ALGORITMOS

Prof. Lourival
lourival@dimap.ufrn.br

SEGUNDA PARTE

CAPÍTULO 7: ESTRUTURA DE DADOS HOMOGÊNEOS

7.1 – INTRODUÇÃO

Até agora, vimos os quatro tipos básicos de informação: Inteiro, Real, Lógico e Literal. Mas eles não são suficientes para representar toda e qualquer informação que possa surgir. Portanto, construiremos novos tipos a partir dos tipos básicos, à medida que se fizerem necessários. Esses novos tipos têm um formato denominado **estrutura de dados**, que define como os tipos básicos estão organizados.

Uma variável simples é uma entidade criada para permitir o acesso a uma posição de memória onde se armazena uma informação de um determinado tipo de dado pela simples referência a um nome simbólico.

Neste capítulo, veremos estrutura de dados homogêneos, como vetores, matrizes e cadeias de caracteres, que nos permitirá referenciar um grupo de variáveis do mesmo tipo pelo mesmo nome simbólico.

7.2 – VARIÁVEIS COMPOSTAS HOMOGÊNEAS

Quando uma determinada Estrutura de dados é composta de variáveis com o mesmo tipo básico, temos um conjunto homogêneo de dados.

Variáveis compostas homogêneas, também chamadas de variáveis indexadas, correspondem a um conjunto de variáveis do mesmo tipo, referenciáveis pelo mesmo nome e individualizadas entre sí através de sua posição dentro desse conjunto (os índices).

Uma variável indexada pode ser definida contendo um ou mais índices. Quando possui um único índice a variável é chamada de **vetor** e quando possui dois índices é chamada de **matriz**. Com três ou mais índices não recebe nome especial, também, na prática, sua ocorrência é pouco freqüente.

Ao número de índices necessários à localização de um componente dentro da variável indexada dá-se o nome de **dimensão**.

7.3 – VETORES

As variáveis indexadas que possui apenas um índice são chamadas de vetores ou variáveis compostas unidimensionais.

Notação:

<nome_variável>[<índice>]

x[i] : variável indexada.
i : índice (expressão inteira positiva)

Exemplos:

a) A[3] : representa o terceiro elemento do vetor A.
b) Nome[p] : representa o p-ésimo elemento do vetor Nome.
c) x[2*i + 3*j - 4*k] : a avaliação da expressão entre colchetes, que deverá ser um número inteiro positivo, dará a posição do elemento no conjunto x.

7.3.1 – DECLARAÇÃO DE VETORES

Como qualquer variável simples, para usarmos vetores precisamos antes declará-lo.

Sintaxe: <tipo> : <nome>[<limite>]

Onde <tipo> poderá ser qualquer dos tipos válidos, <nome> é qualquer nome de uma variável simples representativa do conjunto e <limite> é um número inteiro positivo que limita o valor máximo para o índice da variável indexada, ou seja, número máximo de elementos do vetor.

Exemplos:

Real: vetor1[10], vetor2[20]
Inteiro: pares[30], impares[50]
Lógico: opcoes[20]
Literal[30]: nomes[10], datas[20], cidades[30]

7.3.2 – OPERAÇÕES COM VETORES

Não é possível operar diretamente com conjuntos, como um todo, mas apenas com cada um de seus componentes, um por vez. O acesso individual a cada componente de um conjunto é realizado pela especificação de sua posição, no mesmo, por meio de um ou mais índices. Por exemplo, para somar dois vetores é necessário somar cada um dos seus componentes, dois a dois.

Não se deve confundir o número entre colchetes na declaração de variáveis indexadas com o número entre colchetes no processamento de alguma operação. No primeiro caso, o número representa o limite superior para os índices, enquanto que no segundo caso, o número representa a posição do elemento no conjunto (o índice).

7.3.2.1 – ATRIBUIÇÃO DE VETORES

Cada vez que se processa uma variável indexada, qualquer que seja a operação, o índice deve ser um valor conhecido.

A sintaxe da atribuição para variáveis indexadas é a mesma, sendo que a variável, além do nome, deve conter o(s) índice(s) da componente do conjunto, onde deverá ser armazenado o valor da expressão. A expressão também poderá conter variáveis indexadas.

