Algoritmos

Lógica para Desenvolvimento de Programação

Introdução

Este trabalho é fruto de minha experiência em desenvolvimento de softwares e cursos lecionados na área de programação.

O material aqui apresentado foi recolhido de algumas literaturas de Estrutura de Dados e Tutoriais da encontrados no sites da Internet.

A programação é para mim uma diversão, apesar de complexo, mas vejo como a capacidade de um ser humano dominar e solucionar com recursos computacionais problemas do dia-a-dia de uma Empresa, Corporação, etc...

O raciocínio da lógica para a programação traz ao indivíduo um novo mundo cheio de outros mundos. Dominando a lógica você será capaz de aprender sem grandes dificuldades qualquer outra linguagem de programação seja ela estruturada, orientada a objeto, orientada a eventos.

Alessandro Coelho Porto

Sobre o Autor

Atualmente trabalha como Webdesign / Webmaster na Embrapa Cerrados, sua principal tarefa está no planejamaneto e desenvolvimento de Sites. Utiliza as linguagens HTML, JavaScript e VbScript e Sistemas de Banco de Dados (ASP) para a Internet.

Em sua experiência profissional também encontramos serviços autônomos na área de desenvolvimento na área comercial, as ferramentas por ele utilizadas foram Clipper, Visual Basic e Delphi.

Abordagem Contextual

Muitas pessoas gostam de falar ou julgar que possuem e sabem usar o raciocínio lógico, porém, quando questionadas direta ou indiretamente, perdem esta linha de raciocínio, pois este depende de inúmeros fatores para completá-lo, tais como: calma, conhecimento, vivência, versatibilidade, experiência, criatividade, ponderação, responsabilidade, entre outros.

Bem sem mais delongas podemos dizer que lógica é a ciência que estudas as leis e critéios de validade que regem o pensamento e a demonstração, ou seja, ciência dos princípios formais do raciocínio.

Usar a lógica é um fator a ser considerado por todos, principalmente pelos profissionais de informática (programadores, analistas de sistemas e suporte), pois seu dia-a-dia dentro das organizações é solucionar problemas e atingir os objetivos apresentados por seus usuários com eficiência e eficácia, utilizando recursos computacionais e/ou automatizados macatronicamente. Saber lidar com problemas de ordem administrativa, de controle, de planejamento e de estratégica requer atenção e boa performance de conhecimento de nosso interesse é mostrar como desenvolver e aperfeiçoar melhor esta técnica, lembrando que para isto, você deverá ser persistente e praticá-la constantemente, chegando à exaustão sempre que julgar necessário.

Princípios de Resoluções de Problemas

Primeiramente, devemos entender e compreender a palavra “problema”. Podemos dizer que problema é um proposta duvidosa, que pode Ter inúmeras soluções, ou questão não solvida e que é o objeto de discussão, segundo a definição encontrada no Dicionário Aurélio.

Preferimos dizer que problema é uma questão que foge a uma determinada regra, ou melhor, é o desvio de um percurso, o qual impede de atingir com sucesso um determinado objetivo com eficiência e eficácia.

Uma das soluções utilizadas para a resolução de problemas tem sido a utilização dos diagramas, no decorrer do curso será bastante enfatizado em sala de aula o uso de tais ferramentas com o intuito de auxiliar os aprendizados tanto na resolução de problemas como no conhecimento das técnicas de programação.

ALGORITMO

Um Algoritmo é uma seqüência de instruções ordenadas de forma lógica para a resolução de uma determinada tarefa ou problema.

PROGRAMAÇÃO ESTRUTURADA

Basicamente, a Programação Estruturada consiste numa metodologia de projeto de programas visando:

- facilitar a escrita dos programas;
- facilitar a leitura (o entendimento) dos programas;
- permitir a verificação a priori dos programas;
- facilitar a manutenção e modificação dos programas.

O maior problema em grandes sistemas de software reside na enorme complexidade desses sistemas, cuja a apreensão vai geralmente muito além da capacidade intelectual de um ser humano. Entenda-se aqui por complexidade de um sistema uma medida do número de

seus componentes e do grau de interação entre eles. Para Dijkstra, o indiscutível iniciador da programação estruturada, "a arte de programar consiste na arte de organizar e dominar a complexidade".

A idéia da Programação Estruturada, que vai ao encontro da mencionada tarefa do programador, é reduzir a complexidade, em três níveis:

1. desenvolvimento do programa em diferentes fases por refinamento sucessivo (desenvolvimento Top-Down);
2. decomposição do programa total em módulos funcionais, organizados de preferência num sistema hierárquico;
3. usando dentro de cada módulo só um número muito limitado de estruturas básicas de fluxo de controle.

O curso tem por objetivo apresentar aos alunos os conceitos e desenvolvimento da programação estruturada, utilizando duas ferramentas muito utilizadas em cursos acadêmicos, o Portugol e o Pascal.

PROGRAMA

Um programa é um Algoritmo escrito em uma linguagem computacional.

Mais o que é um Algoritmo?

Algoritmo é um processo de cálculo matemático ou de resolução de um grupo de problemas semelhantes, em que se estipulam, com generalidade e sem restrições. Podemos dizer também, que são regras formais para obtenção de um resultado ou da solução de um problema, englobando fórmulas de expressões aritméticas.

LINGUAGENS DE PROGRAMAÇÃO

São Softwares que permitem o desenvolvimento de programas. Possuem um poder de criação ilimitado, desde jogos, editores de texto, sistemas empresariais até sistemas operacionais.

Existem várias linguagens de programação, cada uma com suas características próprias.

Exemplos:

- Pascal
- Clipper
- C
- Visual Basic
- Delphi e etc.

TÉCNICAS ATUAIS DE PROGRAMAÇÃO

- Programação Seqüencial
- Programação Estruturada
- Programação Orientada a Eventos e Objetos
- Programação Orientada a Objetos

ALGORITMOS EM "PORTUGOL"

Durante nosso curso iremos aprender a desenvolver nossos Algoritmos em uma pseudo-linguagem conhecida como "Portugol" ou Português Estruturado.

"Portugol" é derivado da aglutinação de Português + Algol. Algol é o nome de uma linguagem de programação estruturada usada no final da década de 50.

OPERADORES ARITMÉTICOS

+	➔	Adição
-	➔	Subtração
*	➔	Multiplicação
/	➔	Divisão

OPERADORES RELACIONAIS

>	➔	Maior que
<	➔	Menor que
>=	➔	Maior ou Igual
<=	➔	Menor ou Igual
=	➔	Igual
<>	➔	Diferente

LINEARIZAÇÃO DE EXPRESSÕES

Para a construção de Algoritmos todas as expressões aritméticas devem ser linearizadas, ou seja, colocadas em linhas.

MODULARIZAÇÃO DE EXPRESSÕES

A modularização é a divisão da expressão em partes, proporcionando maior compreensão e definindo prioridades para resolução da mesma.

Como pode ser observado no exemplo anterior, em expressões computacionais usamos somente parênteses "( )" para modularização.

Na informática podemos ter parênteses dentro de parênteses.

Exemplos de prioridades:

(2+2)/2=2

2+2/2=3

OPERADORES ESPECIAIS (MOD e DIV)

MOD ➔ Retorna o resto da divisão entre 2 números inteiros.

DIV ➔ Retorna o valor inteiro que resulta da divisão entre 2 números inteiros.

FUNÇÕES

Uma função é um instrumento (Sub–algoritmo) que tem como objetivo retornar um valor ou uma informação.

