Aula: Alocação Dinâmica (Parte 2)

Introdução a Programação

Túlio Toffolo & Puca Huachi
http://www.toffolo.com.br

BCC201 – 2020/1
Departamento de Computação – UFOP

Aula: Alocação Dinâmica (Parte 2)

1. Mais sobre matrizes ‘dinâmicas’
2. Estruturas heterogêneas
3. Passagem de ponteiros por referência
4. Exemplos e exercícios

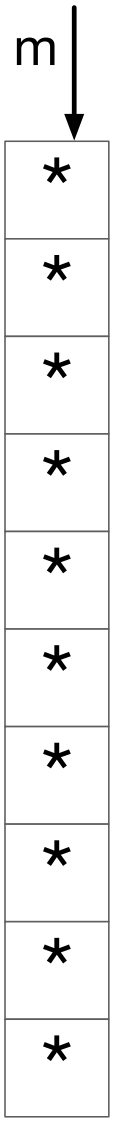
Aula: Alocação Dinâmica (Parte 2)

Matrizes dinâmicas

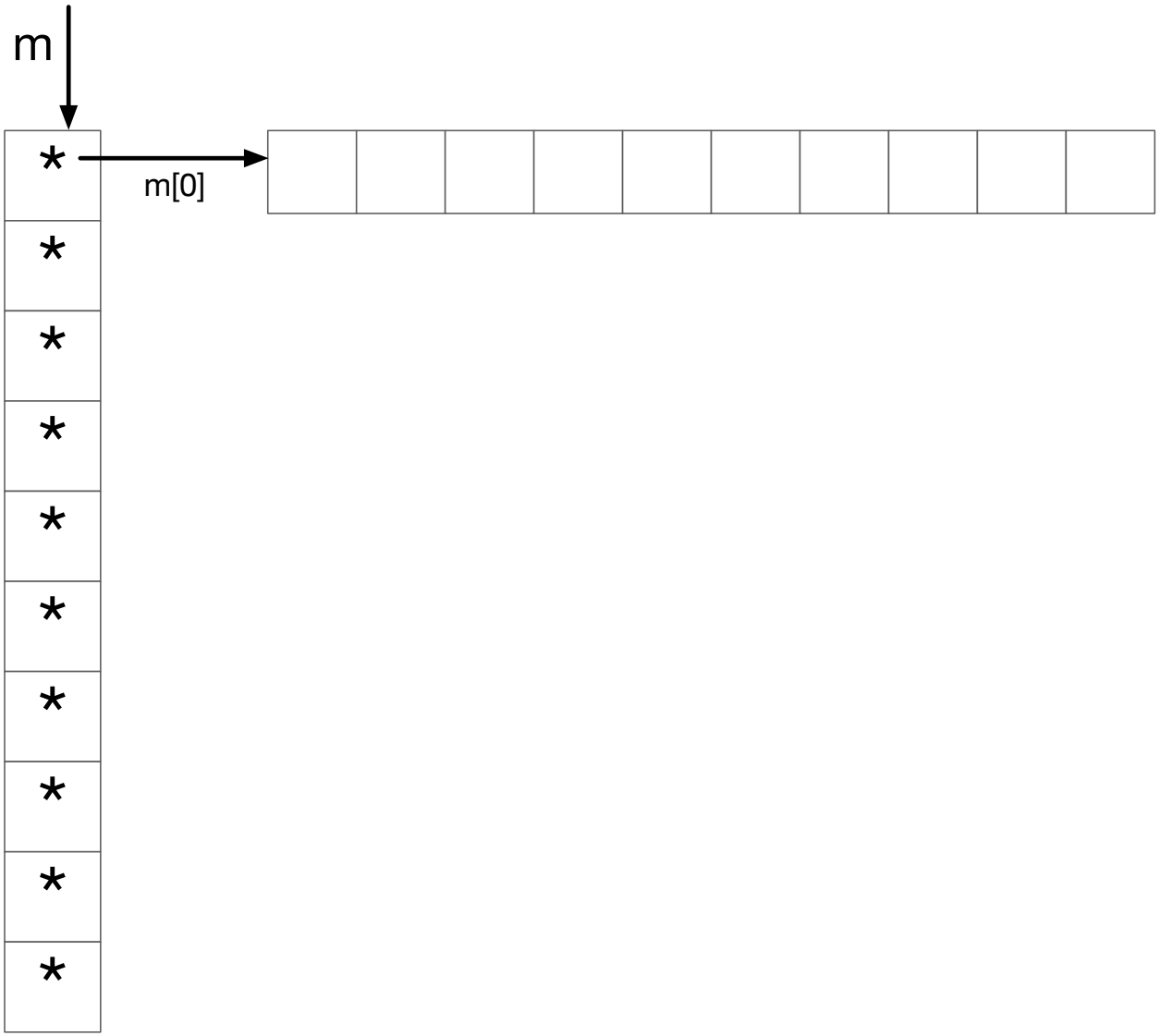
Matrizes dinâmicas são, na verdade, vetores de vetores

```
1  // função que cria uma matriz de tamanho lins x cols
2  int **criaMatriz(int lins, int cols)
3  {
4      int **m;
5      m = malloc(lins * sizeof(int*));
6      for (int i = 0; i < lins; i++)
7          m[i] = malloc(cols * sizeof(int));
8      return m;
9  }
```

