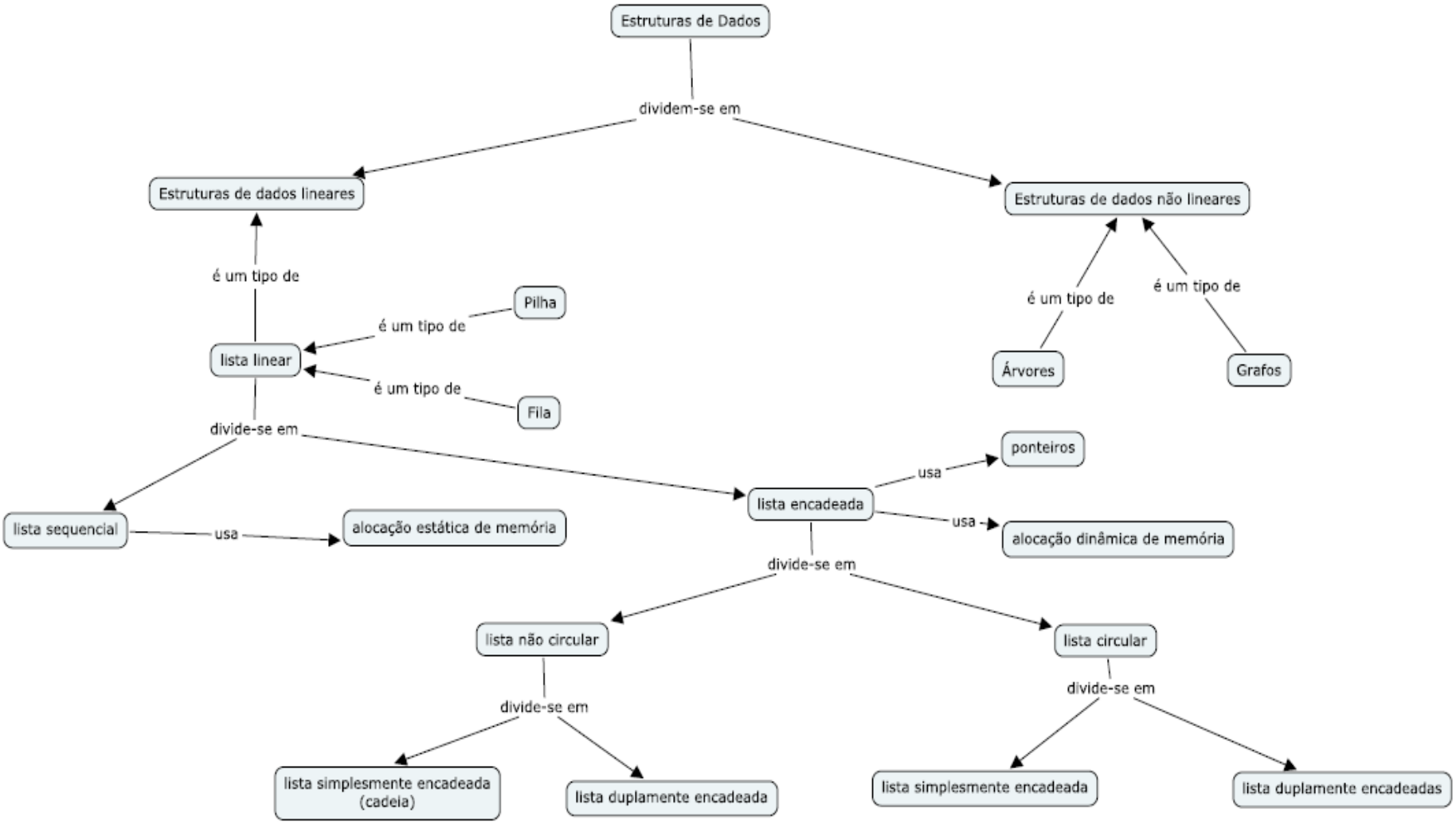

# Estrutura de Dados

## Vetores e Matrizes

Prof. André Lira Rolim

andre.rolim@yahoo.com.br

# Mapa Conceitual



# Conceito

- **Estrutura de Dados** são formas genéricas de se estruturar informações de modo a serem armazenadas e processadas pelo computador.

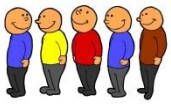
  **Ex**.:
  **vetores;**
  matrizes;
  lista;
  árvores;
  pilha;
  grafos e etc.

