



Algoritmos

Enviado por:
Jedean Carlos Bendlin

Apostila de Lógica de Programação

PLANO DE ENSINO 1º SEMESTRE

EMENTA

Introdução a Algoritmos, Português Estruturado, Variáveis e Operadores Matemáticos e Lógicos, Estrutura de um programa de computador, Procedimentos e Funções, Tipos de dados.

OBJETIVO GERAL

Instruir os conceitos básicos de algoritmos de programação, familiarizando os acadêmicos com os recursos computacionais na resolução de problemas reais. Com a utilização de pseudocódigos ou português estruturado, aplicar tanto prática como teoricamente, as estruturas básicas de programação de computadores. A aplicação da ementa servirá como manual introdutório para as demais disciplinas que tratarão de análise e desenvolvimento de sistemas computacionais.

OBJETIVOS ESPECÍFICOS

- Incentivar os acadêmicos a raciocinarem logicamente na resolução dos problemas propostos, utilizando, de forma adequada, os recursos da linguagem de programação.
- Propor aos acadêmicos a tomada de iniciativa própria, a criatividade e a pesquisa, itens indispensáveis para o desenvolvimento e capacitação profissional na área de sistemas de informação.

CONTEÚDO PROGRAMÁTICO

Conteúdo	C.H.
1.INTRODUÇÃO Conceito Tipos de algoritmos Descrição geral dos algoritmos	8
2. LÓGICA DE PROGRAMAÇÃO Formas de representação gráfica Princípios de resolução de problemas Tipos de variáveis Uso de Constantes Fórmulas matemáticas Instruções básicas	10
3. TOMADAS DE DECISÃO Desvio condicional simples Desvio condicional composto Desvio condicional encadeados Prática com Linguagem Pascal	18
4.LAÇOS DE REPETIÇÃO Looping com teste lógico no início Looping com teste lógico no fim Looping com variável de controle Prática com Linguagem Pascal	20
5. MATRIZES Matrizes de uma dimensão ou vetores Buscas e ordenação Matrizes com mais de uma dimensão Prática com Linguagem Pascal	16

METODOLOGIA DE ENSINO

O método de ensino propõe problemas com a finalidade de desafiar o estudante a utilizar conhecimentos já adquiridos em busca de uma adaptação a uma situação nova. A intervenção do professor é no sentido de pequenas ajudas, devolução de questões, evidenciar falhas e preparar para novas estratégias. As situações didáticas devem favorecer o compartilhamento do conhecimento, a discussão e a ação por parte dos estudantes. Também serão utilizados exemplos práticos e concretos em problemas diversos.

ATIVIDADES DISCENTES

- Os alunos devem desenvolver atividades relacionadas ao conteúdo que simulam situações reais no desenvolvimento de sistemas orientados a objetos.
- Fazer a análise de softwares utilizados em empresas com intuito de criar idéias e as por em prática.
- Desenvolver pesquisas bibliográficas e on-line.
- Desenvolver soluções para problemas apresentados em aula, utilizando conhecimentos adquiridos.

PROCEDIMENTOS DE AVALIAÇÃO

Avaliações Bimestrais - As avaliações serão de forma contínua pela participação dos alunos nas atividades de classe, com peso de 30% da nota integral. Trabalhos feitos no laboratório e atividades extra classe, individualmente ou em grupo de estudos, com peso de 10% da nota integral. Uma prova bimestral para reconhecimento da fixação do conteúdo, com valor de 60% da nota integral.

BIBLIOGRAFIA BÁSICA

FORBELLONE, A. L. V. ; EBERSPACHER, H. F. Lógica de programação. 2.ed. São Paulo: Makron Books, 2000. 3ex.

MANZANO, J.A.N.G. ; YAMATUMI, W. Y. Estudo dirigido de algoritmos. São Paulo: Érica, 1997. 3ex.

LOPES, Anita; GARCIA, Guto. Introdução a Programação – 500 algoritmos resolvidos. Rio de Janeiro: Campus, 2002. 3ex

WOOD, S. Turbo pascal: guia do usuário. São Paulo: Mcgraw-Hill do Brasil, 1987. 1ex.

BIBLIOGRAFIA COMPLEMENTAR

GOODRICH, M. ; TAMASSIA R. Estruturas de Dados e Algoritmos em Java. São Paulo: Bookman, 2002. 6ex.

BEZERRA, Eduardo. Princípios de Análise e Projeto de Sistemas Com UML. Rio de Janeiro: Campus, 2003. 6ex.

WORTMAN, L. A. Programando em turbo pascal: com aplicações. Rio de Janeiro: Campus, 1988. 1ex.

LAURA, Lemay. Aprenda em 21 dias Java 2. São Paulo: Campus, 2003. 3 ex.

SANTOS, Rafael. Introdução à Programação Orientada a Objetos Usando Java. São Paulo: Campus, 2003. 3ex.

NIEMEYER, Patrick. Aprendendo Java. São Paulo: Campus, 2000. 3ex

PLANO DE ENSINO 2º SEMESTRE

EMENTA

Lógica de programação avançada incluindo sub-rotinas, registros, eficiência e correção. Análise de algoritmos de programação estruturados. Aplicação em uma linguagem de programação.

OBJETIVO GERAL

Desenvolver ao aluno habilidades técnicas necessárias ao conhecimento de teorias para composição de programas.

CONTEÚDO PROGRAMÁTICO

Conteúdo	C.H.
1. Estruturas de dados <ul style="list-style-type: none">• Matrizes unidimensionais e multidimensionais• Registros• Matrizes de registros e registros de matrizes• Armazenamento permanente de dados utilizando arquivos	30
2. Modularização de programas <ul style="list-style-type: none">• Procedimentos e funções• Passagem de parâmetros• Units• Exercícios de programação	26
3. Análise de Algoritmos <ul style="list-style-type: none">• Busca• Métodos de Ordenação	8
4. Utilização de Arquivos e Noções de Índices	12

METODOLOGIA DE ENSINO

O método de ensino é baseado na Teoria das Situações de Guy Brousseau (1996. Recherches en Didactique des Mathématiques, vol.9, n.3) em que problemas são propostos com a finalidade de desafiar o estudante a utilizar conhecimentos já adquiridos em busca de uma adaptação a uma situação nova. A intervenção do professor é no sentido de pequenas ajudas, devolução de questões, evidenciar falhas e preparar para novas estratégias. As situações didáticas devem favorecer o compartilhamento do conhecimento, a discussão e a ação por parte dos estudantes. Também serão utilizados materiais concretos em problemas diversos, proporcionando uma mudança de contrato didático.

ATIVIDADES DISCENTES

- Os alunos devem desenvolver atividades relacionadas ao conteúdo que simulam situações reais no desenvolvimento de sistemas orientados a objetos.
- Fazer a análise de softwares utilizados em empresas com intuito de criar idéias e as por em prática.
- Desenvolver pesquisas bibliográficas e on-line.
- Desenvolver soluções para problemas apresentados em aula, utilizando conhecimentos adquiridos.

PROCEDIMENTOS DE AVALIAÇÃO

Avaliações Bimestrais - As avaliações serão de forma contínua pela participação dos alunos nas atividades de classe, com peso de 30% da nota integral. Trabalhos feitos

no laboratório e atividades extra classe, individualmente ou em grupo de estudos, com peso de 10% da nota integral. Uma prova bimestral para reconhecimento da fixação do conteúdo, com valor de 60% da nota integral.

BIBLIOGRAFIA BÁSICA

DEITEL, H. M. Java Como Programar 4 ed, Porto Alegre: Bookman, 2003. 6ex.

LOPES, Anita GARCIA Guto. Introdução a Programação- 500 algoritmos resolvidos. Rio de Janeiro: Campus, 2002.

BIBLIOGRAFIA COMPLEMENTAR

LAURA, Lemay. Aprenda em 21 dias Java 2. São Paulo: Campus, 2003. 3 ex.

SANTOS, Rafael. Introdução à Programação Orientada a Objetos Usando Java. São Paulo: Campus, 2003. 3ex.