Exemplos:

a) x[1] ← 0

b) y[10] ← 2*x**3+1

c) num[3] ← 3*num[1] + 5*num[2]

d) fibo[n] ← fibo[n-2] + fibo[n-1]

e)
```
Para i de 1 até 10 faça
        p[i] ← 3*i-1
Fim_para
```

f)
```
Para u de 1 até n faça
        Se (u/2*2 = u) então
                x[u] ← 0
        senão
                x[u] ← 1
        Fim_se
Fim_para
```

7.3.2.2 – LEITURA DE VETORES

A leitura de um conjunto é feita passo a passo, um componente por vez, usando a mesma sintaxe da instrução primitiva de entrada de dados, ou seja, a instrução **Leia (<lista_de_variáveis>)**, sendo que, mais uma vez, explicitando a posição da componente lida. É bastante freqüente o uso de estruturas de repetição na leitura de variáveis indexadas.

Exemplos:

a) Leia(x[1], x[2], x[3], x[4], x[5])
   Este exemplo é válido, mas pouco prático, principalmente se o conjunto possui muitos elementos.

b) Para i de 1 até 100 faça
   Leia (x[i])
   Fim_para

c) Para k de 1 até n faça
   Repita
   Escreva("Digite um número positivo:")
   Leia (num[k])
   Até (num[k] >0)
   Fim_para

d) Para i de 1 até n faça
   Repita
   Escreva("Digite um número positivo:")
   Leia (p)
   Até (p > 0)
   x[i] ← p
   Fim_para

7.3.2.3 – ESCRITA DE VETORES

A escrita de um conjunto obedece à mesma sintaxe da instrução primitiva de saída, ou seja, a instrução

Escreva (<lista_de_expressões>) .

E de forma análoga ao ítem anterior, utilizamos as estruturas de repetição para escrever todos os elementos de um conjunto.

Exemplos:

a)
```
Para i de 1 até p faça
        Escreva ( x[i] )
Fim_para
```

b)
```
Escreva ("Vetor Solução: ")
Para j de 1 até n faça
        Escreva ( "x[",j,"]=", x[j] )
Fim_para
```

c)
```
Escreva ( " i        X[i]      Y[i]      Z[i] " )
Para i de 1 até n faça
        Escreva ( i , x[i] , y[i] , z[i] )
Fim_para
```

7.3.3 – EXEMPLOS DE ALGORITMOS COM VETORES

1) Imprimir a soma de n números dados.

```
Algoritmo Soma
        Real: x[100], soma
        Inteiro: n, i
Início
        Repita
                Escreva (“Quantos números? “)
                Leia ( n )
        Até ( n > 0 .E. n <= 100 )
        Escreva ( “Digite os “, n , “ números: “ )
        Para i de 1 até n faça
                Leia ( x[i] )
        Fim_para
        soma ← 0
        Para i de 1 até n faça
                soma ← soma + x[i]
        Fim_para
        Escreva ( “Soma = “, soma )
Fim
```

2) Imprimir os n primeiros termos da seqüência de Fibonacci:
1 1 2 3 5 8 13 21 . . .

```
Algoritmo Fibonacci
        Inteiro: f[100], n, i
Início
        Repita
                Escreva(“Quantos termos?”)
                Leia ( n )
        Até ( n>0 .E. n <=100 )
        f[1] ← 1
        f[2] ← 1
        Para i de 3 até n faça
                f[i] ← f[i-2] + f[i-1]
        Fim_para
        Escreva (“Sequência de Fibonacci: “)
        Para i de 1 até n faça
                Escreva ( f[i] )
        Fim_para
Fim
```

3) Dados dois vetores com n componentes cada, calcular o produto escalar entre eles.

```
Algoritmo Produto_escalar
        Real: x[10], y[10], pesc
        Inteiro: n,i
Início
        Repita
                Escreva (“Quantas componentes tem cada
vetor? “)
                Leia ( n )
        Até ( n>0 .E. n<=10 )
        Escreva( “Digite os números do primeiro vetor: “)
        Para i de 1 até n faça
                Leia ( x[i] )
        Fim_para
        Escreva ( “Digite os números do segundo vetor: “)
        pesc ← 0
        Para i de 1 até n faça
                Leia ( y[i] )
                pesc ← pesc + x[i]*y[i]
        Fim_para
        Escreva (“Produto Escalar = “, pesc)
Fim
```