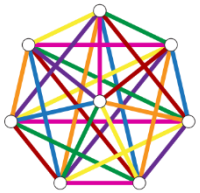
- Contudo estas só adquirem significado quando associadas a um conjunto de **operações**, que visam, de um modo geral, manipulá-las (**algoritmos**).

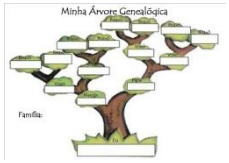
# Tipos

- Os dados manipulados por um computador são classificados de acordo com seus **TIPOS. (inteiro, real, Strings etc...)**
- Os Tipos de dados podem ser classificados como **Atômicos** (int, float, boolean etc..) ou **Complexos** (ou Compostos ou estruturados).
- Para cada tipo de dados existe um conjunto de operações associado ao mesmo.

# Estrutura de Dados (Overview)








**TADs**
**(Alto Nível)**

**Tipos Complexos**
(ex. struct, typedef)

**Estruturas Básicas**
**(Armazenamento em memória)**
(Vetores, Matrizes,Listas de Adjacências etc...)

**Operações Nativas com Tipos Primitivos**

Inteiro

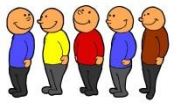
Char

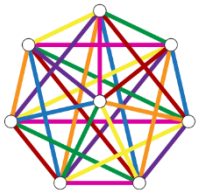
Boolean

Tipos Primitivos

# Estrutura de Dados (Overview)

- 1º Vamos estudar as estruturas básicas.






**TADs**
**(Alto Nível)**

**Tipos Complexos**
(ex. struct, typedef)

**Estruturas Básicas**
**(Armazenamento em memória)**
(Vetores, Matrizes,Listas de Adjacências etc...)

**Operações Nativas com Tipos Primitivos**

Inteiro

Char

Boolean

Tipos Primitivos

# Estruturas de Dados

- Permitem **agrupar diversas informações** dentro de uma mesma variável.
- Um **agrupamento** que obedece sempre **ao mesmo tipo de dado**, são chamadas **homogêneas**.
- A utilização deste tipo de estrutura de dados recebe diversos nomes, tais como: *variáveis indexadas, variáveis compostas,* ***vetores****,* ***matrizes****, tabelas em memória ou arrays.*

# Vetores

- **Matriz de uma dimensão (Vetores)**

Exemplo:

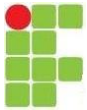
**V = [a1,1 a1,2 a1,3 … a1,n]**

# Vetores

- Estrutura formada por um **conjunto** unidimensional de dados de **mesmo tipo** (**homogêneo**) que possui um número fixo de elementos (**estático**).

- É uma **série de variáveis** do **mesmo tipo** referenciadas por um único **nome**, onde cada variável (**elemento**) é diferenciada através de um número chamado "**índice**".

- É um tipo de variável capaz de armazenar uma **coleção de valores** de um **mesmo tipo de dados**.



# Vetores

- Os elementos do vetor são guardados em posições consecutivas de memória (alocação sequêncial).

**Ex1.:**

- v = vetor de inteiros com 10 elementos

```
int v[10];
```

**Ex2:**

```
int v[5] = { 5, 10, 15, 20, 25 };

ou simplesmente:

int v[] = { 5, 10, 15, 20, 25 };
```

# Vetores

- Os elementos do vetor são guardados em posições consecutivas de memória (alocação sequêncial), e são acessados de forma indexado.

**Ex.**:

```
v[0] = 0;          /*  acessa o primeiro elemento de v     */
...
v[9] = 9;          /*  acessa o último elemento de v       */

v[10]  =10         /*  ERRADO (invasão de memória)         */
```

Neste exemplo os espaços de memória indexados pelos **índices 0,9 e 10** são atualizados com os **valores 0, 9 e 10**.

