AULA 13

# PONTEIROS E MATRIZES

Na aula 12, trabalhamos com ponteiros e vetores. Nesta aula veremos as relações entre ponteiros e matrizes. Neste caso, é necessário estender o conceito de indireção (simples) para indireção dupla. Veremos também como estender esse conceito para indireção múltipla e suas relações com variáveis compostas homogêneas multi-dimensionais.

Esta aula é baseada na referência [7].

## 13.1 Ponteiros para elementos de uma matriz

Sabemos que a linguagem C armazena matrizes, que são objetos representados de forma bi-dimensional, como uma seqüência contínua de compartimentos de memória, com demarcações de onde começa e onde termina cada uma de suas linhas. Ou seja, uma matriz é armazenada como um vetor na memória, com marcas especiais em determinados pontos regularmente espaçados: os elementos da linha 0 vêm primeiro, seguidos pelos elementos da linha 1, da linha 2, e assim por diante. Uma matriz com $n$ linhas tem a aparência ilustrada como na figura 13.1.

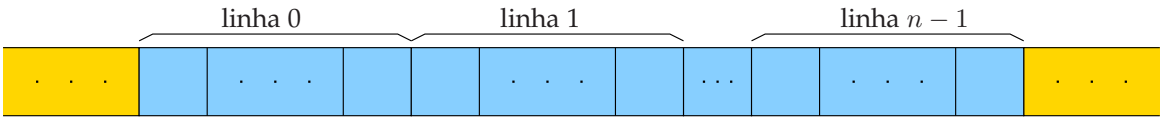


Figura 13.1: Representação da alocação de espaço na memória para uma matriz.

Essa representação pode nos ajudar a trabalhar com ponteiros e matrizes. Se fazemos um ponteiro $p$ apontar para a primeira célula de uma matriz, isto é, o elemento na posição (0,0), podemos então visitar todos os elementos da matriz incrementando o ponteiro $p$ repetidamente.

Por exemplo, suponha que queremos inicializar todos os elementos de uma matriz de números inteiros com 0. Suponha que temos a declaração da seguinte matriz:

```
int A[LINHAS][COLUNAS];
```

A forma que aprendemos inicializar uma matriz pode ser aplicada à matriz $A$ declarada acima e então o seguinte trecho de código realiza a inicialização desejada:

```
int i, j;
⋮
for (i = 0; i < LINHAS; i++)
   for (j = 0; j < COLUNAS; j++)
      A[i][j] = 0;
```

Mas se vemos a matriz $A$ da forma como é armazenada na memória, isto é, como um vetor unidimensional, podemos trocar o par de estruturas de repetição por uma única estrutura de repetição:

```
int *p;
⋮
for (p = &A[0][0]; p <= &A[LINHAS-1][COLUNAS-1]; p++)
   *p = 0;
```

A estrutura de repetição acima inicia com $p$ apontando para `A[0][0]`. Os incrementos sucessivos de $p$ fazem-no apontar para `A[0][1]`, `A[0][2]`, e assim por diante. Quando $p$ atinge `A[0][COLUNAS-1]`, o último elemento da linha 0, o próximo incremento faz com que $p$ aponte para `A[1][0]`, o primeiro elemento da linha 1. O processo continua até que $p$ ultrapasse o elemento `A[LINHAS-1][COLUNAS-1]`, o último elemento da matriz.

## 13.2 Processamento das linhas de uma matriz

Podemos processar uma linha de uma matriz – ou seja, percorrê-la, visitar os conteúdos dos compartimentos, usar seus valores, modificá-los, etc – também usando ponteiros. Por exemplo, para visitar os elementos da linha $i$ de uma matriz $A$ podemos usar um ponteiro $p$ apontar para o elemento da linha $i$ e da coluna 0 da matriz $A$, como mostrado abaixo:

```
p = &A[i][0];
```

ou poderíamos fazer simplesmente

```
p = A[i];
```

já que a expressão `A[i]` é um ponteiro para o primeiro elemento da linha $i$.