4) Dados n números inteiros positivos, imprimi-los, separadamente, como pares e ímpares.

```
Algoritmo Par_ímpar
        Inteiro: x[30], par[30], impar[30], n, k, p, i
Início
        Repita
                Escreva (“Quantos números?”)
                Leia ( n )
        Até ( n>0 .E. n<=30 )
```

```
        Escreva (“Digite os”, n, “ números inteiros
positivos:”)
        p ← 0
        i  ← 0
        Para k de 1 até n faça
                Repita
                        Leia ( x[k] )
                Até ( x[k] > 0 )
                Se ( x[k]/2*2 = x[k] ) então
                        p ← p+1
                        par[p] ← x[k]
                senão
                        i ← i+1
                        impar[i] ← x[k]
                Fim_se
        Fim_para
        Escreva(“Números pares: “)
        Para k de 1 até p faça
                Escreva (par[k])
        Fim_para
        Para k de 1 até i faça
                Escreva(ímpar[k])
        Fim_para
Fim
```

5) Calcular e imprimir o conjunto interseção entre dois conjuntos numéricos quaisquer dados.

```
Algoritmo Interseção
        Real: a[20], b[20], c[20]
        Inteiro: m, n, i, j, k
Início
        Repita
                Escreva(“Quantos números tem o primeiro
conjunto? “)
                Leia( m )
```

```
        Até( m>0 .E. m<=20 )
        Escreva("Digite os números do primeiro conjunto:")
        Para i de 1 até m faça
                Leia( a[i] )
        Fim_para
        Repita
                Escreva("Quantos números tem o segundo
        conjunto?")
                Leia ( n )
        Até( n>0 .E. n<=20 )
        Escreva("Digite os números do segundo conjunto:")
        Para i de 1 até n faça
                Leia( b[i] )
        Fim_para
        k ← 0
        Para i de 1 até m faça
                Para j de 1 até n faça
                        Se( a[i] = b[j] ) então
                                k ← k+1
                                c[k] ← a[i]
                        Fim_se
                Fim_para
        Fim_para
        Se ( k=0 ) então
                Escreva("Interseção Vazia. ")
        Senão
                Escreva("Conjunto Interseção:")
                Para i de 1 até k faça
                        Escreva( c[i] )
                Fim_para
        Fim_se
Fim
```

6) **Pesquisa seqüencial**.

Pesquisa seqüencial ou linear é o método para se encontrar um elemento particular num conjunto não_classificado. Vejamos um algoritmo para ler um número e verificar se o mesmo se encontra num vetor com n elementos.

```
Algoritmo Pesquisa_sequencial
        Real: x[100], num
        Inteiro: n, i
        Lógico: achou
Início
        Repita
                Escreva("Quantos números?")
                Leia( n )
        Até( n>0 .E. n<=100)
        Escreva("Digite todos os números:")
        Para i de 1 até n faça
                Leia( x[i] )
        Fim_para
        Escreva("Digite o número que procura:")
        Leia (num)
        achou ← .F.
        i ← 1
        Enquanto ( i <= n .E. .Não. achou )faça
                Se( x[i] = num) então
                        achou ← .V.
                senão
                        i ← i+1
                Fim_se
        Fim_enquanto
        Se( achou ) então
                Escreva("Número encontrado. ")
        Senão
                Escreva("Número não encontrado.")
        Fim_se
Fim
```

7) **Pesquisa binária**.

Pesquisa binária é semelhante à pesquisa seqüencial quanto ao objetivo, sendo que os elementos do vetor estão previamente classificados segundo algum critério. Vejamos um algoritmo, supondo que os números do vetor estão classificados na ordem crescente.

```
Algoritmo Pesquisa_binária
        Real: x[100], num
        Inteiro: n, i, meio, alto, baixo
        Lógico: achou
Início
        Repita
                Escreva("Quantos números?")
                Leia( n )
        Até ( n>0 .E. n<=100 )
        Escreva("Digite todos os números:")
        Para i de 1 até n faça
                Leia ( x[i] )
        Fim_para
        Escreva ("Digite o número que procura: ")
        Leia( num )
        alto ← n
        baixo ← 1
        achou ← .F.
        Enquanto( baixo<=alto .E. .Não. achou ) faça
                meio ← (baixo + alto)/2
                Se( num < x[meio] )então
                        alto ← meio – 1
                senão
                        Se( num > x[meio] ) então
                                baixo ← meio + 1
                        senão
                                achou ← .V.
                        Fim_se
                Fim_se
        Fim_enquanto
        Se( achou ) então
                Escreva("Número encontrado. ")
        Senão
                Escreva("Número não encontrado. ")
        Fim_se
Fim
```

8) **Classificação na ordem crescente**.