NIEMEYER, Patrick. Aprendendo Java. São Paulo: Campus, 2000. 3ex

1 – Introdução

Conceito de 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.

A técnica mais importante no projeto da lógica de programas é chamada programação estruturada, a qual consiste em uma metodologia de projeto, objetivando:


- Agilizar a codificação da escrita da programação;
- Permitir a verificação de possíveis falhas apresentadas pelos programas;
- Facilitar as alterações e atualizações dos programas.


2 – Lógica de Programação


Formas de representação gráfica

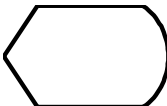
São vários os símbolos utilizados na programação estruturadas, a seguir, uma relação das simbologias mais utilizadas:

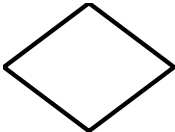
 Terminal: indica o início ou o fim de um programa

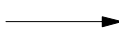
 Conector: serve para a união dos fluxos de informações

 Entrada: indica a entrada de informações passadas pelo usuário, seja por scanner, teclado, mouse ou qualquer dispositivo de entrada

 Processamento: utilizado para fazer cálculos e atribuição de valores

 Saída: indica a saída de informações processadas para o usuário por qualquer dispositivo de saída como monitor, impressora, etc.

 Condição: indica a divisão do fluxo por uma determinada condição.

 Seta de fluxo de informações: indica o caminho que o processamento pode tomar.

Princípios de resolução de problemas

Para desenvolver um diagrama correto, devemos levar como procedimentos prioritários, os itens a seguir:

1 – Os diagramas de blocos devem ser feitos e quebrados em vários níveis. Os primeiros devem conter apenas as idéias gerais, deixando para as etapas posteriores os detalhes necessários;

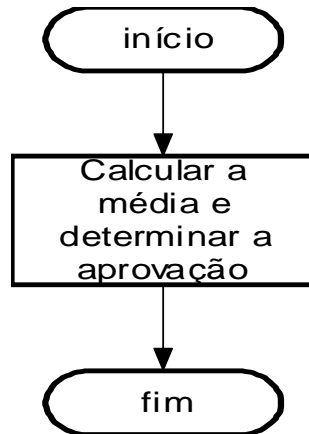
2 – Para o desenvolvimento correto de um diagrama de bloco, sempre que possível deve ser feito de cima para baixo e da esquerda para a direita;

3 – É incorreto ocorrer cruzamento das linhas de fluxo de dados de um diagrama de bloco.

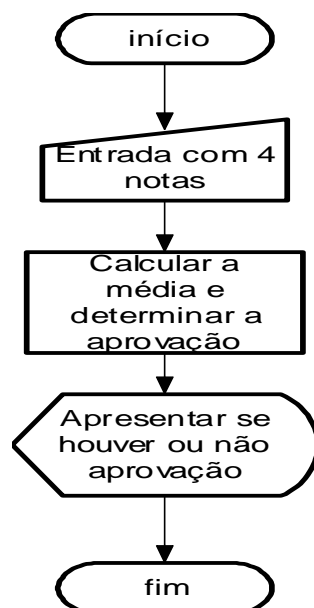
4 – Transcrever o diagrama de bloco em pseudolinguagem

Tomemos como exemplo uma escola qualquer, onde o cálculo da média é

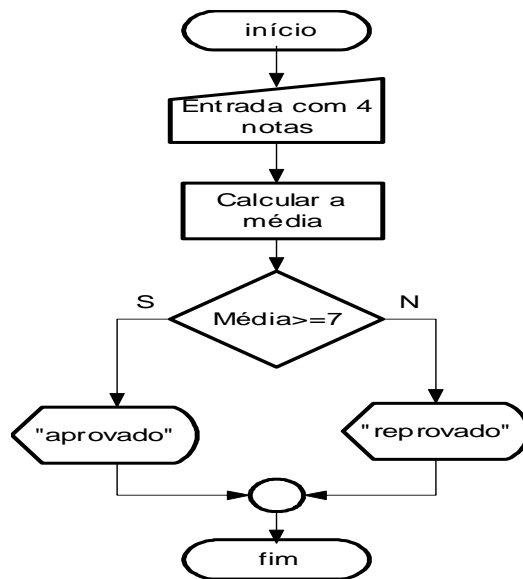
realizado por quatro notas bimestrais que determinam a aprovação ou reprovação dos seus alunos. Considere ainda, que o valor da média deve ser maior ou igual a 7 para que haja aprovação. A primeira etapa se inicia e termina com um terminador e existe apenas um processamento que indica a idéia geral do problema:



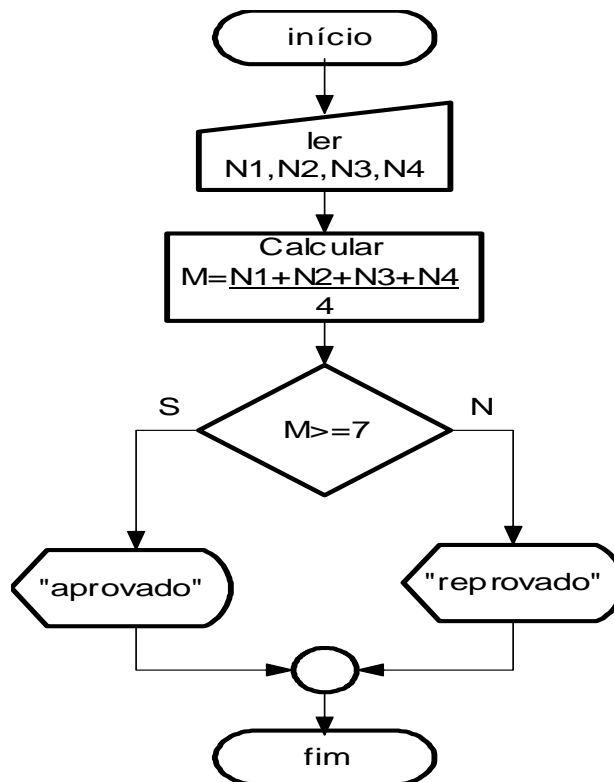
O segundo detalhamento está no que se refere a entrada e saída dos dados do problema:



A terceira etapa consiste em trabalhar o termo “determinar a aprovação”. Para ser possível determinar algo é necessário estabelecer uma condição. Esta condição decide sobre o resultado da aprovação:



Esta terceira etapa deve ser aperfeiçoada para trabalhar com variáveis.



A Quarta fase consiste em escrever o diagrama de bloco de forma narrativa denominada pseudocódigo.

```

Programa média
Var
    Resultado : caractere
    N1,n2,n3,n4,soma,media : real
Inicio
    Leia(n1,n2,n3,n4)
    Soma<-n1+n2+n3+n4
    Media <-soma/4
    Se (media >=7) então
        Resultado <-“APROVADO”
    Senão
        Resultado<- “REPROVADO”
    Fim_se
    Escreva(Resultado)
fim

```

Tipos de variáveis

Tipos Inteiros: São tipos inteiros, os dados numéricos positivos ou negativos, excluindo-se, destes qualquer fracionário. Como exemplo deste tipo de dado, têm-se os valores : 35,0,-56;

Tipos Reais: São tipos reais, os dados numéricos positivos, negativos e números fracionários. Ex. : 35,0,-56,1.2,-45.897

Tipos Caracteres: São caracteres, as sequências contendo letras, números e símbolos especiais. Uma sequência de caractere deve ser indicada entre aspas(“”). Ex.: “PROGRAMAÇÃO”, “Rua Alfa”, “”, “98”.

Tipos Lógicos: São tipos lógicos ou booleanos, os dados com valores verdadeiro ou falso, e deve ser escrito entre pontos. Ex.: .Falso.,.Verdadeiro..

