

Estruturas de Dados I

2-Vetores, Matrizes e Alocação Dinâmica

Lidio Mauro Lima de Campos
lidio@ufpa.br



Universidade Federal do Pará – UFPA
Faculdade de Computação

Referências

- Waldemar Celes, Renato Cerqueira, José Lucas Rangel, *Introdução a Estruturas de Dados*, Editora Campus (2004)
- Kernigan, B.W. Ritchie, D.M. A Linguagem de Programação C. Campus.
- Ascencio, Ana Fernades G., de Araújo, Graziela Santos. Estruturas de Dados. Prentice Hall, 2010.
- GRAHAM, N., *Learning C++* . McGRAW-HILL, 1991

Tópicos

- Vetores
- Alocação dinâmica
- Vetores locais e funções
- Matrizes e Alocação Dinâmica

Vetor

- Estrutura de dados definindo um conjunto enumerável
- Exemplo:
- v = vetor de inteiros com 10 elementos
 - `int v[10];`

Vetor

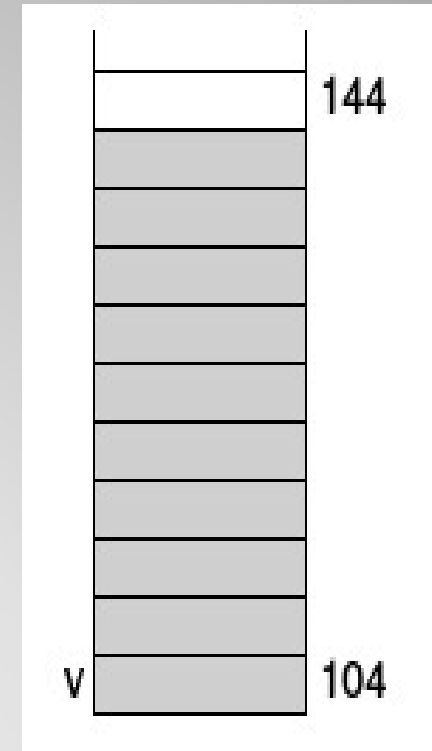
- vetor pode ser inicializado
- Exemplo:
 - `v = vetor de inteiros com 10 elementos`
 - `int v[5] = { 5, 10, 15, 20, 25 };`
 - ou simplesmente:
 - `int v[] = { 5, 10, 15, 20, 25 };`

Vetor

- acesso a cada elemento é feito através de indexação da variável
- Exemplo:
 - `v = vetor de inteiros com 10 elementos`
 - `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) */`

Vetor

- vetor é alocado em posições contíguas de memória
- Exemplo:
- v = vetor de inteiros com 10 elementos
 - espaço de memória de v =
 $10 \times \text{valores inteiros de 4 bytes} = 40 \text{ bytes}$



Vetor

- Exemplo:
 - 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

Vetor


```
/* 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) */
}
```

endereço da i-ésima
posição de v



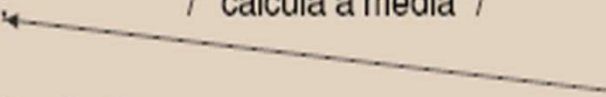
Vetor

```
/* 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"



Vetor

- Passagem de vetor para função:
 - Consiste em passar o endereço da primeira posição do vetor.
 - função deve ter parâmetro do tipo ponteiro para armazenar valor.
 - Elementos do vetor não são copiados para a função
 - Exemplo: c
 - Chamada a função passando vetor de int
 - função deve ter um parâmetro do tipo int

Vetor

```
/* Cálculo da média e da variância de 10 reais (segunda versão) */
```

```
#include <stdio.h>
```

```
/* Função para cálculo da média */
```

```
float media (int n, float* v)
```

```
{
```

```
    int i;
```

```
    float s = 0.0;
```


```
    for (i = 0; i < n; i++)
```

```
        s += v[i];
```

```
    return s/n;
```

```
}
```

parâmetro do tipo
ponteiro para float



Vetor

```
/* Função para cálculo da variância */  
float variancia (int n, float* v, float m)  
{  
    int i;  
    float s = 0.0;  
    for (i = 0; i < n; i++)  
        s += (v[i] - m) * (v[i] - m);  
    return s/n;  
}
```

Vetor

```
int main ( void )
{
    float v[10];
    float med, var;
    int i;