Colocar na ordem crescente n números quaisquer dados. Vamos utilizar o algoritmo conhecido como método da bolha.

```
Algoritmo Ordem_crescente
        Real: x[100], aux
        Inteiro: n, i, j
Início
        Repita
                Escreva("Quantos números?")
                Leia( n )
        Até( n>0 .E. n<=100 )
        Escreva("Digite os números:")
        Para i de 1 até n faça
                Leia ( x[i] )
        Fim_para
        Para j de n até 2 passo  –1 faça
                Para i de 1 até j –1 faça
                        Se( x[i] > x[i+1] ) então
                                aux ← x[i]
                                x[i] ← x[i+1]
                                x[i+1] ← aux
                        Fim_se
                Fim_para
        Fim_para
        Escreva("Vetor ordenado: ")
        Para i de 1 até n faça
                Escreva ( x[i] )
        Fim_para
Fim
```

7.3.4 – EXERCÍCIOS PROPOSTOS

1) Dados dois vetores com n componentes cada um, calcular e imprimir a soma deles.

2) Calcular o cosseno do ângulo formado por dois vetores dados, com o mesmo número de coordenadas, através da fórmula:

$$\text{Cosseno} = (A.B) / (|A| \, |B|)$$

Onde A.B é o produto escalar entre os vetores A e B, e |A| é o módulo do vetor A, dado por $|A| = \sqrt{A.A}$.

3) Calcular a média aritmética de n números reais dados.

4) Calcular o n-ésimo termo da seqüência de Fibonacci.

5) Leia n números maiores que 1, e imprima-os, separadamente, como primos e não-primos.

6) Dado um vetor A com n números reais, obter um outro vetor B, também com n números, da seguinte forma:

```
B[1] = 2*A[1]
B[2] = 3*A[1] + 2*A[2]
B[3] = 4*A[1] + 3*A[2] + 2*A[3]
.
.                ( ...e assim por diante,)
.
```

7) Calcular o Desvio padrão de uma amostra x de n números quaisquer dados.

$$\text{Desvio padrão} = \sqrt{ (\sum (x[i] - M)^2) / (n - 1) }$$

Onde M é a média dos n números dados.

8) Leia um conjunto com n números e informe se existe algum elemento repetido no conjunto.

9) Leia n números quaisquer e imprima-os sem repetições.

10) Dado um número inteiro positivo, do sistema decimal, obtenha o seu valor correspondente no sistema binário.

7.4 – MATRIZES

Os vetores, ou variáveis compostas unidimensionais, têm como principal característica a necessidade de apenas um índice para endereçamento. Uma estrutura que precise de mais de um índice, como no caso de matrizes, será denominada estrutura composta multidimensional.

As variáveis compostas multidimensionais, mais utilizadas são as bidimensionais, ou matrizes, devido à sua relação direta com muitas aplicações. A aplicação com as demais variáveis indexadas, isto é, com três índices, quatro índices, etc., se faz por analogia a esta.

Notação:

<nome_variável>[<índice1>,<índice2>, ... , <índicen>]

x[i,j,k] : variável indexada de dimensão três.
i, j, k: índices (valores inteiros positivos).

Exemplos:

a) x[2,3] : elemento de uma matriz (variável indexada bidimensional) da segunda linha e terceira coluna (posição na matriz).
b) mat[i+1,j-1] : os índices podem ser expressões desde que sejam inteiras.

7.4.1 – DECLARAÇÃO DE MATRIZES

Veremos, de agora em diante, apenas matrizes(dois índices) como variáveis compostas multidimensionais .