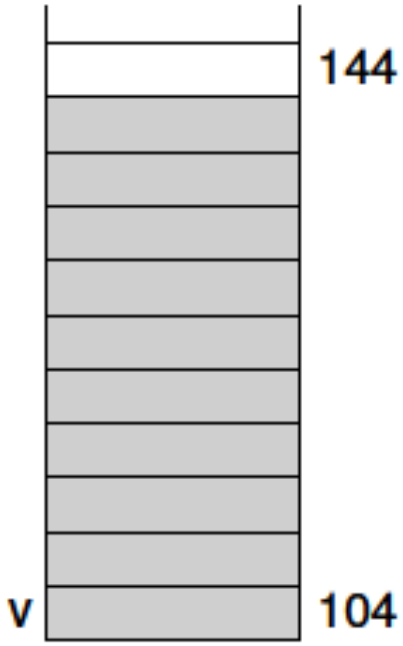
OBS.: Notem que em C, o primeiro índice começa com ‘0’.

# Vetores

- Os Vetores são alocados em posições contíguas de Memória.

**EX.:**

- v = vetor de inteiros com 10 elementos
- espaço de memória de v =
  10 x valores inteiros de 4 bytes =
  40 bytes

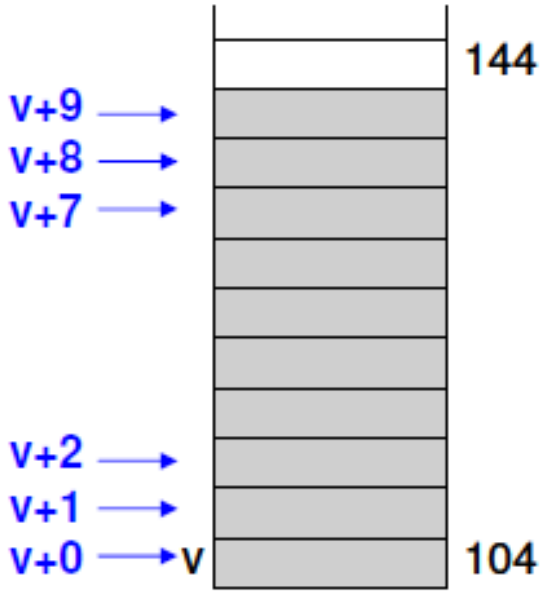


# Acesso em memória

Então, em C, os acessos funcionam da seguinte maneira:

- nome do vetor aponta para endereço inicial
- C permite aritmética de ponteiros
- Exemplo:
  - v+0 : primeiro elemento de v

    ...
  - v+9 : último elemento de v
- Exemplo:
  - &v[i] é equivalente a (v+i)
  - *(v+i) é equivalente a v[i]



# Vetores em C

Calculando média e variância.

Ex.:

- cálculo da média e da variância de um conjunto de 10 números reais

$$m = \frac{\sum x}{N}, \quad v = \frac{\sum (x - m)^2}{N}$$

- implementação
  - valores são lidos e armazenados em um vetor de 10 posições
  - cálculos da média e da variância efetuados sobre o conjunto de valores armazenado

# Vetores em C

```
/* Cálculo da média e da variância de 10 números reais */

#include <stdio.h>

int main ( void )
{
   float v[10];          /* declara vetor com 10 elementos */
   float med, var;       /* variáveis para a média e a variância */
   int i;                /* variável usada como índice do vetor */

   /* leitura dos valores */
   for (i = 0; i < 10; i++)      /* faz índice variar de 0 a 9 */
      scanf("%f", &v[i]);        /* lê cada elemento do vetor (digitados na mesma linha) */
```

# Vetores em C

```
/* cálculo da média */
   med = 0.0;                    /* inicializa média com zero */
   for (i = 0; i < 10; i++)
      med = med + v[i];          /* acumula soma dos elementos */
   med = med / 10;               /* calcula a média */

   /* cálculo da variância */
   var = 0.0;                    /* inicializa com zero */
   for ( i = 0; i < 10; i++ )
     var = var+(v[i]-med)*(v[i]-med);        /* acumula */
   var = var / 10;               /* calcula a variância */

   /* exibição do resultado */
   printf ( "Media = %f   Variancia = %f  \n", med, var );
   return 0;
}
```

comando não pertence ao corpo do “for”

# Vetores em C

int a, b[20]; /* declara uma variável simples e um vetor */
float c[10]; /* declara um vetor */
double d[30], e, f[5]; /* declara dois vetores e uma variável simples */

int v[5] = {12, 5, 34, 32, 9};  /*Declara e já inicializa*/

```
#include <stdio.h>

int main (void)
{
   int i;
   float v[6] = {2.3, 5.4, 1.0, 7.6, 8.8, 3.9};

   for (i=0; i<6; i++) {
      printf("%f", v[i]);
   }

   return 0;
}
```