A chamada de uma função é feita através da citação do seu nome seguido opcionalmente de seu argumento inicial entre parênteses.

As funções podem ser predefinidas pela linguagem ou criadas pelo programador de acordo com o seu interesse.

BIBLIOTECAS DE FUNÇÕES

Armazenam um conjunto de funções que podem ser usadas pelos programas.

FUNÇÕES PRÉ-DEFINIDAS

ABS( )	VALOR ABSOLUTO
SQRT( )	RAIZ QUADRADA
SQR( )	ELEVA AO QUADRADO
TRUNC( )	VALOR TRUNCADO
ROUND( )	VALOR ARREDONDADO
LOG( )	LOGARITMO
SIN( )	SENO
COS( )	COSENO
TAN( )	TANGENTE

As funções acima são as mais comuns e importantes para nosso desenvolvimento lógico, entretanto, cada linguagem possui suas funções própias. As funções podem ser aritméticas, temporais, de texto e etc.

VARIÁVEIS

Variáveis são endereços de memória destinados a armazenar informações temporariamente.

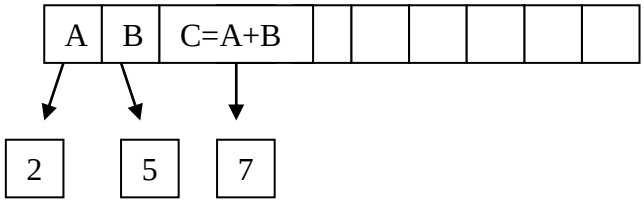
* Todo Algoritmo ou programa deve possuir variável!

VARIÁVEIS DE ENTRADA E SAÍDA

Variáveis de Entrada armazenam informações fornecidas por um meio externo, normalmente usuários ou discos.

Variáveis de Saída armazenam dados processados como resultados.

Exemplo:



De acordo com a figura acima A e B são Variáveis de Entrada e C é uma Variável de Saída.

CONSTANTES

Constantes são endereços de memória destinados a armazenar informações fixas, inalteráveis durante a execução do programa.

Exemplo:

PI = 3.1416

IDENTIFICADORES

São os nomes dados a variáveis, constantes, Tipo de Dados, Registros e programas.

Regras Para construção de Identificadores:

- Não podem ter nomes de palavras reservadas (comandos da linguagem);
- Devem possuir como 1º caractere uma letra ou Underscore (_);
- Ter como demais caracteres letras, números ou Underscore;
- Ter no máximo 127 caracteres;
- Não possuir espaços em branco;
- A escolha de letras maiúsculas ou minúsculas é indiferente.

Exemplos:

NOME	TELEFONE	IDADE_FILHO
NOTA1	SALARIO	PI
UMNOMEMUITOCOMPRIDOEDIFICILDELER UM_NOME_MUITO_COMPRIDO_E_FACIL_DE_LER		

TIPOS DE DADOS

Todas as Variáveis devem assumir um determinado tipo de informação.

O tipo de dado pode ser:

- Primitivo ➔ Pré-definido pela linguagem;
- Sub-Faixa ➔ É uma parte de um tipo já existente;
- Escalar ➔ Definidos pelo programador.

TIPOS PRIMITIVOS DE DADOS

INTEIRO	ADMITE SOMENTE NÚMEROS INTEIROS. GERALMENTE É UTILIZADO PARA REPRESENTAR UMA CONTAGEM (QUANTIDADE).
REAL	ADMITE NÚMEROS REAIS (COM OU SEM CASAS DECIMAIS). GERALMENTE É UTILIZADO PARA REPRESENTAR UMA MEDIÇÃO.

CARACTERE	ADMITE CARACTERES ALFANUMÉRICOS. OS NÚMEROS QUANDO DECLARADOS COMO CARACTERES TORNAM SE REPRESENTATIVOS E PERDEM A ATRIBUIÇÃO DE VALOR.
LÓGICO	ADMITE SOMENTE VALORES LÓGICOS(VERDADEIRO/FALSO).

COMANDOS DE I/O (INPUT/OUTPUT)

LEIA ➔ Comando de entrada que permite a leitura de Variáveis de Entrada.
ESCREVERA ➔ Comando de saída que exibe uma informação na tela do monitor.
IMPRIMA ➔ Comando de saída que envia uma informação para a impressora.

SINAL DE ATRIBUIÇÃO

Uma Variável nunca é eternamente igual a um valor, seu conteúdo pode ser alterado a qualquer momento. Portanto para atribuir valores a variáveis devemos usar o sinal de "**:=".**

Exemplos:
A **:=** 2;
B **:=** 3;
C **:=** A + B;

SINAL DE IGUALDADE

As constantes são eternamente iguais a determinados valores, portanto usamos o sinal de "=".

Exemplos:
PI **=** 3.1416;
Empresa = 'Colégio de Informática L.T.D.A.'
V = Verdadeiro

CORPO GERAL DE UM PROGRAMA

```
PROGRAMA <<identificador>>;
CONST
        <<identificador>> = <<dado>>
VAR
        <<identificador>> :  <<tipo>>;
ÍNICIO
        {
        COMANDOS DE ENTRADA,PROCESSAMENTO E SAÍDA
        <<comando1>>;
        <<comandoN>>
        }
FIM.
```

ESTRUTURAS SEQÜÊNCIAIS

Como pode ser analisado no tópico anterior, todo programa possui uma estrutura seqüencial determinada por um ÍNICIO e FIM.

TÉCNICAS BÁSICAS DE PROGRAMAÇÃO

A partir deste ponto de estudo, você terá um contato direto com a parte mais prática deste trabalho. Anteriormente nos preocupamos em proporcionar a você, leitor, um conhecimento teórico básico, de algns pontos que são por vezes causadores de dúvidas até a alguns profissionais de informática considerados experientes. Sendo assim, daqui para frente você terá um contato maior com a aplicação prática dos Algoritmos.

ENTRADA, PROCESSAMENTO E SAÍDA

Para se criar um programa que seja executável dentro de um computador, você deverá term em mente três pontos de trabalho: a entrada de dados, o seu processamamento e a saída dos mesmos. Sendo assim, todo programa estará trabalhando com estes três conceitos. Se os dados forem entrados de forma errada, serão consequentemente e resultarão em respostas erradas. Desta forma, dizer a alguém que foi erro do computador é ser um pouco “medíocre”.

O processo de execução de um programa ocorre segundo o exposto, após a entrada de dados com a instrução leia e saída dos mesmos com a instrução escreva, o processamento será uma conseqüência da manipulação das variáveis de ação.

Uma entrada e uma saída poderão ocorrer dentro de um computador de diversas formas. Por exemplo, uma entrada poderá ser feita via teclado, moddem, leitores óticos, disco, entre outras. Uma saída poderá ser feita em víde, impressora, disco, entre outras formas. Devido a esta grande variedade, nossos programas escrito em português estruturado farão menção às instruções leia e escreva.