O nome de uma variável é utilizado para sua identificação e posterior uso dentro de um programa, sendo assim, é necessário estabelecer algumas regras de utilização das mesmas:

- Nomes de variáveis poderão ser atribuídos com um ou mais caracteres;
 - O primeiro caractere do nome de uma variável não poderá ser em hipótese alguma, um número, sempre deverá ser uma letra;
 - O nome de uma variável não poderá possuir espaços em branco;
 - Não poderá ser nome de uma variável, uma instrução de programa;
 - Não poderão ser utilizados outros caracteres a não ser letras e números.
- Obs.: o caracter “_” Under-line ou sub-linha é considerado uma letra.

Uso de Constantes

Tem-se como definição de constante tudo aquilo que é fixo ou estável. E existirão vários momentos em que este conceito deverá estar em uso. Por exemplo, o valor 1.23 da fórmula a seguir é uma constante: RESULTADO=ENTRADA*1.23.

Operadores aritméticos

Operador	Operação	Prioridade Matemática
+	Manutenção de sinal (números positivos)	1

-	Inversão de sinal (números negativos)	1
Seta para cima	Exponenciação	2
/	Divisão	3
*	Multiplicação	3
+	Adição	4
-	Subtração	4

Fórmulas matemáticas

Na programação é comum usar fórmulas matemáticas, mas para facilitar a escrita da programação as fórmulas são escritas de uma forma diferenciada, como mostrados nos exemplos a seguir:

(Exemplificar a fórmula do raio e da área)

Instruções básicas

Cada linguagem de programação usa um conjunto de palavras onde cada uma desenvolve uma ação. Estas palavras reservadas de cada linguagem são conhecidas como comandos ou instruções. Daqui para frente vamos utilizar instruções em português estruturado, tais como : início, fim, var , programa, enquanto, se, então, senão, para, escreva, leia, faça, repita e até que, entre outras que serão estudadas.

Exemplo de algoritmo

Ler dois valores
Efetuar a soma destes valores
Apresentar o resultado

Exercícios

1 – Indique o tipo de variável que poderão armazenar os valores abaixo.

I : inteiro;

R : real;

S : String ou Literal ou Caractere;

L : Lógico ou Booleano.

() 1000	() "0"	() "-900"	().Verdadeiro.
() -456	() 34	() "Casa 8"	() 0
() -1.56	() ".falso."	() 1.87	() .F.

2 – Assinale com X os nomes válidos de uma variável

() Endereco	() 21brasil	() Fone\$com
() NomeUsuario	() Nome_usuario	() Nome*usuario

☐ End-A ☐ Cidade3 ☐ #Cabec
☐ 23 ☐ N23

3 – Desenvolva a lógica de um programa que efetue o cálculo da área de uma circunferência, apresentando a medida da área calculada.

Fórmula : $Area = \pi * Raio^2$

4 – Elaborar um algoritmo que calcule o salário de um professor onde o usuário deverá entrar com o número de aulas dadas, o valor da hora aula e o percentual de desconto do INSS.

5 – Ler dois valores A e B, efetuar a troca dos valores de forma que a variável A passe a possuir o valor da variável B e que a variável B passe a possuir o valor da variável A. Apresentar os valores trocados

6 – Ler dois valores A e B, efetuar as operações de adição, subtração, multiplicação e divisão de A por B, apresentando no final os quatro resultados obtidos.

7 – Efetuar o cálculo do valor de uma prestação em atraso, utilizando a fórmula: $PRESTAÇÃO = VALOR + (VALOR * (TAXA / 100) * TEMPO)$.

8 – Indique o tipo de variável que poderão armazenar os valores abaixo.

I : inteiro; R : real; S : String ou Literal ou Caractere; L : Lógico ou Booleano.

☐ -900 ☐ 0 ☐ "-900" ☐ .Verdadeiro.
☐ 4.54 ☐ 34 ☐ ".falso." ☐ "Real"

9 – Assinale com X os nomes válidos de uma variável

☐ Dia 21 ☐ 7quedas ☐ C/C
☐ x*y ☐ Senha_conta2 ☐ Nome Cliente

10 – Desenvolva um fluxograma com os 3 níveis de detalhamento e o português estruturado de um programa que permita ao usuário digitar um número e que apresente como resultado este número elevado ao quadrado.

3 – Tomadas de Decisões

Você já aprendeu como trabalhar com entrada, processamento e saída. Apesar de já conseguir solucionar problemas e transformá-los em programas, os recursos aqui estudados são limitados, pois haverá momentos em que um determinado valor dentro de um programa necessitará ser tratado para se efetuar um processamento mais adequado.

Desvio condicional simples

Para aprender-mos desvio condicional devemos conhecer os operadores utilizados para se estabelecer uma condição. São eles:

Operadores Relacionais

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

Operador Lógico .e.

Condição1	Condição 2	Resultado
F	F	F
V	F	F
F	V	F
V	V	V

Operador Lógico .ou.

Condição1	Condição 2	Resultado
F	F	F
V	F	V
F	V	V
V	V	V

Operador Lógico .não.

Condição	Resultado
V	F
F	V

Imagine a seguinte situação: um programa que apresente a média escolar de um aluno. Baseado em sua média, se o aluno tiver nota maior ou igual a 7 deverá apresentar uma mensagem parabenizando o aluno.

Para solucionar o problema proposto será necessário trabalhar uma nova instrução: SE...ENTÃO...FIM_SE. Esta instrução tem por finalidade tomar uma decisão. Sendo a condição verdadeira, serão executadas todas as instruções que estejam entre a instrução se...então e a instrução fim_se.

Sintaxe :

SE <condição> ENTÃO

 <instruções para condição verdadeira>

FIM_SE

 <instruções executadas se a condição for falsa ou após condição ser verdadeira>

Exemplo : elaborar um programa que pessa dois valores para as variáveis A e B. Efetuar a soma dos dois valores. Apresentar o resultado se a soma for maior que 10.

Desvio condicional composto

O desvio condicional composto utiliza a estrutura SE...ENTÃO...SENAO...FIM_SE, como mostra a sintaxe :

SE <condição> ENTÃO

 <instruções para condição verdadeira>

SENAO

 <instruções para condição falsa>

FIM_SE

Exemplo : Ler dois valores A e B. Efetuar a soma dos dois valores e implicar em X. Verificar se X é maior ou igual a 10, caso sim, mostre X+5, senão, mostre X-7.

Exercícios :

1 – Determine o resultado lógico das expressões abaixo baseado nos seguintes valores: X=1, A=3,B=5,C=8 e D=7.

.não.(x>3) (X<1).e..não.(b>d)

.não.(d<0).e.(c>5)

.não(x>3).ou.(c<7) (a>b).ou.(c<=5)

(x>=2)

(x<1).e.(b>=d) (d<0).ou.(c>5)

.não.(d>3).ou..não.(b<7)

(a>b).ou..não.(c>b)

2 – Efetuar a leitura de dois valores e apresentá-los em ordem crescente.

Desvio condicional encadeados

Sintaxe :

```
SE <condição> ENTÃO
    SE <condição> ENTÃO
        <instruções para condição verdadeira>
    SENAO
        <instruções para condição falsa>
FIM_SE
SENAO
    SE <condição> ENTÃO
        <instruções para condição verdadeira>
    SENAO
        <instruções para condição falsa>
FIM_SE
FIM_SE
```

Exemplo : Ler 4 notas escolares de um aluno. Se o a média do aluno for maior ou igual a 7 apresentar a média e uma mensagem dizendo que foi aprovado, senao, efetuar a leitura da nota do exame, somar à média e dividir por dois, se a média for maior ou igual a 5 apresentar a média e uma mensagem dizendo que está aprovado, senao apresentar uma mensagem que está reprovado.

Exercícios:

1 – Elaborar um programa que efetue o cálculo do reajuste de salário de um funcionário. Considere que o funcionário deverá receber um reajuste de 15% caso seu salario seja menor que 500. Se o salario for maior ou igual a 500 mas n\menor ou igual a 1000, seu reajuste será de 10%, e caso seja ainda maior que 1000, o reajuste deverá ser de 5%.

ler um valor para o salario

verificar se o valor de salario < 500; se sim, reajustar em 15%

verificar se o valor de salario <=1000; se sim, reajustar em 10%

verificar se o valor de salario > 1000; se sim, reajustar em 5%

apresentar o valor reajustado.