# Somatório

```
#include <stdio.h>

int main (void)
{
    int i;
    float v[6] = {2.3, 5.4, 1.0, 7.6, 8.8, 3.9};
    float s = 0.0;

    for (i=0; i<6; i++) {
        s = s + v[i];
    }

    printf("%f", s);

    return 0;
}
```

# Máximo

```c
#include <stdio.h>

int main (void)
{
    int i;
    float v[6] = {2.3, 5.4, 1.0, 7.6, 8.8, 3.9};
    float maior_valor = v[0];

    for (i=0; i<6; i++) {
      if(v[i]>maior_valor)
        maior_valor = v[i];
    }

    printf("%f", maior_valor);

    return 0;
}
```

# Vetores Passados para Funções

- Além de passarmos variáveis simples como parâmetros, podemos passar **vetores** também:

– Quando passamos um vetor como parâmetro, a função chamada recebe uma **referência para o vetor**.

– Isso significa que a função chamada, quando acessa os elementos, acessa as mesmas posições de memória que a função que declarou o vetor.

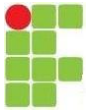
– Por essa razão, se atribuirmos um valor a um elemento do vetor passado como parâmetro, este elemento também é alterado no vetor original.

– Portanto, podemos declarar um vetor numa função e chamar uma outra função auxiliar para acessar e/ou modificar seus elementos.

# Exemplo

```
#include<stdio.h>
#define NUM_ALUNOS 6

void ler_dados (float vet[], int num) {
  int i;
  /* leitura dos dados para armazenar no vetor */
  for(i=0;i<num;i++) {
    printf("Entre com o valor %d: ", i+1);
    scanf("%f", &vet[i]);
  }
}

float calcula_media (float vet[], int num) {
  float soma = 0.0;
  int i;
  for(i=0;i<num;i++)
    soma = soma + vet[i];
  return soma/num;
}

int main (void) {
  float notas[NUM_ALUNOS];
  ler_dados(notas, NUM_ALUNOS);
  printf("Media da turma = %.2f\n.", calcula_media(notas, NUM_ALUNOS));
  return 0;
}
```

**Testem:**
**vetor[a]++**



# Buscas

```
int busca(int n, int[] vet, int elem) {
  int i;
  for(i = 0 ; i < n ; i++) {
    if(vet[i] == elem) {
      return i;
    }
  }
  return -1;
}
```

*Retorna o índice da primeira ocorrência do elemento procurado!*

*E se quiséssemos retornar a última ocorrência?...*

# Buscas

```
int busca(int n, int[] vet, int elem) {
  int i;
  int ind_elem = -1;
  for(i = 0 ; i < n ; i++) {
    if(vet[i] == elem) {
      ind_elem = i;
    }
  }
  return ind_elem;
}
```

*Percorre o vetor até o final procurando o elemento desejado. Se achar, guarda seu índice, se não a variável ind_elem já está com o valor -1 de retorno.*

# Matrizes

# Matrizes

Suponha que estamos trabalhando com **100 provas de 100 alunos**. Seria muito cansativo criar 100 vetores para as notas.

Para resolver esse problema podemos utilizar **matrizes**. Uma matriz ou vetor que possui **duas ou mais dimensões**, resolve o problema em questão.

Podemos criar uma única variável com duas dimensões, onde a primeira dimensão seria para os **100 alunos** e a segunda dimensão seria para as **100 notas** do aluno.

# Matrizes

Caracteriza-se por ser definida uma **única variável** vinculada a cada dimensão com um determinado **tamanho**.

A dimensão de uma matriz é constituída por **constantes inteiras e positivas**. Os nomes dados às matrizes seguem as mesmas regras de nomes utilizados para indicar as variáveis simples.

