ALGORITMO E PROGRAMAÇÃO

FELIPE DENIS M. DE OLIVEIRA

Natal / RN / BRASIL
E-Mail : fdenis_natal@yahoo.com.br

Índice

1. Algoritmos

O que é:

Seqüência ordenada, finita de operações bem definidas e eficazes que, quando executadas sobre dados convenientes, produz a solução de um dado problema.

Programa: Algoritmo escrito numa linguagem que o computador possa executar direta ou indiretamente.

Linguagem de Programação: Conjunto de instruções e regras utilizadas para se escrever programas, a partir dos algoritmos. As linguagens de Programação são classificadas da seguinte forma:

- Linguagem de Máquina: instruções representadas por zeros e uns (0, 1 => BIT)
- Linguagem Assembly: instruções representadas por mnemônicos
- Linguagem de Alto Nível: instruções representadas por palavras de uma linguagem comum.

Algoritmo não é a solução de um problema, pois, se assim fosse, cada problema teria um único algoritmo. Algoritmo é um caminho para a solução de um problema, e em geral, os caminhos que levam a uma solução são muitas.

O aprendizado de algoritmos não se consegue a não ser através de muitos exercícios.

Algoritmos não se aprende:

- Copiando Algoritmos
- Estudando Algoritmos

Algoritmos só se aprendem:

- Construindo Algoritmos
- Testando Algoritmos

2. Fases de um Algoritmo

Quando temos um problema e vamos utilizar um computador para resolvê-lo inevitavelmente temos que passar pelas seguintes etapas:

a) Definir o problema.

b) Realizar um estudo da situação atual e verificar quais a(s) forma(s) de resolver o problema.

c) Terminada a fase de estudo, utilizar uma linguagem de programação para escrever o programa que deverá a princípio, resolver o problema.

d) Analisar junto aos usuários se o problema foi resolvido. Se a solução não foi encontrada, deverá ser retornado para a fase de estudo para descobrir onde está a falha.

Estas são de forma bem geral, as etapas que um programador passa, desde a apresentação do problema até a sua efetiva solução. Iremos, neste curso, nos ater as etapas de estudo, também chamada de análise, e a etapa de programação. Mas antes vamos definir o seguinte conceito: Programar um computador consiste em elaborar um conjunto finito de instruções, reconhecidas pela máquina, de forma que o computador execute estas instruções.

Estas instruções possuem regras e uma Sintaxe própria, como uma linguagem tipo português ou inglês, sendo isto chamadas de linguagem de computador.

No mundo computacional existe uma grande variedade de linguagens: Pascal, C, C++, Cobol, Fortran, etc.

3. Formas de Representação de Algoritmos:

- Descrição Narrativa – linguagem natural
- Fluxograma – representação gráfica onde formas geométricas diferentes implicam em ações distintas.
- Pseudocódigo – usa linguagem natural normalmente dirigida para uma dada linguagem de programação. Ex: Fortran, Cobol, Pascal, C... Nós utilizaremos o estilo de pseudocódigo baseado em Pascal e C ANSI.

4. COMENTÁRIOS

Servem para realizar uma descrição de algum item do algoritmo ao qual não venha a ser utilizado como linha ativa no código, ou seja, serve como auxílio ao programador para enfatizar algum trecho do código do algoritmo que seja relevante para o mesmo. Pode ser expresso de duas formas:

- Comentário de uma linha: Inicia a linha com //, seguido do comentário. Pode também ser utilizado no final de uma instrução ou estrutura, mas nunca no meio ou no início dela, sob risco de provocar falhas ou anular a linha de código;
- Comentário de múltiplas linhas: Inicia-se com /*, seguido do comentário, em uma ou mais linhas, finalizando com */.

OBS: Os caracteres utilizados para criação dos comentários podem variar dependendo da linguagem de programação. No item 5, um exemplo da inserção de comentários em um algoritmo.

5. Estrutura de Algoritmos

Antes de utilizarmos uma linguagem de computador, é necessário organizar as ações a serem tomadas pela máquina de forma organizada e lógica, sem nos atermos as regras rígidas da Sintaxe de uma linguagem. Para isto utilizaremos uma forma de escrever tais ações, conhecida como algoritmo, ou pseudo-código. Para facilitar o entendimento da disciplina, vamos utilizar a linguagem de programação Pascal para traduzir o algoritmo.

Os algoritmos terão a seguinte estrutura:

```
ALGORITMO....: Nome_do_arquivo.alg
/*AUTOR.....:   nome do autor
DATA.....: data automática do sistema
DESCRIÇÃO....:  dizer, aqui, o que o algoritmo irá fazer  */

Algoritmo Novo

  //Declaração das variáveis

início

  //Inicialização das variáveis


  //Código do Algoritmo


fim
```

6. Tipos de Dados

Os algoritmos destinados a resolver um problema no computador manipulam, basicamente, dois tipos de informação:

- Instruções (comandos): leitura, gravação, operações, etc.
- Dados (Constantes, variáveis): valores a serem processados.

6.1 – *Constantes* **–** valor fixo que não se modifica ao longo do tempo, durante a execução de um programa.

6.1.1 **Numérica** (Inteira, Real,...) é um número representado no sistema decimal podendo conter ou não uma parte decimal (após o ponto).

Ex. 96; 0; -99 -> inteiros 0.0; 85.6; -9.55 -> reais

Lógica – usada para representar dois únicos valores possíveis: Verdadeiro ou Falso

Ex: .V.; 1; Sim -> Verdadeiro .F.; 0; Não -> Falso

Caractere – representa, apenas, um caractere.

Ex: ‘J’; ‘1’; ‘3’; ‘ ‘

Literal ou String – seqüência de caracteres (letras, números ou sinais que o computador consegue processar), também chamada de String de Caracteres ou String.

Ex: “UFRN”; “133”; “2001”.

7. Variáveis

O computador possui uma área de armazenamento conhecida como memória. Todas as informações existentes no computador estão ou na memória primária (memória RAM), ou na memória secundária (discos, fitas, CD-ROM etc). Nós iremos trabalhar, neste curso, somente com a memória primária, especificamente com as informações armazenadas na RAM (memória de acesso aleatório).