Exemplo 01

Segue um Algoritmo que lê o nome e as 4 notas bimestrais de um aluno. Em seguida o Algoritmo calcula e escreve a média obtida.

```
PROGRAMA MEDIA_FINAL
VAR
        NOTA1, NOTA2, NOTA3, NOTA4, MEDIA: INTEIRO
        NOME : CARACTERE
INICIO
        LEIA (NOME)
        LEIA (NOTA1, NOTA2, NOTA3, NOTA4)
        MEDIA <= (NOTA1 + NOTA2 + NOTA3 + NOTA4) / 4
        ESCREVER (NOME, MEDIA)
FIM.
```

Exemplo 02

Segue um Algoritmo que lê o raio de uma circunferência e calcula sua área.

```
PROGRAMA AREA_CIRCUNFERENCIA
CONST PI = 3.1416
VAR RAIO, AREA : REAL
INICIO
        LER (RAIO) {PROCESSAMENTO}
        AREA <= PI * SQR(RAIO) {ENTRADA}
        ESCREVER (“AREA =”, AREA) {SAÍDA}
FIM.
```

{LINHAS DE COMENTÁRIO}

Podemos inserir em um Algoritmo comentários para aumentar a compreensão do mesmo, para isso basta que o texto fique entre Chaves “{}”.

Exemplo:

LER (RAIO); **{ENTRADA}**

Exercícios de Aprendizagem

1) Construir um programa que efetue o cálculo de um salário líquido de um professor. Para fazer este programa você deverá possuir alguns dados, tais como: valor da hora aula, número de aulas dadas no mês e percentual de desconto do INSS. Em primeiro lugar, deve-se estabelecer qual será o seu salário bruto para efetuar o desconto e Ter o valor do salário líquido.
2) Efetuar o cálculo da quantidade de litros de combustível gastos em uma viagem, utilizando-se um automóvel que faz 12 Km por litro. Para obeter o cálculo, o usuário deverá fornecer o tempo gasto na viagem e a velocidade média durante a mesma. Desta forma, será possível obter a distância percorrida com a fórmula DISTÂNCIA <= TEMPO * VELOCIDADE. Tendo o valor da distância, basta calcular a quantidade de litros de combustível utilizada na viagem com a fórmula: LITROS_USADOS <= DISTÂNCIA / 12. O programa deverá apresentar os valores da velocidade média, tempo gasto na viagem, a distância percorrida e a quantidade de litros usados na viagem.
3) Ler dois números inteiros e efetuar as operações de adição, subtração, multiplicação e divisão de A por B apresentando ao final os quadro resultados obtidos.

Estrutura de Controloe – A tomada de Decisões

Executa uma seqüência de comandos de acordo com o resultado de um teste.

A estrutura de decisão pode ser Simples ou Composta, baseada em um resultado lógico.

```
Simples:
SE <<CONDIÇÃO>>
        ENTÃO <<COMANDO1>>
```

```
Composta 1:
SE <<CONDIÇÃO>>
        ENTÃO <<COMANDO1>>
SENÃO <<COMANDO1>>
```

```
Composta 2:
SE <<CONDIÇÃO>>
        ENTÃO INICIO
                <<COMANDO1>>;
                <<COMANDON>>
            FIM;
        SENÃO INICIO
                <<COMANDO1>>; <<COMANDON>>
            FIM;
```

OPERADORES LÓGICOS

Atuam sobre expressões retornando sempre valores lógicos como Falso ou Verdadeiro.

E	RETORNA VERDADEIRO SE AMBAS AS PARTES FOREM VERDADEIRAS.
OU	BASTA QUE UMA PARTE SEJA VERDADEIRA PARA RETORNAR VERDADEIRO.
NÃO	INVERTE O ESTADO, DE VERDADEIRO PASSA PARA FALSO E VICE-VERSA.

TABELA VERDADE

A	B	A E B	A OU B	NÃO (A)
V	V	V	V	F
V	F	F	V	F
F	V	F	V	V
F	F	F	F	V

EXPRESSÕES LÓGICAS

As expressões compostas de relações sempre retornam um valor lógico.
Exemplos:

2+5>4 ➔ Verdadeiro | 3<>3 ➔ Falso

De acordo com a necessidade, as expressões podem ser unidas pelos operadores lógicos.

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

Exemplo 01

Segue um Algoritmo que lê 2 números e escreve o maior.

```
PROGRAMA ACHA_MAIOR;
VAR A, B : INTEIRO;
INICIO
        LER (A, B);
        SE A>B
                ENTÃO ESCREVER (A)
                SENÃO ESCREVER (B)
FIM.
```

Exemplo 02

Segue um Algoritmo que lê o nome e as 4 notas bimestrais de um aluno. Em seguida o Algoritmo calcula e escreve a média obtida pelo aluno escrevendo também se o aluno foi aprovado ou reprovado.

Média para aprovação = 6

```
PROGRAMA MEDIA_FINAL;
VAR
        NOTA1, NOTA2, NOTA3, NOTA4, MEDIA: REAL;
        NOME : CARACTERE [35]
INICIO
        LER (NOME);
        LER (NOTA1, NOTA2, NOTA3, NOTA4);
        MEDIA := (NOTA1 + NOTA2 + NOTA3 + NOTA4) / 4;
        SE MEDIA>=6
                ENTÃO ESCREVER ('APROVADO')
                SENÃO ESCREVER ('REPROVADO')
                ESCREVER (NOME, MEDIA)
FIM.
```

NINHOS DE SE

Usados para tomadas de decisões para mais de 2 opções.

Forma Geral:

```
SE <<CONDIÇÃO>>
        ENTÃO <<COMANDO1>>
        SENÃO SE <<CONDIÇÃO>>
                        ENTÃO <<COMANDO1>>
                        SENÃO <<COMANDO1>>
```

Exemplo 01

Segue um Algoritmo que lê 3 números e escreve o maior.

```
PROGRAMA ACHA_MAIOR;
VAR A, B, C : INTEIRO;
INICIO
        LER (A, B, C);
        SE (A>B) E (A>C)
                ENTÃO ESCREVER (A)
                SENÃO SE (B>A) E (B>C)
                                ENTÃO ESCREVER (B)
                                SENÃO ESCREVER (C)
FIM.
```

ESTRUTURAS DE CONDIÇÃO

A estrutura de condição eqüivale a um ninho de SE'S.
Forma Geral:

```
FACA CASO
        CASO <<CONDIÇÃO1>>
                <<COMANDO1>>;
        CASO <<CONDIÇÃON>>
                <<COMANDO1>>;
        OUTROS CASOS
                <<COMANDO1>>;
FIM DE CASO
```

Exemplo 01

Segue um Algoritmo que lê 3 números e escreve o maior.

```
PROGRAMA ACHA_MAIOR;
VAR A, B, C : INTEIRO;
INICIO
        LER (A, B, C);
        FACA CASO
                CASO (A>B) E (A>C)
                        ESCREVER (A);
                CASO (B>A) E (B>C)
                        ESCREVER (B);
                OUTROS CASOS
                        ESCREVER (C);
        FIM DE CASO
FIM.
```

Exercícios de Aprendizagem

1) Ler três valores para os lados de um triângulo, considerando lados como: A, B e C. Verificar se os lados fornecidos formam realmente um triângulo, e ser for esta condição verdadeira, deverá ser indicado qual tipo de triângulo for formado: isósceles, escaleno ou equilátero.

2) Fazer a leitura de 3 valores para extrair uma equação do 2º grau. Deverá ser verificado se existem raízes reais, uma ou duas raízes conforme necessário.
3) Ler quatro valores referentes a quatro notas escolares e um aluno e imprimr uma mensagem dizendo que o alunos foi aprovado, se o valor da média escolar for maior ou igual a 5. Se o aluno não foi aprovado, indicar uma mensagem informando esta condição. Apresentar junto das mensagem o valor da média do aluno para qualquer condição.
4) Ler quatro valores referentes a quatro notas escolares de um aluno e imprimir uma mensagem dizendo que o aluno foi aprovado, se o valor da média escolar for maior ou igual a 7. Se o valor da média for menor que 7, solicitar a nota de exame, somar com o valor da média e obter nova média. Se a nova média for maior a 5, apresentar uma mensagem dizendo que o aluno foi aprovado em exame, se o aluno não foi aprovado, indicar uma mensagem informando esta condição. Apresentar com as mensagens o valor da média do aluno, para qualquer condição.

