Alocação dinâmica de matriz em C: vetor de tamanho variável

Na linguagem C, podemos alocar uma matriz M de dimensões $n \times m$ dinamicamente da seguinte maneira:

```
int n = ...
int m = ...

float M[n][m];
```

Desta forma, a matriz M fica alocada na memória e pode ser usada normalmente:

```
M[i][j] = 3.12;
```

Também é possível definir os valores da matriz de uma forma bem sucinta, por exemplo:

```
int n = 3;
int m = 2;

float M[n][m] = {{1,2},{3,4},{5,6}};
```

Se quisermos definir um método que recebe como parâmetro uma matriz, temos que fazer assim:

```
void metodo(..., int n, int m, float M[n][m], ...){

    ...

}
```

Note que é necessário passar as dimensões da matriz como parâmetros também.

Alocação dinâmica de matriz em C: vetor de vetores

(Obs.: Esta forma pode ser usada e C++ também.)

Na linguagem C, podemos alocar uma matriz M de dimensões $n \times m$ dinamicamente da seguinte maneira:

```
int n = ...
int m = ...

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

Desta forma, a matriz M fica alocada na memória e pode ser usada normalmente:

```
M[i][j] = 3.12;
```

Uma vez criada a matriz dessa forma, o tipo dela na linguagem C é

```
float**
```

Se quisermos definir um método que recebe como parâmetro uma matriz, temos que fazer assim:

```
void metodo(..., int n, int m, float** M, ...){

    ...

}
```

Além disso, as dimensões da matriz devem ser mantidas em variáveis, pois, uma vez criada a matriz dinamicamente, não é possível (em C padrão) recuperar as suas dimensões.

Finalmente, pode ser conveniente liberar o espaço alocado para a matriz uma vez que ela não é mais necessária. Isso pode ser feito desta forma:

```
for (int i = 0; i < n; i++){
    free(M[i]);
}
free(M);
```

Os métodos `malloc()` e `free` fazem parte da biblioteca `stdlib.h` (em C++ as biblioteca é a `cstdlib`).

Alocação dinâmica de matriz em C++: `new` e `delete`

A alocação dinâmica de matrizes em C++ é semelhante à forma feita em C. A principal diferênça é que em C++ usaremos as palavras reservadas `new` e `delete` ao invés dos métodos `malloc()` e `free()`.

Para alocar uma matriz M de dimensões $n \times m$, fazemos o seguinte:

```
float** M = new float*[n];
for (int i = 0; i < n; i++){
    M[i] = new float[m];
}
```

Se quisermos definir um método que recebe como parâmetro uma matriz, temos que fazer assim:

```
void metodo(..., int n, int m, float** M, ...){

    ...

}
```

Além disso, as dimensões da matriz devem ser mantidas em variáveis, pois, uma vez criada a matriz dinamicamente, não é possível (em C padrão) recuperar as suas dimensões.

Para desalocar:

```
for (int i = 0; i < n; i++){
    delete [] M[i];
}
delete [] M;
```

Novamente, se lembre de manter as dimensões da matriz de alguma forma acessível.

Modelo 1: C

```c
#include <stdio.h>

/*
 * Matriz como vetor dinâmico
 *
 * uma matriz é definida assim
 *
 *          int n = ...
 *          int m = ...
 *          float M[n][m];
 *
 * podemos acessar os elementos da matriz assim
 *
 *          M[i][j] = ...
 *
 * para declarar um método que recebe uma matriz, fazemos assim
 *
 *          void metodo(..., int n, int m, float M[n][m], ...)
 *
 * ou seja, além da matriz, temos que passar também as dimensões da matriz
 *
 */

void imprimir_matriz(int n, int m, float M[n][m]){

    for (int i = 0; i < n; i++){
        for (int j = 0; j < m; j++){
            printf("%f ", M[i][j]);
        }
        printf("\n");
    }

}


void trocar_linhas(int i, int j, int n, int m, float M[n][m]){

    // ...

}


void multiplicar_escalar_linha(int i, float j, int n, int m, float M[n][m]){

    // ...

}


void somar_linhas(int i, int j, float a, int n, int m, float M[n][m]){

    // ...