Podemos dizer que uma variável é uma posição de memória, representada por um Nome simbólico (atribuído pelo usuário), a qual contém, num dado instante, uma informação.

7.1 Formação de Variáveis ou Indentificadores

Uma variável é formada por uma letra ou então por uma letra seguida de letras ou dígitos. Não é permitido o uso de espaços em branco ou de qualquer outro caractere, que não seja letra ou dígito, na formação de um identificador.

Na formação do nome da variável de um nome significativo, para que se possa ter idéia do seu conteúdo sem abri-la. Se utilizar palavras para compor o nome da variável utilize o “_ “ underline para separar as palavras. Resumindo, uma variável possui **Nome, Tipo e Conteúdo**.

7.2 Tipos de Variáveis

- Inteiro - Real - Caractere - String - Lógico

8. Declaração de Variáveis

Para declarar uma ou mais variáveis, basta colocarmos o nome da variável, seguido do seu tipo, separando as duas partes pelo sinal de dois pontos (:). Abaixo segue exemplo de algumas definições de variáveis:

a)
```
ALGORITMO Teste
VARIÁVEIS
    Palavra : String;
INICIO
    <comandos>
FIM.
```

b)
```
ALGORITMO Teste;
VARIÁVEIS
    Letra, Caracter: Caractere;
INICIO
    <comandos>
FIM.
```

c)
```
ALGORITMO Teste;
VARIÁVEIS
    Letra, Caracter: Caractere;
    Número:  Inteiro;
INICIO
    <comandos>
FIM.
```

Obs.: Os nomes dados as variáveis não podem ser os mesmos nomes de palavras reservadas das linguagens de programação ou da própria estrutura do algoritmo, tais do Pascal, tais como Início, Fim, PROGRAM, BEGIN, END, VER, etc…

8.1 Atribuição de valores às variáveis

Quando definimos uma variável é natural atribuirmos a ela uma informação. Uma das formas de colocar um valor dentro de uma variável, conseqüentemente colocado este dado na memória do computador, é através da atribuição direta, do valor desejado que a variável armazena. Para isto utilizaremos o símbolo = , que significa: **recebe**, ou seja, a posição, de memória que uma variável representa, receberá uma informação, a qual será armazenada no interior desta variável.

Exemplo:

```
ALGORITMO Teste
VARIÁVEIS
    Numero:  INTEIRO;
INICIO
    Número = 10;
FIM
```

O Exemplo acima nos informa que:

a) Foi definido uma variável, a qual demos o Nome de “Numero”, e informamos que esta variável, ou posição de memória, só poderá aceitar dados, que sejam numéricos tipo inteiro .

b) Atribuímos à variável “Numero” o valor 10

8.1.1 Exercícios 0:

1) Dar o tipo de cada uma das constantes

a) 613
b) 613.0
c) -613
d) “613”
e) $–3.012 * 10^{15}$
f) $17*10^{12}$
g) $–28.3 * 10^{-23}$
h) “Fim de Questão”
i) ‘1’

2) Faça um PROGRAMA para atribuir a variáveis as seguintes informações:

a) 12345
b) 123456
c) -1122
d) 10
e) VERDADE
f) 12345605

3) No seguinte Algoritmo existe algum erro? Onde?

```
ALGORITMO Teste
VARIÁVEIS
        Maria   : String;
        idade   : INTEIRO;
        letra   : CARACTER;
        Maria   : REAL;
INICIO
        idade   = 23;
        idade   = 678;
        idade   = letra;
        letra   = ABC;
        letra   = A;
        letra   = 2;
FIM.
```

4) Qual a diferença existente nas seguintes atribuições?

a)
```
Letra   = ‘A’
Nome    = “João”
```

b)
```
Letra   = A
Nome    = João
```

5) É correto definir uma variável como sendo Caracter e atribuirmos a ela o valor: “PEDRO”? E se a variável fosse definida como STRING, a mesma poderia receber um valor do tipo CARACTER?

6) Identifique os nomes de variáveis válidos com V e os inválidos com I.

( ) abc	( ) 3abc	( ) 123a	( ) a123
( ) soma_1	( ) ab-cd	( ) a_	( ) sim
( ) Início	( ) Windows	( ) %@!$_	

9. Operadores

9.1 Operadores matemáticos

Os operadores matemáticos são os seguintes:

a) Adição : +
b) Subtração : -
c) Multiplicação : *
d) Divisão : /
e) Divisão inteira : DIV (QUOCIENTE)
f) Resto da divisão : MOD (RESTO) **OBS: Varia de acordo com a linguagem!!!!**
g) Exponenciação : ** ou ^ **OBS: Varia de acordo com a linguagem!!!!**
h) Igualdade : == **OBS: Varia de acordo com a linguagem!!!!**
i) Diferença : != **OBS: Varia de acordo com a linguagem!!!!**
j) Menor Igual : <=
k) Maior Igual : >=

OBS: PRECENDÊNCIA DOS OPERADORES

OPERADOR	PRECEDÊNCIA
not	maior
* , / , div , mod , and	↓
+ , - , or	
< , <= , > , >= , == , <>	menor

OS PARÊNTESIS ( ) SEMPRE TERÃO PRECÊNCIA MÁXIMA EM RELAÇÃO A QUALQUER OUTRO OPERADOR

9.2 Operadores Lógicos

Os operadores lógicos, realizam as operações da álgebra booleana. Os operadores são os seguintes:

a) AND (.E.)
b) OR (.OU.)
c) NOT (.NÃO.)