ESTRUTRA DE CONTROLE – LAÇOS OU MALHAS DE REPETIÇÃO

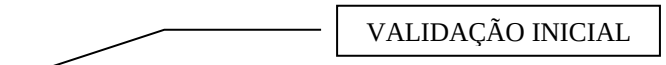
Existem ocasiões onde se é necessário efetuar a repetição de um trecho de programa um determinado número de vezes. Neste caso, poderá ser criado um looping que efetue o processamento de um determinado trecho, tantas vezes quantas forem necessárias. Os loopings também são chamados de: laços de repetição ou malhas de repetição.

Na programação estrutura ocorre a existênciade comandos apropriados para efetuar a repetição de determinados trechos de programas o número de vezes que for necessário. A principal vantagem deste recurso é que o programa passa a Ter um tamanho menor, podendo sua amplitude de processamento ser aumentada em se alterar o tamanho do código de programação. Desta forma, pode-se determinar repetiçoes com números variados de vezes.

Repetição do Tipo: Teste lógico no Início do Looping

É usada para repetir N vezes uma ou mais instruções. Tendo como vantagem o fato de não ser necessário o conhecimento prévio do número de repetições.

Forma Geral 1:




```
ENQUANTO <<CONDIÇÃO>> FACA
        <<COMANDO1>>;
```

Forma Geral 2:

```
ENQUANTO <<CONDIÇÃO>> FACA
        ÍNICIO
                <<COMANDO1>>;
                <<COMANDON>>
        FIM;
```

Exemplo

Segue um algoritmo que calcule a soma dos salários dos funcionários de uma empresa. O programa termina quando o usuário digitar um salário menor que 0.

```
PROGRAMA SOMA_SALARIOS;
VAR SOMA, SALARIO : REAL;
INICIO
        SOMA:=O;
        SALARIO:=1;
        ENQUANTO SALARIO>=0
                INICIO
                        LER (SALARIO);
                        SOMA:=SOMA+SALARIO
                FIM;
        ESCREVER (SOMA)
FIM.
```

A estrutura **enquanto...faça...fim_enquanto** tem o seu funcionamento controlado por decisão. Sendo assim poderá executar um determinado conjunto de instruções enquanto a condição verificada for Verdadeira. No momento em que esta condição se torna falsa, o processamento da rotina é desviado para fora do looping. Se a condição for Falsa logo de início, as instruções contidas no looping são ignoradas.

TODAS AS VARIÁVEIS QUE ACUMULAM VALORES DEVEM RECEBER UM VALOR INICIAL.

Assim como a estrutura ENQUANTO É usada para repetir N vezes uma ou mais instruções.

Sua validação é final fazendo com que a repetição seja executada pelo menos uma vez.

Forma Geral;

```
REPITA
        <<COMANDO1>>;
        <<COMANDON>>
ATE <<CONDIÇÃO>>
```

Exemplo 01

Segue um algoritmo que calcule a soma dos salários dos funcionários de uma empresa. O programa termina quando o usuário digitar um salário menor que 0.

```
PROGRAMA SOMA_SALARIOS;
VAR
        SOMA, SALARIO : REAL;
INICIO
        SOMA:=O;
        REPITA
                LER (SALARIO);
                SOMA:=SOMA+SALARIO
        ATE SALARIO<0;
        ESCREVER (SOMA)
FIM.
```

Exemplo 02

Segue um algoritmo que escreve os 100 primeiros números pares.

```
PROGRAMA PARES_2;
VAR I, PAR, CONTADOR : INTEIRO;
INICIO
        CONTADOR := 0;
        PAR := 0;
        REPITA
                ESCREVER (PAR);
                PAR := PAR+2;
                CONTADOR := CONTADOR+1;
        ATE CONTADOR=100
FIM.
```

A estrutura **repita..até_que** tem o seu funcionamento controlado por decisão. Porém, irá efetuar a execução de um conjunto de instruções pelo menos uma vez antes de verificar a validade da condição estabelecida. Diferente da estrutura **enquanto** que executa somente um conjunto de instruções, enquanto a condição é verdadeira.
Desta forma **repita** tem seu funcionamento em sentido contrário a **enquanto**, pois sempre irá processar um conjunto de instruções, no mínimo uma vez até que a condição se torne Verdadeira. Pra a estrutura **repita** um conjunto de instruções é executado enquanto a condição se mantém Falsa e até que ela seja Verdadeira.

REPETIÇÃO DO TIPO: VARIÁVEL DE CONTROLE

Anteriormente, formas vistas duas formas de se elaborar looping. Uma usando o conceito **enquanto** e a outra usando o conceito **repita**. Foi visto também como se estabelecer rotinas que efetuarem a execução de um looping um determinado número de vezes através da utilização de um contador (através de uma variável de controle).

Porém, existe uma possibilidade de faciliar o uso de contadores finitos, sem fazer uso das estruturas anteriores, deixando-as para a utilização de loopings onde não se conhece de antemão o número de vezes que uma determinada sequência de instruções deverá ser executada. Os looping que possuem um número finito de execuções poderão ser processados através de estrutura de laços contados **para**, sendo conseguida com a utilização do conjunto de instruções **para..de..até..passo.faça..fim_para**.

A estrutura **para..de..até..passo.faça..fim_para** tem seu funcionamento controlado por uma variável denominada contador. Sendo assim, poderá executar um determinado conjunto de instruções um determinado número de vezes.

Sua sintaxe é a seguinte:

Para <variável**> de <**inicio**> até <**fim**> passp<**incremento**> faça**

<instruções**>**

fim_para

Exemplo

Pedir a leitura de um valor para a variável X, multiplicar este valor por 3 implicando-o à variável de resposta R e apresentar o valor obtido, repetindo esta següência por cinco vezes.

```
Programa Looping_Para

Var

        X, R ,CONT: Inteiro
Inicio

        Para CONT de 1 até 5 passo 1 faça
                Leia (x)
                R <= X * R
                Escreva(R)
        Fim_para
Fim.
```

Será executado o conjunto de instruções entre a instrução **para** e a instrução **fim_para**, sendo a variável CONT (variável de controlo) inicializada com valor 1 e incrementada de mais 1 através da instrução **passo** até o valor 5. Este tipo de estrutura de repetição poderá ser utilizado todas as vezes que se tiver a necessidade de repetir trechos finitos, onde se conhece o valor inicial e o valor final.

Exercícios de Aprendizagem

(Desenvolver Cada Exercício utilizando as três estruturas de repetição apresentadas)

1) Apresentar os quadrados do números inteiros de 15 a 200.
2) Apresentar os resultados de uma tabuada de um número qualquer. Esta deverá ser impressa no seguinte formato:

   Considerando como exemplo o fornecimento do número 2

   2 X 1 = 2
   2 X 2 = 4
   2 X 3 = 6
   2 X 4 = 8
   2 X 5 = 10

   (...)

