

# Estruturas de Dados

## Matrizes

# Matrizes

- Conjuntos bidimensionais declarados estaticamente
  - `float mat[4][3];`
- Declaração de um vetor (estática ou dinâmica?)
  - `int v[10]`
  - `int *v;`
  - `v = (int*) malloc (n*sizeof(int));`
  - `v = (int*) realloc (v, m*sizeof(int));`

# Vetor – declaração estática



- `int v[10]`
- Precisamos saber o tamanho do vetor antes da execução do programa
- `v` armazena o endereço de memória ocupado pelo primeiro elemento do vetor
  - `*v` e `v[0]`
  - `*(v+1)` e `v[1]`
- Escopo de declaração local
  - Se o vetor for declarado dentro da função, não pode ser acessado fora da função

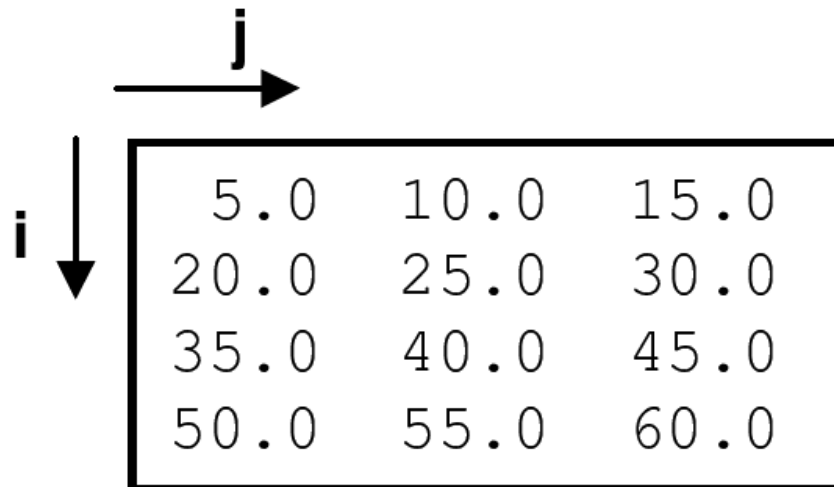
## Vetor – declaração dinâmica



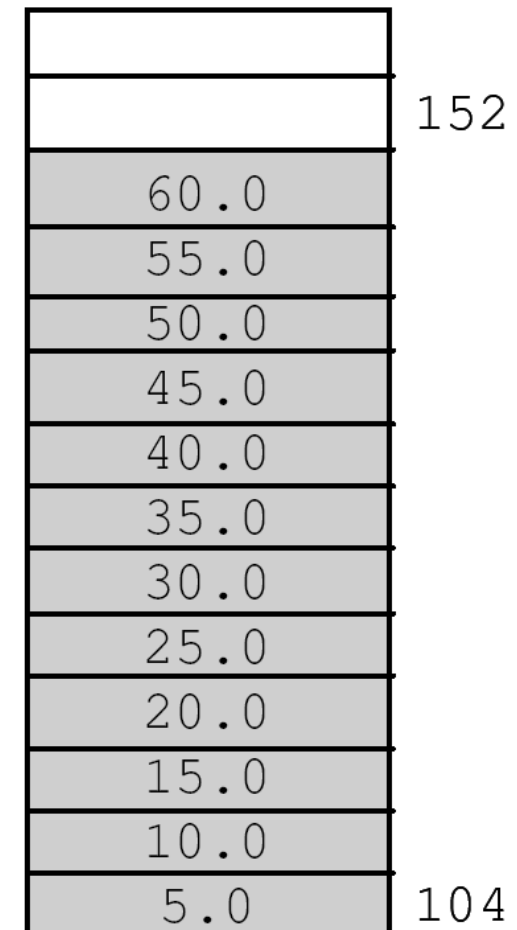
- `int *v;`
- `v = (int*) malloc (n*sizeof(int));`
- Tamanho do vetor pode ser definido em tempo de execução do programa
- Variável ponteiro aponta para a primeira posição do vetor
- Área ocupada pelo vetor permanece fora das funções até que seja liberada explicitamente por `free()`

# Matrizes

```
float mat[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}};
```



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



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

## Vetores bidimensionais (matrizes)

- Elementos acessados pela indexação `m[i][j]`
  - `i` acessa linha e `j` coluna
- Elemento inicial `m[0][0]`
- “`m`” representa um ponteiro para o primeiro “vetor-linha”, composto por 3 elementos.
- Pode ser inicializada na declaração:
  - `float mat [4][3] = {1,2,3,4,5,6,7,8,9,10,11,12};`
  - `float mat[][3] = {1,2,3,4,5,6,7,8,9,10,11,12};`