    /* leitura dos valores */
    for ( i = 0; i < 10; i++ )
        scanf("%f", &v[i]);

    med = media(10,v);
    var = variancia(10,v,med);

    printf ( "Media = %f  Variancia = %f \n", med, var);
    return 0;
}
```

Vetor

- Passagem de vetor para função (cont.):
 - função pode alterar os valores dos elementos do vetor pois
 - recebe o endereço do primeiro elemento do vetor

(e não os elementos propriamente ditos)

– Exemplo:

- Função incrementando todos os elementos de uma unidade

Vetor

```
/* Incrementa elementos de um vetor */
#include <stdio.h>
void incr_vetor ( int n, int *v )
{
    int i;
    for (i = 0; i < n; i++)
        v[i]++;
}

int main ( void )
{
    int a[ ] = {1, 3, 5};
    incr_vetor(3, a);
    printf("%d %d %d \n", a[0], a[1], a[2]);
    return 0;
}
```

saída do programa será 2 4 6

Vetor

- Exercícios Fixação
- 1) Faça um programa que leia 10 números inteiros, coloque-os num vetor e mostre-os em ordem inversa.
- 2) Faça um programa que leia uma lista de 20 números, colocando-os em um vetor e após o termino da leitura , mostre os elementos com índice maior ou igual a 10.
- 3)Faça um programa que leia uma lista de 100 números INTEIROS E MOSTRE os números impares.

Vetor

1)

```
#include<stdio.h>
```

```
#include<stdlib.h>
```

```
int main(void)
```

```
{
```

```
    int i,j;
```

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

```
    for(i=0;i<10;i++)
```

```
        {scanf("%d",&v[i]);}
```

```
    for(i=9;i>=0;i--)
```

```
        {printf("%d",v[i]);}
```

```
        return 0;
```

```
}
```

Vetor

2)

```
#include<stdio.h>
```

```
#include<stdlib.h>
```

```
int main(void)
```

```
{
```

```
    int i,j;
```

```
    int v[20];
```

```
    for(i=0;i<20;i++)
```

```
        {scanf("%d",&v[i]);}
```

```
    for(i=10;i<20;i++)
```

```
        {printf("%d",v[i]);}
```

```
    return 0;
```

```
}
```

Vetor

```
3) #include<stdio.h>
#include<stdlib.h>
```

```
int main(void)
{
    int i; int v[20] ; int temp;

    for(i=0;i<10;i++)
    { scanf("%d",&v[i]); }

    for(i=0;i<10;i++)
    {temp=v[i];
      if(temp%2==1)
      {
          printf("%d",temp);
      }
    }

    return 0;
}
```

Vetor

4) Faça um programa que lê dois vetores de 10 elementos e calcula a soma e armazena num terceiro vetor.

5) Faça um programa que lê um vetores de 100 elementos e calcula o fatorial de cada elemento do vetor e armazena esses valores em outro vetor usar funções e recursividade para o calculo do fatorial.

6) Faça um programa que lê um vetores de 20 elementos correspondente a receita na venda de um produto e tab. lê as despesas de produção e venda, ao final calcula-se o lucro obtido na venda dado por $LUCRO = RECEITA - DESPESA$ IMPRIMIR O lucro obtido.

Vetor

- Uso da memória:
 - alocação dinâmica:
 - Espaço de memória é requisitada em tempo de execução.
 - Espaço permanece reservado até que seja explicitamente liberado.
 - depois de liberado, espaço estará disponibilizado para outros usos e não pode mais ser acessado.
 - espaço alocado e não liberado explicitamente, será automaticamente liberado quando ao final da execução.

Vetor

Alocação Dinâmica

- Uso da memória:
 - **uso de variáveis globais e estáticas:**
 - Espaço reservado para uma variável global existe enquanto o programa estiver sendo executado.
 - **uso de variáveis locais:**
 - Espaço existe apenas enquanto a função que declarou a variável está sendo executada.
 - Liberado para outros usos quando a execução da função termina.
 - **variáveis globais ou locais podem ser simples ou vetores:** para vetor, é necessário informar o número máximo de elementos pois o compilador precisa calcular o espaço a ser reservado.

Vetor

Alocação Dinâmica

- Uso da memória:
 - **memória estática:**
 - código do programa
 - variáveis globais
 - variáveis estáticas
 - **memória dinâmica:**
 - variáveis alocadas dinamicamente
 - **memória livre**
 - variáveis locais

memória estática	Código do programa
	Variáveis globais e Variáveis estáticas
memória dinâmica	Variáveis alocadas dinamicamente
	Memória livre
	Variáveis locais (Pilha de execução)

Vetor

- **Alocação Dinâmica**

- **Função "sizeof":**

- retorna o número de bytes ocupado por um tipo

- **Função "malloc":**

- recebe como parâmetro o número de bytes que se deseja alocar.

- retorna um ponteiro genérico para o endereço inicial da área de memória alocada, se houver espaço livre:

- ponteiro genérico é representado por void*

- ponteiro é convertido automaticamente para o tipo apropriado.

- retorna um endereço nulo, se não houver espaço livre:

- representado pelo símbolo NULL.

Vetor

- **Alocação Dinâmica**

- **Exemplo:**

- alocação dinâmica de um vetor de inteiros com 10 elementos.

- malloc retorna um ponteiro genérico, representado por `void*`, o mesmo é convertido explicitamente, para o tipo apropriado na atribuição.
 - ponteiro de inteiro recebe endereço inicial do espaço alocado para armazenar o vetor.

- **`int *v;`**

- **`v = (int *) malloc(10*sizeof(int));`**

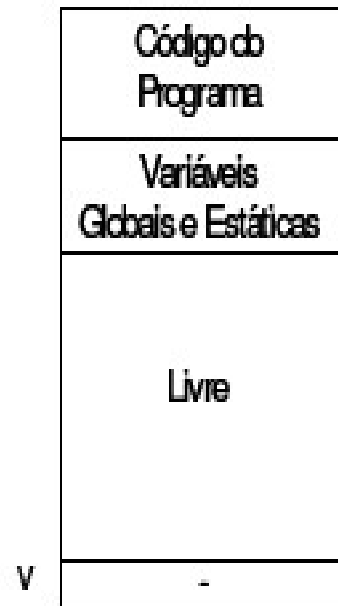
Vetor

Exemplo (cont.):