   2 X 10 = 20
3) Apresentar o total da soma do cem primeiros números inteiros (1 + 2+ 3 + 4...+ 99 + 100).
4) Elaborar um programa que apresente no final o somatório dos valores pares existentes na faixa de 1 até 500.
5) Apresentar todos os valores numéricos inteiros ímpares situados nafaixa de 0 a 20. Para verificar se o número é ímpar, efetuar dentro da malha a verificação lógica desta condição com a instrução **se**, perguntando se o número é ímpar, sendo mostre-o, não sendo, passe para o próximo passo.
6) Apresentar todos os números divisíveis por 4 que sejam menores que 200. Para verificar se o número é divisível por 4, efetuar dentro da malha de verificação lógica desta condição com a instrução **se**, perguntando se o número é divisível, sendo, mostre-o, não sendo passe para o próximo passo. A variável que controlará o contador deverá ser iniciada com valor 1.
7) Apresentar as potências de 3 variando de 0 a 15. Deve ser considerado que qualquer número elevado a zero é 1, e elevado a 1 é ele próprio.
8) Elaborar um programa que apresente o valor de uma potência de uma base qualquer elevada a um expoente qualquer, ou seja, de N^M .
9) Escreva um programa que apresente a série de Fibobnacci atéo décimo quinto termo. A série de Fibonacci é formada pela seqüência: 1,1,2,3,5,8,13,21,34,... etc. Esta série se caracteriza pela soma de um termo posterior com o seu anterior subseqüente.

ESTRUTURA BÁSICA DE DADOS – TABELAS EM MEMÓRIAS

Estrutura de Dados Homogêneas

Durante os pontos estudados anteriormente, percebemos que o poder de programação se tornou maior. Porém, tendo domínio das técnicas anteriores, ainda correr-se à o risco de não conseguir resolver alguns tipos de problemas, pois foram trabalhadas até aqui apenas variáveis simples, variáveis que somente armazenam um valor por vez.

Neste capítulo, será apresentado uma técnica de programação que permitirá trabalhar com o agrupamento de várias informações dentro de uma mesma variável. Vale salientar que este agrupamento ocorrerá obedecendo sempre ao mesmo tipo de dado, e por esta razão chamado de estruturas de dados homogêneas. Agrupamentos de tipo de dados diferentes setão estudados mais adiante quando forme abordadas as estruturas de dados heterogêneas.

A utilização deste tipo de estrutura de dados recebe diversos nomes, com: variáveis indexadas, variáveis compostas, variáveis subscritas, arranjos, vetores, matrizes, tabelas em memória ou arrays (do inglês). São vários os nomes encontrados na literatura voltada para o estudo de técnicas de programação que envolvem a utilização das estruturas homogêneas de dados. Por nós, serão definidas como matrizes.

As matrizes (tabelas em memória) são tipo de dados que podem ser "constituídos" à medida que se fazem necessários , pois não é sempre que os tipos básicos (real, inteiro, caractere ou lógico) e /ou variáveis simples são suficiente para representar a estrutura de dados utilizado em um programa.

Matrizes de uma Dimensão ou Vetores

Este tipo de estrutura em particular é também denominado por alguns profissionais como matrizes unidimensionais. Sua utilização mais comum está vinculada à criação de tabelas. Caracteriza-se por ser definida uma única variável dimensionada com um determinado tamanho. A dimensão de uma matriz é constituida por constantes inteiras e positivas. Os nomes dados às matrizes seguem as mesmas regras de nomes utilizados para indicar as variáveis simples.

Operações Básiccas com Matrizes do Tipo Vetor

Uma matriz de uma dimensão ou vetor será, neste trabalho, representada por seu nome e seu tamanho (dimensão) entre colchetes, desta forma seria uma matriz MD [1..n], onde MD é o seu nome e possuindo uum tamanho de 1 a n. Isto significa que poderão ser armazenados em MD até um valor definido pelo programador. Perceba que na utilização de variáveis simples existe uma regra: uma variável pode conter apenas um valor por vez. No caso das matrizes, esta poderão armazenar mais um valor por vez. No caso das matrizes, esta poderão armazenar mais de um valor por vez, pos são dimensionadas, exatamente para este fim. Desta forma poder-se á manipular uma quantidade maior de informação com pouco trabalho de processamento. Deve-se apenas considerar que com relação à manipulação dos elementos de uma matriz, eles ocorrerão de forma individualizada, pois nào é possível efetuar a manipulação de todos os elementos do conjunto ao mesmo tempo.

Atribuição de uma Matriz

Anteriormente, foram utilizadas várias instruções em português estruturado para poder se definir e montar um programa. No caso da utilização de matrizes, será definida a instrução **conjunto** que indicará em português estruturado a utilização de uma matriz, tendo como sintaxe: Variável : **conjunto** [<dimensão>] **de** <tipo de dado>. Onde <dimensão> será a indicação dos valores inicial e final do tamanho do vetor e <tipo de dado> se o vetor em questão irá utilizar valores reais, inteiros, lógicos ou caracteres.

Leitura dos Dados de uma Matriz

A leitura de uma matriz é processada passo a passo, um elemento por vez. A instrução de leitura é leia seuida da variável mais o índice. Abaixo, é apresentado a codificação em portuguol da leitura de 8 notas de 8 alunos, cálculo da média e a apresentação da mesma.

```
Programa Media_Turma

Var
    Md : conjunto [1..8] de real
    Soma, media : real
    I : inteiro
Inicio
    Soma <= 0
    Para I de 1 ate 8 faça
        Leia (Md[I])
        Soma <= Soma + MD[I]
    Fim_Para
    Média <= Soma / 8
    Escreva (Media)
Fim.
```

Escrita dos Dados de uma Matriz

O processo de escrita de uma matriz é bastante parecido com o processo de leitura de seus elementos. Para esta ocorrência deverá ser utilizada a instrução **escreva** seguida da indicação da variável e seu índice. Supondo que após a leitura das 8 notas, houvesse a necessidade de apresentá-las antes da apresentação do valor da média. Abaixo é exibido a codificação da escritados 8 alunos antes de ser apresentado o cálculo da média.

```
Programa Media_Turma

Var
    Md : conjunto [1..8] de real
    Soma, media : real
    I : inteiro
Inicio
    Soma <= 0
    Para I de 1 ate 8 faça
        Leia (Md[I])
        Soma <= Soma + MD[I]
    Fim_Para
    Para I de 1 ate 8 faça
        Escreva (Md[I])
    Fim_Para
    Média <= Soma / 8
    Escreva (Media)
Fim.
```

Exercícios de Aprendizagem

1) Desenvolver um programa que efetue a leitura de 10 elementos de uma matriz A do tipo vetor. Construir uma matriz B de mesmo tipo, observando a seguinte lei de formação: Se o valor do índice for par, o valor deverá ser multiplicado por 5, sendo ímpar deverá ser somado por 5. Ao final mostrar os conteúdos das duas matrizes.
2) Desenvolver um programa que efetue a leitura de 5 elementos de uma matriz A do tipo vetor. No final, apresente o total da soma de todos os elementos que sejam ímpares.