Sintaxe: <tipo>: <nome>[<limite1>,<limite2>]

Exemplos:

a) Real: mat1[10,10]
b) Inteiro: x[15,15], y[20,20], z[10,10]
c) Lógico: achou[5,5]
d) Literal[20]: nomes[15,15]

7.4.2 – OPERAÇÕES COM MATRIZES

Percebemos que uma estrutura bidimensional é um conjunto de estruturas unidimensionais, ou seja, uma matriz é um conjunto de vetores. Portanto, para operarmos com matrizes, geralmente, lançamos mão de duas estruturas de repetição (para vetores utilizamos uma).

7.4.2.1 – ATRIBUIÇÃO DE MATRIZES

A atribuição é uma das formas de qualquer variável armazenar algum valor. Como não operamos diretamente com a matriz, somente seus elementos armazenam valores numa atribuição.

Exemplos:

a)
```
mat[3,4] ← 3.75
```

b)
```
Para i de 1 até 10 faça
        Para j de 1 até 10 faça
                Se(i = j) então
                        x[i,j] ← 1
                senão
                        x[i,j] ← 0
                Fim_se
        Fim_para
Fim_para
```

c)
```
Para i de 1 até m faça
    Para j de 1 até n faça
        Se ( i > j ) então
                a[i,j] ← 2*i + 3*j
        senão
                Se ( i = j ) então
                        a[i,j] ← i**2
                senão
                        a[i,j] ← 5*i**3 – 2*j**2
                Fim_se
        Fim_se
    Fim_para
Fim_para
```

7.4.2.2 – LEITURA DE MATRIZES

De forma análoga a leitura de vetores, utilizamos dois laços para a leitura de matrizes.

Exemplos:

a)
```
Para i de 1 até m faça
    Para j de 1 até n faça
        Leia ( mat[i,j] )
    Fim_para
Fim_para
```

b)
```
Escreva("Digite números positivos:")
Para a de 1 até p faça
    Para b de 1 até p faça
        Repita
            Escreva("Atenção! Positivo.")
            Leia( x )
        Até ( x > 0 )
        matriz[a,b] ← x
    Fim_para
Fim_para
```

7.4.2.3 – ESCRITA DE MATRIZES

De forma semelhante a leitura fazemos a escrita de matrizes.

Exemplos:

a)
```
Para i de 1 até m faça
    Para j de 1 até n faça
        Escreva ( matriz[i,j] )
    Fim_para
Fim_para
```

b)
```
Para i de 1 até m faça
    Para j de 1 até n faça
        Escreva("x[",i,",",j,"] = ",x[i,j] )
    Fim_para
Fim_para
```

7.4.3 – EXEMPLOS DE ALGORITMOS COM MATRIZES

1) Imprimir a matriz identidade de ordem n.

```
Algoritmo Matriz_identidade
        Inteiro: ident[20,20], n, i, j
Início
        Repita
                Escreva("Digite a ordem da matriz <=20:")
                Leia ( n )
        Até ( n>0 .E. n<=20 )
        Para i de 1 até n faça
                Para j de 1 até n faça
                        Se ( i = j ) então
                                ident[i,j] ← 1
                        senão
                                ident[i,j] ← 0
                        Fim_se
                Fim_para
        Fim_para
        Para i de 1 até n faça
                Para j de 1 até n faça
                        Escreva( ident[i,j] )
                Fim_para
        Fim_para
Fim
```

2) Dada uma matriz quadrada de ordem n>3, calcular:

a) a soma dos elementos da primeira linha;
b) a soma dos elementos da terceira coluna;
c) a soma dos elementos da diagonal principal;
d) a soma dos elementos da diagonal secundária;
e) a soma de todos os elementos da matriz.

```
Algoritmo Cálculos_matriciais
        Real: mat[15,15], sa, sb, sc, sd, se
        Inteiro: n, i, j
Início
        Repita
                Escreva("Digite a ordem da matriz >3:")
                Leia ( n )
        Até ( n>3 .E. n<=15 )
        Escreva("Digite os elementos da matriz por linha:")
        Para i de 1 até n faça
                Para j de 1 até n faça
                        Leia( mat[i,j] )
                Fim_para
        Fim_para
        sa ← 0
        sb ← 0
        sc ← 0
        sd ← 0
        se ← 0
        Para i de 1 até n faça
                sa ← sa + mat[1,i]
                sb ← sb + mat[i,3]
                sc ← sc + mat[i,i]
                sd ← sd + mat[i,n – i + 1]
                Para j de 1 até n faça
                        se ← se + mat[i,j]
                Fim_para
        Fim_para
        Escreva("sa=",sa," sb=",sb," sc=",sc," sd=",sd," se=", se)
Fim
```