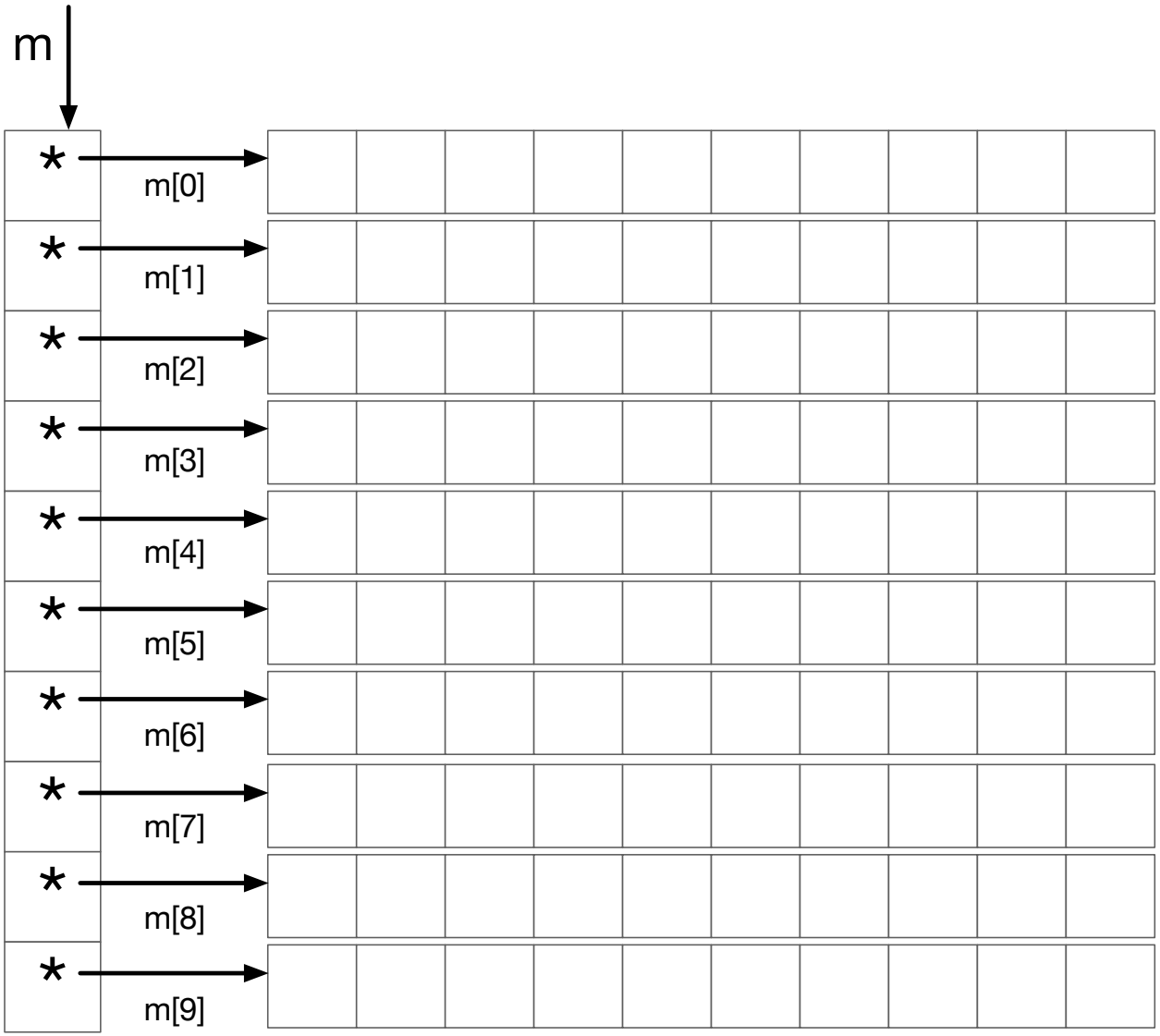
Matrizes dinâmicas são vetores de vetores



