGraw-Hill, 1990.

COSTA, Renato. **Apostila de Lógica de Programação - Criação de Algoritmos e Programas**. Disponível em:< http://www.meusite.pro.br/apostilas2.htm>. Acesso em: 17 jul. 2010.

FARRER, Harry et al. **Algoritmos Estruturados**. Rio de Janeiro: Editora Guanabara Koogan S.A, 1989. 252p.

FARRER, Harry. **Programação Estruturada de Computadores**. Rio de Janeiro: Ed. LTC, 1989.

FORBELLONE, André. **Lógica de Programação - A Construção de Algoritmos e Estruturas de Dados.** São Paulo: Ed. Makron Books, 1993.

GOMES, Abel. **Algoritmos, Fluxogramas e Pseudo-código - Design de Algoritmos.** Disponível em: <http://mail.di.ubi.pt/~programacao/capitulo6.pdf>. Acesso em: 13 set. 2010.

GOTTFRIED, Byron S. **Programação em Pascal**. Lisboa: McGraw Hill, 1994. 567p.

GRILLO, Maria Célia Arruda. **Turbo Pascal 5.0 e 5.5**. Rio de Janeiro: LTC - Livros Técnicos e Científicos, 1990. 396p.

GUIMARÃES, Angelo de Moura. **Algoritmos e Estruturas de Dados**. Rio de Janeiro: Ed. LTC, 1985.

HERGERT, Douglas. **Dominando o Turbo Pascal 5**. Rio de Janeiro: Editora Ciência Moderna, 1989. 550p.

KNUTH, D. E. **The Art of Computer Programming.** v. 3. Sort and Searchim. Addison Wesley, Reading, Mass., 1973

KOZAK, Dalton V. **Técnicas de Construção de Algoritmos**. Disponível em: <http://minerva.ufpel.edu.br/~rossato/ipd/apostila_algoritmos.pdf>. Acesso em: 12 ago. 2010.

MANZANO, José Augusto N. G.; OLIVEIRA, Jayr Figueiredo. **Algoritmos:** Lógica Para Desenvolvimento de Programação. São Paulo: Érica, 1996. 270p.

MARTINS, Luiz E. G.; ZÍLIO, Valéria M. D. **Apostila da Disciplina Introdução à Programação.** Disponível em:< http://www.unimep.br/~vmdzilio/apostila00.doc>. Acesso em: 14 jun. 2010.

MECLER, Ian; MAIA, Luiz Paulo. **Programação e Lógica com Turbo Pascal**. Rio de Janeiro: Campus, 1989. 223p.

MICHAELIS: **Moderno dicionário da língua portuguesa**. São Paulo: Companhia Melhoramentos, 1998(Dicionários Michaelis). 2259p.

MULLER, Nicolas. **Como fazer um fluxograma?.** Disponível em: <http://www.oficinadanet.com.br/artigo/desenvolvimento/como_fazer_um_fluxograma> Acesso em: 13 fev. 2012.

NASSI, Ike. **Ike Nassi's Home Page**. Disponível em:< http://www.nassi.com/ike.htm>. Acesso em: 11 jun. 2010.

NASSI, Ike; SHNEIDERMAN, Ben. **Flowchart Techniques for Structured Programming.** ACM SIGPLAN Notices, v. 8, n. 8, p.12-26, August 1973

ORTH, Afonso Inácio. **Algoritmos**. Porto Alegre: Editora Pallotti, 1985. 130p.

OSIER, Dan. **Aprenda em 21 dias Delphi 2**. Rio de Janeiro:Campus, 1997. 840p.

PALMER, Scott D. **Guia do Programador Turbo Pascal for Windows**. Rio de Janeiro: Ed. Ciência Moderna Ltda, 1992. 470p.

PREUSS, Evandro. **Algoritmos e Estrutura de Dados I**. Disponível em: <http://pt.scribd.com/doc/55726984/7/Diagrama-de-Chapin>. Acesso em: 21 jan. 2012.

RINALDI, Roberto, **Turbo Pascal 7.0:** comandos e funções. São Paulo: Érica, 1993. 525p.

SALIBA, Walter Luís Caram. **Técnicas de Programação:** Uma Abordagem Estrutura. São Paulo: Makron, McGraw-Hill, 1992. 141p.

SALIBA, Walter Luís Caram. **Técnicas de Programação:** Uma Abordagem Estrutura. São Paulo: Ed. Makron Books, 1992.

SALVETTI, Dirceu Douglas e Barbosa, L. M. **Algoritmos.** São Paulo: Ed. Makron Books, 1998.

SANTANA, João. **Algoritmos & Programação**. Disponível em:< http://www.iesam.com.br/paginas/cursos/ec/1ano/aulas/08/joao/APunidade-1.pdf>. Acesso em: 15 jul. 2010.

TERADA, Routo; SETZER, Valdemar W. **Introdução à Computação e à Construção de Algoritmos**. São Paulo: Ed. Makron Books, 1992.

TONET, Bruno; KOLIVER, Cristian. **Introdução aos Algoritmos.** Disponível em:< http://dein.ucs.br/napro/Algoritmo/manuais/Manual 20Visualg.pdf>. Acesso em: 21 mar. 2010.