Matrizes Estáticas em C

Em C, matrizes estáticas são blocos de memória fixos onde armazenamos valores, tipicamente números, em uma estrutura de dados bidimensional. Para entender melhor o funcionamento dessas matrizes, vamos abordar a representação de memória, a manipulação, e as limitações e possibilidades que elas oferecem.

1. Declaração e Representação de Matrizes Estáticas

Uma matriz estática em C é declarada da seguinte forma:

```
int matriz[3][3];
```

Neste caso, declaramos uma matriz 3x3 (3 linhas e 3 colunas), onde cada elemento é do tipo `int`. Ao ser criada, essa matriz ocupa um bloco contínuo de memória. No C, a matriz é armazenada em **row-major order** (ordem de linha), onde todos os elementos de uma linha estão em sequência na memória antes dos elementos da linha seguinte.

2. Representação na Memória

Ao declararmos `int matriz[3][3]`, a memória para essa matriz é alocada de forma contínua. Supondo que cada `int` ocupa 4 bytes, uma matriz `3x3` precisará de \(3 imes 3 imes 4 = 36\) bytes de memória.

A disposição na memória para a matriz seria assim:

Endereço	Elemento da Matriz	Valor Inicial
0x1000	matriz[0][0]	(valor inicial)
0x1004	matriz[0][1]	(valor inicial)
0x1008	matriz[0][2]	(valor inicial)
0x100C	matriz[1][0]	(valor inicial)
0x1010	matriz[1][1]	(valor inicial)
0x1014	matriz[1][2]	(valor inicial)
0x1018	matriz[2][0]	(valor inicial)
0x101C	matriz[2][1]	(valor inicial)
0x1020	matriz[2][2]	(valor inicial)

Essa estrutura permite que o acesso aos elementos seja feito de forma direta, pois cada posição da matriz pode ser calculada pela fórmula:

\[ext{Endereço de matriz}[i][j] = ext{Endereço de matriz}[0][0] + ((i * ext{Número de Colunas}) + j) * ext{tamanho de } int\]

3. Inicialização de Matrizes

Podemos inicializar uma matriz diretamente na declaração:

```
int matriz[3][3] = {
   {1, 2, 3},
   {4, 5, 6},
   {7, 8, 9}
};
```

Aqui, a matriz `3x3` é inicializada com valores explícitos. O compilador organiza esses valores na memória em ordem de linha, seguindo a ordem declarada.

4. Acessando Elementos

Para acessar ou modificar elementos na matriz, utilizamos o índice de linha e coluna:

```
#include <stdio.h>

int main() {
    int matriz[3][3] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };
    // Exibindo todos os elementos da matriz
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            printf("%d ", matriz[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Aqui, os dois `for` loops percorrem todos os elementos da matriz. O `printf` imprime cada elemento, onde `matriz[i][j]` acessa diretamente o valor na posição `i, j`.

5. Manipulando Matrizes

Além de acessar elementos individuais, é possível fazer operações em matrizes, como somar duas matrizes, multiplicar, transpor, etc.

Exemplo de transposição de uma matriz:

```
#include <stdio.h>

int main() {
  int matriz[2][3] = {
    {1, 2, 3},
    {4, 5, 6}
  };
  int transposta[3][2]; // Matriz transposta terá tamanho 3x2
  // Transpondo a matriz
  for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 3; j++) {
      transposta[j][i] = matriz[i][j];
    }
  }
  // Exibindo a matriz transposta
  for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 2; j++) {
      printf("%d ", transposta[i][j]);
    }
    printf("\n");
  }
```

```
    return 0;
}
```

Neste exemplo, a matriz `matriz` de tamanho `2x3` é transposta para uma nova matriz `transposta` de tamanho `3x2`. Cada elemento `matriz[i][j]` é mapeado para `transposta[j][i]`.

6. Limitações de Matrizes Estáticas

- Tamanho fixo: O tamanho de uma matriz estática é fixado em tempo de compilação. Não é possível alterá-lo durante a execução do programa.

- Uso de memória: Matrizes estáticas alocam memória contínua, o que pode limitar o uso em sistemas com pouca memória.

- Falta de flexibilidade: Em casos onde o tamanho da matriz depende de condições dinâmicas, o uso de matrizes dinâmicas (com `malloc`) é mais apropriado.

7. Acessando Memória com Ponteiros

Em C, uma matriz pode ser acessada com ponteiros, pois sua estrutura interna é uma sequência contínua de memória. Isso permite percorrer a matriz com aritmética de ponteiros:

```
#include <stdio.h>

int main() {
    int matriz[3][3] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };
    int *ptr = &matriz[0][0];

    // Exibindo elementos usando ponteiro
    for (int i = 0; i < 9; i++) {
        printf("%d ", *(ptr + i));
    }
    return 0;
}
```

Neste exemplo, o ponteiro `ptr` aponta para o primeiro elemento de `matriz`. Usando aritmética de ponteiros (`*(ptr + i)`), acessamos os elementos sequencialmente.

8. Funções com Matrizes

Ao passar uma matriz para uma função, é necessário especificar pelo menos o número de colunas, para que o compilador entenda a estrutura de memória:

```
#include <stdio.h>

void imprimirMatriz(int matriz[3][3], int linhas, int colunas) {
    for (int i = 0; i < linhas; i++) {
        for (int j = 0; j < colunas; j++) {
            printf("%d ", matriz[i][j]);
        }
        printf("\n");
    }
}

int main() {
    int matriz[3][3] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };

    imprimirMatriz(matriz, 3, 3);
    return 0;
}
```

Aqui, `imprimirMatriz` recebe a matriz `3x3` e imprime seus elementos. O parâmetro `int matriz[3][3]` indica ao compilador o número de colunas, o que é necessário para acessar corretamente cada elemento.

9. Resumo

- Matrizes estáticas em C são alocadas de forma contínua na memória, e o acesso a seus elementos pode ser feito por índice ou por aritmética de ponteiros.

- Matrizes em C são armazenadas em **row-major order**, onde todas as linhas são armazenadas em sequência na memória.

- Passar matrizes para funções requer que o número de colunas seja informado.

- Apesar de eficientes, matrizes estáticas são limitadas, especialmente em casos que exigem maior flexibilidade de alocação.