3) Dada uma matriz quadrada de ordem n, calcular e imprimir a soma dessa matriz com a sua transposta.

```
Algoritmo Soma_transposta
        Real: x[10,10], y[10,10]
        Inteiro: n, i, j
Início
        Repita
                Escreva("Digite a ordem da matriz <=10")
                Leia ( n )
        Até ( n>0 .E. n<=10)
        Escreva("Digite os números da matriz:")
        Para i de 1 até n faça
                Para j de 1 até n faça
                        Leia( x[i,j] )
                Fim_para
        Fim_para
        Para i de 1 até n faça
                Para j de 1 até n faça
                        y[i,j] ← x[i,j] + x[j,i]
                Fim_para
        Fim_para
        Escreva("Matriz resultante:")
        Para i de 1 até n faça
                Para j de 1 até n faça
                        Escreva( y[i,j] )
                Fim_para
        Fim_para
Fim
```

4) Dada uma matriz A qualquer de números inteiros positivos, gerar uma matriz B tal que:
B[i,j] = 0, se A[i,j] é par e B[i,j] = 1, se A[i,j] é ímpar.

```
Algoritmo Matriz_B
    Inteiro: A[10,10], B[10,10], m, n, i, j
Início
  Repita
    Escreva("Digite o número de linhas e o número de colunas <=10:")
    Leia ( m, n )
  Até( m>0 .E. m<=10 .E. n>0 .E. n<=10 )
  Escreva("Digite os números da matriz de inteiros positivos: ")
  Para i de 1 até m faça
    Para j de 1 até n faça
      Repita
        Leia ( A[i,j] )
      Até ( A[i,j] > 0 )
      Se ( A[i,j]/2*2 = A[i,j] ) então
        B[i,j] ← 0
      senão
        B[i,j] ← 1
      Fim_se
    Fim_para
  Fim_para
  Escreva("Matriz pedida:")
  Para i de 1 até m faça
    Paraj de 1 até n faça
      Escreva( B[i,j] )
    Fim_para
  Fim_para
Fim
```

5) Obtenha o produto entre duas matrizes conformes.

Obs. Dadas duas matrizes de tipos possíveis de se efetuar o produto entre elas, por exemplo a matriz A do tipo m por n, e a matriz B do tipo n por p, então obteremos uma outra matriz C do tipo m por p.

```
Algoritmo Produto_matricial
    Real: a[10,10], b[10,10], c[10,10]
    Inteiro: m, n, p, i, j, k
Início
    Repita
        Escreva("Digite o número de linhas e o número de colunas da primeira matriz, e também o número de colunas da segunda matriz: ")
        Leia ( m, n, p )
    Até(m>0 .E. m<=10 .E. n>0 .E. n<=10 .E. p>0 .E. p<=10)
    Escreva("Digite os números da primeira matriz: ")
    Para i de 1 até m faça
        Para j de 1 até n faça
            Leia ( a[i,j] )
        Fim_para
    Fim_para
    Escreva("Digite os números da segunda matriz: ")
    Para i de 1 até n faça
        Para j de 1 até p faça
            Leia ( b[i,j] )
        Fim_para
    Fim_para
    Para i de 1 até m faça
        Para k de 1 até p faça
            c[i,k] ← 0
            Para j de 1 até n faça
                c[i,k] ← c[i,k] + a[i,j]*b[j,k]
            Fim_para
        Fim_para
    Fiom_para
    Escreva("Matriz Produto: ")
    Para i de 1 até m faça
        Para j de 1 até p faça
            Escreva( c[i,j] )
        Fim_para
    Fim_para
Fim
```

7.4.4 – EXERCÍCIOS PROPOSTOS

1) Gerar e imprimir uma matriz com m linhas e n colunas onde seus elementos são da forma:

$$A[i,j] = \begin{cases} 2*i + 7*j - 2 & \text{se } i < j \\ 3*i**2 - 1 & \text{se } i = j \\ 4*i**3 - 5*j**2 + 1 & \text{se } i > j. \end{cases}$$

2) Calcular a soma dos elementos de uma matriz numérica quadrada qualquer dada, que estão acima da diagonal principal.

