Ponteiros e matrizes

Aula 12

Diego Padilha Rubert

Faculdade de Computação
Universidade Federal de Mato Grosso do Sul

Algoritmos e Programação II

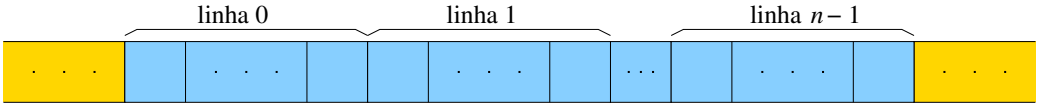
Conteúdo da aula

Ponteiros para elementos de uma matriz

- Na linguagem C, matrizes são armazenadas como uma sequência contínua de compartimentos de memória, como um vetor
- Há demarcações de onde começa e onde termina cada uma de suas linhas
- Podemos fazer um pointeiro *p* apontar para a primeira célula de uma matriz, isto é, o elemento na posição (0,0), e então visitar todos os elementos da matriz incrementando o ponteiro *p*

Ponteiros para elementos de uma matriz

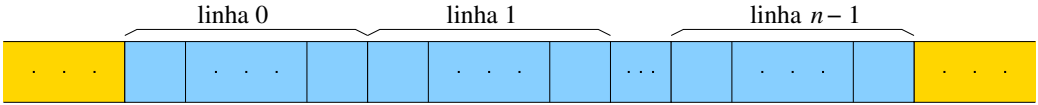
- Na linguagem C, matrizes são armazenadas como uma sequência contínua de compartimentos de memória, como um vetor
- Há demarcações de onde começa e onde termina cada uma de suas linhas



- Podemos fazer um pointeiro *p* apontar para a primeira célula de uma matriz, isto é, o elemento na posição (0,0), e então visitar todos os elementos da matriz incrementando o ponteiro *p*

Ponteiros para elementos de uma matriz

- Na linguagem C, matrizes são armazenadas como uma sequência contínua de compartimentos de memória, como um vetor
- Há demarcações de onde começa e onde termina cada uma de suas linhas



- Podemos fazer um pointeiro *p* apontar para a primeira célula de uma matriz, isto é, o elemento na posição (0,0), e então visitar todos os elementos da matriz incrementando o ponteiro *p*

Ponteiros para elementos de uma matriz

- Se queremos zerar todos elementos de uma matriz `int A[L][C]`, podemos fazer:

```
int i, j;
for (i = 0; i < L; i++)
   for (j = 0; j < C; 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 fazer:

```
int *p;
for (p = &A[0][0]; p <= &A[L-1][C-1]; p++)
   *p = 0;
```

Ponteiros para elementos de uma matriz

- Se queremos zerar todos elementos de uma matriz `int A[L][C]`, podemos fazer:

```
int i, j;
for (i = 0; i < L; i++)
   for (j = 0; j < C; 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 fazer:

```
int *p;
for (p = &A[0][0]; p <= &A[L-1][C-1]; p++)
   *p = 0;
```

Ponteiros para elementos de uma matriz

- Se queremos zerar todos elementos de uma matriz `int A[L][C]`, podemos fazer:

```
int i, j;
for (i = 0; i < L; i++)
   for (j = 0; j < C; 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 fazer:

```
int *p;
for (p = &A[0][0]; p <= &A[L-1][C-1]; p++)
   *p = 0;
```

Ponteiros para elementos de uma matriz

- Se queremos zerar todos elementos de uma matriz `int A[L][C]`, podemos fazer:

```
int i, j;
for (i = 0; i < L; i++)
   for (j = 0; j < C; 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 fazer:

```
int *p;
for (p = &A[0][0]; p <= &A[L-1][C-1]; p++)
   *p = 0;
```

Processamento das linhas de uma matriz

- Para visitar os elementos da linha *i* de uma matriz *A* podemos fazer um ponteiro *p* apontar para o elemento da linha *i* e da coluna 0 de *A*:

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

ou simplesmente:

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

Processamento das linhas de uma matriz

- Para visitar os elementos da linha *i* de uma matriz *A* podemos fazer um ponteiro *p* apontar para o elemento da linha *i* e da coluna 0 de *A*:

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

ou simplesmente:

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

Processamento das linhas de uma matriz

- Para visitar os elementos da linha *i* de uma matriz *A* podemos fazer um ponteiro *p* apontar para o elemento da linha *i* e da coluna 0 de *A*:

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

ou simplesmente:

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

Processamento das linhas de uma matriz

- Para inicializar com zeros a linha *i* da matriz *A*:

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

Processamento das linhas de uma matriz

- 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
- A função `max` da aula anterior pode ser chamada com a linha *i* da matriz *A* como argumento:

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

Processamento das linhas de uma matriz

- 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
- A função `max` da aula anterior pode ser chamada com a linha *i* da matriz *A* como argumento:

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

Processamento das colunas de uma matriz

- Na linguagem C, podemos declarar um ponteiro para um vetor:

```
int (*p)[C];
```

- Nesse caso, *p* deverá apontar para um vetor de dimensão `C`
- A expressão `p++` avança *p* um bloco inteiro, de dimensão `C`
- Caso *p* aponte para uma linha de uma matriz, `p++` avança *p* para a próxima linha

Processamento das colunas de uma matriz

- Na linguagem C, podemos declarar um ponteiro para um vetor:

```
int (*p)[C];
```

- Nesse caso, *p* deverá apontar para um vetor de dimensão `C`
- A expressão `p++` avança *p* um bloco inteiro, de dimensão `C`
- Caso *p* aponte para uma linha de uma matriz, `p++` avança *p* para a próxima linha

Processamento das colunas de uma matriz

- Na linguagem C, podemos declarar um ponteiro para um vetor:

```
int (*p)[C];
```

- Nesse caso, *p* deverá apontar para um vetor de dimensão `C`
- A expressão `p++` avança *p* um bloco inteiro, de dimensão `C`
- Caso *p* aponte para uma linha de uma matriz, `p++` avança *p* para a próxima linha

Processamento das colunas de uma matriz

- Na linguagem C, podemos declarar um ponteiro para um vetor:

```
int (*p)[C];
```

- Nesse caso, *p* deverá apontar para um vetor de dimensão `C`
- A expressão `p++` avança *p* um bloco inteiro, de dimensão `C`
- Caso *p* aponte para uma linha de uma matriz, `p++` avança *p* para a próxima linha

Processamento das colunas de uma matriz

- Para inicializar com zeros a coluna *j* da matriz *A*:

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

Processamento das colunas de uma matriz

- Da mesma forma que o identificador de um vetor pode ser utilizado como um ponteiro, o identificador de uma matriz também o pode
- Considerando a matriz `A[L][C]`:
  - Sabemos que `A[0]` é um ponteiro do tipo `int *` para o elemento `A[0][0]`
  - Contudo, *A* é um ponteiro do tipo `int (*)[C]` para a linha `A[0]`

Processamento das colunas de uma matriz

- Da mesma forma que o identificador de um vetor pode ser utilizado como um ponteiro, o identificador de uma matriz também o pode
- Considerando a matriz `A[L][C]`:
  - Sabemos que `A[0]` é um ponteiro do tipo `int *` para o elemento `A[0][0]`
  - Contudo, *A* é um ponteiro do tipo `int (*)[C]` para a linha `A[0]`

Processamento das colunas de uma matriz

- Da mesma forma que o identificador de um vetor pode ser utilizado como um ponteiro, o identificador de uma matriz também o pode
- Considerando a matriz `A[L][C]`:
  - Sabemos que `A[0]` é um ponteiro do tipo `int *` para o elemento `A[0][0]`
  - Contudo, *A* é um ponteiro do tipo `int (*)[C]` para a linha `A[0]`

Processamento das colunas de uma matriz

- Da mesma forma que o identificador de um vetor pode ser utilizado como um ponteiro, o identificador de uma matriz também o pode
- Considerando a matriz `A[L][C]` :
  - Sabemos que `A[0]` é um ponteiro do tipo `int *` para o elemento `A[0][0]`
  - Contudo, *A* é um ponteiro do tipo `int (*)[C]` para a linha `A[0]`

Processamento das colunas de uma matriz

- Para inicializar com zeros a coluna *j* da matriz *A*, ao invés de:

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

- Podemos fazer:

```
int A[L][C], (*p)[C], j;
⋮
for (p = A; p < A + L; p++)
   (*p)[j] = 0;
```

Processamento das colunas de uma matriz

- Para inicializar com zeros a coluna *j* da matriz *A*, ao invés de:

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

- Podemos fazer:

```
int A[L][C], (*p)[C], j;
⋮
for (p = A; p < A + L; p++)
   (*p)[j] = 0;
```

Processamento das colunas de uma matriz

- Também podemos fazer o compilador acreditar que uma variável composta homogênea multi-dimensional é unidimensional, isto é, é um vetor:

```
M = max(L*C, A[0]);
```

Processamento das colunas de uma matriz

- Também podemos fazer o compilador acreditar que uma variável composta homogênea multi-dimensional é unidimensional, isto é, é um vetor:

```
M = max(L*C, A[0]);
```

Exercícios

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.
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[MAX][MAX])
{
   int i, j, soma = 0;

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

   return soma;
}
```