```
v = (int *) malloc(10*sizeof(int));
```

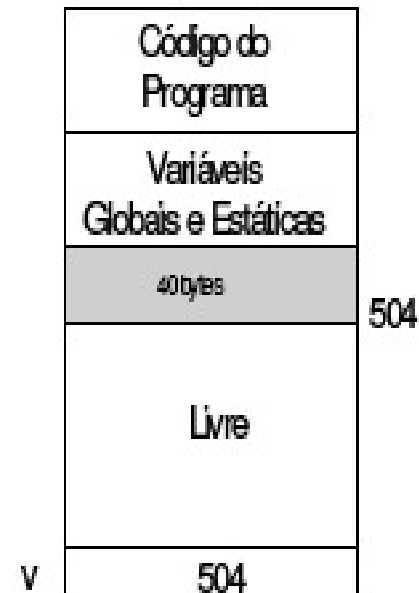
1 - Declaração: `int *v`

Abre-se espaço na pilha para o ponteiro (variável local)



2 - Comando: `v = (int *) malloc(10*sizeof(int))`

Reserva espaço de memória da área livre e atribui endereço à variável



Vetor

- Exemplo (cont.):
 - tratamento de erro após chamada a malloc imprime mensagem de erro.
 - aborta o programa (com a função exit).

```
...  
v = (int*) malloc(10*sizeof(int));  
if (v==NULL)  
{  
    printf("Memoria insuficiente.\n");  
    exit(1); /* aborta o programa e retorna 1 para o sist. operacional */  
}  
...
```

Vetor

- **Alocação Dinâmica**
- Função "free":
 - recebe como parâmetro o ponteiro da memória a ser liberada.
 - a função free deve receber um endereço de memória que tenha sido alocado dinamicamente.

```
free (v);
```

Vetor

```
/* Cálculo da média e da variância de n reais */
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
...
```