Exemplo:

a) Operador .E. :

V e V => Verdadeiro
F e F => Falso
F e V => Falso
F e F => Falso

b) Operador .OU. :

V ou V => Verdadeiro
V ou F => Verdadeiro
F ou V => Verdadeiro
F ou F => Falso

c) Operador NÃO:

Não V => Falso
Não F => Verdadeiro

9.2.1 Exercícios 1 :

1) indique qual o resultado será obtido das seguintes expressões:
   a) 1 / 2
   b) 1 DIV 2
   c) 1 MOD 2
   d) (200 DIV 10) MOD 4
   e) 6 +19-23
   f) 3.0* 5.0 +1
   g) 1/4+2
   h) 29.0/7+4
   i) 3/6.0-7

2) Indique o resultado das seguintes expressões:
   a) 2 > 3
   b) (6 < 8) OR (3 > 7)
   c) NOT (2 < 3)

3) Construa o algoritmo que calcule as seguintes expressões:
   a) 2 + 3 * { 23 - 12 + [{ (123 / 34) + 10 } / 2 - 1 * (45 - 12)] / 3 }
   b) (2 + [2 * 3 - (4 / 20)])/ (10 * { [(7 * 3) - 2] * 3 })

Obs.: O operador "+" caso seja usado entre variáveis do tipo Caractere ou STRING, causará uma ação conhecida por **concatenação**, ou seja, juntar os caracteres ou STRING's usados na operação em uma só STRING.

Exemplo:

```
ALGORITMO Concatena
VARIÁVEIS
        Letra1, Letra2          : CARACTER;
        Nome1, Nome2, Nome :  STRING;
INICIO
        Letra1 = 'D';
        Letra2  =  'a';
        Nome1 = "Joao";
        Nome2 = "Silva";
        Nome =  Nome1 + Letra1 + Letra2 + Nome2; //OBS: varia, de acordo com a linguagem.
FIM
```

As instruções acima resultarão no armazenamento do Nome 'João Da Silva" na variável rotulada de "Nome".

10. Comandos de Entrada e Saída

O computador não é uma máquina isolada, pois ele precisa se comunicar com o mundo exterior com vídeo, impressora, teclado, discos, fitas etc. Para realizar esta comunicação existem comandos que permitem que informações sejam exibidas, por Exemplo, no vídeo, como também existem comandos que permitem que informações sejam colocadas na memória do computador através do teclado do micro.

Os comandos que iremos ver são os comandos LEIA ou LEIAL e ESCREVA ou ESCREVAL, respectivamente, comando de entrada e de Saída.

Exemplo 1: Escrever um algoritmo para ler um valor numérico do teclado e atribuí-lo a uma variável do tipo numérica.

```
ALGORITMO LeNúmero
VARIÁVEIS
        Num     : INTEIRO;
INICIO
        LEIA(Num);
FIM
```

O algoritmo acima, executará os seguintes comandos:

a) Define uma variável rotulada "Num", a qual só poderá armazenar dados numéricos inteiros, sendo que inicialmente o conteúdo desta variável está indefinido.
b) interrompe o processamento até que uma informação seja digitada, seguida do pressionamento da tecla ENTER. Caso a informação digitada seja compatível com o tipo (INTEIRO), este valor será armazenado dentro da variável "Num".

Exemplo 2: Fazer um algoritmo para escrever no vídeo do PC uma mensagem qualquer.

```
ALGORITMO EscreveMsg
INICIO
        ESCREVA ("Alo Mundo");
FIM
```

Obs.: A mensagem está entre aspas porque representa uma String de caracteres.

É perfeitamente possível mandar escrever o conteúdo de variáveis no vídeo. Desta forma, o Exemplo acima poderia ser escrito do seguinte modo:

```
ALGORITMO EscreveMsg
VARIÁVEIS
        Msg:    : STRING;
INICIO
        Msg     = "Alo Mundo";
        ESCREVA  (Msg);
FIM
```

10.1.1 Exercícios 2:

1) Faça um algoritmo para ler as seguintes informações de uma pessoa: Nome, Idade, Sexo, Peso, Altura, Profissão, Rua, Bairro, Cidade, Estado, CEP, Telefone.

2) Faça um algoritmo para ler a base e a altura de um triângulo. Em seguida, escreva a área do mesmo.
Obs.: Área = (Base * Altura) / 2

3) Faça um algoritmo que calcule a média aritmética de 4 valores inteiros.

4) O preço de um automóvel é calculado pela soma do preço de fábrica com o preço dos impostos (45% do preço de fábrica) e a percentagem do revendedor (28% do preço de fábrica). Faça um algoritmo que leia o nome do automóvel e o preço de fábrica e imprima o nome do automóvel e o preço final.

11. Processos de Repetição e Seleção (Laços)

11.1 Estrutura de Repetição ENQUANTO-FAÇA

a) ALGORITMO

```
ALGORITMO ExEnquanto
Início
        ENQUANTO <Condição for verdadeira> FAÇA
        Início
                <Comandos>;
        FIM
Fim.
```

Exemplo : Faça um algoritmo para ler e escrever o Nome de 20 pessoas.

```
ALGORITMO LeEscreve
VARIÁVEIS
        Nome    : STRING;
        Total   : INTEIRO;
INICIO
        Total  = 0;
        ENQUANTO Total<20 FACA
          Início
                LEIA(Nome);
                ESCREVA ("Nome=", Nome);
                Total    =  Total  +  1;
        FIM
Fim.
```

11.1.1 Exercícios 3:

1) Faça um algoritmo para calcular um valor A elevado a um expoente B. Os valores A e B deverão ser lidos. Não usar A** B.

2) Faça um algoritmo para:
   a) Ler um valor x qualquer
   b) Calcular Y = (x+1)+(x+2)+(x+3)+(x+4)+(x+5)+…(x+100).

3) Faça um algoritmo para ler e escrever o Nome, idade e sexo de um número de alunos informados, previamente, via teclado. Ao final escreva o total de alunos lidos.

4) Faça um algoritmo que calcule a hipotenusa de 10 triângulos.
   $hipotenusa^2 = cateto^2 + cateto^2$

11.2 Estrutura de repetição REPITA-ATÉ (Repeat Until)

```
ALGORITMO
        .
        .
        .
        REPITA

                <Comandos>;
        ATE <Condição for verdadeira>;
```

Exemplo: Faça um ALGORITMO para ler e escrever o Nome de 20 pessoas.

```
ALGORITMO LeEscreve
VARIÁVEIS
        Nome   : STRING;
        Total    : INTEIRO;
INICIO
        Total    =  0;
        REPITA
                LEIA(Nome);
                ESCREVA("Nome=",Nome);
                Total    =  Total + 1;
        ATÉ Total >=20;
FIM.
```

11.2.1 Exercícios 4:

1) Leia 20 valores reais e escreva o seu somatório.

