MCTA028-15: Programação Estruturada

Aula 9: Alocação Dinâmica (Segunda Parte)

Wagner Tanaka Botelho
wagner.tanaka@ufabc.edu.br / wagtanaka@gmail.com
Universidade Federal do ABC (UFABC)
Centro de Matemática, Computação e Cognição (CMCC)

Alocação de *Arrays* Multidimensionais

Alocação de *Arrays* Multidimensionais

- Existem várias soluções na Linguagem C para alocar um *array* com mais de uma dimensão:
  - Usando *array* unidimensional;
  - Usando ponteiro para ponteiro;
  - Ponteiro para ponteiro para *array*;
  - ...

Usando *Array* Unidimensional

Usando *Array* Unidimensional

- Apesar de terem o comportamento de estruturas com mais de uma dimensão, os dados dos *arrays* multidimensionais são armazenados LINEARMENTE na memória;
- O uso dos COLCHETES cria a impressão de estarmos trabalhando com MAIS de uma DIMENSÃO:
  - Por exemplo, a matriz `int mat[3][3]`, apesar de ser bidimensional, é armazenada como um **SIMPLES** *array* na memória:

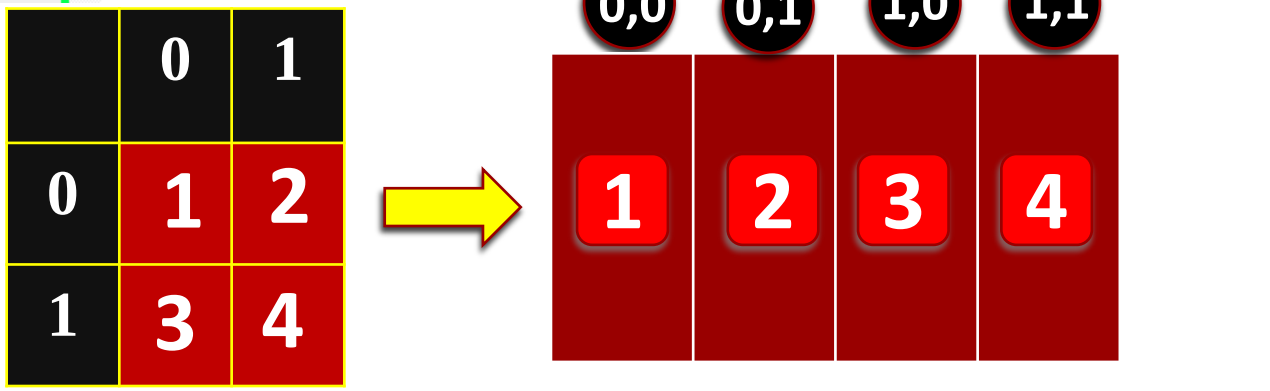
	0	1	2
0	5	7	5
1	1	9	7
2	0	2	1

x_10.c

```c
#include <stdio.h>
#include <stdlib.h>
void main(){
    int *p=NULL;
    int i, j, Nlinhas = 2, Ncolunas = 2;

    p = (int *) malloc(Nlinhas * Ncolunas * sizeof(int));

    for (i = 0; i < Nlinhas; i++){
        for (j = 0; j < Ncolunas; j++){
            scanf("%d", &p[i * Ncolunas + j]);
        }
    }

    for (i = 0; i < Nlinhas; i++){
        for (j = 0; j < Ncolunas; j++){
            printf("%d ",p[i * Ncolunas + j]);
        }
    }
}
```

2*2*4 = 16 bytes

PONTEIRO para a PRIMEIRA POSIÇÃO do *array* ALOCADO



	0	1
0	1	2
1	3	4

Linha	*p	i	J	Nlinhas	NColunas
4	NULL				
5				2	2
7	[0]...[4]				
9		0			
10			0		
11	[0]=(1)				
10			1		
11	[1]=(2)				
10			2		
9		1			
10			0		
11	[2]=(3)				
10			1		
11	[3]=(4)				
10			2		
9		2			
15		0			

x_10.c

```c
#include <stdio.h>
#include <stdlib.h>
void main(){
    int *p=NULL;
    int i, j, Nlinhas = 2, Ncolunas = 2;

    p = (int *) malloc(Nlinhas * Ncolunas * sizeof(int));

    for (i = 0; i < Nlinhas; i++){
        for (j = 0; j < Ncolunas; j++){
            scanf("%d", &p[i * Ncolunas + j]);
        }
    }

    for (i = 0; i < Nlinhas; i++){
        for (j = 0; j < Ncolunas; j++)
            printf("%d ",p[i * Ncolunas + j]);
        }
    }
    free(p);
    p=NULL;
}
```

	0	1
0	1	2
1	3	4

Lixos na Memória!