3) Ler duas matrizes A e B do tipo vetor com 20 elementos. Construir uma matriz C, onde cada elemento de C é a subtração do elemento correspondente de A com B.
4) Ler 15 elementos de uma matriz A do tipo vetor. Construir uma matriz B de mesmo tipo. Observando a seguinte lei de formação: “Todo elemento de B deverá ser o quadrado do elemento de A correspondente”.
5) Ler uma matriz A do tipo vetor com 15 elementos. Construir uma matriz B de mesmo tipo, sendo que cada elemento da matriz B seja o fatorial do elemento correspondente da matriz ª
6) Ler duas matrizes A e B do tipo vetor com 15 elementos cada. Construir uma matriz C, sendo esta a junção das duas outras matrizes. Desta forma, C deverá Ter a capacidade de armazenar 50 elementos.
7) Ler 20 elementos de uma matriz A tipo vetor e construir uma matriz B de mesma dimensão com os mesmos elementos de A, sendo que estes deverão estar invertidos. Ou seja, o primeiro elemento de A passa a ser o último elemento de B. O segundo elemento de A passa a ser o penúltimo de B e assim por diante.

MATRIZES COM MAIS DE UMA DIMENSÃO

Anteriormente, você teve contato com o uso de uma única variável indexada com apenas uma dimensão (uma coluna e várias linhas), quanto foi utilizado o exemplo para efetuar o cálculo da média geral das médias dos oito alunos. A partir deste ponto, serão apresentadas as tabelas com mais colunas, sendo assim teremos variáveis no sentido horizontal e vertical.

Com o conhecimento adquirido até este ponto, você teria condições suficientes para elaborar um programa que efetuasse a leitura das notas dos alunos, o cálculo da média de cada aluno e no final apresentar a média do grupo, utilizando-se de matrizes unidemensionais. Porém, há de se condiderar que o trabalho seria grande, uma vez que se necessitaria manter um controle de cada índice em cada matriz para um mesmo aluno.

Para facilitar o trabalho com estruturas deste porte é que serão utilizadas matrizes com mais dimensões. A mais comum é a matriz de duas dimensões por se relacionar diretamente com a utilização de tabelas. Matrizes com mais de duas dimensões são utilizadas com menos freqüência, mas poderão ocorrer com momentos em que se necessite trabalhar com um número maior de dimensões, estas serão fáceis de serem utilizadas se você dominar bem a utilização de uma matrz com duas dimensões.

Um importante aspecto a ser considerado é que na manipulação de uma matriz é utilizada uma única instrução de looping(**enquanto, para** ou **repita**). No caso de matrizes com mais dimensões, deverá ser utilizado o número de loopings relativos ao tamanho de sua dimensão. Desta forma, uma matriz de duas dimensões deverá ser controlada com dois loopings, sendo que de três dimensões deverá ser controlada por três loopings e assim por diante.

Em matrizes de mais uma dimensão os seus elementos serão também manipulados de forma individualizada, sendo a referência feita sempre através de dois índices: o primeiro para indicar a linha e o segundo para indicar a coluna. Desta forma, TABELA[2,3], indica que está sendo feita uma referência ao elemento armazenadona linha 3 coluna 3. Pode-se considerar que uma matriz com mais de uma dimensão é também um vetor, sendo válido para este tipo de matriz tudo o que já foi utilizado anteriormente para as matrizes de uma dimensão.

Operações Básicas com Matrizes de Duas Dimensões

Uma matriz de das dimensões está sempre fazendo menção a linhas e colunas e será representada por seu nome e seu tamanho (dimensão) entre colchetes, desta forma seria uma matriz de duas dimensõesTABELA[1..8, 1..5], onde TABELA é o seu nome, possuindo um tamanho de 8 linhas (de 1 a 8) e 5 colunas (de 1 a 5), ou seja, é uma matriz de 8 por 5 (8X5). Isto significa que poderão ser armazenados em TABELA até 40 elementos. A figura 8.1 apresenta a matriz TABELA com a indicação dos endereços (posições) que poderão ser utilizadas para armazenamento de seus elementos.

Atribuição de uma Matriz

Uma matriz de duas dimensões será atribuída pelas instrução **conjunto** já utilizada para definir o uso de uma matriz de uma dimensão, sendo bastante parecidos em sua referência. A sintaze será: VARIAVEL : **conjunto** [<dimensão1: dimensão2>] **de** <tipo de dado>, onde <dimensão1> e <dimensão2> serão indicaçã do tamanho da tabela e <tipo de dado> o tipo da matriz, que poderá ser formada por valores reais, inteiros, lógicos ou caracteres.

Leitura dos Dados de uma Matriz

A leitura de uma matriz de duas dimensões assim como as matrizes de uma dimensão é processada passo a passo, um elemento por vez, sendo utilizada a instrução leia seguida da vairável mais os seus índices. A seguir é apresentado a codificação em português estruturado da leitura das 4 notas bimestrais de 8 alunos, sem considerar o cálculo da média.

```
Programa Ler_Elementos

Var
    Notas :  conjunto [1..8, 1..4] de real
    I, J :  inteiro
Inicio
    Para I de 1 até 8 faça
        Para J de 1 até 4 faça
            Leia(notas[I,J])
        Fim_Para
    Fim_Para
Fim.
```

Escrita dos Dados de uma Matriz

O precesso de escrita será bastante parecido com o processo de leitura de seus elementos. Supondo que após a leitura das notas dos 8 alunos, houvesse a necessidade de efetuar a apresentação das notas. Abaixo é apresentado a codificação em português estruturado da escrita das 4 notas dos 8 alunos.

```
Programa Ler_Elementos

Var
    Notas :  conjunto [1..8, 1..4] de real
    I, J :  inteiro
Inicio
    Para I de 1 até 8 faça
        Para J de 1 até 4 faça
            Escreva(notas[I,J])
        Fim_Para
    Fim_Para
Fim.
```

Exercícios de Aprendizagem

1) Ler duas matrizes A e B, cada uma de duas dimensões com 5 linhas e 3 colunas. Construir uma matriz C de mesma dimensão, onde C é formada pela soma dos elementos da matriz A com os elementos da matriz B.

ESTRUTURA DE DADOS HETEROGÊNEAS

No tópicos anteriores, você teve um contato com técnicas de programação que envolveram o uso de estruturas de dados homogêneas, através da utilização de matrizes de uma e duas dimensões. Observou que somente foi possível trabalhar com um tipo de dado por matriz. No momento em que se precisou trabalhar com dois tipos de dados diferentes, foi necessária a utilização também de duas matrizes, uma de cada tipo.

Estrutura de um Registro

Agora você terá contato com a utilização da principal estrutura de dados, o registro, a qual consiste em trabalhar vários dados de tipos diferentes (os campos) em uma mesma estrutura. Por esta razão, este tipo de dado é considerado heterogêneo.

Atribuição de Registros

Os tipos registro devem ser declarados ou atribuídos antes da variáveis, pois poderá ocorrer a necessidade de se declarar uma variável com o tipo registro anteriormente atribuído.

Para se declarar um tipo registro em português estruturado deverá ser utilizada a instrução tipo em conjunto com a instrução **registro...fim_registro**, conforme a sintaxe indicada abaixo.

Tipo
<identificador> = **registro**
<lista dos campos e seus tipos>
fim_registro
var
<variável> : <identificador>

Onde indentificador é o nome do tipo registro em caracteres maiúsculos, em itálico como as variáveis, e lista dos campos e seus tipos é a relação de variáveis que serão usadas como campos, bem como o seu tipo de estrutura de dados, podendo ser: real, inteiro, lógico ou caractere.

Após a instrução **var**, deverá ser indicada a variável tipo registro e a declaração do seu tipo de acordo com um identificador definido anteriormente. Perceba que a instrução **tipo** deverá ser utilizada antes da instrução **var**, pois ao definir um tipo de variável, pode-se fazer uso deste tipo definido.