# Matrizes Dinâmicas

- Conjuntos bidimensionais não podem ser alocados dinamicamente no C
- Abstrações conceituais com vetores são necessárias para alocarmos matrizes dinamicamente

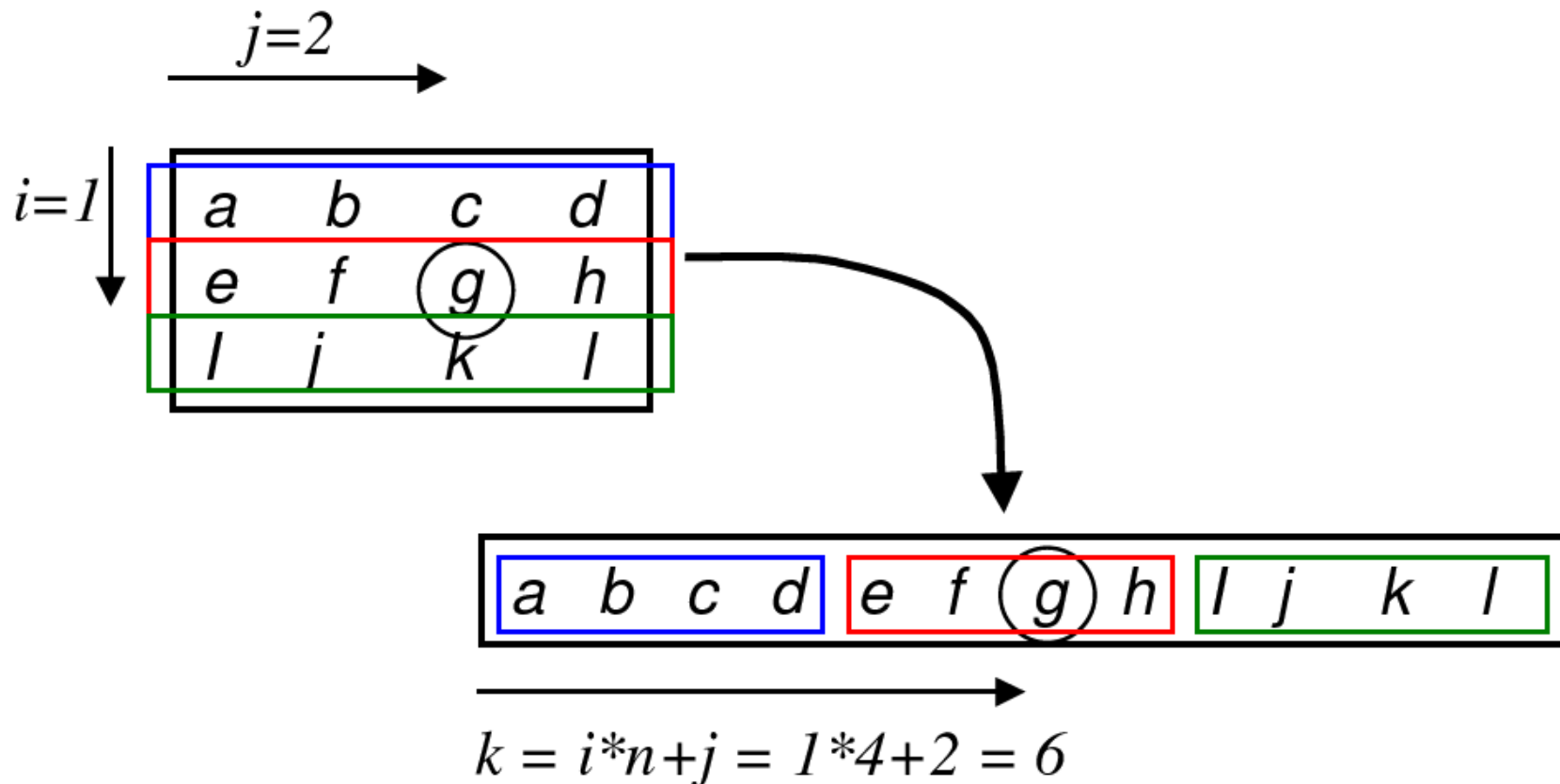
# Matrizes representadas por vetor simples

- Matriz é representada por um vetor unidimensional
  - Primeiras posições do vetor armazenam os elementos da primeira linha
  - As posições seguintes armazenam da segunda linha, e assim por diante
- Conceitualmente, estamos trabalhando com uma matriz
- Concretamente, estamos representando um vetor unidimensional
- Exige disciplina para acessar os elementos



