Alocação dinâmica de matrizes

As matrizes **estáticas** podem ser definidas e acessadas de forma bastante simples:

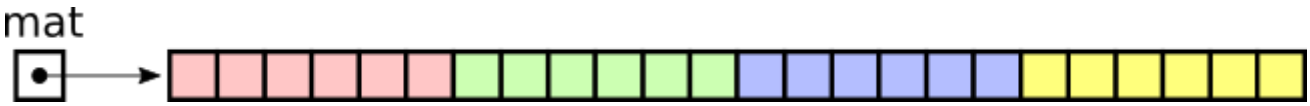
```
int matriz[100][100] ;

for (i=0; i < 100; i++)
  for (j=0; j < 100; j++)
    matriz[i][j] = i+j ;
```

Por outro lado, matrizes **dinâmicas** não são tão simples de alocar ou de usar. Esta seção visa mostrar alguns exemplos de como fazê-lo.

Nas figuras abaixo, cada faixa de blocos de uma mesma cor representa uma linha da matriz.

Método 1: alocação única

Com a *alocação única*, os elementos da matriz são alocados em um único vetor, linearmente. Os elementos da matriz são acessados explicitamente através de aritmética de ponteiros.

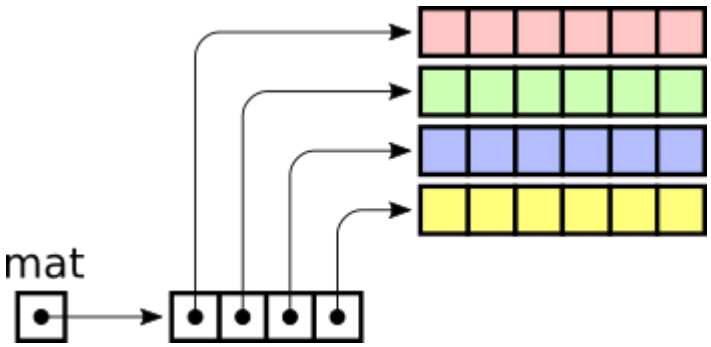



```
#define LIN 4
#define COL 6

int *mat ;
int i, j ;

// aloca um vetor com todos os elementos da matriz
mat = malloc (LIN * COL * sizeof (int)) ;

// percorre a matriz
for (i=0; i < LIN; i++)
   for (j=0; j < COL; j++)
      mat[(i*COL) + j] = 0 ; // calcula a posição de cada elemento

// libera a memória alocada para a matriz
free (mat) ;
```

Método 2: vetor de ponteiros de linhas separadas

Com a alocação de um *vetor de ponteiros de linhas separadas*, a matriz é vista e alocada como um vetor de ponteiros para linhas, que são vetores de elementos. O vetor de ponteiros de linhas e os vetores de cada linha devem ser alocados separadamente.

A vantagem deste método é que o acesso aos elementos da matriz usa a mesma sintaxe do acesso a uma matriz estática, o que torna a programação mais simples.

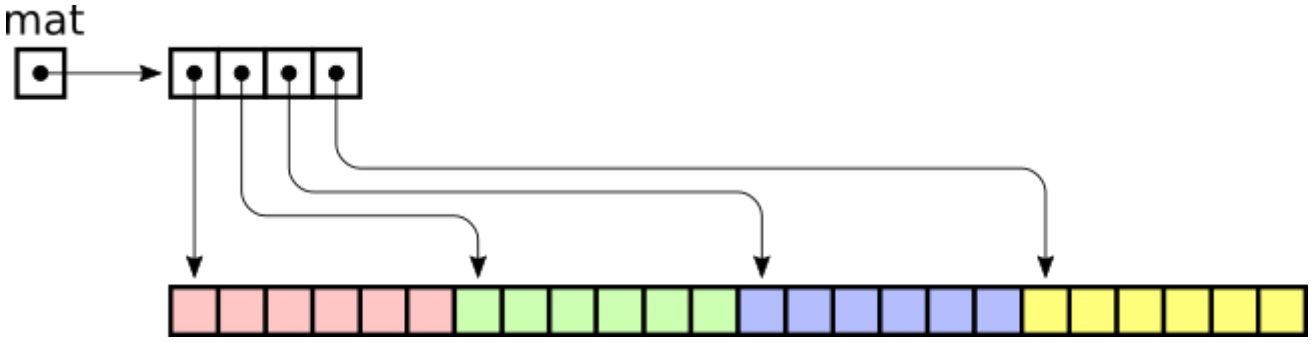



```
#define LIN 4
#define COL 6

int **mat ;
int i, j ;

// aloca um vetor de LIN ponteiros para linhas
mat = malloc (LIN * sizeof (int*)) ;

// aloca cada uma das linhas (vetores de COL inteiros)
for (i=0; i < LIN; i++)
   mat[i] = malloc (COL * sizeof (int)) ;

// percorre a matriz
for (i=0; i < LIN; i++)
   for (j=0; j < COL; j++)
      mat[i][j] = 0 ;        // acesso com sintaxe mais simples

// libera a memória da matriz
for (i=0; i < LIN; i++)
   free (mat[i]) ;
free (mat) ;
```

Método 3: vetor de ponteiros de linhas contíguas

Com a alocação de um *vetor de ponteiros de linhas contíguas*, a matriz é vista e alocada como um vetor de ponteiros para linhas, mas as linhas são alocadas como um único vetor de elementos. O vetor de ponteiros de linhas e o vetor de elementos são alocados separadamente.

Além de usar a mesma sintaxe do acesso que o método anterior (mesma que uma matriz estática), este método tem mais duas vantagens: (i) somente precisa de duas operações de alocação de memória, e (ii) todos os elementos da matriz estão alocados sequencialmente na memória, o que facilita operações de cópia de matrizes (usando "memcpy") ou de leitura/escrita da matriz para um arquivo (usando "fread" ou "fwrite").




```
#define LIN 4
#define COL 6
```

```
int **mat ;
int i, j ;

// aloca um vetor de LIN ponteiros para linhas
mat = malloc (LIN * sizeof (int*)) ;

// aloca um vetor com todos os elementos da matriz
mat[0] = malloc (LIN * COL * sizeof (int)) ;

// ajusta os demais ponteiros de linhas (i > 0)
for (i=1; i < LIN; i++)
   mat[i] = mat[0] + i * COL ;

// percorre a matriz
for (i=0; i < LIN; i++)
   for (j=0; j < COL; j++)
      mat[i][j] = 0 ;

// libera a memória da matriz
free (mat[0]) ;
free (mat) ;
```

Exercícios

1. Utilizando alocação dinâmica de matrizes, escreva um programa para receber duas matrizes de tamanho 3x3 e calcular seu produto.

2. Um quadrado mágico é uma matriz com números distintos na qual a soma dos elementos de cada linha, coluna e diagonal é igual. Elabore um algoritmo que encontre e imprima na tela um quadrado mágico de tamanho 3x3 e cuja soma dos elementos de cada linha, coluna e diagonal seja 15.

3. Escreva um programa que recebe uma sequência de 10 palavras e as armazena em uma matriz do tipo "char**", em seguida, o programa deve trocar a ordem das palavras para ordená-las em ordem alfabética.

4. Utilizando alocação dinâmica, escreva um programa que aloca uma matriz de 3 dimensões e preenche cada elemento dessa matriz com a soma dos índices do elemento. Por exemplo, Matriz[1][2][3] = 1 + 2 + 3 = 6

5. Reescreva o programa anterior utilizando desta vez alocação única para alocar a matriz de 3 dimensões.