3) Obtenha e imprima um vetor que seja a soma dos elementos de cada coluna de uma matriz numérica qualquer dada.

4) Verificar se uma matriz quadrada dada é ou não simétrica.

5) Gerar e imprimir a matriz quadrada de ordem n abaixo.

$$\begin{vmatrix} 1 & n-1 & n-2 & ... & 1 \\ 2 & 1 & n-2 & ... & 1 \\ 3 & 3 & 1 & ... & 1 \\ . & . & . & ... & . \\ . & . & . & ... & . \\ . & . & . & ... & . \\ n & n & n & ... & 1 \end{vmatrix}$$

6) A matriz abaixo, quadrada 4x4, foi obtida de acordo com uma determinada definição para seus elementos. Faça um algoritmo que leia um número inteiro positivo n e imprima essa matriz com n linhas e n colunas.

$$\begin{vmatrix} 3 & 4 & 5 & 6 \\ 5 & 5 & 6 & 7 \\ 7 & 8 & 7 & 8 \\ 9 & 10 & 11 & 9 \end{vmatrix}$$

7.5 – CADEIAS E SUBCADEIAS DE CARACTERES

A maior parte dos processamentos efetuados atualmente exige a manipulação de cadeias de caracteres alfanuméricos. Uma cadeia de caracteres é uma sequência de letras, algarismos ou símbolos(sinais de pontuação, parênteses, etc.). Cada caractere é uma informação e a cadeia de caracteres é um conjunto de informações.

A relação de ordem entre as cadeias depende da relação de ordem vigente para os caracteres e esta depende da linguagem de programação utilizada na codificação do algoritmo.

Conforme já vimos na apresentação do tipo de dados Literal, esta ordem está estabelecida pela ordem dos caracteres no sistema de codificação utilizado(ASCII).

A representação de uma cadeia de caracteres como um dado de processamento é feita através da seqüência de seus caracteres entre aspas duplas. Por exemplo: “JANEIRO”, “abcdefg”, “Rio Grande do Norte”, “etc.”.

As variáveis do tipo Literal armazenam cadeias de caracteres .

Lembramos que o único operador literal, que nos permite operar cadeias de caracteres, é o operador de concatenação + (“abc” + “de” resulta “abcde”).

Exemplo:

mes ← “FEVEREIRO”
nome ← “Ana Maria Duarte”
endereco ← “Rua Afonso Pena, 625. Tirol”

7.5.1 – DECLARAÇÃO DE CADEIA DE CARACTERES

A declaração de variáveis do tipo literal, é feita como se fosse um vetor onde seus elementos são caracteres, sendo que o número que indicaria o limite máximo de elementos do vetor não fica junto à variável, e sim, ao lado do tipo, mas o objetivo continua sendo reservar espaço na memória para armazenamento de constantes. No momento da declaração o espaço reservado fica cheio de caracteres vazios(“”) e, durante o processamento, a medida que a variável vai recebendo caracteres, os vazios vão sumindo, mas o programador deve providenciar para que tenhamos pelo menos um caractere vazio em qualquer momento do processamento, pois isso o ajudará a saber quantos caracteres poderão ser acessados, ou seja, ele deverá declarar n caracteres e utilizar, no máximo, n-1,sendo n uma constante inteira maior que 1.

Sintaxe: Literal[<limite>]: <lista_variáveis>

Exemplos:

a) Literal[41]: nome, endereco

nome e endereco podem armazenar até 40 caracteres.

b) Literal[21]: nomes[50], cidades[50]

nomes e cidades são conjuntos (vetores) com, no máximo, 50 cadeias de caracteres, contendo, no máximo, 20 caracteres cada cadeia.

Como em um vetor, todos os caracteres de uma cadeia são armazenados em unidades consecutivas de armazenamento de caracteres (1 byte para cada caractere). O número de caracteres (ou de unidades de armazenamento) de uma cadeia é o comprimento da cadeia, e a cada caractere corresponde um número (posição) dentro da cadeia (como o índice nos vetores).