Linha	*p	I	J	Nlinhas	NColunas
15		0			
16			0		
17	[0]={1}				
16			1		
17	[1]={2}				
16			2		
15		1			
16			0		
17	[2]={3}				
16			1		
17	[3]={4}				
16			2		
15		2			
20	liberar o bloco de memória				
21	NULL				

Usando Ponteiro para Ponteiro

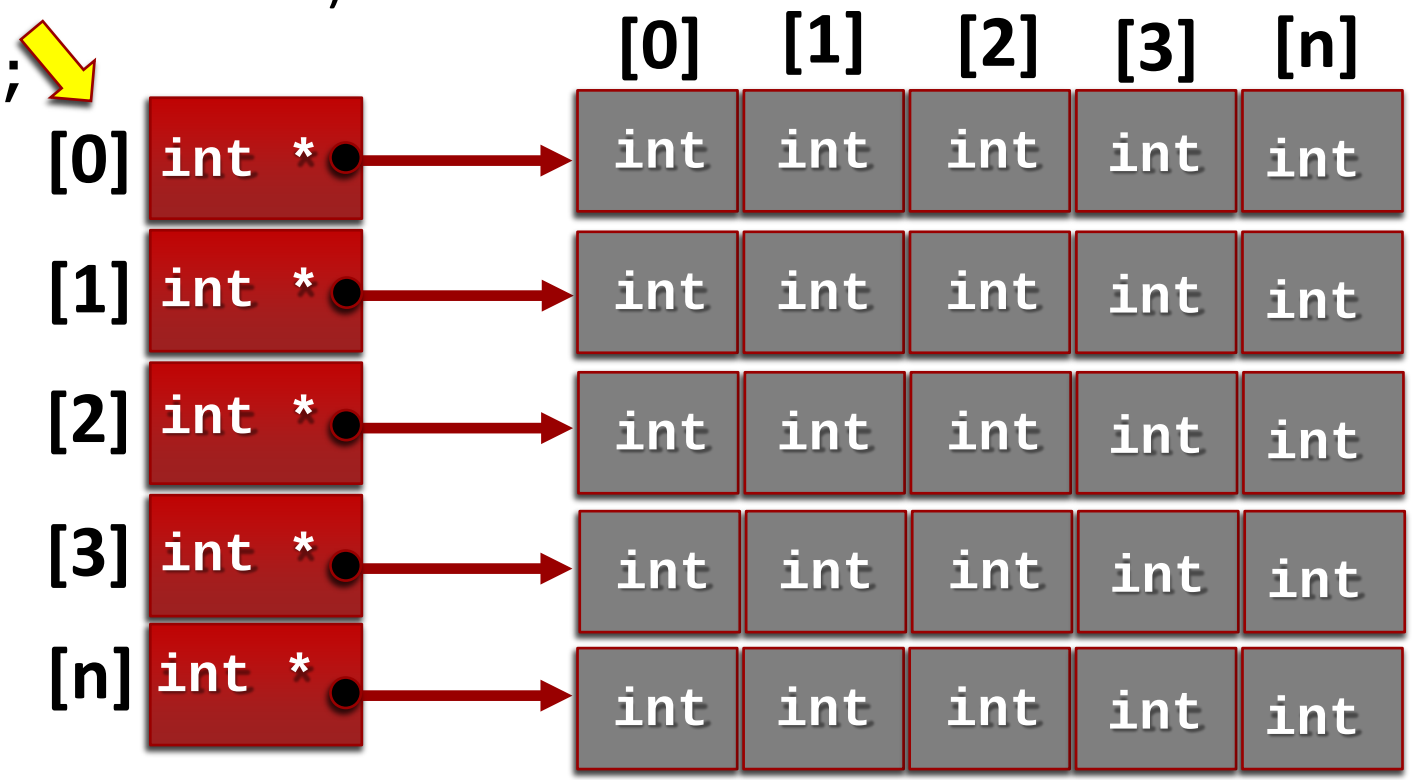
Usando Ponteiro para Ponteiro

- Para alocar um *array* com mais de uma dimensão e manter a NOTAÇÃO DE COLCHETES "[i][j]" para cada dimensão:
  - Deve-se utilizar o conceito de "ponteiro para ponteiro".
- Basicamente, para alocar uma matriz (*array* com DUAS dimensões), utiliza-se um PONTEIRO com DOIS níveis:
  - Em um ponteiro para ponteiro, cada nível do ponteiro permite criar uma nova dimensão no *array*.
- Para um *array* com **DUAS** dimensões, precisa-se de um ponteiro com **DOIS** níveis (**):
  - Se quisermos TRÊS dimensões, é necessário um ponteiro com **TRÊS** níveis (***).

No primeiro nível é alocado um ***ARRAY* DE PONTEIROS** (**LINHAS DA MATRIZ**).