# Matrizes representadas por vetor simples

- `mat [i][j]`
  - `V[k]`, com  $k = i*n+j$ , onde  $n$  é o número de colunas da matriz



## Matriz representada por vetor simples

- `mat [i][j]` mapeado para `v[i*n + j]`
- `m` e `n` são as dimensões da matriz (de tamanho `m*n` elementos)

```
float *mat;
```

```
...
```

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

```
...
```

# Operações com Matrizes

- Exemplo: função transposta
  - Dada uma matriz, cria dinamicamente a matriz transposta
- Matriz de entrada: mat (m x n)
- Matriz de saída: trp
- Uma matriz Q é a matriz transposta de M, se  $Q_{ij} = M_{ji}$

```
float* transposta (int m, int n, float* mat);
```

## Exemplo – matriz com vetor simples

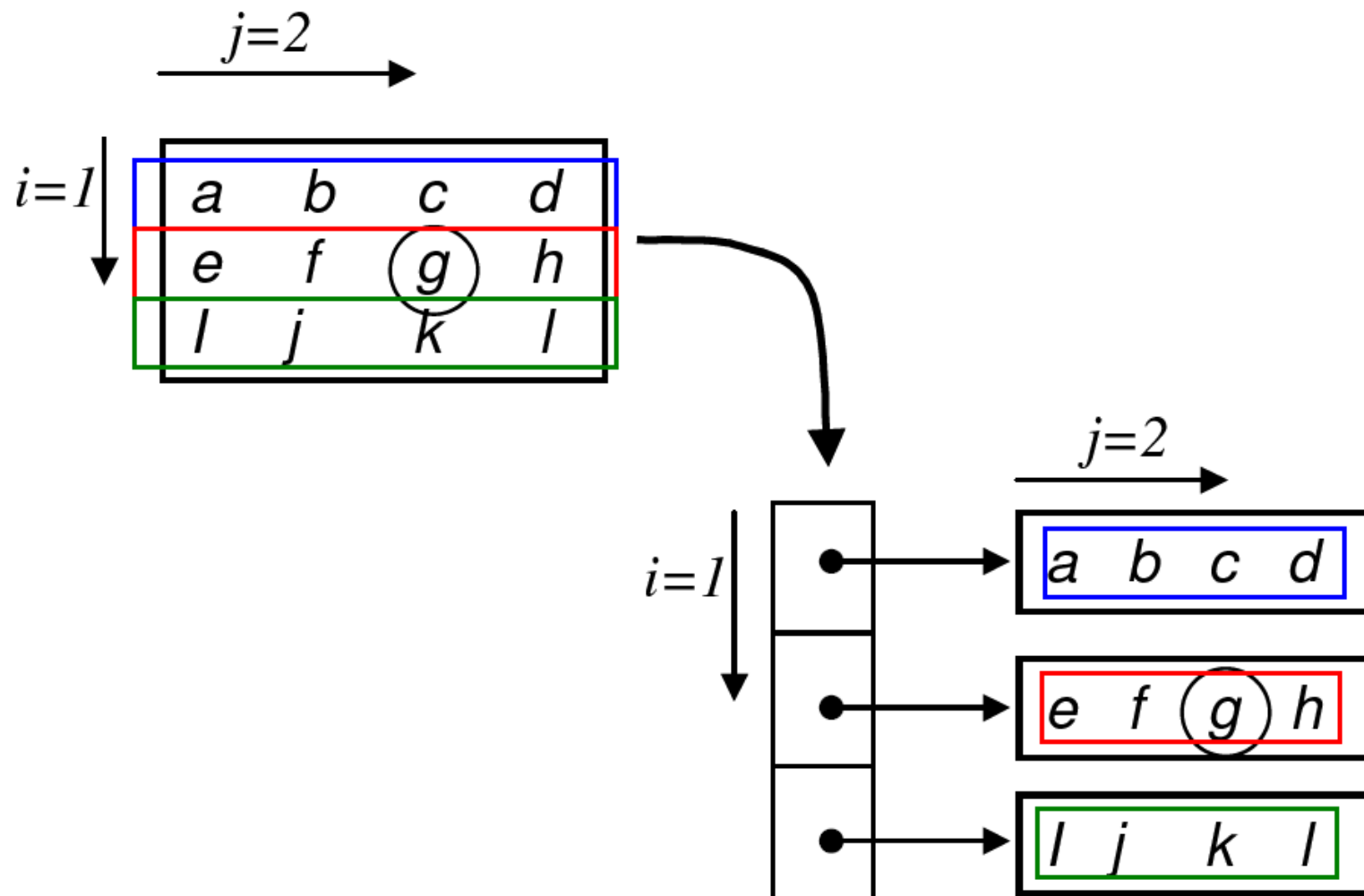
```
float* transposta (int m, int n, float* mat);
{ int i, j;
  float* trp;
  trp = (float*) malloc (n*m*sizeof (float));
  for (i=0; i<m; i++)
    for (j=0; j<n; j++)
      trp[j*m+i] = mat [i*n+j];
  return trp;
}
```