7.5.2 – SUBCADEIA DE CARACTERES

Uma subcadeia de caracteres é uma seqüência de um ou mais caracteres consecutivos dentro de uma cadeia. Por exemplo, “JANE” é uma subcadeia de “JANEIRO”, mas “JAIRO” não é.

O recurso da subcadeia propicia a manipulação (o processamento) de partes de uma determinada cadeia de caracteres.

Notação da subcadeia:

<nome_cadeia>[<início_cadeia>:<fim_cadeia>]

<nome_cadeia> é um nome qualquer de uma variável declarada do tipo Literal.

<início_cadeia> é um número inteiro positivo que indica a posição dentro da cadeia onde a subcadeia inicia.

<fim_cadeia> é um número inteiro positivo que indica a posição dentro da cadeia onde a subcadeia termina.

Exemplos: x[3:6], nome[4:10], mês[3:3]

Exemplo: Seja a cadeia vogal ← “AEIOU”

então:	a subcadeia	vogal[3:4]	corresponde a	“IO”
	a subcadeia	vogal[1:5]	corresponde a	“AEIOU”
	a subcadeia	vogal[2:2]	corresponde a	“E”

7.5.3 – EXEMPLOS DE ALGORITMOS COM CADEIAS E SUBCADEIAS DE CARACTERES

1) Diga como será a saída do algoritmo abaixo.

```
Algoritmo Cadeia_caracteres
        Literal[11]: cadeia
Início
        cadeia[1:3] ← “ABC”
        cadeia[4:7] ← “DEFG”
        cadeia[6:10] ← “HIJKL”
        Escreva( cadeia )
Fim
```

Saída:

ABCDEHIJKL

2) Diga como será a saída do algoritmo abaixo.

```
Algoritmo Branco
        Literal[52]: branco, z
        Inteiro: n
Início
        Para n de 1 até 50 faça
                branco[n:n] ← “ “
                z ← branco[1:n] + “Z”
                Escreva( z )
        Fim_para
Fim
```

Saída:

```
 Z
  Z
   Z
    Z
     .
      .
       .
```

3) Calcular o comprimento de uma cadeia de caracteres dada.

```
Algoritmo Comprimento
        Literal[81]: nome
        Inteiro: n
Início
        Escreva("Digite a cadeia: ")
        Leia( nome )
        n ← 0
        Enquanto( nome[n+1:n+1] <> ""faça
                n ← n + 1
        Fim_enquanto
        Escreva("Comprimento = ", n)
Fim
```

4) Converter uma letra minúscula em letra maiúscula.

```
Algoritmo Maiúsculo
        Literal[27]: alfa, ALFA
        Literal[2]: letra
        Inteiro: i
Início
        Repita
                Escreva("Digite uma letra minúscula: ")
                Leia( letra )
        Até( letra>="a" .E. letra<="z" )
        alfa ← "abcdefghijklmnopqrstuvwxyz"
        ALFA ← "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
        i ← 1
        Enquanto ( letra <> alfa[i:i] ) faça
                i ← i + 1
        Fim_enquanto
        letra ← ALFA[i:i]
        Escreva (letra)
Fim
```

7.5.4 – EXERCÍCIOS PROPOSTOS

1) Converta uma letra maiúscula em letra minúscula.

2) Faça um algoritmo para converter uma cadeia de caracteres de letras maiúsculas em letras minúsculas.

3) Dado o nome completo de uma pessoa imprimir apenas o primeiro nome.

4) Dado o nome completo de uma pessoa imprimir apenas as iniciais seguidas cada uma de ponto e espaço.

5) Dada uma cadeia de caracteres, inserir um caractere dado numa posição da cadeia também dada e imprimir a nova cadeia.

6) Dada uma cadeia de caracteres, eliminar o caractere de uma posição dada nessa cadeia e imprimir a nova cadeia.

7) Dado um texto com, no máximo, 80 caracteres, diga quantas consoantes existem no texto.

8) Dado um texto com, no máximo, 80 caracteres, diga quantas vogais existem no texto.

9) Calcular o custo de um telegrama, onde os 30 primeiros caracteres custam 9 centavos cada, os demais, a partir daí, custam 6 centavos cada e os espaços em branco não são cobrados.

10) Dado um número inteiro positivo do sistema binário, converte-lo no número correspondente do sistema decimal.