**Matriz com mais de uma dimensão(Matrizes)**

Exemplo:

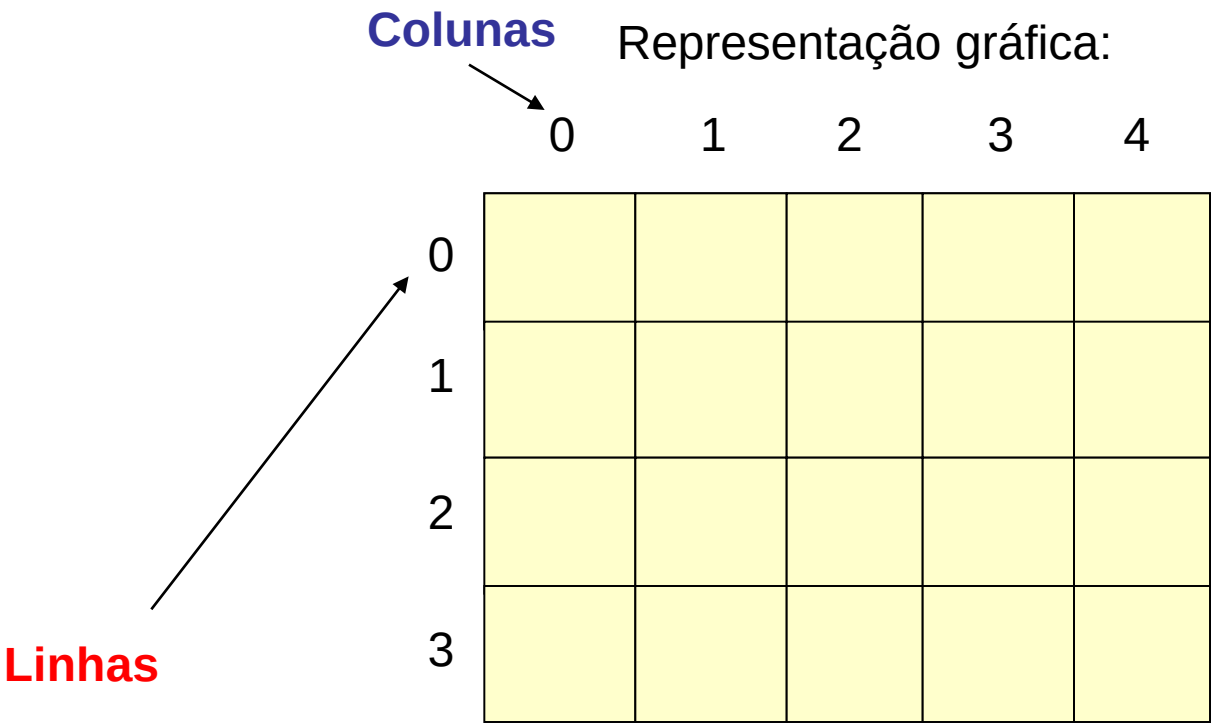
**M = [a1,1 a1,2 a1,3 … a1,n]**

**[a2,1 a2,2 a2,3 … a2,n].**

# Matrizes

- Os elementos da matriz são guardados em posições consecutivas de memória. As posições representam linhas e colunas.

**Ex**.: int matriz[3][4]



# Matrizes e C

- A definição de uma matriz em C se dá pela sintaxe:

tipo_do_dado nome_da_matriz[ quantidade_linhas ] [ quantidade_colunas ]

- A definição de uma matriz de várias dimensões em C se dá pela sintaxe:

tipo_do_dado nome_da_matriz[tamanho_dimensão_1] [tamanho_dimensão_2] [tamanho_dimensão_3] ... [tamanho_dimensão_n]