```
int main ( void )
```

```
{
```

```
    int i, n;
```

```
    float *v;
```

```
    float med, var;
```

```
    /* leitura do número de valores */
```

```
    scanf("%d", &n);
```

```
    /* alocação dinâmica */
```

```
    v = (float*) malloc(n*sizeof(float));
```

```
    if (v==NULL) {
```

```
        printf("Memoria insuficiente.\n");
```

```
        return 1;
```

```
    }
```

Vetor

```
/* leitura dos valores */  
for (i = 0; i < n; i++)  
    scanf("%f", &v[i]);  
med = media(n,v);  
var = variancia(n,v,med);  
printf("Media = %f  Variancia = %f \n", med, var);  
/* libera memória */  
free(v);  
return 0;  
}
```

Matrizes

- Alocação estática versus dinâmica.
- Vetores bidimensionais – matrizes.
- Matrizes dinâmicas.
- Operações com matrizes.

Matrizes

- **Alocação Estática Versus Dinâmica**

- **Alocação estática de vetor:**

- é necessário saber de antemão a dimensão máxima do vetor.
 - variável que representa o vetor armazena o endereço ocupado.
 - vetor declarado dentro do corpo de uma função não pode ser usado fora do corpo da função.

- `#define N 10`
 - `int v[N];`

Matrizes

- **Alocação Dinâmica**

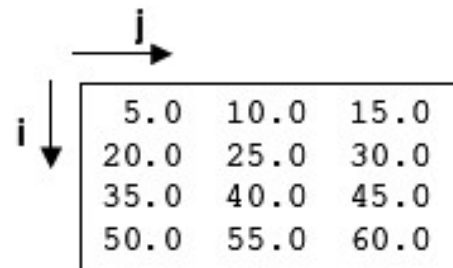
- dimensão do vetor pode ser definida em tempo de execução.
- variável do tipo ponteiro recebe o valor do endereço do primeiro elemento do vetor.
- área de memória ocupada pelo vetor permanece válida até que seja explicitamente liberada (através da função `free`).
- vetor alocado dentro do corpo de uma função pode ser usado fora do corpo da função, enquanto estiver alocado

- `int* v;`
- `...`
- `v = (int*) malloc(n * sizeof(int));`

Matrizes

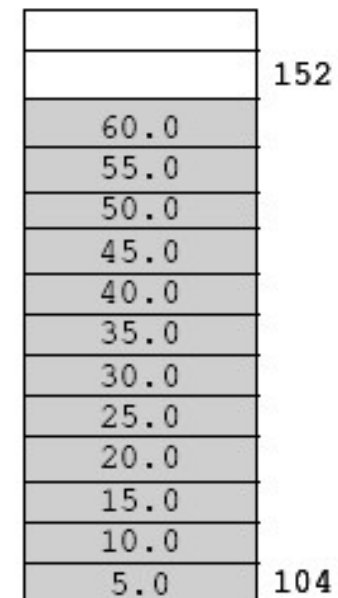
- Vetor bidimensional (ou matriz):

```
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}};
```



A diagram showing a 4x3 matrix. A horizontal arrow labeled 'j' points to the right above the matrix, and a vertical arrow labeled 'i' points downwards to the left of the matrix.

5.0	10.0	15.0
20.0	25.0	30.0
35.0	40.0	45.0
50.0	55.0	60.0



A diagram showing a 1D array representing the 2D array in memory. The array contains 12 elements, with the first two empty and the last one highlighted. The values are listed in descending order from top to bottom.

60.0
55.0
50.0
45.0
40.0
35.0
30.0
25.0
20.0
15.0
10.0
5.0

152

104

Matrizes

- **Vetor bidimensional (ou matriz):**

- declarado estaticamente.
- elementos acessados com indexação dupla $m[i][j]$
 - i acessa a linha e j acessa a coluna.
 - indexação começa em zero:
 - $m[0][0]$ é o elemento da primeira linha e primeira coluna
- variável representa um ponteiro para o primeiro “vetor-linha”
 - matriz também pode ser inicializada na declaração

Matrizes

declaração da matriz na função principal:

```
float mat[4][3]
```

protótipo da função (opção 1): parâmetro declarado como "vetor-linha"

```
void f (... , float (*mat)[3], ...);
```

protótipo da função (opção 2): parâmetro declarado como matriz, omitindo o número de linhas