2 – Efetuar a leitura de três valores (a,b,c) apresentá-los em ordem crescente.

3 – Efetuar a leitura de três valores (a,b,c) e efetuar o cálculo da equação de segundo grau, apresentando as duas raízes, se para os valores informados for possível efetuar o referido cálculo.

4 – Ler dois valores numéricos e apresentar a diferença do maior para o menor.

5 – Efetuar a leitura de um número inteiro positivo ou negativo e apresentá-lo como sendo positivo.

6 – Efetuar a leitura de 4 numeros inteiros e apresentar os que são divisíveis por 2 ou 3.

4 – Laços de Repetição

Looping com teste lógico no início

Este tipo de estrutura efetua um teste lógico no início do looping, verificando se é permitido executar o trecho de instruções subordinado a este looping. Esta estrutura é conhecida como: enquanto, através do conjunto de instruções ENQUANTO...FAÇA...FIM_ENQUANTO.

Esta estrutura, tem o seu funcionamento controlado por decisão, podendo 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.

SINTAXE

```
ENQUANTO <condição> FAÇA
    <conjunto de instruções para condição verdadeira>
FIM_ENQUANTO
```

Exemplo: Considere o problema: Pedir a leitura de um valor para a variável x, multiplicar este valor por 3, implicando-o á variável de resposta R e apresentar o valor R obtido, repetindo esta seqüência por 5 vezes.

Passos:

- ✓ criar uma variável para servir como contador com valor inicial 1;
- ✓ enquanto o valor do contador for < ou = a 5, processar os passos 3,4 e 5;
- ✓ Ler um valor para a variável X;
- ✓ Efetuar a multiplicação do valor de x por 3, implicar o resultado em r;
- ✓ apresentar o valor calculado contido na variável R;
- ✓ Acrescentar +1 á variável contador;
- ✓ Quando contador for maior que 5, encerrar o processamento do looping.

Sugestão: fazer o mesmo exercício enquanto o usuário desejar.

Exercício de aprendizagem

Elaborar o algoritmo de um programa que efetue o calculo da fatorial do número 5. Fatorial é o produto dos números naturais desde 1 até o interiroy n. Sendo assim o cálculo de um fatorial é conseguido pela multiplicação sucessiva do número de termos. Exemplo:

CONTADOR	FATORIAL	RESULTADO
1	1	1
2	1	2
3	2	6
4	6	24
5	24	120

Logo vemos que o fatorial de 5 é 120.

```

PROGRAMA FATORIAL
VAR  contador, resultado: INTEIRO;
INICIO
    Resultado<-1
    Contador<-1
    ENQUANTO contador<=5 FAÇA
        Resultado<-resultado+contador
        Contador<-contador+1
    FIM_ENQUANTO
    ESCREVA('O fatorial de 5 é ',resultado)
FIM

```

Exercícios de Fixação:

- 1 - Apresentar todos os valores numéricos ímpares situados na faixa de 0 a 20.
- 2 - Apresentar o total da soma obtida dos cem primeiros números inteiros.
- 3 - Apresentar os resultados de uma tabuada de um número apresentado pelo usuário. Todo o programa deve se repetir enquanto o usuário desejar.
- 4 - Apresentar os quadrados dos números inteiros de 15 a 200

Looping com teste lógico no fim

Caracteriza-se por uma estrutura que efetua um teste lógico no fim de um looping, denominada repita, sendo conseguida com a utilização do conjunto de instruções repita...até_que.

A estrutura repita...até_que tem o seu funcionamento controlado também 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.

Exemplo: "Pedir a leitura de um valor para a variável X, multiplicar este valor por 3, colocando o resultado em uma variável R e apresentar o valor. Tudo isso deverá ser repetido por 5 vezes".

```

Criar uma variável contador;
Ler um valor para a variável X;
Efetuar a multiplicação do valor de X por 3, implicando o resultador em R;
Apresentar o valor calculado contido na variável R;
Acrescentar 1 ao contador;
Repetir os passos 2,3,4 e 5 até que o contador seja maior que 5.

```

Exercícios:

- a) Apresentar todos os valores numéricos inteiros pares situados na faixa de 100 a 200.
- b) Apresentar o total da soma obtido dos cinco primeiros números inteiros.
- c) Apresentar a tabuada de um número qualquer. Todo o programa deve se repetir enquanto o usuário desejar.
- d) Apresentar todos os números divisíveis por 4 que sejam menores que 20.
- e) Apresentar os quadrados dos números inteiros de 2 a 50.
- f) Elaborar um programa que apresente no final, o somatório dos valores pares existentes na faixa de 10 até 20.

Looping com variável de controle

Os loopings que possuem um número finito de execuções poderão ser processados através de estrutura para, sendo conseguida com a utilização das instruções para...de...até...passo...faça...fim_para.

Esta estrutura tem o seu funcionamento controlado por uma variável denominada contador. Sendo assim, poderá executar um determinado conjunto de instruções um determinado número de vezes.

Sintaxe:

```
PARA <variável> DE <início> ATÉ <fim> PASSO <incremento> FAÇA
    <instruções>
FIM_PARA
```

Para exemplificar considere o problema anterior : "Pedir a leitura de um valor para a variável X, multiplicar este valor por 3, colocando o resultado em uma variável R e apresentar o valor. Tudo isso deverá ser repetido por 5 vezes".

Definir um contador variando de 1 a 5;

Ler um valor para a variável X;

Efetuar a multiplicação do valor de X por 3, implicando o resultador em R;

Apresentar o valor calculado contido na variável R;

Repetir os passos 2,3,4 e 5 até que o contador seja maior que 5.

Exercícios:

- a) Apresentar todos os valores numéricos inteiros ímpares situados na faixa de 1000 a 1500.
- b) Apresentar o total da soma obtido de N números inteiros onde N é um número digitado pelo usuário.
- c) Apresentar a tabuada de um número qualquer.
- d) Apresentar os números divisíveis por 3 que sejam menores que 12.
- e) Elaborar um programa que apresente no final, o somatório dos valores pares existentes na faixa de 10 até 20.

5 - Matrizes

As matrizes consistem em uma técnica de programação que permite trabalhar com o agrupamento de várias informações dentro de uma mesma variável. Este agrupamento ocorrerá obedecendo sempre ao mesmo tipo de dado, e por esta razão é chamado de estruturas de dados homogêneas ou matriz.

Matrizes de uma dimensão ou vetores

Este tipo de estrutura é também denominado por alguns 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 constituída por constantes interias e positivas. Os nomes dados às matrizes seguem as mesmas regras de nomes utilizados para indicar as variáveis simples.

Para se Ter uma idéia de como utilizar matrizes em uma determinada situação, considere o seguinte problema: "Calcular e apresentar a média geral de uma turma de 8 alunos. A média a ser obtida deve ser a média geral das médias de cada aluno obtida durante o ano letivo". Desta forma será necessário somar todas as médias e dividí-las por 8. A tabela a seguir, apresenta o número de alunos, suas notas bimestrais e respectivas médias anuais.

ALUNO	MÉDIA
1	4.5
2	6.5
3	8.0
4	6.5
5	6.0
6	7.0
7	6.5
8	6.0

Agora basta escrever um programa para efetuar o cálculo das 8 médias de cada aluno. Para representar a média do primeiro aluno será utilizada a variável MD1, para o segundo MD2 e assim por diante. Então tem-se:

MD1=4.5;MD2=6.5;MD3=8.0;MD4=6.5;MD5=6.0;MD6=7.0;MD7=6.5;MD8=6.0.

Com o conhecimento adquirido até este momento, seria então elaborado um programa que efetuaria a leitura de cada nota, a soma das mesmas e a divisão do valor da soma por 8, obtendo-se desta forma a média, conforme exemplo abaixo:

```

Programa média_turma
Var
    MD1,MD2,MD3,MD4,MD5,MD6,MD7,MD8 : real
    Soma, media : real
Inicio
    Soma<-0
    Leia MD1,MD2,MD3,MD4,MD5,MD6,MD7,MD8
    Soma<- MD1+MD2+MD3+MD4+MD5+MD6+MD7+MD8
    Media<-soma/8
    Escreva media
Fim

```

Perceba que para receber a média foram utilizada 8 variáveis. Com a técnica de matrizes poderia Ter sido utilizada apenas uma variável com a capacidade de armazenar 8 valores.