11.3 Estrutura de repetição PARA-FAÇA

ALGORITMO

```
PARA <Variável> = <INICIO> Até <Variável> = <FIM> PASSO <Valor> FAÇA
 Início
        <Comandos>;
Fim
```

Exemplo: Faça um algoritmo para ler e escrever o Nome de 20 pessoas.

```
ALGORITMO LeEscreve
VARIÁVEIS
        Nome   : STRING;
        Cont    : INTEIRO;
INICIO
        PARA Cont = 1 ATE Cont = 20 PASSO 1 FACA
        Início
                LEIA(Nome);
                ESCREVA(Nome);
        FIM
FIM.
```

Obs.:

a) A variável de controle, no Exemplo acima é a variável "Cont", é automaticamente incrementada em um unidade, a partir do valor inicial (1 pelo Exemplo acima), até que seja ultrapassado o limite final definido (20 no Exemplo acima). Se quiséssemos que o passo aumentasse ou diminuísse, bastava acrescentar o valor desejado. Ex: PASSO -2, para decrementar o contador de 2.

B) Quando existe a necessidade de interromper o processamento antes de ser alcançado o limite final definido para a estrutura, não deverá ser usado a estrutura PARA-FAÇA, pela simples razão que a variável de controle não deve ser alterada propositadamente pelo usuário. A alteração do conteúdo das variáveis de controle do PARA-FAÇA são de inteira responsabilidade da própria estrutura.

11.3.1 Exercícios 5:

1) Uma empresa possui 100 vendedores que ganham por comissão sobre cada produto vendido. Cada vendedor em um determinado mês vendeu X produtos, onde cada produto pode ou não ter preços iguais. A empresa deseja obter um relatório com o Nome, o total de vendas e o valor a ser pago a cada vendedor. A comissão paga pela empresa é de 30% sobre o valor de cada produto vendido.

2) Dado uma relação de 1000 números em graus Célcius, faça um PROGRAMA que imprima o seguinte relatório:

Graus Fahrenheit	Graus Célcius
Xxxxxx	xxxxxx

Obs.:

$$C = \frac{5}{9}(F - 32)$$

12. Estrutura de seleção (condicionais)

12.1 Estrutura SE…ENTÃO…SENÃO

ALGORITMO

SE <Condição FOR verdade> ENTÃO
<Comandos>
[SENÃO
<Comandos>] Colchete indica que o comando é opcional
FIMSE

Exemplo: Dado dois valores A e B quaisquer, faça um algoritmo que imprima se A > B, ou A < B , ou A = B

```
ALGORITMO Maior
VARIÁVEIS
        A,B     :  INTEIRO
INICIO
        ESCREVA("Digite os valores A e B");
        SE A > B ENTÃO
                ESCREVA("A é maior que B");
        SENÃO
                SE A < B ENTÃO
                        ESCREVA("A é menor que B");
                SENÃO
                        ESCREVA('A é igual a B');
FIM
```

12.1.1 Exercícios 6:

1) Faça um algoritmo que leia os valores A, B, C e diga se a soma de A + B é menor que C.

2) Faça um algoritmo que leia dois valores inteiros A e B se os valores forem iguais deverá se somar os dois, caso contrário multiplique A por B ao final do calculo atribuir o valor para uma variável C.

3) Faça um algoritmo que leia o nome e as três notas de uma disciplina de uma aluno e ao final escreva o nome do aluno, sua média e se ele foi aprovado a média é 8 (a média é aritmética).

4) Faça um algoritmo que leia 3 números inteiros e imprima o menor deles.

5) Dado uma relação de 100 carros imprima quantos são da cor azul. Sendo para cada carro tem-se uma ficha contento o nome e a cor.

6) Dados três valores distintos, fazer um algoritmo que, após a leitura destes dados coloque-os em ordem crescente.

7) Suponha que para cada aluno de sua sala exista uma ficha contendo o nome e a idade do aluno. Supondo que exista 50 alunos, faça uma algoritmo que determine quantos alunos tem idade maior que 30.

8) Dado Nome e notas (total de 6) de n alunos, faça um algoritmo que:
   a) Imprima Nome e média dos alunos aprovados. Média > = 7.0.
   b) Imprima Nome e média dos alunos em recuperação. 5.0 >= Média <7.0.
   c) Imprima Nome e média dos alunos reprovados. Média <5.0.

12.2 Estrutura CASO

Possibilita a escolha de um conjunto de comandos que serão executados, dentre várias alternativas de escolha.

Sintaxe

```
CASO <variável  de escolha> DE
  < lista de constantes > : < comandos > ;
  < lista de constantes > : < comandos > ;
  ...
  < lista de constantes > : < comandos > ;
  senão < comandos > ;
fim
```

Onde:

- < variável de escolha > é uma expressão do tipo **inteira** ou **caractere** ;

- < lista de constantes > é uma sequência de constantes do tipo inteira ou caractere, separadas por vírgula (ao invés de uma constante é possível usar um intervalo de constantes, que consiste em duas constantes separadas por um par de pontos)

A cláusula **senão** não é obrigatória, e os comandos associados a essa cláusula serão executados **somente se nenhuma outra opção** do caso foi selecionada.

Exemplo

```
Algoritmo Menu
  Variáveis
    opcao : inteiro ;

  Início
    escreva (“Entre com uma opcao: “ );
    leia ( opcao );

    // escolha da opção
   caso opcao de
    1 : escreva( “Você escolheu a opção 1...” );

    2 : escreva( “Você escolheu a opção 2...” );
    3 : escreva( “Você escolheu a opção 3...” );
    senão
        escreva( “Você escolheu uma opção diferente de 1, 2, 3...”);
   fim
 fim.
```

A estrutura CASO corresponde ao comando SE-ENTÃO mas de uma forma mais compacta nas operações de seleção. Por Exemplo, seja a seguinte estrutura escrita com o comando SE-ENTÃO:

Obs.: Por não aceitar valores do tipo REAL e STRING, a estrutura CASO não substitui totalmente a estrutura SE...SENÃO...ENTÃO.