Tomando como exemplo a proposta de se criar um registro denominado ALUNO, cujos campos são NOME, NOTA1, NOTA2, NOTA3 e NOTA4, este seria assim declarado:

```
tipo

   CAD_ALUNO = registro
                  NOME : caractere
                  NOTA1 : real
                  NOTA2 :  real
                  NOTA3 :  real
                  NOTA4 :  real
                  fim_registro

var
   ALUNO : CAD_ALUNO
```

Observe que é especificado um registro denominado CAD_ALUNO, o qual é um conjunto de dados heterogêneos (um campo tipo caractere e quatro do tipo real). Desta forma é possível guardar em uma mesma estrutura vários tipos diferentes de dados.

A título de comparação, pode-se dizer que um tipo registro é também um vetor (matriz de uma dimensão). Pois tem-se a variável ALUNO do tipo cad_aluno como um vetor com os índices NOME, NOTA1, NOTA2, NOTA3 e NOTA4.

Leitura de Registros

A leitura de um registro é efetuada com a instrução **leia** seguida do nome da variável registro e seu campo correspondente separado por um caractere "." ponto.

Programa Leitura

```
tipo

   CAD_ALUNO = registro
                  NOME : caractere
                  NOTA1 : real
                  NOTA2 :  real
                  NOTA3 :  real
                  NOTA4 :  real
                  fim_registro

var
   ALUNO : CAD_ALUNO
Início

   Leia (Aluno.Nome)
   Leia (Aluno.Nota1)
   Leia (Aluno.Nota2)
   Leia (Aluno.Nota3)
   Leia (Aluno.Nota4)
Fim
```

Escrita de Registros

O processo de escrita de um registro é feito com a instrução **escreva** de forma semelhante ao processo de leitura.

Programa Escrita

```
tipo

   CAD_ALUNO = registro
                  NOME : caractere
                  NOTA1 : real
                  NOTA2 :  real
                  NOTA3 :  real
                  NOTA4 :  real
                  fim_registro

var
   ALUNO : CAD_ALUNO
Início

   Escreva (Aluno.Nome)
   Escreva (Aluno.Nota1)
   Escreva (Aluno.Nota2)
   Escreva (Aluno.Nota3)
   Escreva (Aluno.Nota4)
Fim
```

Estrutura de um Registro de Conjuntos

No tópico anterior, foi apresentado o conceito de se trabalhar com um registro. No ponto de aprendizado em que estamos, podemos até jurar que você esteja perguntando: Será que não possível definir um vetor ou mesmo uma matriz dentro de um registro, para não ser necessário utilizar somente os tipos primitivos de dados? Isto é realmente possível. Considere ainda o exemplo do registro ALUNO, onde temos o campo NOME tipo caractere e mais quatro campos tipo real para o armazenamento de duas notas sendo, NOTA1, NOTA2, NOTA3 e NOTA4. Veja que se pode definir um vetor chamado NOTA com quatro
Indices, uma para cada nota.

Atribuição de Registros de Conjuntos

Tomando como exemplo a proposta de se criar um registro denominado ALUNO, cujas notas serão informadas em um vetor, este seria bastante declarado:

```
tipo
   BIMESTRE = conjuno [1..4] de real
   CAD_ALUNO = registro
                  NOME : caractere
                  NOTA : BIMESTRE
                  fim_registro
```

Observe que ao ser especificado o registro CAD_ALUNO, existe nele um campo chamado NOTA do tipo bimestre, onde bimestre é a especificação de um tipo de conjunto matricial de um única dimensão com capacidade para quatro elementos. Veja que o tipo de bimestre foi anteriormente definido, pois se caracteriza por um tipo criado, assim como o tipo cad_aluno atribuído à variável de registro ALUNO.

Leitura de Registro de Conjuntos

A leitura de um registro de conjunto é efetuada com a instrução **leia** geralmente dentro de um laço de repitação. Assim sendo, observe o código abaixo:

```
Programa Leitura

tipo
   BIMESTRE = conjunto [1..4] de real
   CAD_ALUNO = registro
                     NOME : caractere
                     NOTA : bimestre
                     fim_registro
var

   ALUNO : CAD_ALUNO
   I : inteiro

Inicio

   Leia (ALUNO.NOME)
   Para de I ate 4 faça
         Leia (ALUNO.NOTA[I])
   Fim_para
Fim.
```

Estrutura de um conjunto de Registros

Com as técnicas de programação anteriormente apresentadas, passou-se a Ter uma mobilidade bastante grande , podendo-se trabalhar de uma forma mais adequada com diversos problemas, principalmente os que envolvem a utilização de dados heterogêneos, facilitando a construção de programas mais eficientes. Porém, os programas apresentados até aqui com a utilização de registros, só fizeram menção à leitura e escrita de um único registro.

Neste momento, você terá contato com o conjunto de registros que permite a construção de programas, onde é possível se fazer a entrada, processamento e saída de diversos registros.

Atribuição de Conjunto de Registros

Para se declarar un conjunto de registros é necessário em primeiro lugar, possuir a definição de um registro, ou seja é necessário Ter um formato de um único registro para então definir o número de registros que será utilizado pelo programa. Para exemplificar o que está sendo exposto, considere que você deverá fazer um programa que leia o nome e as quatro notas escolares de 8 alunos. Isto já e familiar. Veja a seguir a codificação do conjunto de registros para os oito alunos:

Tipo

```
BIMESTRE =  conjunto [1..4] de real
CAD_ALUNO = registro
        NOME : caractere
        NOTA: BIMESTRE
Fim_registro
```

Var

```
ALUNO:  conjunto [1..8] de CAD_ALUNO
```

Observe que após a instrução **var,** é indicada a variável de registros ALUNO, sendo esta um conjunto de 8 registros do tipo Cad_Aluno, que por sua vez é Bimestre é um conjunto de quatro valores reais.

Leitura de Conjunto de Registros

A leitura de forma semelhante às anteriores. No entanto,m serão utilizadas dois laços, pois além de controlar a entrada das quatro notas de cada aluno, tem-se que controlar a entrada de 8 alunos. Esta estrutura é bastante similar a uma matriz de duas dimensões. Assim sendo, observe o código em português estruturado:

Programa LEITURA

Tipo

```
BIMESTRE =  conjunto [1..4] de real
CAD_ALUNO = registro
        NOME : caractere
        NOTA: BIMESTRE
Fim_registro
```

Var

```
ALUNO:  conjunto [1..8] de CAD_ALUNO
I, J :  inteiro
```

início

```
Para J de 1 até 8 faça
        Leia (ALUNO[J].NOME)
        Para I de 1 até 4 faça
                Leia (ALUNO[J].NOME[I])
        Fim_para
Fim_para
```

Fim

Veja que o looping da variável J controla o número de alunos da turma, no caso 8, e o looping da variável I controla o número de notas, até 4 por aluno. Para cada movimentação de mais uma variável J existem quatro movimentações na variável I.

Escrita de Conjunto de Registros

O processo de escrita de um conjunto de registros é similar aos modos anteriores, já estudados. Assim sendo, veja o código:

```
Programa LEITURA

Tipo

   BIMESTRE =  conjunto [1..4] de real
   CAD_ALUNO = registro
         NOME : caractere
         NOTA: BIMESTRE
   Fim_registro

Var
   ALUNO:  conjunto [1..8] de CAD_ALUNO
   I, J :  inteiro

início

   Para J de 1 até 8 faça
         Escreva (ALUNO[J].NOME)
         Para I de 1 até 4 faça
                  Escreva (ALUNO[J].NOME[I])
         Fim_para
   Fim_para

Fim
```

PROGRAMAÇÃO MODULAR

Será estudado a partir deste ponto, a aplicação de subrotinas em algoritmos, também conhecidas pela denominação módulos ou subalgoritmos. Na realidade, não importa como são chamadas, o que importa é a forma como funcionam e como devem ser aplicadas em um programa, e é isto que você aprenderá.