```
p = (int**) malloc(nLin*sizeof(int *));
```

```
int **p;
```



Aloca o ***ARRAY* DE PONTEIROS** usando o tamanho de um **PONTEIRO** para `int`.

Aloca o ***ARRAY*** usando o tamanho de um `int`.

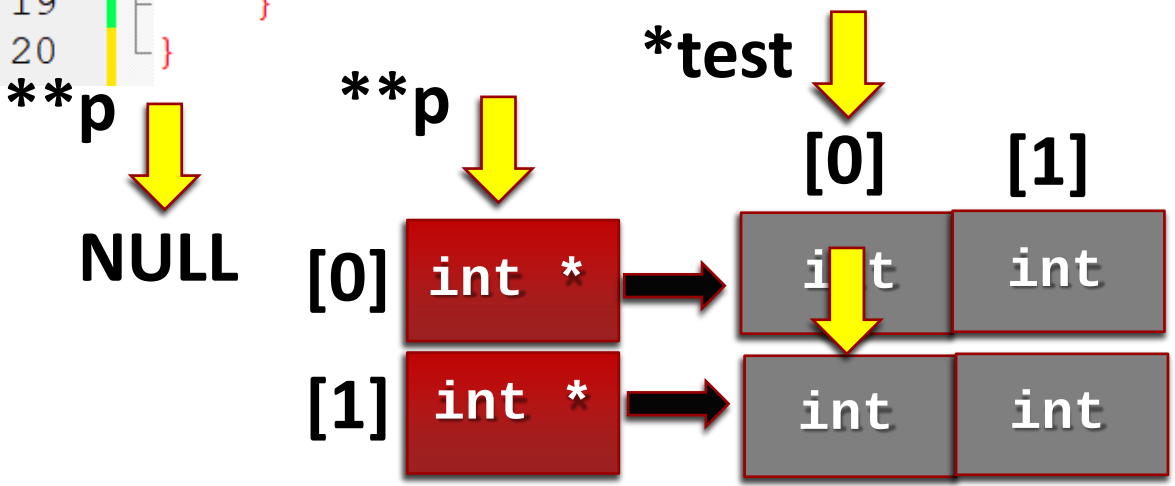
No segundo nível, para cada posição do ***ARRAY* DE PONTEIROS,** é alocado um ***ARRAY* DE INTEIROS (COLUNAS DA MATRIZ).**

```
p[i] = (int *)malloc(nCol*sizeof(int));
```

*Ex_12.c

```c
#include <stdio.h>
#include <stdlib.h>
void main(){
    int **p = NULL, *test = NULL;
    int nLin = 2, nCol = 2;

    p = (int**) malloc(nLin*sizeof(int *));

    for (int i = 0; i < nCol; i++){
        test = (int *)malloc(nCol*sizeof(int));
        p[i] = test;
        //p[i] = (int *)malloc(nCol*sizeof(int));
    }

    for (int i = 0; i < nLin; i++){
        for (int j = 0; j < nCol; j++){
            scanf("%d", &p[i][j]);
        }
    }
}
```

Não estão no teste de mesa.