12.2.1 Exercícios 7:

1) Faça um algoritmo que leia um número que represente um determinado mês do ano. Após a leitura escreva por extenso qual o mês lido. Caso o número digitado não esteja na faixa de 1..12 escreva uma mensagem informando o usuário do erro da digitação.

13. Variáveis Compostas Homogêneas

Vimos, no início deste curso, ser possível dar um Nome para uma posição de memória, sendo que a esta será associado um valor qualquer. Pois bem, acontece que, muitas vezes, esta forma de definição, ou melhor dizendo, de alocação de memória, não é suficiente para resolver certos problemas computacionais. Imagine por Exemplo, como faríamos para construir um algoritmo, para ler o Nome de N Pessoas e que imprimisse um relatório destes mesmos nomes, mas ordenados alfabeticamente? Não seria uma tarefa simples, haja visto não ser possível determinar quantos nomes seriam lidos, mesmo que soubéssemos o número de pessoas, digamos 1.000 pessoas, teríamos que definir 1.000 variáveis do tipo STRING, como é mostrado abaixo:

```
ALGORITMO Loucura
VARIÁVEIS
        Nome1,
        Nome2,
        Nome3,
        .
        .
        .
        Nome999,
        Nome1000: STRING;
INICIO
        <Comandos>
FIM
```

Considere o tamanho do algoritmo, e o trabalho braçal necessário para construi-lo. Isto só com 1.000 Nome, imagine agora 1.000.000 de pessoas. A construção deste algoritmo começaria a ficar inviável na pratica. Para resolver problemas como este, e outros, foi criado um novo conceito para alocação de memória sendo, desta forma, também criado uma nova maneira de definir variáveis, a qual foi denominada de variável indexada.

Uma variável indexada corresponde a uma seqüência de posições de memória, a qual daremos único Nome, sendo que cada uma destas pode ser acessada através do que conhecemos por índice. O índice corresponde a um valor numérico (exceto REAL), ou a um valor caracter (exceto STRING). Cada uma das posições de memória de uma variável indexada pode receber valores no decorrer do algoritmo como se fosse uma variável comum, a única diferença reside na Sintaxe de utilização desta variável.

13.1 Variáveis Indexadas Unidimensionais(Vetores)

Também conhecida por “Vetor”. Uma variável unidimensional, como o próprio Nome já indica, possui apenas uma dimensão, sendo ser possível definir variáveis com quaisquer tipo de dados validos do Pascal.

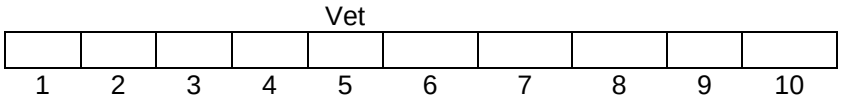
a) Definição:

```
ALGORITMO Define;
VARIÁVEIS
        Nome[Tamanho] : <tipo>;
INICIO
        <Comandos>;
FIM.
```

Exemplo Definir uma variável indexada como sendo do tipo REAL, sendo que a mesma deverá corresponder a 10 posições de memória.

```
ALGORITMO Exemplo
VARIÁVEIS
        Vet10]: REAL;
INICIO
        <Comandos>;
FIM.
```

No **Exemplo** acima, após a definição da variável, a memória estará como mostrado no esquema abaixo:



Os Valores numéricos apresentados acima correspondem aos índices da variável.

B) Atribuição

```
<Nome>[<Índice>] = Valor;
LEIA (<Nome>[<Índice>]);
```

Exemplo:

```
ALGORITMO Atribui
VARIÁVEIS
        Nomes[20]: STRING;
        i        : INTEIRO;
Início
        Nomes[1] =  "João da Silva";
        PARA  i = 2 ATÉ i = 20 PASSO 1  FACA
                LEIA(Nomes[i]);
FIM.
```

13.1.1 Exercícios 8:

1) Faça um algoritmo que leia, via teclado, 200 valores do tipo inteiro e os guarde na memória.

2) Amplie o exercício anterior emitindo um relatório com todos os números pares que você leu.

3) Faça um algoritmo que determine qual o menor valor existente no vetor do exercício número 1.

4) Faça um algoritmo que determine qual o maior valor existente no vetor do exercício número 1.

5) Faça um algoritmo que leia 10 nomes e os guarde na memória.

6) Faça um algoritmo que leia um nome e o imprima ao contrário

7) Faça um algoritmo que leia, Nome idade e sexo de N pessoas. Após a leitura faça:

a) Imprima o Nome, idade e sexo das pessoas cuja idade seja maior que a idade da primeira pessoa.
b) Imprima o Nome e idade de todas as mulheres.
c) Imprima o Nome dos homens menores de 21 anos.

8) Faça uma algoritmo para ler 20 valores e que imprima os que são maiores que a média dos valores.

9) Faça um algoritmo para ler 50 valores inteiros. Após imprima tais valores ordenados crescentemente.

10) Continuando o exercício anterior, emita um relatório com os valores ordenados decrescentemente.

13.2 Variáveis Indexadas Bidimensionais(Matrizes)

Também conhecida por "Matriz". Uma variável Bidimensional, como o próprio Nome já indica, possui duas dimensões, sendo ser possível definir variáveis com quaisquer tipo de dados validos do Pascal.

A =

a11	a12
a21	a22
a31	a33

Matriz A 3x2 (Linha x Coluna)

a) Definição:

```
ALGORITMO Define;
VARIÁVEIS
        Nome[I][J]:  <tipo>;
INICIO
        <Comandos>;
FIM.
```

Exemplo Definir uma variável indexada bidimensional para armazenar os dados de uma matriz 4 por 4 de números do tipo REAL, sendo que a mesma deverá corresponder no total a 16 posições de memória.

```
ALGORITMO Exemplo
VARIÁVEIS
        MAT[4][4]:  REAL;
INICIO
        <Comandos>;
FIM.
```

No **Exemplo** acima, após a definição da variável, a memória estará como mostrado no esquema abaixo:

MAT

1,1	1,2	1,3	1,4
2,1	2,2	2,3	2,4
3,1	3,2	3,3	3,4
4,1	4,2	4,3	4,4

Os Valores numéricos apresentados acima correspondem aos índices da variável.