Neste capítulo, será introduzido o conceito da criação estruturada de programas, pois para escrever um programa de computador necessita-se de estudo (levamtamento de todas as necessidades e detalhes do que deverá ser feito) e metodologia (regras básicas que deverão ser seguidas). Sem a aplicação de métodos não será possível resolver grandes problemas, quanto muito, pequenos problemas.

As Sub-Rotinas

No geral, problemas complexos exigem algoritmos complexos. Mas sempre é possível dividir um problema grande em problemas menores. Desta forma, cada parte menor tem um algoritmo mais simples, e é estre trecho menor que é chamado de sub-rotina. Uma sub-rotina é na verdade, um programa, e sendo um programa poderá efetuar diversas operações computacionais (entrada, processamento e saída) e deverá ser tratada como foram os programas projetados até este momento. As sub-rotinas são utilizadas na divisão de

algoritmos complexos, permitindo assim possuir a modularização de um determinado problema, considerado grande e de difícil solução.

Ao se trabalhar com esta técnica, pode-se deparar com a necessidade de se dividir uma sub-rotina em outras tantas quantas forem necessárias, buscando uma solução mais simples de uma parte do problema maior. O processo de dividir sub-rotinas em outras é denominado *Método de Refinamento Sucessivo*.

Procedimentos

Um procedimento é um bloco de programa, contendo início e fim e será identificado por um nome, através do qual será referenciado em qualquer parte do programa principal ou do programa chamador da rotina. Quando uma sub-rotina é chamada por um programa, ela é executada e ao seu término o controle de processamento retorna automaticamente para a primeira linha de instrução após a linha que efetuou a chamada da sub-rotina.

Procedimento <nome do procedimento>
var
<variáveis>
início

<instruções>

fim

A melhor maneira de se entender como trabalhar com sub-rotinas é fazer a sua aplicação em um programa mais complexo. Para tanto, imagne o seguinte problema:

Exercício de Aprendizagem

Cria um programa calculadora que apresente um menu de seleções no programa principal. Este menu deverá dar ao usuário a possibilidade de escolher uma entre quadra operações aritméticas. Escolhida a opção desejada, deverá ser solicitada a entrada de dois números, e processada a operação deverá ser exibido o resultado.

Note que este programa deverá ser um conjunto de 5 rotinas, uma principal e 4 secundárias. A rotina principal efetuará o controle sobre as 4 rotinas secundárias, que por sua vez pedirão a leitura de dois valores, farão a operação e apresentarão o resultado obtido. Observe agora a codificação:

```
Programa calculadora

Var
    Opcao : caractere

{sub-rotinas de cálculos}

procedimento ROTSOMA
var
    r, a, b : real
início
    Escreva (“Rotina de Soma”)
    Escreva (“Entre um valor para A: “)
    Leia(A)
    Escreva (“Entre um valor para B:”)
    Leia(B)
    R <= A + B
    Escreva (“A soma de A com B é = “, R)
Fim

procedimento ROTSSUBTRACAO
var
    r, a, b : real
início
    Escreva (“Rotina de Subtração”)
    Escreva (“Entre um valor para A: “)
    Leia(A)
    Escreva (“Entre um valor para B:”)
    Leia(B)
    R <= A - B
    Escreva (“A subtração de A com B é = “, R)
Fim

procedimento ROTMULTIPLICACAO
var
    r, a, b : real
início
    Escreva (“Rotina de Multiplicação”)
    Escreva (“Entre um valor para A: “)
    Leia(A)
    Escreva (“Entre um valor para B:”)
    Leia(B)
    R <= A * B
    Escreva (“A multiplicação de A com B é = “, R)
Fim

procedimento ROTDIVISAO
var
    r, a, b : real
início
    Escreva (“Rotina de Divisão”)
    Escreva (“Entre um valor para A: “)
```

```
    Leia(A)
    Escreva ("Entre um valor para B:")
    Leia(B)
    R <= A / B
    Escreva ("A Divisão de A com B é = ", R)
Fim

{Programa Principal}
inicio
    opcao <= "0"
    enquanto (opcao <>"5 ") faça
            escreva ("1 – Adição")
            escreva ("2 – Subtração")
            escreva ("3 – Multiplicação")
            escreva ("4 – Divisão")
            escreva ("5 – Fim do Programa")
            escreva ("Escolha uma opção")
            leia (opcao)
            se opcao = "1" então
                rotsoma
            fim_se
            se opcao = "2" então
                rotsubtracao
            fim_se
            se opcao = "3" então
                rotmultiplicacao
            fim_se
            se opcao = "4" então
                rotdivisao
            fim_se
    fim_enquanto
fim.
```

Obs: Observe que em cada sub-rotina possuia suas próprias variáveis, estas porém só podem ser acessadas dentro de cada procedimento, neste caso são chamadas de variáveis locais. No caso da variável *opcao* é uma variável global, pode ser acessada em qualquer local do programa, tanto nos procedimentos como no programa principal.

Funções

Uma função também é um bloco de programa como são os procedimentos, contendo início e fim e sendo identificado por um nome, através do qual também será referenciada em qualquer parte do programa principal. Uma sub-rotina de função é na verdade muito parecida com uma sub-rotina de procedimento. A sintaxe em português estruturado será também idêntica ao estudo anterior. Observe a seguir, o código em português estruturado de uma função.

Português Estruturado

Função <nome da função> (parâmetros) : <tipo da função>
Var

<variáveis>
início
<instruções>
fim.

A sua principal diferença está no fato de uma função retornar um determinado valor, que é retornado no próprio nome da função. Quando se diz valor, devem ser levados em consideração os valores numéricos, lógicos ou literais (caracteres).

Você deve ter observado a presença de um item na sintaxe da função, que são os parâmetros, que tanto podem ser referenciados nos procedimentos, mas vejamos sua utilidade.

Òs parâmetros têm por finalidade servir como um ponto de comunicação bidirecional entre uma sub-rotina e o programa principal ou com uma outra sub-rotina hierarquicamente de nível mais alto. Desta forma, é possível passar valores de uma sub-rotina ou rotina chamadora à outra sub-rotina e vice-versa, através do uso de parâmetros que poderão ser formais ou reais.

Serão considerados parâmetros formais quando forem declarados através de variáveis juntamente com a identificação do nome da sub-rotina, os quais serão tratados exatamente da mesma forma que são tratadas as variáveis globais ou locais.

Serão considerados parâmetros Reais quando estes substituírem os parâmetros formais, quando a utilização da sub-rotina por um programa principal ou por uma rotina chamadora.

Utilização de Funções

Criaremos uma função que calcule a raiz cúbica de um determinado número inserido pelo usuário. Observe o Programa:

```
Programa Raiz_Cubica
Var
  I : Inteiro
Função Cubo(N) : Real
Inicio
    Cubo <=  N * N * N
Fim
Inicio
    Leia (I)
    Escreva(Cubo(I))
Fim.
```

Observe que neste caso o parâmetro formal foi a variável " N ", enquanto o parâmetro real foi a variável "I ".