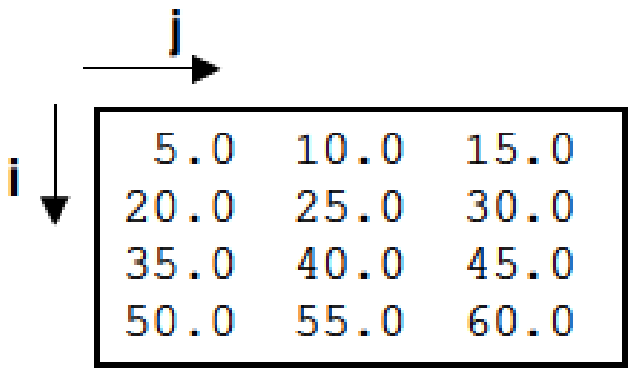
# Matrizes e C

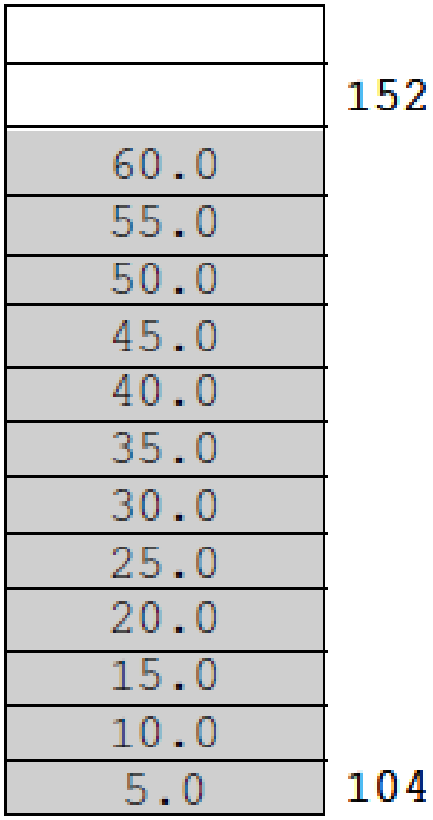
```
float notas[2][5];

float notas[3][5] = { {8.0, 7.5, 8.5, 9.0, 8.0 },
                      {8.9, 9.0, 8.6, 8.4, 8.0 },
                      {6.8, 7.1, 7.0, 7.6, 6.5 } };
```

# Matrizes e C

- Vetor bidimensional (ou matriz):

```
float m[4][3] = {{ 5.0,10.0,15.0},
                 {20.0,25.0,30.0},
                 {35.0,40.0,45.0},
                 {50.0,55.0,60.0}};
```

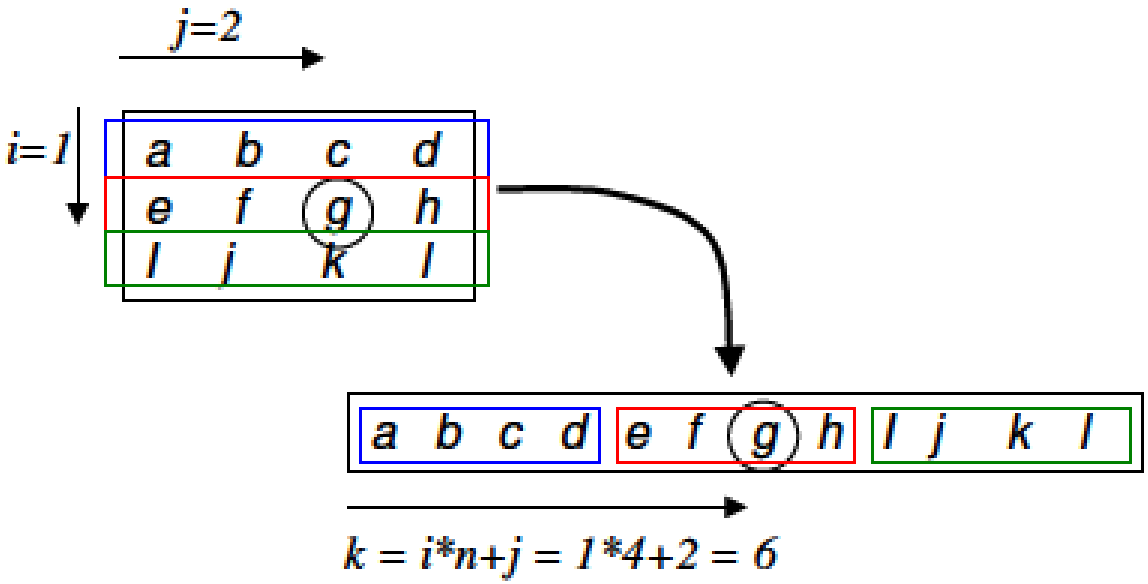


# Matrizes em um vetor

## Matrizes Representadas por um vetor simples:

- matriz mat com n colunas representada no vetor v:
  - mat[ i ][ j ] mapeado em v[ k ] onde k = i * n + j



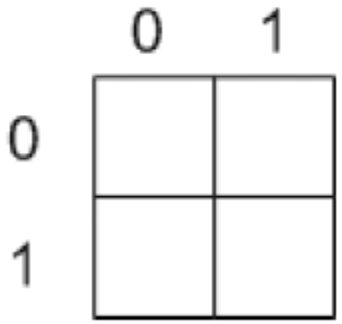
# Acesso em memória

*Posij = endereço inicial + ((i-1) * C * tamanho do tipo do elemento) + ((j-1) * tamanho do tipo do elemento)*.