## Matriz representada por vetor de ponteiros



- Cada linha da matriz é representada por um vetor separado
- A matriz é representada por um vetor de vetores
  - Vetor de ponteiros
  - Cada elemento do vetor armazena o endereço de memória do primeiro elemento de cada linha

# Matriz representada por vetor de ponteiros



## Matriz representada por vetor de ponteiros

- Precisamos alocar memória para o vetor de ponteiros e atribuir os endereços das linhas da matriz

```
int i;
float **mat; /*vetor de ponteiros*/
...
mat = (float**)malloc (m*sizeof(float*));
for (i=0; i<m; i++)
    mat[i] = (float*) malloc (n*sizeof(float));
```

## Matriz representada por vetor de ponteiros



- Para liberar o espaço de memória alocado

...

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

```
    free (mat[i]);
```

```
free (mat);
```



# Operações com Matrizes

- Exemplo: função transposta
  - Dada uma matriz, cria dinamicamente a matriz transposta
- Matriz de entrada: mat (m x n)
- Matriz de saída: trp
- Uma matriz Q é a matriz transposta de M, se  $Q_{ij} = M_{ji}$

`float* transposta (int m, int n, float* mat);`

`float** transposta (int m, int n, float** mat);`

## Exemplo – matriz com vetor de ponteiros

```
float** transposta (int m, int n, float** mat);
{
    int i, j;
    float** trp;
    trp = (float**) malloc (n*sizeof (float*));
    for (i=0; i<n; i++)
        trp[i] = (float*) malloc(m*sizeof(float));
    for (i=0; i<m; i++)
        for (j=0; j<n; j++)
            trp[j][i] = mat [i][j];
    return trp;
}
```

## Resumindo

- Matriz representada por vetor bidimensional estático:
  - elementos acessados com indexação dupla  
`m[i][j]`
- Matriz representada por um vetor simples:
  - conjunto bidimensional representado em vetor unidimensional
- Matriz representada por um vetor de ponteiros:
  - cada elemento do vetor armazena o endereço do primeiro elemento de cada linha da matriz

## Exercício 1

- Implemente a função multiplicação de matriz usando a abordagem de alocação dinâmica de matrizes (vetor de ponteiros)
- Multiplicação de matrizes:
  - entrada:
    - matriz A de dimensão m x p
    - matriz B de dimensão p x n
  - saída: matriz M de dimensão m x n, definida como:

$$M_{i,j} = \sum_{k=1}^p A_{i,k} \times B_{k,j}$$

para i = 0 até m - 1, de 1 em 1

para j = 0 até n - 1, de 1 em 1

M[ i , j ] = 0

para k = 0 até p - 1, de 1 em 1

M[ i , j ] = M[ i , j ] + A[ i , k ] \* B[ k , j ]

# Resposta

```
/* Multiplicação de Matrizes (representadas por vetor de ponteiros) */
void mult ( int m, int p, int n, int** A, int** B, int** M)
{ int i, j, k, t;
  for (i = 0; i<m; i++)
    for (j = 0; j<n; j++)
    {
      t = 0;
      for (k = 0; k<p; k++)
        t = t + A[ i ][ k ]*B[ k ][ j ];
      M[ i ][ j ] = t;
    }
}
```

# Resposta

```

/* Alocação Dinâmica das Matrizes */
int main ( void )
{
    int **A, **B, **M;
    int m = 4, p = 3, n = 5;
    int i,j;
    /* Alocação das matrizes */
    A = (int**) malloc(m*sizeof(int*));
    for (i=0; i<m; i++)
        A[i] = (int*) malloc(p*sizeof(int));
    B = (int**) malloc(p*sizeof(int*));
    for (i=0; i<p; i++)
        B[i] = (int*) malloc(n*sizeof(int));
    M = (int**) malloc(m*sizeof(int*));
    for (i=0; i<m; i++)
        M[i] = (int*) malloc(n*sizeof(int));
    ...
    mult(m,p,n,A,B,M);
    ...
    /* inicialização das matrizes */
    /* multiplicação das matrizes */
    /* etc... */

```

## Exercício 2

- Implemente a função multiplicação usando a abordagem de alocação dinâmica de matrizes (representada por vetor simples)