Matrizes dinâmicas são vetores de vetores



Matrizes dinâmicas são vetores de vetores



O que acontecerá na memória?

```c
#include <stdlib.h>

// cria uma matriz de tamanho lins x cols
int **criaMatriz(int lins, int cols)
{
    int **m;
    m = malloc(lins * sizeof(int*));
    for (int i = 0; i < lins; i++)
        m[i] = malloc(cols * sizeof(int));
    return m;
}

// libera uma matriz com lins linhas
void liberaMatriz(int **m, int lins) {
    for (int i = 0; i < lins; i++)
        free(m[i]);
    free(m);
}

int main() {
    int **matriz = criaMatriz(2, 2);
    matriz[1][0] = 1;
    liberaMatriz(matriz, 2);
    return 0;
}
```

Aula: Alocação Dinâmica (Parte 2)

Alocando memória para estruturas

Assim como alocamos memória para um `int` ou `double`, podemos, é claro, alocar memória para tipos definidos pelo programador:

```
1  int *x = malloc(sizeof(int));
2  *x = 10;
3  ...
4  free(x);
```

```
1  typedef struct {
2      int matricula;
3      int media;
4      char nome[100];
5  } Aluno;
```

```
1  Aluno *aluno = malloc(sizeof(Aluno));
2  (*aluno).matricula = 10;
3  aluno->matricula = 10; // lembram desta sintaxe?
4  ...
5  free(aluno);
```

Alocando memória para estruturas

Nós podemos, inclusive, ter ponteiros dentro de estruturas:

```
1  typedef struct {
2      int matricula;
3      double *notas;
4  } Aluno;
5
6  int main() {
7      Aluno a;
8      ...
9      a.notas = malloc(10 * sizeof(double));
10     ...
11     free(a.notas);
12     return 0;
13 }
```

Alocando memória para estruturas

E podemos alocar dinamicamente memória tanto para Aluno quanto para um campo dentro de Aluno:

```
1  typedef struct {
2      int matricula;
3      double *notas;
4  } Aluno;
5
6  int main() {
7      Aluno *a = malloc(sizeof(Aluno));
8      ...
9      a->notas = malloc(10 * sizeof(double));
10     ...
11     free(a->notas);
12     free(a);
13     return 0;
14 }
```

Exemplo/exercício

Crie um programa que lê e armazena os seguintes dados de n alunos:

1. número de matrícula;
2. tempo no curso (número de semestres);
3. notas em k avaliações.

Seu programa deve ler os valores de n e k no início da execução.
Utilize alocação dinâmica.

Aula: Alocação Dinâmica (Parte 2)

Referência de ponteiros

Em alguns casos, pode ser útil passar a referência de um ponteiro para uma função.

Exemplo:

- Alocar memória para dois vetores
- Armazenar o endereço em ponteiros passados por parâmetro

```
1 /* Função fictícia que aloca memória para dois vetores de tamanho n */
2 void alocaVetores(int **vetor1, int **vetor2, int n);
```

- Note o tipo `int**`: trata-se de um ponteiro para ponteiro
- Como implementar a função acima?

```c
/* Função fictícia que aloca memória para dois vetores de tamanho n */
void alocaVetores(int **vetor1, int **vetor2, int n) {
    *vetor1 = malloc(n * sizeof(int));
    *vetor2 = malloc(n * sizeof(int));
}
```

Observações:

- Note que o conteúdo de um ponteiro para ponteiro é um **ponteiro**!
- Note o exemplo abaixo, de como chamar a função acima:

```c
int main() {
    int n = 100;
    ...
    int *vetor1, *vetor2;
 → alocaVetores(&vetor1, &vetor2, n); // passagem por referência
    ...
    free(vetor1);
    free(vetor2);
    return 0;
}
```

Referência de ponteiros

Atenção: cuidado para não confundir com matrizes:

- A referência de um ponteiro `int*` é do tipo `int**`
- A referência de uma matriz `int**` é do tipo `int***`

Aula: Alocação Dinâmica (Parte 2)

Exercícios

Exercício 1

Crie um procedimento `alocaMatriz` que recebe por parâmetro:

1. referência de um ponteiro para alocar e retornar uma matriz de inteiros;
2. número de linhas (`n`);
3. número de colunas (`m`).

A função deve alocar uma matriz `n`$\times$`m` dinamicamente e preencher todos os campos com valor zero.

Importante: implemente a função `main` para mostrar um exemplo de uso da função criada anteriormente.

Perguntas?