B) Atribuição

<Nome>[<Índice>][<Índice>] = Valor;
LEIA (<Nome>[<Índice>][<Índice>]);

Exemplo:

```
ALGORITMO Atribui;
VARIÁVEIS
        Nomes[4][4]:   STRING
        I,J        : INTEIRO;
INÍCIO
        PARA  I =1 ATE 4 PASSO 1   FACA
                PARA  J = 1 ATE J = 4 PASSO 1  FAÇA
                        LEIA (Nomes [ I][J] );

FIM.
```

13.2.1 Exercícios 9:

1) Faça um algoritmo para ler e imprimir uma matriz 2x4 de números inteiros.

2) Dado uma matriz de ordem 3x3 faça um algoritmo que:
a) Calcule a soma dos elementos da primeira coluna;
b) Calcule o produto dos elementos da primeira linha;

c) Calcule a soma de todos os elementos da matriz;
d) Calcule a soma do diagonal principal;

13.2.2 Algoritmo de Ordenação Bolha Bouble Sort

A ordenação bolha é feita através de dois laços. O laço mais externo, da variável j, determina qual elemento do vetor será usado como base de comparação. O laço mais interno, da variável i, compara cada item com o elemento base do primeiro laço e, quando encontrar um item menor que o de base, faz a troca.

O processo de troca usa uma variável temporária para guardar o valor do menor valor, inicialmente; depois, põe na posição que antes era ocupada pelo item do laço mais interno, o valor do item de base; finalmente, atribui o valor armazenado na variável temporária (o menor valor) ao elemento base. A troca esta concluída.

Veja, abaixo, o algoritmo completo, baseado em um vetor inicializado com 3 elementos sem ordenação definida:

```
ALGORITMO ORDENA_VETOR;

Variáveis

  Vet[3]: inteiro;

  cont1,cont2,temp: inteiro;

Início
  //inicializa o vetor
  vet[1] = 12;
  vet[2] =3;
  vet[3]= 8;
  cont1=0; cont2=0; temp = 0;
  para  de cont1 = 1 até cont = 2 PASSO 1faça  // fixa, incialmente, a bolha
    para cont2 = cont1+1 até cont2 = 3  PASSO 1faça  //compara o 1o. elemento com o
2o. e assim sucessivamente...
    início
      se vet[cont1] > vet [cont2] então //se o 1o. elem. do vetor for maior que o 2o.... - 1a.
passagem
      início
        temp = vet[cont2]; // a variável temporária recebe o menor elemento
        vet[cont2] = vet[cont1]; // o 2a. elemento recebe o valor do 1o., ou seja, do      maior
        vet[cont1] = temp; //finalmente, o 1o. elemento recebe o nr. menor da var. temp
      fim
    fim

  // exibindo os valores
 escreval ("O Vetor ordenado é: ");
 para cont1 = 1 até 3 faça
        escreva(vet[cont1]);

fim.
```

---------- Comprovando o algoritmo. Fazendo o chinês ----------

j	i	temp	vet[j]	vet[i]	vet[j] > vet[i]?
1	2	00	12	03	*** passo 1 (externo)
1	2	03	03	12	*** passo 1 (interno) com o teste e troca de variáveis
1	3	03	03	08	*** passo 2 (interno) sem teste, sem troca de variáveis
2	3	03	12	08	*** passo 2,1 (2 externo e 1 interno)
2	3	08	08	12	*** passo 2,2 (fim passos)

********** final: posicao 1 = 3; posicao 2 = 08; posicao 3 = 12**

14. Programação Modular

A programação estruturada tem sua origem na programação modular. Grande parte da filosofia estruturada é construída sobre a filosofia de modularização de dividir-para-conquistar. A programação modular utiliza essa filosofia para resolver o problema da complexidade do programa. Quando um programa é dividido em partes independentes, fáceis de serem entendidas, sua complexidade pode ser grandemente reduzida. A filosofia estruturada utiliza dividir-para-conquistar e amplia a filosofia de modularização acrescentando os importantes conceitos de organização hierárquica e níveis de abstração para controlar as relações intermodulares.

A *programação modular* pode ser definida como a organização de um programa em unidades independentes, chamadas módulos, cujo comportamento é controlado por um conjunto de regras. Suas metas são as seguintes:

- Decompor um programa em partes independentes
- Dividir um problema complexo em problemas menores e mais simples
- Verificar a correção de um módulo de programa, independentemente de sua utilização como uma unidade em um sistema maior

A modularização pode ser aplicada em diferentes níveis. Pode ser usada para separar um problema em sistemas, um sistema em programas e um programa em módulos.

14.1 – Vantagens da Programação Modular

- Os programas modulares são mais fáceis de entender, porque a análise do programa pode ser feita através da análise dos módulos, um de cada vez.
- O teste do programa é mais simples.
- Os erros do programa são mais fáceis de serem isolados e corrigidos.
- As mudanças no programa podem ser limitadas a apenas alguns módulos, em vez de percutirem pela maior parte do código do programa.
- Pode-se aumentar mais facilmente a eficiência do programa.
- Os módulos do programa podem ser reutilizados como blocos de construção de outros programas.
- O tempo de desenvolvimento do programa pode ser diminuído, porque diferentes módulos podem ser designados a diferentes programadores, que podem trabalhar com maior ou menor independência.

14.2. Esquemas de Modularização

Dividir um programa em módulos pode ser um meio muito eficaz de controlar a complexidade. O problema que surge é quanto à melhor forma de dividir o programa. Mesmo para um programa simples, existem inúmeras maneiras de dividi-lo em um conjunto de módulos estruturados. A maneira escolhida pode afetar significativamente a complexidade do programa. Por exemplo, definir módulos grandes demais ou pequenos demais aumenta a complexidade, os custos e a probabilidade de erros no programa.