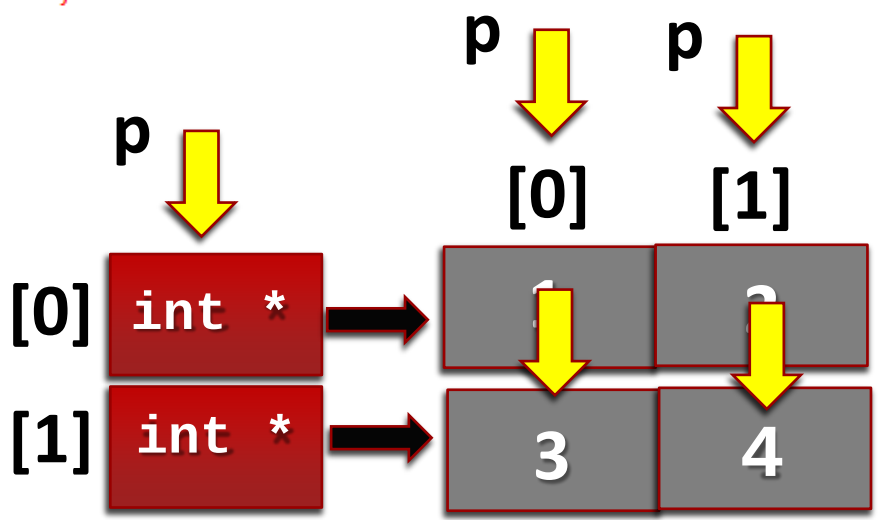


Linha	p	test	i	j
4, 5	**p = NULL	*test = NULL		
7	[0]=0x731710 - int*			
7	[1]=0x731718 - int*			
9			0	
10		[0]=0x1b1750 - int		
10		[1]=0x1b1754 - int		
11	[0] -> 0x1b1750			
9			1	
10		[0]=0x771790 - int		
10		[1]=0x771794 - int		
11	[1] -> 0x771790			
9			2	
15			0	

*Ex_12.c

```c
#include <stdio.h>
#include <stdlib.h>
void main(){
    int **p = NULL, *test = NULL;
    int nLin = 2, nCol = 2;

    p = (int**) malloc(nLin*sizeof(int *));

    for (int i = 0; i < nCol; i++){
        test = (int *)malloc(nCol*sizeof(int));
        p[i] = test;
        //p[i] = (int *)malloc(nCol*sizeof(int));
    }

    for (int i = 0; i < nLin; i++){
        for (int j = 0; j < nCol; j++){
            scanf("%d", &p[i][j]);
        }
    }
}
```

Não estão no t. mesa.



Linha	p	test	i	j
15			0	
16				0
17	[0][0]=(1)			
16				1
17	[0][1]=(2)			
16				2
15			1	
16				0
17	[1][0]=(3)			
16				1
17	[1][1]=(4)			
16				2
15			2	

