matrizes

vetores bidimensionais - matrizes

- a linguagem C permite a criação de vetores bidimensionais, declarados estaticamente
  - para declarar uma matriz de valores reais com 4 linhas e 3 colunas: `float mat [4] [3];`
  - essa declaração reserva um espaço de memória necessário para armazenar os 12 elementos da matriz, que são organizados de maneira contínua, organizados linha a linha
- os elementos da matriz são acessados com indexação dupla: `mat [i] [j]` (i → linha, j → coluna)
  - o elemento da primeira linha e primeira coluna é acessado por `mat [0] [0]`
- as matrizes também podem ser inicializadas na declaração:
  - `float mat [4] [3] = { {1,2,3}, {4,5,6}, {7,8,9}, {10,11,12} }`
- ou podemos iniciar sequencialmente:
  - `float mat [4] [3] = {1,2,3,4,5,6,7,8,9,10,11,12}`
- o número de elementos por linha pode ser omitido numa inicialização, mas o número de colunas deve ser sempre fornecido.
  - `float mat [ ] [3] = {1,2,3,4,5,6,7,8,9,10,11,12}`

passagem de matrizes para funções

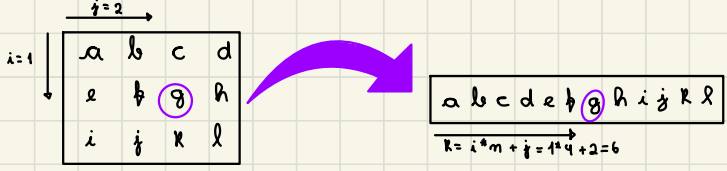
- `void f (..., float (*mat) [3], ...);` ou `void f (..., float mat [ ] [3], ...);`
  - de qualquer modo, o acesso aos elementos da matriz dentro da função é feito da forma usual, com indexação dupla

matrizes dinâmicas

- para trabalhar com matrizes alocadas dinamicamente, temos que criar abstrações conceituais com vetores para representar conjuntos bidimensionais

MATRIZ REPRESENTADA POR UM VETOR SIMPLES: reservemos as primeiras posições do vetor para armazenar os elementos da primeira linha, seguidos dos elementos da segunda linha e assim por diante.

- conceitualmente, trabalharemos com um conjunto bidimensional mas, de fato, temos um vetor unidimensional
- a estratégia de endereçamento para acessar os elementos é a seguinte: se quisermos acessar o que seria o elemento `mat [i] [j]` de uma matriz, devemos acessar os elementos `v [k]`, com $k = i * m + j$, onde m representa o número de colunas da matriz



↳ se quisermos acessar elementos da terceira linha (i=2) linha da matriz, temos de pular duas linhas de elementos (i*m) e depois incrementar o elemento da linha com j

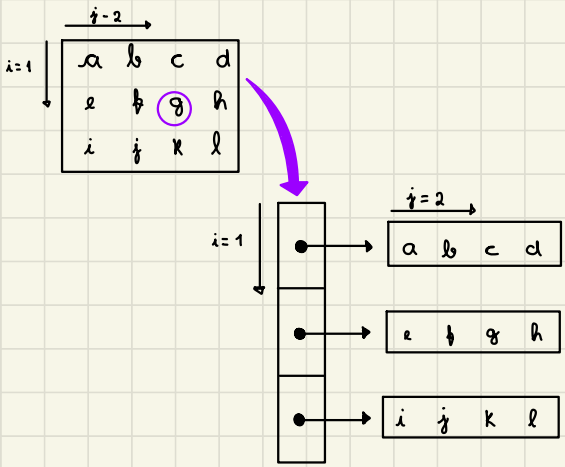
↳ com essa estratégia, a alocação da "matriz" recai em uma alocação de vetor com m*n elementos, onde m e n representam as dimensões da matriz

```
1  float *mat; //matriz representada por um vetor
2  ...
3  mat = (float*) malloc(m*n*sizeof(float));
4  ...
```

↳ mas somos obrigados a usar uma notação desconfortável, v[i*m+j], para acessar os elementos, o que pode deixar o código pouco legível

MATRIZ REPRESENTADA POR UM VETOR DE PONTEIROS: cada linha da matriz é representada por um vetor independente

↳ a matriz é representada por um vetor de vetores, ou vetor de ponteiros, no qual cada elemento armazena o endereço do primeiro elemento de cada linha



• A alocação agora é mais elaborada

↳ primeiro temos de alocar o vetor de ponteiros, em seguida alocamos cada uma das linhas da matriz, atribuindo seus endereços aos elementos do vetor de ponteiros criado

```
1  int i;
2  float **mat; //matriz representada por um vetor de ponteiros
3  ...
4  mat = (float**) malloc(m*sizeof(float));
5  for(i=0; i<m; i++)
6      mat[i] = (float**) malloc(n*sizeof(float));
```

• a grande vantagem dessa estratégia é o acesso aos elementos ser feito da mesma forma que quando temos uma matriz criada estaticamente

• a liberação do espaço de memória ocupado pela matriz também exige a construção de um laço, pois temos de liberar cada linha antes

de liberar o vetor de ponteiros

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

operações com matrizes

MATRIZ COM VETOR SIMPLES: `float* Transposta (int m, int n, float* mat);`

p/ linhas
p/ colunas

```
float* transposta (int m, int n, float* mat){
    int i, j;
    float* trp;

    //aloca matriz transportada:
    trp = (float*) malloc(m*n*sizeof(float));

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

MATRIZ COM VETOR DE PONTEIROS: `float** transposta (int m, int n, float** mat);`

```
float** transporta(int m, int n, float** mat){
    int i, j;
    float** trp;

    //aloca matriz transposta: n linhas, 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;
}
```