Vamos utilizar como representação de nome a forma mais comum usada pelas linguagens de programação, que é o nome da variável e em seguida, entre colchetes, a dimensão da mesma. Desta forma, teríamos uma matriz MD[1..8], onde seu nome é MD, possuindo um tamanho de 1 a 8.

```

MD[1]=4.5;
MD[2]=6.5;
MD[3]=8.0;
MD[4]=3.5;
MD[5]=6.0;
MD[6]=7.0;
MD[7]=6.5;
MD[8]=6.0

```

Observe que o nome é um só, o que muda é a informação indicada dentro dos colchetes. A esta informação dá-se o nome de índice, sendo este o endereço onde o elemento está armazenado.

Para definir as variáveis no português estruturado utilizaremos a seguinte sintaxe:

VARIAVEL : Matriz[<dimensão>] de <tipo de dado>

Desta forma utilizaria-mos no exemplo :

```

Var
    MD : Matriz[1..8] de real;

```

Obs.: outros autores poderão utilizar outros tipos de declaração, isto também acontece com as linguagens de programação.

A leitura e escrita da matriz é feita passo a passo, um elemento por vez com as

instruções leia e escreva, como as variáveis simples, embora deva-se observar o índice.
Exemplo:

Escreva MD[2] {este comando escrevera o valor 6.5}

Leia MD[1] {este comando efetua a leitura de um valor para a nota do 1º aluno}

Abaixo é apresentado o algoritmo da leitura das notas dos 8 alunos, cálculo da média e a apresentação da mesma e apresentação das notas lidas.

```
Programa média_turma
Var
  MD : matriz[1..8] de real
  Soma, media : real;
  I : inteiro;
Inicio
  Soma<-0
  Para i de 1 até 8 passo 1 faça
    Leia MD[i]
    Soma<-soma+MD[i]
  Fim_para
  Media<-soma/8
  Escreva media
  Para i de 1 até 8 passo 1 faça
    Escreva MD[i]
  Fim_para
Fim
```

Exercícios

1 - Desenvolva um programa que efetue a leitura de 10 valores para uma matriz A e que esses valores sejam passados para a matriz B acrescentando 10 por cento a cada elemento.

2 - Desenvolva um programa que efetue a leitura de 10 valores para uma matriz A e que esses valores sejam passados para a matriz B verificando se o índice for par deverá ser multiplicado por 5 e se for ímpar, somado a 5.

3 - Desenvolva um programa que efetue a leitura de 10 elementos para uma matriz A e apresente no final a soma dos elementos ímpares.

4 - Ler duas matrizes A e B com 20 elementos. Construir uma matriz C, sendo esta formada pelos elementos de A subtraídos dos elementos de B.

5 - Ler duas matrizes A e B de 4 elementos cada. Construir uma matriz C, sendo esta a junção das duas outras matrizes. Desta forma, C deverá Ter o dobro de elementos, ou seja, 8.

6 - Ler 20 elementos de uma matriz A 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 de B, o segundo elemento de A passa a ser o penúltimo de B e assim por diante.

Matrizes com mais de uma dimensão

Em matrizes unidimensionais a manipulação dos elementos "é feita através de um único looping (enquanto, para ou repita). No caso de matrizes com mais dimensões, deverá ser utilizado o número de loopings relativo 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.

Nestas matrizes, 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 para referenciar um elemento que está na Segunda linha e na terceira coluna de uma matriz, tem-se tabela[2,3]. Para declarar uma matriz de 8 linhas e 5 colunas, por exemplo, ficaria :

```
VAR <nomevariavel> : MATRIZ[dimensão1,dimensão2] DE <tipodedado>;
```

Caso 1 - Como modelo, vamos utilizar dos exemplos, o primeiro, que efetue a leitura de 4 notas para 4 bimestres e apresente no final as médias de cada bimestre :

Existe uma tabela aqui

PROGRAMA EXEMPLO1

{declaração das variáveis}

VAR

Notas : MATRIZ[1..4,1..4] DE REAL;

Nota, Bim : integer;

Soma : REAL;

INICIO

{leitura dos elementos da matriz}

PARA bim DE 1 ATE 4 FACA

PARA nota DE 1 ATE 4 FACA

LEIA(notas[bim,nota])

FIM_PARA

FIM_PARA

{cálculo e apresentação das médias}

PARA bim DE 1 ATE 4 FACA

Soma <-0

PARA nota DE 1 ATE 4 FACA

Soma <- soma +notas[bim,nota]

FIM_PARA

ESCREVA('A nota do bimestre ',bim,' é ',soma/4)

FIM_PARA

FIM

Caso2 - Imaginemos a seguinte matriz tridimensional:

jk	Gf	sa	R	1	5	7	8	k	l	j	N
a	Ee	x	f	A	g	d	F	h	f	h	H

Considerando esta matriz com 2 linhas e 3 colunas e cada elemento (célula) da tabela é dividida em 4 partes, temos uma matriz com 3 dimensões que pode ser declarada da seguinte maneira:

```
VAR
  Tabela : array[1..2,1..3,1..4] de caractere;
```

A maneira mais correta de efetuar a leitura e escrita de matrizes é utilizando um laço de repetição para cada dimensão da matriz. `

Para efetuar a leitura do elemento que está na linha 2 coluna 3 elemento 4 utilizamos:

```
Escreva('Digite o elemento da linha 2 coluna 3 elemento 4:');
Leia(tabela[2,3,4];
```

Ou para ler todos os elementos:

```
Programa Leitura;
Var
  Tabela: array[1..2,1..3,1..4] de caractere;
  l,c,e: integer;
inicio
  para l de 1 ate 2 passo 1 faça {percorre as linhas de 1 a 2}
    para c de 1 ate 3 passo 1 faça {percorre as colunas de 1 a 3}
      para e de 1 até 4 passo 1 faça {percorre os elementos de 1 a 4}
        escreva('Digite o elemento da linha ',l,' coluna ',c,' elemento ',e,' : ');
        leia(tabela[l,c,e];
      fim_para
    fim_para
  fim_para
fim
```


Exercícios

1 – Desenvolver um programa de agenda que cadastre o nome, endereço, cep de 4 pessoas e em seguida liste-os.

2 – Crie um programa que faça a leitura de 10 nomes de alunos onde cada aluno deverá ter 4 notas. Para isso crie uma matriz unidimensional para armazenar os nomes e uma bidimensional para as notas. Após efetuar a leitura o programa deve apresentar os alunos e as médias.

6 - Buscas e ordenação de matrizes

Busca sequencial

Caso1: Efetuar a leitura de 5 nomes para uma matriz A. Solicitar um valor para pesquisa do usuário, pesquisar na matriz os nomes que coincidem com o valor e apresentar os nomes e as posições dos elementos encontrados. Caso não tenha encontrado, apresente uma mensagem que o valor de busca não foi encontrado.

Caso2: Efetuar a leitura de 5 alunos para uma matriz “alunos” e suas 5 médias para uma matriz chamada “medias”. Apresente os alunos que têm nota maior que 7 dizendo que estão para exame. Se não foi encontrado nenhum aluno, apresente uma mensagem que nenhum aluno ficou para exame.