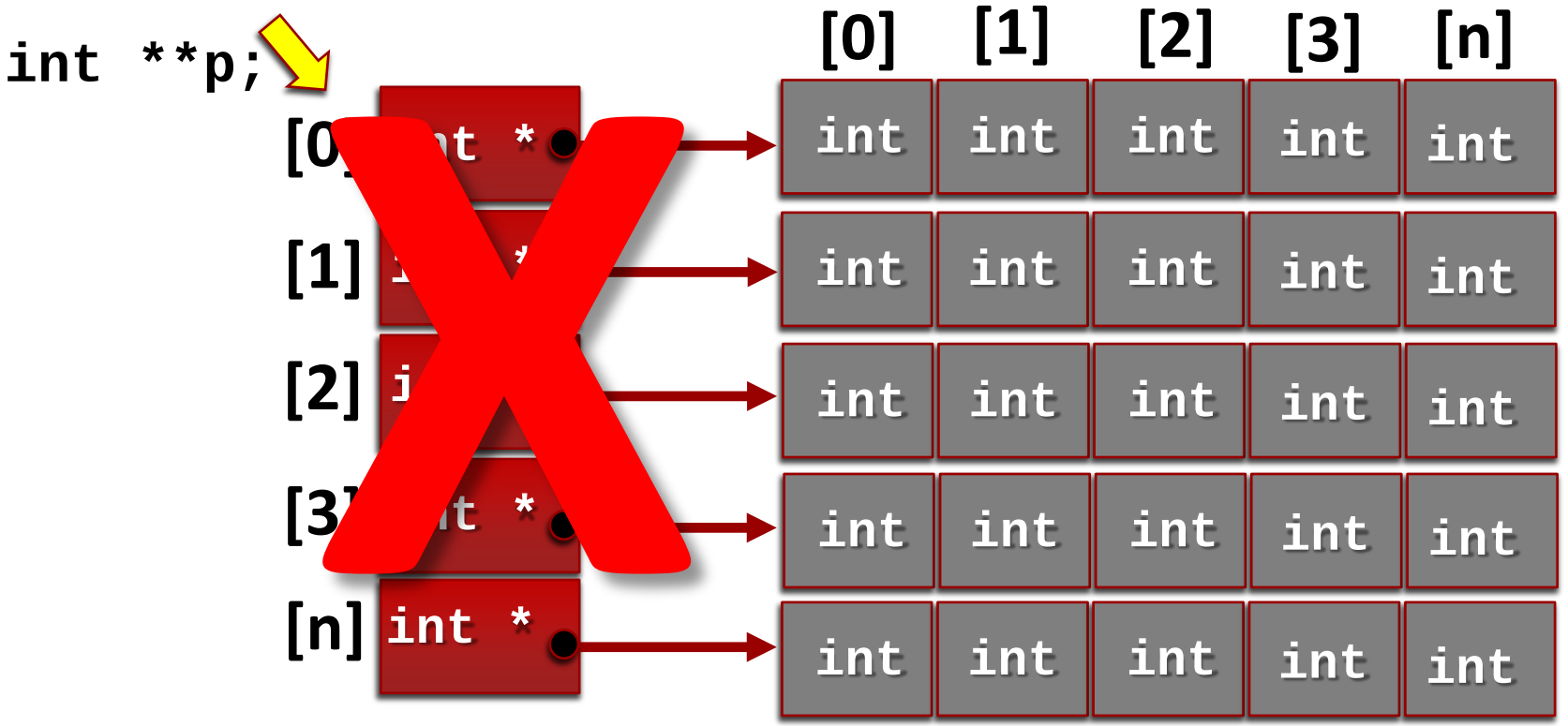
Liberar Memória

Ex_13.c

```c
#include <stdio.h>
#include <stdlib.h>
void main(){
    int **p = NULL, *test = NULL;
    int nLin = 2, nCol = 2;

    p = (int**) malloc(nLin*sizeof(int *));

    for (int i = 0; i < nCol; i++){
        test = (int *)malloc(nCol*size
        p[i] = test;
        //p[i] = (int *)malloc(nCol*si
    }

    for (int i = 0; i < nLin; i++){
        for (int j = 0; j < nCol; j++)
            scanf("%d", &p[i][j]);
        }
    }

    for (int i = 0; i < nCol; i++){
        free(p[i]);
    }

    free(p);
}
```

Essa ordem deve ser respeitada porque, se liberar **PRIMEIRO** as LINHAS, os **PONTEIROS** para onde estão alocadas as COLUNAS serão **PERDIDOS** e, assim, não sendo possível liberá-los.

Deve-se liberar a memória no sentido inverso da alocação: **PRIMEIRO** libera-se as COLUNAS, para depois liberar as LINHAS da matriz.



Os ponteiros são perdidos, caso as linhas sejam liberadas primeiro.

Referências

- Slides do Prof. Luiz Rozante;
- SALES, André Barros de; AMVAME-NZE, Georges. Linguagem C: roteiro de experimentos para aulas práticas. 2016;
- BACKES, André. Linguagem C Completa e Descomplicada. Editora Campus. 2013;
- SCHILDT, Herbert. C Completo e Total. Makron Books. 1996;
- DAMAS, Luís. Linguagem C. LTC Editora. 1999;
- DEITEL, Paul e DEITEL, Harvey. C Como Programar. Pearson. 2011.