A forma de dividir um programa em módulos é chamada de seu esquema de modularização. Um grande objetivo da programação modular é definir um bom esquema de modularização (aquele que reflita rigorosamente os componentes do problema a ser programado, minimize a complexidade e possa ser fácil e eficazmente implantado). A seguir, veremos duas das principais técnicas utilizadas para dividir um programa em módulos, que são os **procedimentos** e as **funções**. Como estamos estudando algoritmos, vamos discriminar uma forma genérica utilizada na maioria das linguagens de programação.

15. Procedimentos e Funções

Ao falarmos de procedimentos e funções temos que, antes, falar sobre **variáveis locais e variáveis globais**; As variáveis locais serão declaradas e utilizadas **apenas no módulo ao qual elas pertençam**; variáveis globais são declaradas **sempre no módulo principal do algoritmo e servirão para todos os módulos subseqüentes**. Até agora, tínhamos declarado variáveis globais e não havíamos percebido. As variáveis globais não são destruídas nem inicializadas no decorrer do algoritmo, enquanto que as locais só vingam enquanto estão sendo utilizadas por aquele módulo. Ao final do mesmo, **elas são destruídas!**

15.1. Implementação dos procedimentos

Veja, abaixo, a forma genérica de se implementar um procedimento. Perceba que um procedimento pode ou não conter parâmetros e que **nunca retornam nada para o módulo que o chamou**. O procedimento deve ser sempre declarado **antes do bloco do módulo principal**.

```
Procedimento  nome_do_ procedimento;
Variáveis
        Local reservado as variáveis locais;
Início
        Instruções;
        .
        .
        .
Fim
```

Exemplo – supondo que quiséssemos fazer a multiplicação de dois números quaisquer através do procedimento Multiplica:

```
ALGORITMO....: Multiplica_procedure.alg
/*AUTOR.......:
DATA.........: 16/10/2006
DESCRIÇÃO....:              */
Algoritmo mult;
//criação do procedimento

Procedimento Multiplica;
Variáveis
  num1, num2: real; //variáveis locais
início
   escreval("Algoritmo Multiplica");
   escreva("Digite o primeiro e o segundo numeros, respectivamente");
   Leia(num1, num2);
   Escreval("O resultado da multiplicacao entre os 2 números eh: ", (num1*num2));
Fim

//******** Módulo principal *******
Variáveis
    num1, num2: real;
início
   Multiplica(); //chama o procedimento Multiplica
  Leia;

Fim.
```

15.1.1. Criação de procedimentos com passagem de parâmetros

Às vezes, torna-se necessária a passagem de parâmetros para dentro de um procedimento. Neste caso, a maneira genérica de se fazer isso é:

```
Procedimento  nome_do_ procedimento([lista de argumentos]: <tipo>);
Variáveis
        Local reservado as variáveis locais;
Início
        Instruções;
        .
        .
        .
Fim
```

Exemplo – supondo que quiséssemos fazer a multiplicação de dois números quaisquer através do procedimento Multiplica(num1, num2):

```
ALGORITMO....: Multiplica_procedure2.alg
/*AUTOR.....:
DATA.....: 16/10/2006
DESCRIÇÃO....:              */
Algoritmo mult;

Procedimento Multiplica(numero1, numero2:real);
início
   Escreval("O resultado da multiplicacao entre os 2 números eh: ", (numero1*numero2));
Fim

//******** MÓDULO PRINCIPAL ********
Variáveis
    num1, num2: real;
início
   escreval("Algoritmo Multiplica");
   escreva("Digite o primeiro e o segundo numeros, respectivamente");
   Leia(num1, num2);
   Multiplica(num1, num2);
  Leia;

Fim.
```

15.2. Implementação das funções

A implementação das funções tem uma grande semelhança com a implementação dos procedimentos. A única diferença é que, em uma função, *necessariamente temos de retornar um valor, mesmo que ele seja a instrução **nulo** do processador (NULL ou NILL)* **e toda a função deve ser chamada no bloco principal através do comando Escreva ou Escreval (característica exclusiva do Pascal).**

```
Função  nome_da_ função(null:<tipo>): <tipo>;
Variáveis
        Local reservado as variáveis locais;
Início
        Instruções;
        .
        .
        .
        retorne(<valor_a_ser_devolvido_ao_bloco_que_chamou_a_função>);
Fim
```

Exemplo:

```
ALGORITMO....: Multiplica_funcao.alg
AUTOR.....:
DATA.....: 16/10/2006
DESCRIÇÃO....:          */
Algoritmo Multiplica_funcao;
Variáveis   //globais
    null: real; // variável global criada Para nao receber argumentos em uma função

//criação da função Multiplica
função Multiplica(null:real) : real ; // Definição da Função que não possui argumentos
Variáveis //locais
  Num1, Num2 : real;
início
   escreval(“Algoritmo Multiplica”);
   escreva(“Digite o primeiro e o segundo numeros, respectivamente”);
   Leia(num1, num2);
   Escreva(“O resultado da multiplicacao entre os 2 números eh: “);


   retorne(Num1 * Num2);  // equivale ao retorno da função.
Fim

início
// *******Programa Principal*******

  Escreva (Multiplica()); //Chama a função multiplica, sem parâmetros


 Leia;
Fim.
```

15.2.1. Criação de funções com passagem de parâmetros

Às vezes, torna-se necessária a passagem de parâmetros para dentro de uma função. Neste caso, a maneira genérica de se fazer isso é:

```
Procedimento  nome_da_ função(lista de argumentos:<tipo>):<tipo>;
Variáveis
        Local reservado as variáveis locais;
Início
        Instruções;
        .
        .
        .
Fim
```