}


void zerar_posicao_abaixo(int i, int j, int n, int m, float M[n][m]){

    // ...
```

```
}


int main()
{
    // criando matriz e inicializando

    int n = 3;
    int m = 3;

    float Mat[n][m];
    Mat[0][0] = 1; Mat[0][1] = 2; Mat[0][2] = 3;
    Mat[1][0] = 1; Mat[1][1] = 3; Mat[1][2] = 2;
    Mat[2][0] = 2; Mat[2][1] = 1; Mat[2][2] = 3;

    // imprimindo a matriz

    imprimir_matriz(n, m, Mat);

    return 0;
}
```

Modelo 2: C e C++

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


/*
 * Matriz como vetor de vetores alocados dinamicamente
 *
 * uma matriz é definida assim
 *
 *          int n = ...
 *          int m = ...
 *          float** M = (float**) malloc(n*sizeof(float*));
 *          for (int i = 0; i < n; i++)
 *              M[i] = (float*) malloc(m*sizeof(float));
 *
 * podemos acessar os elementos da matriz assim
 *
 *          M[i][j] = ...
 *
 * para declarar um método que recebe uma matriz, fazemos assim
 *
 *          void metodo(..., int n, int m, float** M, ...)
 *
 * ou seja, além da matriz, temos que passar também as dimensões da matriz
 *
 */


void imprimir_matriz(int n, int m, float** M){

    for (int i = 0; i < n; i++){
        for (int j = 0; j < m; j++){
            printf("%f ", M[i][j]);
        }
        printf("\n");
    }

}


void trocar_linhas(int i, int j, int n, int m, float** M){

    // ...

}


void multiplicar_escalar_linha(int i, float j, int n, int m, float** M){

    // ...

}


void somar_linhas(int i, int j, float a, int n, int m, float** M){

    // ...

}
```

```c
void zerar_posicao_abaixo(int i, int j, int n, int m, float** M){

    // ...

}


int main()
{
    // criando matriz e inicializando

    int n = 3;
    int m = 3;

    float** Mat = (float**) malloc(n*sizeof(float*));
    for (int i = 0; i < n; i++)
        Mat[i] = (float*) malloc(m*sizeof(float));

    Mat[0][0] = 1; Mat[0][1] = 2; Mat[0][2] = 3;
    Mat[1][0] = 1; Mat[1][1] = 3; Mat[1][2] = 2;
    Mat[2][0] = 2; Mat[2][1] = 1; Mat[2][2] = 3;

    // imprimindo a matriz

    imprimir_matriz(n, m, Mat);

    return 0;
}
```

Modelo 3: C++

```cpp
#include <cstdio>

/*
 * Matriz alocada dinamicamente usando 'new' em C++
 *
 * uma matriz é definida assim
 *
 *          int n = ...
 *          int m = ...
 *          float** M = new float*[n];
 *          for (int i = 0; i < n; i++)
 *              M[i] = new float[m];
 *
 * podemos acessar os elementos da matriz assim
 *
 *          M[i][j] = ...
 *
 * para declarar um método que recebe uma matriz, fazemos assim
 *
 *          void metodo(..., int n, int m, float** M, ...)
 *
 * ou seja, além da matriz, temos que passar também as dimensões da matriz
 *
 */



void imprimir_matriz(int n, int m, float** M){

    for (int i = 0; i < n; i++){
        for (int j = 0; j < m; j++){
            printf("%f ", M[i][j]);
        }
        printf("\n");
    }

}


void trocar_linhas(int i, int j, int n, int m, float** M){

    // ...

}


void multiplicar_escalar_linha(int i, float j, int n, int m, float** M){

    // ...

}


void somar_linhas(int i, int j, float a, int n, int m, float** M){

    // ...

}
```

```
void zerar_posicao_abaixo(int i, int j, int n, int m, float** M){

    // ...

}


int main()
{
    // criando a matriz

    int n = 3;
    int m = 3;

    float** Mat = new float*[n];
    for (int i = 0; i < n; i++)
        Mat[i] = new float[m];

    // inicializando a matriz

    Mat[0][0] = 1; Mat[0][1] = 2; Mat[0][2] = 3;
    Mat[1][0] = 1; Mat[1][1] = 3; Mat[1][2] = 2;
    Mat[2][0] = 2; Mat[2][1] = 1; Mat[2][2] = 3;

    // imprimindo a matriz

    imprimir_matriz(n, m, Mat);

    return 0;
}
```