```
void f (... , float mat[][3], ...);
```

Matrizes

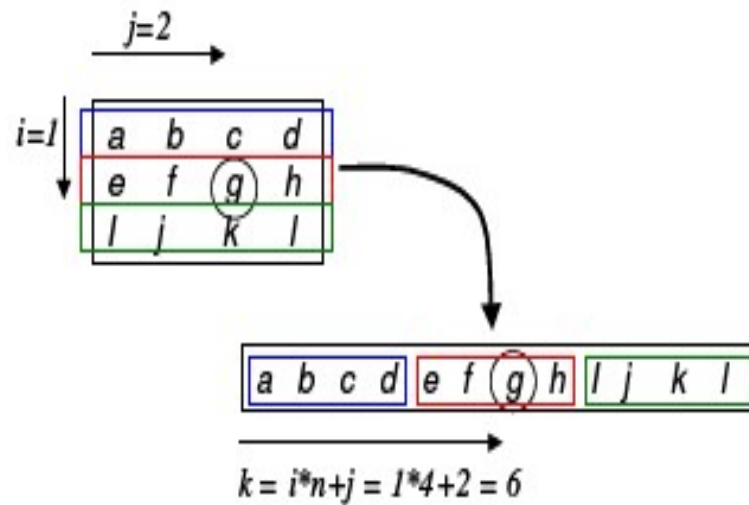
- **Limitações:**
 - **alocação estática de matriz:**
 - é necessário saber de antemão suas dimensões
 - **alocação dinâmica de matriz:**
 - C só permite alocação dinâmica de conjuntos unidimensionais
 - é necessário criar abstrações conceituais com vetores para representar matrizes (alocadas dinamicamente)

Matrizes

- **Matrizes Alocação Dinâmica**
- **Matriz representada por um vetor simples:**
 - conjunto bidimensional representado em vetor unidimensional.
 - estratégia:
 - primeiras posições do vetor armazenam elementos da primeira linha.
 - seguidos dos elementos da segunda linha, e assim por diante.
 - exige disciplina para acessar os elementos da matriz.

Matrizes

- Matriz representada por um vetor simples (cont.):
 - matriz **mat** com **n** colunas representada no vetor **v**:
 - **mat[i][j]** mapeado em **v[k]** onde $k = i * n + j$



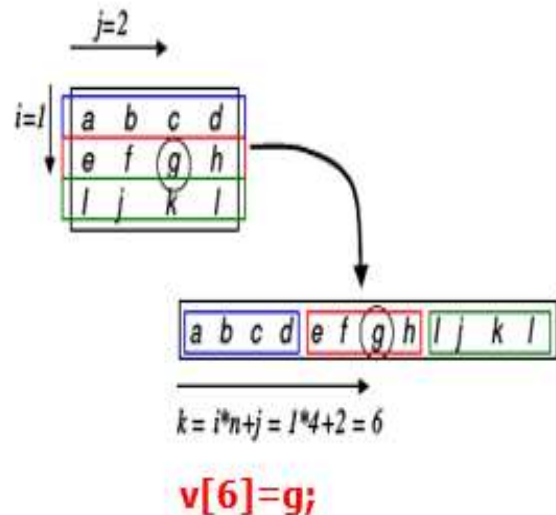
$v[6] = g;$

Matrizes

- Matriz representada por um vetor simples (cont.):
 - `mat[i][j]` mapeado em `v[i * n + j]`

```
float *mat;    /* matriz m x n representada por um vetor */  
...  
mat = (float*) malloc(m*n*sizeof(float));
```

Matrizes



Representação por Vetor Simples

1) ALOCAÇÃO

```
float* mat;  
mat=(float*)malloc(n*m*sizeof(float));
```

2) PREENCHIMENTO

```
for(i=0;i<m;i++)  
    for(j=0;j<n;j++)  
        scanf("%f",&mat[i*n+j]);
```

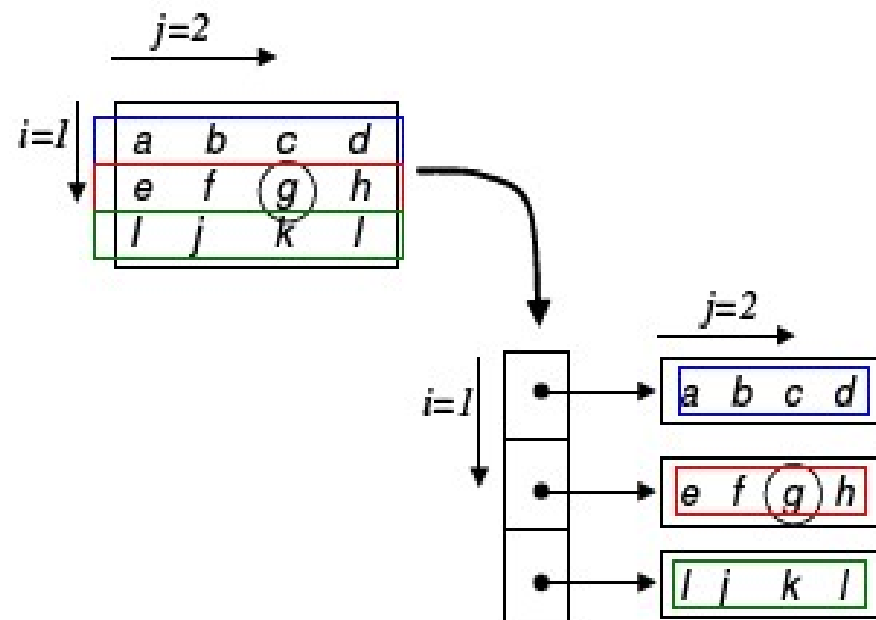
3) LIBERAR ESPAÇO DE MEMÓRIA