Exemplo – supondo que quiséssemos fazer a multiplicação de dois números quaisquer através da função Multiplica(num1, num2):

```
ALGORITMO....: Multiplica_funcao2.alg
/*AUTOR.....:
DATA.....: 16/10/2006
DESCRIÇÃO....:              */
Algoritmo mult_funcao2;

Função Multiplica(numero1, numero2: real):real;

início
   retorne(numero1* numero2); //retorna a multiplicação dos dois números para quem chamou a função
Fim

Variáveis
    num1, num2: real;
início
   escreval("Algoritmo Multiplica");
   escreva("Digite o primeiro e o segundo numeros, respectivamente");
   Leia(num1, num2);
   escreval("O resultado da multiplicacao eh: ", Multiplica(num1, num2));

  Leia

Fim.
```

15.3. Funções recursivas

Agora que sabemos como as funções funcionam, vamos examinar uma aplicação que pode ser dada às funções chamada *recursão*. Recursão ou recursividade é uma situação em que uma função **chama a si mesma**. Estas funções são conhecidas como *funções recursivas*. Vejamos um exemplo de função recursiva que calcula fatoriais. Um fatorial é uma função matemática. Por exemplo, o fatorial de 5 (escreve-se 5!) é igual a 120 (5 x 4 x 3 x 2 x 1).

```
Algoritmo funcao_fatorial;

função fatorial (n :integer ) : inteiro ;

início

   Se n > 1 então

     retorne( n * fatorial (n-1));

   senão

     retorne(1);

Fim

//********* módulo principal *********
Variáveis //locais
   num1: inteiro ;
Início
   Escreval('Algoritimo fatorial');
   Escreva('Digite o numero a ser calculado seu fatorial');
   Leia(num1);
   Escreval('O fatorial de ', num1, ' eh: ',fatorial(num1));
Fim.
```

16. ARQUIVOS

Entende-se por arquivo uma coleção de dados, onde os mesmos possuem alguma relação entre si. Os arquivos podem ser de vários tipos, os quais destacam-se:

A) Arquivos texto (oriundos de editores ou processadores de texto);
B) Arquivos gráficos (advindos de algum editor de imagens)
C) Arquivos de dados (originados de um banco de dados)
D) Arquivos de programas (executam um programa específico)

Vamos nos deter, em nosso estudo, nos arquivos tipo A e C (de texto e de dados).

16.1. Operações com arquivos

Qualquer tipo de arquivo deve ser capaz de permitir as seguintes operações:

A) Inclusão de dados
B) Alteração dos dados inseridos
C) Exclusão das informações existentes
D) Consulta de todos os dados nele armazenados

Porém, para realiza-las, é necessário a execução das seguintes operações:

- Abertura (leitura) do arquivo, que permitirá a realização de alguma ou todas as operações listadas acima;
- Fechamento do arquivo, que evitará que falhas possam corromper o arquivo outrora aberto.
- Cada linguagem possui a sua forma peculiar de lidar com arquivos

16.2. Tipo A: Arquivos texto

Apresentam-se, essencialmente, na forma de conteúdo string (editores de texto) ou binário (processadores de texto). Para exemplificarmos as operações de leitura e escrita em um arquivo texto, construímos os algoritmos abaixo. Note que o usuário entrará com o nome do arquivo desejado. Observe com atenção os novos comandos, e a lógica do algoritmo, que é de simples entendimento.
OBS: Na verdade, qualquer arquivo poderá ser lido, porém, o único a ser legível na tela serão arquivos gerados por um editor de textos (ex: Bloco de Notas do Windows).

16.3. Tipo C: Arquivos de dados

Todos os aquivos que envolvem cadastro podem ser chamados de arquivos de dados. Eles são subdivididos em:

A) **Registro**: Conjunto de campos. Por exemplo, na criação de uma agenda de telefones, temos os campos:

- Matrícula (inteiro)
- Nome (string, com 20 posições)
- Telefone (string, com 9 posições).

B) **Campo**: Conjunto de informações pertinentes a ele. Os campos podem ser comparados a “variáveis de memória”, inclusive possuindo os mesmos tipos de dados.

- **CONSISTÊNCIA DE UM ARQUIVO DE DADOS**

Para que um arquivo de dados exista de maneira confiável, o mínimo de consistência deve ser imposta. As principais inconsistências são as seguintes:

- Campos-chave duplicados (exemplo um funcionário com o mesmo CPF ou mesma matrícula)
- Exclusão inadequada de registros (selecionar um registro para excluir e o algoritmo excluir outro ou não excluir, simplesmente)
- Corrupção dos dados do arquivo (ocorre quando não se fecha adequadamente o arquivo, falta ou interrupção rápida de energia ou quando um sistema ou programa provoca um erro)

As linguagens de programação atuais, por terem a possibilidade de trabalhar com gerenciadores de banco de dados, quase não enfrentam problemas de inconsistência. No caso do Pascal, devemos tomar o máximo de cuidado para evitarmos as inconsistências listadas acima, quando da criação e manipulação dos arquivos de dados.

17. BIBLIOGRAFIA

CARVALHO, Sérgio E. R. – *Introdução à Programação com Pascal.* Campus, 1985.

COLLINS, William J. – *Programação Estruturada com Estudos de Casos em Pascal*. McGraw-Hill, 1988.

FARRER, Harry et al – *Algoritmos Estruturados.* Guanabara Dois, 1989.

GRILLO, Maria Célia. A. – *Turbo Pascal 5.0 e 5.5*. LTC, 1991.

GUIMARÃES & LAJES – *Algoritmos e Estruturas de Dados*. LTC, 1985.

MANZANO, José Augusto N. G. & YAMATUMI, Wilson Y. – *Programando em Turbo Pascal 7.0*. Érica.

RINALDI, Roberto – *Turbo Pascal 7.0 Comandos e Funções*. Érica, 1993.

SCHIMTZ, Eber A. & TELES, Antonio A. S. – *Pascal e Técnicas de Programação.* LTC, 1985.

TREMBLAY, Jean-Paul & BUNT, Richard B. – *Ciência dos computadores - Uma abordagem Algorítmica.* Mcgraw-Hill, 1983.

WIRTH, Niklaus – *Programação Sistemática em Pascal.* Campus, 1989.

WORTMAN, Leon A. – *Programando em Turbo Pascal com Aplicações*. Campus, 1988.