*C* a quantidade de colunas por linhas.

i → Linha
j → Coluna

Matriz M
i/j



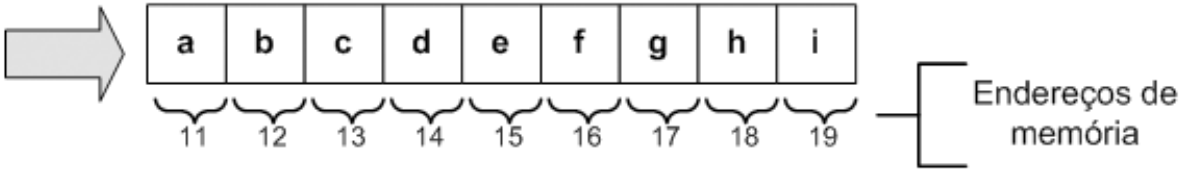

```
1 begin
2 |   M00 ⟵ endereço Inicial
3 |   M01 ⟵ endereço Inicial + 1 * tamanho Elemento
4 |   M10 ⟵ endereço Inicial + i * C * tamanho Elemento
5 |   M11 ⟵ endereço Inicial + i * C * tamanho Elemento + j * tamanho Elemento
6 end
```

Cálculo da posição de índices de uma matriz na memória

# Matrizes

$$\begin{vmatrix} a_{11} & b_{12} & c_{13} \\ d_{21} & e_{22} & f_{13} \\ g_{31} & h_{32} & i_{33} \end{vmatrix}$$

Representação na memória

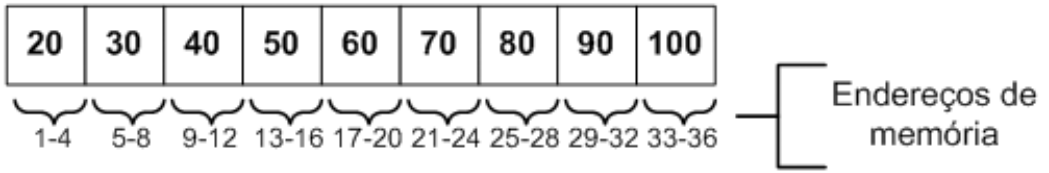



Posição de $h_{32}$ = 11 + ((3-1) * 3 * 1) + ((2-1)*1) = 18

C = 3 e Tamanho = 1 (caracter)

$$\begin{vmatrix} 20_{11} & 30_{12} & 40_{13} \\ 50_{21} & 60_{22} & 70_{13} \\ 80_{31} & 90_{32} & 100_{33} \end{vmatrix}$$

Representação na memória

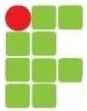



Posição de $80_{31}$ = 1 + ((3-1) * 3 * 4) + ((1-1)*4) = 25

C = 3 e Tamanho = 4 (inteiro)

# Operações

```c
#include<stdio.h>
#include<conio.h>
int main (void )
{
  int matriz[2][2],i, j;

  printf ("\nDigite valor para os elementos da matriz\n\n");

  for ( i=0; i<3; i++ )
    for ( j=0; j<3; j++ )
    {
       printf ("\nElemento[%d][%d] = ", i, j);
       scanf ("%d", &matriz[ i ][ j ]);
    }

   printf("\n\n******************* Saida de Dados ********************** \n\n");

   for ( i=0; i<3; i++ )
    for ( j=0; j<3; j++ )
    {
       printf ("\nElemento[%d][%d] = %d\n", i, j,matriz[ i ][ j ]);
    }

  getch();
  return(0);
}
```



# Operações

## Ex.: Matriz Transposta:

- entrada: mat matriz de dimensão m x n
- saída: trp transposta de mat, alocada dinamicamente
  - Q é a *matriz transposta* de M se e somente se *Qij = Mji*

```
/* Solução 1: matriz alocada como vetor simples */
float* transposta (int m, int n, float* mat)
{
  int i, j;
  float* trp;

  /* aloca matriz transposta com n linhas e m colunas */
  trp = (float*) malloc(n*m*sizeof(float));

  /* preenche matriz */
  for (i=0; i<m; i++)
    for (j=0; j<n; j++)
      trp[ j*m+i ] = mat[ i*n+j ];

  return trp;
}
```

# Referência

Waldemar Celes, Renato Cerqueira, José Lucas Rangel, *Introdução a Estruturas de Dados*, Editora Campus (2016) 2ª Edição.