Algoritmos de busca

(matéria aplicada a estrutura de dados)

Ordenação

Imagine um programa com 5 elementos inteiros em uma matriz A dispostos como mostrado a tabela abaixo :

MATRIZ A	
INDICE	ELEMENTO
1	9
2	8
3	7
4	5
5	3

Ou seja :

A[1]=9

A[2]=8

A[3]=7

A[4]=5

A[5]=3

Para efetuar o processo de troca é necessário aplicar o método de propriedade distributiva, sendo assim, o elemento que estiver em A[1] deverá ser comparado com os elementos que estiverem em A[2], A[3], A[4] e A[5]. Depois, o elemento que estiver em A[2] não necessita ser comparado com o elemento que estiver em A[1], pois já foram anteriormente comparados, passando a ser comparado somente com os elementos que estiverem em A[3], A[4] e A[5]. Na sequência, o elemento que estiver em A[3] é comparado com os elementos que estiverem em A[4] e A[5] e por fim o elemento de A[4] comparado com A[5]. A cada comparação se o primeiro elemento comparado for maior que o segundo os valores deverão ser trocados utilizando-se uma variável de troca.

Exemplo :

```

Programa ordenacao;
Var
    A : matriz[1..5] de inteiros;
    I, J, troca : inteiro;
Inicio
    Para i de 1 até 5 passo 1 faça
        Leia A[i]
    Fim_para
    Para i de 1 até 5 passo 1 faça
        Para j de i+1 até 5 passo 1 faça
            Se A[j]<A[i] então
                Troca <- A[i]
                A[i] <- A[j]
                A[j] <- troca
            Fim_se
        Fim_para
    Fim_para
    Para i de 1 até 5 passo 1 faça
        Escreva A[i]
    Fim_para
Fim

```

7 - Registros

Até agora aprendemos a trabalhar com uma estrutura de dados de tipos homogêneos (matrizes). Os registros são estruturas de dados que se diferenciam das matrizes, devido o fato de possuírem campos onde definimos tipos de dados diferentes para cada campo.

Sintaxe:

```
Type <identificador> = record
    <lista dos campos e seus tipos>
end;
```

```
Var <variáveis> : <identificador>;
```

No capítulo anterior, um exercício de aprendizagem solicitava que fosse informado o nome de 10 alunos e suas 4 notas bimestrais, o que obrigou a utilização de duas matrizes, uma para conter os nomes, por seus valores serem do tipo caractere e a outra para conter as notas, por seus valores serem do tipo real. Com registros podemos armazenar os mesmos dados em apenas uma estrutura que seria:

```
TYPE
    CAD_ALUNO = RECORD
        NOME : STRING;
        NOTA1 : REAL;
        NOTA2 : REAL;
        NOTA3 : REAL;
        NOTA4 : REAL;
    END;
VAR
    ALUNO : CAD_ALUNO;
```

O acesso a estes dados, tanto para leitura quanto para a escrita seria :

```
READLN(ALUNO.NOME)
READLN(ALUNO.NOTA1)
```

```
WRITELN(ALUNO.NOME)
WRITELN(ALUNO.NOTA1)
```

Estrutura de um registro de conjunto

Consiste em estabelecer uma estrutura de registro onde um campo do registro é do tipo matriz. Exemplo:

```
TYPE
```

```

    CAD_ALUNO = RECORD
        CODIGO : INTEGER;
        NOME : STRING;
        NOTAS : ARRAY[1..4] OF REAL;
    END;
VAR
    ALUNO : CAD_ALUNO;

Leitura : READLN(ALUNO.NOTAS[1]);
Escrita : WRITELN(ALUNO.NOTAS[2]);

```

Estrutura de um conjunto de registros

Consiste em estabelecer uma matriz de tipo registro. Exemplo:
TYPE

```

    CAD_ALUNO = RECORD
        CODIGO : INTEGER;
        NOME : STRING;
        NOTA1 : REAL;
        NOTA2 : REAL;
        NOTA3 : REAL;
        NOTA4 : REAL;
    END;
VAR
    ALUNO : ARRAY[1..10] OF CAD_ALUNO;

Leitura : READLN(ALUNO[1].NOTA1);
Escrita : WRITELN(ALUNO[2].NOME);

```

Exercícios:

1 – Elaborar um programa que faça a leitura de 10 produtos contendo descrição, quantidade, quantidade mínima e valor. Após a leitura dos dados apresentar uma ficha de compra onde deverão ser listados todos os produtos que estão abaixo da quantidade mínima. No final da lista apresentar o total a ser gasto para repor estes produtos. {conjunto de registro}

2 - Efetuar a leitura de 4 notas bimestrais para 8 alunos. Apresentar o boletim escolar dos alunos.

3 – Elaborar um programa que contenha em um menu do seguinte formato :

PROGRAMA AGENDA DE TELEFONES

- 1 – CADASTRAR
- 2 – PESQUISAR
- 3 – ALTERAR
- 4 – CLASSIFICAR
- 5 – LISTAR
- 6 - SAIR

ESCOLHA UMA OPÇÃO : []

Construa rotinas que desempenham cada uma das funções do menu acima. Os dados da agenda deverão ser : nome, telefone e celular

8 – Utilização de Subrotinas

Os procedimentos e funções são utilizados na divisão de um programa complexo, permitindo assim possuir a modularização de um determinado problema, considerado grande e de difícil solução. A modularização do código nos dá várias vantagens na linguagem pascal :

- Para tarefas que devem ser efetuadas mais do que uma vez num programa, a modularidade evita a necessidade de programação redundante (repetida) de essencialmente o mesmo conjunto de instruções. Dessa forma, cada módulo pode ser definido apenas uma vez e depois acedido de vários pontos do programa.
- Um conjunto diferente de dados pode ser processado de cada vez que o módulo é acessado.
- Facilidade na interpretação do código.

O uso de módulos de programa pode, portanto, reduzir apreciavelmente o tamanho de um programa.

Uma sub-rotina é, na verdade, um programa, e sendo um programa poderá efetuar diversas operações computacionais (entrada, processamento e saída). 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.

Sub – Rotinas do tipo procedimento

```
Sintaxe :  
PROCEDIMENTO <nome do procedimento>  
VAR  
    <variáveis>  
INICIO  
    <instruções>  
FIM
```

Exemplo : efetuar um programa que utilize um procedimento para determinar o maior número entre 3 inteiros.

Exercícios :

1 – Elaborar um programa que possua uma sub-rotina que apresente o total do somatório dos N primeiros números inteiros. $(1+2+3+4+...+N)$

2 – Elabore um programa que solicite do usuário 3 números. Se a soma dos 3 números for menor que 10, solicite os 3 números novamente e saia do programa. Como o programa pode solicitar 2 vezes os 3 números, crie um procedimento para solicitar estes 3 números.

Utilização de Parâmetros

Parâmetros têm por finalidade servir como um ponto de comunicação bidirecional entre as sub-rotinas e o programa principal.

```
Exemplo:
PROCEDIMENTO TROCA(A,B : INTEIRO);
VAR X : INTEIRO;
INICIO
    X<-A;
    A<-B;
    B<-X;
FIM;
```

Exercícios :

1 – Desenvolva um programa que solicite do usuário o seu salário e o percentual de acréscimo. Após a leitura chame uma sub-rotina para calcular e apresentar o salário atualizado.

2 – Desenvolva um programa que solicite do usuário a base e a altura de um triângulo retângulo e chame um procedimento para calcular e apresentar a área do triângulo.

3 – Elaborar um programa que possua uma sub-rotina que apresente o total do somatório dos N primeiros números inteiros. $(1+2+3+4+...+N)$

4 – Elaborar um programa que utilize um procedimento para calcular a potência de um número, ou seja, chamando-se o procedimento POTENCIA(2,3) o procedimento deve calcular e apresentar o resultado 8.

5 – Construa uma procedure que receba como parâmetro uma frase e um número inteiro, esta procedure deve escrever a frase na tela repetidamente o número de vezes que foi passado como parâmetro.

Sub-rotinas do tipo função

Uma função é um bloco de programa como os procedimentos, sua principal diferença está no fato de a função retornar um determinado valor. O valor de uma função é retornado em seu nome.

```
Sintaxe :
FUNCAO <nome da função>(<parametro1>,<parametro2> : <tipo>) : <tipo de retorno>;
VAR
    <variáveis>
INICIO
    <instruções>
    <nome da função> := <valor de retorno>;
FIM
```

Exemplo : Desenvolva um programa com uma função que retorne o maior número

entre dois valores.