```
free(mat);
```

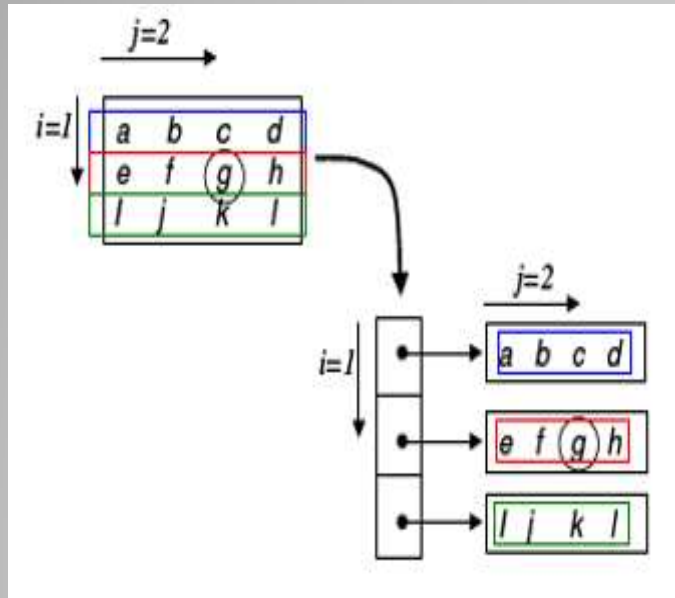
n = número de colunas da matriz

Matrizes

- Matriz representada por um vetor de ponteiros:
 - cada elemento do vetor armazena o endereço do primeiro elemento de cada linha da matriz



Matrizes



Representação por Vetor de Ponteiros

1) ALOCAÇÃO

```
float** mat;  
mat=(float**)malloc(m*sizeof(float*));  
for(i=0;i<n;i++)  
{ mat[i]=(float*)malloc(n*sizeof(float));}
```

2) PREENCHIMENTO

```
for(i=0;i<m;i++)  
for(j=0;j<n;j++)  
scanf("%f",&mat[i][j]);
```

3) LIBERAR ESPAÇO DE MEMÓRIA

```
for(i=0;i<m;i++)  
free(mat[i]);  
free(mat);
```

m=número de linhas

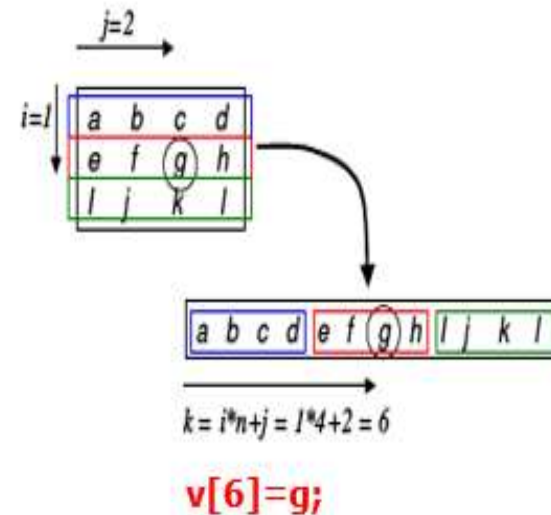
n = número de colunas

Matrizes

- Exemplo – função `transposta`:
 - entrada: `mat` matriz de dimensão $m \times n$
 - saída: `trp` transposta de `mat`, alocada dinamicamente
 - Q é a *matriz transposta* de M se e somente se $Q_{ij} = M_{ji}$
- Solução 1: matriz alocada como vetor simples
- Solução 2: matriz alocada como vetor de ponteiros

Matrizes

```
/* 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;  
}
```



Matrizes

```
/* Solução 2: matriz alocada como vetor de ponteiros */
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*sizeof(float*));
    for (i=0; i<n; i++)
        trp[i] = (float*) malloc(m*sizeof(float));

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

    return trp;
}
```