A justificativa para essa afirmação vem da aritmética com ponteiros. Lembre-se que para um vetor $A$, a expressão `A[i]` é equivalente a `*(A + i)`. Assim, `&A[i][0]` é o mesmo que `&(*(A[i] + 0))`, que é equivalente a `&*A[i]` e que, por fim, é equivalente a `A[i]`. No trecho de código a seguir usamos essa simplificação para inicializar com zeros a linha $i$ da matriz $A$:

```
int A[LINHAS][COLUNAS], *p, i;
⋮
for (p = A[i]; p < A[i] + COLUNAS; p++)
   *p = 0;
```

Como `A[i]` é um ponteiro para a linha $i$ da matriz $A$, podemos passar `A[i]` para um função que espera receber um vetor como argumento. Em outras palavras, um função que foi projetada para trabalhar com um vetor também pode trabalhar com uma linha de uma matriz. Dessa forma, a função `max` da aula 12 pode ser chamada com a linha $i$ da matriz $A$ como argumento:

```
M = max(COLUNAS, A[i]);
```

## 13.3 Processamento das colunas de uma matriz

O processamento dos elementos de uma coluna de uma matriz não é tão simples como o processamento dos elementos de uma linha, já que a matriz é armazenada linha por linha na memória. A seguir, mostramos uma estrutura de repetição que inicializa com zeros a coluna $j$ da matriz $A$:

```
int A[LINHAS][COLUNAS], (*p)[COLUNAS], j;
⋮
for (p = &A[0]; p < A[LINHAS]; p++)
   (*p)[j] = 0;
```

Nesse exemplo, declaramos $p$ como um ponteiro para um vetor de dimensão `COLUNAS`, cujos elementos são números inteiros. Os parênteses envolvendo `*p` são necessários, já que sem eles o compilador trataria $p$ como um vetor de ponteiros em vez de um ponteiro para um vetor. A expressão `p++` avança $p$ para a próxima linha. Na expressão `(*p)[j]`, `*p` representa uma linha inteira de $A$ e assim `(*p)[j]` seleciona o elemento na coluna $j$ da linha. Os parênteses são essenciais na expressão `(*p)[i]`, já que, sem eles, o compilador interpretaria a expressão como `*(p[i])`.

## 13.4 Identificadores de matrizes como ponteiros

Assim como o identificador de um vetor pode ser usado como um ponteiro, o identificador de uma matriz também pode e, na verdade, de qualquer o identificador de qualquer variável composta homogênea pode. No entanto, alguns cuidados são necessários quando ultrapassamos a barreira de duas ou mais dimensões.

Considere a declaração da matriz a seguir:

```
int A[LINHAS][COLUNAS];
```

Neste caso, $A$ *não* é um ponteiro para `A[0][0]`. Ao contrário, é um ponteiro para `A[0]`. Isso faz mais sentido se olharmos sob o ponto de vista da linguagem C, que considera $A$ não como uma matriz bidimensional, mas como um vetor. Quando usado como um ponteiro, $A$ tem tipo `int (*)[COLUNAS]`, um ponteiro para um vetor de números inteiros de tamanho `COLUNAS`.

Saber que $A$ aponta para `A[0]` é útil para simplificar estruturas de repetição que processam elementos de uma matriz. Por exemplo, ao invés de escrever:

```
for (p = &A[0]; p < &A[LINHAS]; p++)
   (*p)[j] = 0;
```

para inicializar a coluna $j$ da matriz $A$, podemos escrever

```
for (p = A; p < A + LINHAS; p++)
   (*p)[j] = 0;
```

Com essa idéia, podemos fazer o compilador acreditar que uma variável composta homogênea multi-dimensional é unidimensional, isto é, é um vetor. Por exemplo, podemos passar a matriz $A$ como argumento para a função `max` da aula 12 da seguinte forma:

```
M = max(LINHAS*COLUNAS, A[0]);
```

já que `A[0]` aponta para o elemento na linha 0 e coluna 0 e tem tipo `int *` e assim essa chamada será executada corretamente.

## Exercícios

13.1 Escreva uma função que preencha uma matriz quadrada de dimensão $n$ com a matriz identidade $I_n$. Use um único ponteiro que percorra a matriz.

13.2 Reescreva a função abaixo usando aritmética de ponteiros em vez de índices de matrizes. Em outras palavras, elimine as variáveis $i$ e $j$ e todos os `[]`. Use também uma única estrutura de repetição.

```
int soma_matriz(int n, const int A[DIM][DIM])
{
   int i, j, soma = 0;

   for (i = 0; i < n; i++)
      for (j = 0; j < n; j++)
         soma = soma + A[i][j];

   return soma;
}
```