Exercício :

1- Desenvolva um programa que solicite do usuário 2 valores e um operador (+, -, /, *), chame um procedimento que receba como parâmetro os valores solicitados ao usuário e devolva a soma dos valores caso o operador seja +, a subtração dos valores caso o operador seja - e assim por diante.

2 – Desenvolva um programam com uma subrotina que receba um número N e retorne ao programa principal o somatório dos N primeiros números inteiros. Apresente o valor retornado pela função.

3 – Crie um programa que solicite do usuário a base e a altura de um triângulo retângulo, chame uma função que calcule e retorne a área da figura. Apresente o valor retornado pela função.

4 – Baseado no exemplo dado:

```
PROGRAM SOMA;  
USES CRT;  
VAR A,B,R : INTEGER;  
  
BEGIN  
  CLRSCR;  
  WRITE('Entre com o valor de A : ');  
  READLN(A);  
  WRITE('Entre com o valor de B : ');  
  READLN(B);  
  R:=A+B;  
  Writeln('O resultado da soma de A e B , : ',R);  
END.
```

- A) construa um programa com uma procedure que calcule e apresente o resultado da soma dos dois valores A e B passados como parâmetro;
- B) construa um programa com uma função que calcule a soma dos dois valores A e B passados como parâmetro e retorne o resultado da soma para que este seja apresentado.

5 – Construa uma procedure que replique uma seqüência de um caracter o número de vezes passados como parâmetro.

Ex.: replica('-',20) esta chamada a procedure deve escrever vinte sinais de menos.

6 – Construa uma função que faça o acréscimo em porcentagem do valor passado como parâmetro. Ex.: ValorAtual:=percentual(ValorAntigo,perc_acrescimo);
percentual(200,10)=220;

9 – Arquivos

Um arquivo é de suma importância nos programas computacionais, desde o tempo em que o primeiro computador surgiu, pois, para que um programa faça algum tipo de operação, o mesmo precisa ser alimentado com informações: estas, ou são fornecidas pelo teclado, o que atualmente torna-se inviável, ou são fornecidos através de um arquivo.

Conceitos

ITEM ou CAMPO: unidade de informação; o valor de um item pode ser um número inteiro, uma cadeia de caracteres, um número decimal ou, ainda, uma data.

REGISTRO: conjunto de itens ou uma linha de uma tabela

ARQUIVO: conjunto de registros.

Exemplo de um arquivo de alunos:

R.A. N(6)	NOME C(30)	ENDEREÇO C(50)	MENS. Dec(8,2)
1	Ana Paula	Rua...	250,00
2	Cristina	Av...	260,00
3	Renata	Rua...	255,00
.			
.			
200	Pedro	Av...	270,00

Campos do arquivo: R.A., nome, endereço e mensal.

Tamanho do registro: 96 bytes

Total de registros: 200

Tamanho total do arquivo: 19200

Lembrando que:

1K – 1024 bytes 1M – 1024 K 1 G – 1024 M 1 T – 1024 G

CHAVE PRIMÁRIA (PK): É uma chave que representa um valor diferente para cada registro do arquivo, de tal forma que, dado um valor da chave primária, é identificado um único registro do arquivo.

Exemplo de chave primária para o arquivo de alunos: R.A

Uma chave primária pode ser composta por mais de um campo.

CHAVE SECUNDÁRIA: difere de uma primária pela possibilidade de não possuir um valor diferente para cada registro. Assim, um valor de uma chave secundária identifica um conjunto de registros. É utilizada principalmente para buscas e ordenação.

CHAVE DE ESTRANGEIRA (FK): é a chave usada para identificar o(s) registro(s) desejado(s) em uma operação de acesso a um outro arquivo.

EXERCÍCIOS

Com base no arquivo de funcionários abaixo, resolva as questões:

Filial	Matrícula	Nome	Cargo	Data Adm	Salário
1	1024	Vagner	Porteiro	25/05/91	350,00
1	1050	Maria	Zeladora	16/03/92	200,00
2	1029	Marcelo	Digitador	05/04/95	600,00
2	1080	Valéria	Analista Sistemas	06/09/96	1500,00
.					
.					

- Quais são os campos do arquivo?
- Qual o 3º registro do arquivo?
- Defina o tamanho de cada campo.
- Calcule o tamanho dos registros (desconsidere a “,” do valor e a “/” da data).
- Calcule o espaço em disco que este arquivo irá ocupar, considerando que o mesmo tem um total de 400 registros.
- Defina a chave primária do arquivo.
- Defina uma chave secundária para o arquivo, considerando que deverá ser emitido um relatório ordenado por FILIAL, CARGO e MATRÍCULA.

O PASCAL, possui dois tipos de arquivos, os quais são:

- Arquivos FILE
- Arquivos TEXT

Arquivos FILE

Um arquivo do tipo FILE, também conhecido por arquivo randômico, ou de acesso aleatório, é o arquivo mais importante do Pascal, sendo desta forma também o mais utilizado. Um arquivo randômico é caracterizado pelo fato de ser possível buscar uma determinada informação em qualquer posição que a mesma se encontre, sem haver a necessidade de se percorrer todo o arquivo até se alcançar a informação desejada.

Sintaxe :

<Nome da variável> : FILE OF<tipo>

Observação: Um arquivo FILE deve ser apenas um tipo de dado, ou seja : INTEGER, REAL, RECORD, STRING, BYTE, etc.

Exemplo: Crie um programa que defina uma variável como sendo um arquivo FILE de STRING's, crie também neste mesmo programa um tipo Arquivo de INTEGERS.

```
PROGRAM Exemplo;  
TYPE  
    Meu_tipo = FILE OF INTEGER;  
VAR  
    Minha_Variável = FILE OF STRING;  
BEGIN  
END.
```

Estrutura Interna do Arquivo:

Quando um arquivo FILE é criado, o mesmo possui a seguinte estrutura:

Posição Física	Informação
0	
1	
2	
...	
n	

A posição física corresponde a um número que é gerado automaticamente no instante que uma informação qualquer é incluída no arquivo. Este número, corresponde ao "Endereço" da informação no arquivo, sendo que é através deste Endereço que é possível recuperar qualquer informação, sem precisar percorrer todo o arquivo em busca da mesma, ao invés disto basta fornecer o número da posição física da informação no arquivo.

Observação: Passaremos daqui por diante a chamar as informações armazenadas em um arquivo de "Registros".

Sub-Rotinas para Tratamento de Arquivos FILES

Existem uma grande quantidade de sub-Rotinas construídas especialmente para manipular arquivos FILE. As principais são:

Rotina : ASSIGN()

Função : Serve para associar um determinado Nome de arquivo, no disco ou disquete com o arquivo definido pelo programador.

Sintaxe : ASSIGN(Meu_Arquivo, STRING_Com_Nome_Arquivo_DOS).

Exemplo:

```
PROGRAM TESTE
TYPE
    Registro = RECORD
        Nome : STRING;
        Idade : BYTE;
    END;
VAR
    Arquivo : FILE OF Registro;
BEGIN
    ASSIGN (Arquivo, 'dados.dat');
END.
```

Rotina : REWRITE()

Função : Cria e abre para E\S um arquivo. Caso o arquivo não exista, o mesmo será criado. Caso o arquivo já exista, todos os dados existentes nele serão apagados.

Sintaxe : REWRITE(Meu_Arquivo);

Exemplo:

```
PROGRAM Teste;
TYPE
    Registro = RECORD
        Nome : STRING;
        Idade : BYTE;
    END;
VAR
    Arquivo : FILE OF Registro;
BEGIN
    ASSIGN (Arquivo, 'Dados.Dat');
    REWRITE (Arquivo);
END.
```

Rotina : **RESET()**

Função : Abre para E/S um arquivo que já exista. Caso o arquivo não exista ocorrerá um erro de execução e o programa será abortado.

Sintaxe : RESET(Meu_Arquivo)

Exemplo:

```
PROGRAM Teste;
TYPE
    Registro = RECORD
        Nome : STRING;
        Idade : BYTE;
    END;
VAR
    Arquivo : FILE OF Registro;
BEGIN
    ASSIGN (Arquivo, 'Dados.Dat');
    RESET (Arquivo);
END.
```

Rotina : **CLOSE()**

Função : Fecha um arquivo que tenha sido aberto com RESET\REWRITE.

Sintaxe : CLOSE(Meu_Arquivo)

Exemplo:

```
PROGRAM Teste;
TYPE
    Registro = RECORD
        Nome : STRING;
        Idade : BYTE;
    END;
VAR
    Arquivo : FILE OF Registro;
BEGIN
    ASSIGN (Arquivo, 'Dados.Dat');
    REWRITE (Arquivo);
    CLOSE (Arquivo);
END.
```

Rotina : **WRITE()**

Função : A Rotina WRITE tem a mesma Função de saída de informações como até agora já tínhamos trabalhado, somente que ao invés da informação ser apresentada no vídeo, a mesma será armazenada em um arquivo.

Sintaxe : WRITE (Meu_Arquivo, Registro)

Exemplo:

```
PROGRAM Teste;
TYPE
    Registro = RECORD
        Nome : STRING;
        Idade : BYTE;
    END;
VAR
    Arquivo : FILE OF Registro;
    Reg : Registro;
BEGIN
    ASSIGN (Arquivo, 'Dados.Dat');
    REWRITE (Arquivo);
    WRITE ('Digite o Nome: ');
    READ (Reg.Nome);
    WRITE ('Digite a Idade: ');
    READ (Reg.Idade);
    WRITE (Arquivo, Reg);
    CLOSE (Arquivo);
END.
```

Rotina : **READ()**

Função : A Rotina READ tem a mesma Função de entrada de informações como até agora já tínhamos trabalhado, somente que ao invés da leitura ser feita pelo teclado, a mesma será feita de um arquivo.

Sintaxe : READ (Meu_Arquivo, Registro)

Exemplo:

```
PROGRAM Teste;
TYPE
    Registro = RECORD
        Nome : STRING;
        Idade : BYTE;
    END;
VAR
    Arquivo : FILE OF Registro;
    Reg : Registro;
BEGIN
    ASSIGN (Arquivo, 'Dados.Dat');
    RESET (Arquivo);
    READ (Arquivo, Reg);
    WRITE ('Nome = ', Reg.Nome);
    WRITE ('Idade = ', Reg.Idade);
    CLOSE (Arquivo);
END.
```

Observação: Após cada operação READ/WRITE no arquivo, o endereço do registro corrente no arquivo é incrementado em uma unidade. Assim por Exemplo, se o endereço do registro corrente é igual a 10, após uma operação de READ/WRITE, o registro corrente passará a ser o número 11.

Rotina : **FILEPOS()**

Função : Retorna um número inteiro indicando qual o registro corrente em um arquivo.

Sintaxe : Registro_Corrente := FILEPOS (Meu_Arquivo)

Exemplo:

```
PROGRAM Teste;
TYPE
    Registro = RECORD
        Nome : STRING;
        Idade : BYTE;
    END;
VAR
    Arquivo : FILE OF Registro;
    Corrente: INTEGER;
BEGIN
    ASSIGN (Arquivo, 'Dados.Dat');
    RESET (Arquivo);
    corrente := FILEPOS(Arquivo);
    WRITE (corrente);
    CLOSE (Arquivo);
END.
```

Rotina : **FILESIZE()**

Função : Retorna quantos registro existem armazenados no arquivo.

Sintaxe : Tamanho_Arquivo := FILESIZE (Meu_Arquivo)

Exemplo:

```
PROGRAM Teste;
TYPE
    Registro = RECORD
        Nome : STRING;
        Idade : BYTE;
    END;
VAR
    Arquivo : FILE OF Registro;
    Total : INTEGER;
BEGIN
    ASSIGN (Arquivo, 'Dados.Dat');
    RESET (Arquivo);
    Total := FILESIZE (Arquivo);
    WRITE (Total);
    CLOSE (Arquivo);
END.
```

Rotina : **SEEK ()**

Função : Posiciona o ponteiro do arquivo em um registro determinado, para que o mesmo possa ser processado.

Sintaxe : SEEK(Meu_Arquivo, Endereço_Registro)

Exemplo:

```
PROGRAM Teste;
TYPE
    Registro = RECORD
        Nome : STRING;
        Idade : BYTE;
    END;
VAR
    Arquivo : FILE OF Registro;
    Reg : Registro;
BEGIN
    ASSIGN (Arquivo, 'Dados.Dat');
    RESET (Arquivo);
    SEEK (Arquivo, 10);
    READ (Arquivo, Reg);
    WRITE ('Nome = ', Reg.Nome);
    WRITE ('Idade = ', Reg.Idade);
    CLOSE (Arquivo);
END.
```

Rotina : **EOF()**

Função : Retorna TRUE caso se alcance o final do arquivo, FALSE caso contrário.

Sintaxe : Chegou_Final := EOF (Meu_Arquivo)

Exemplo:

```
PROGRAM Teste;
TYPE
    Registro = RECORD
        Nome : STRING;
        Idade : BYTE;
    END;
VAR
    Arquivo : FILE OF Registro;
    Reg : Registro;
BEGIN
    ASSIGN (Arquivo, 'Dados.Dat');
    RESET (Arquivo);
    WHILE NOT EOF(Arquivo) DO
        BEGIN
            READ (Arquivo, Reg);
            WRITE ('Nome = ', Reg.Nome);
            WRITE ('Idade = ', Reg.Idade);
        END;
    CLOSE (Arquivo);
END.
```


Rotina : **ERASE ()**

Função : Elimina o arquivo do disco. É importante notar que o arquivo a ser eliminado não pode estar aberto.

Sintaxe : ERASE (Meu_Arquivo)

Exemplo:

```
PROGRAM Teste;
TYPE
    Registro = RECORD
        Nome      : STRING;
        Idade     : BYTE;
    END;
VAR
    Arquivo : FILE OF Registro;
BEGIN
    ASSIGN (Arquivo, 'Dados.Dat');
    ERASE(Arquivo);
END.
```

Exercícios:

- Faça um programa de inclusão de dados em um arquivo chamado pessoas.dat. Os registros deste arquivo deverão ser formados de código, nome e telefone de uma pessoa.
- Faça outro programa que abra o arquivo criado no exercício anterior e liste todos os dados no vídeo.
- Elabore um programa para incluir produtos em um arquivo qualquer. A inclusão deverá ser efetuada enquanto o usuário desejar. O produto é composto de código, descrição, quantidade e valor. Quando o usuário não desejar incluir mais produtos, chame uma procedure para listar os dados inclusos e o total em Reais do estoque.
- Incluir um módulo de consulta no arquivo de produtos do exercício anterior.
- Faça um programa com as seguintes rotinas:

1 – Inclusão de funcionários: Deverão ser incluídos clientes no arquivo func.dat enquanto o usuário desejar. Os dados dos funcionários são : código, nome, salário, nºfilhos.

2 - Processamento Folha de pagamento: Para cada funcionário deverá ser criada uma folha de pagamento contendo: código, codfun, salário. O salário da folha deverá ser igual ao salário de seu cadastro mais 1% para cada filho que tiver.

3 – Listagem de folha de pagamento: Deverá ser uma listagem com o seguinte lay-out :

Cod. Folha	Cod. Func.	Nome Func.	NºFilhos	Salário
...				
...				
Total da Folha:				

f) Crie um arquivo de peças, com o seguinte Lay-Out: Nome de Peça, cor , quantidade, tamanho e deletado. O campo “Deletado” será um campo Boolean, setado inicialmente para FALSE , informando se o registro está ou não deletado do arquivo.

g) Faça uma Rotina para deletar um, ou mais, registros do arquivo de peças. A deleção consiste em setar o campo “deletado” do arquivo para TRUE.

- h) Percorrer o arquivo de peças imprimindo somente as peças que não foram deletadas
- i) Faça uma Rotina que elimine fisicamente os registros do arquivo de peças que foram marcadas para deleção, isto é , onde o campo “deletado” está setado para TRUE.