Ponteiros para Estruturas, Matrizes Alocadas Dinamicamente

INF0446 — Introdução à Computação

Prof. Me. Raphael Guedes

raphaelguedes@ufg.br

2024

INF
INSTITUTO DE INFORMÁTICA

UFG
UNIVERSIDADE FEDERAL DE GOIÁS

Relembrando Struct

```
struct ponto {
    float x;
    float y;
};


typedef struct ponto {
    float x;
    float y;
}Ponto;
```

Ponteiro para Estrutura

- Podemos ter variáveis do tipo ponteiro para estruturas.

```
typedef struct ponto {
     float x_value;
     float y_value;
}Ponto;


int main() {
     Ponto ponto_ini;
     Ponto *ponto_ptr;
     ponto_ptr = &ponto_ini;
     ...
}
```

A variável `ponto_ptr` armazena o endereço de uma estrutura (`ponto_ini`)

Acessando os membros do ponteiro p/ struct

- Os membros de uma estrutura são acessados usando seu nome seguido do operador ponto.
  - `estrutura.membro`
- Podemos acessar os membros do mesmo jeito utilizando ponteiros?

Acessando os membros do ponteiro p/ struct

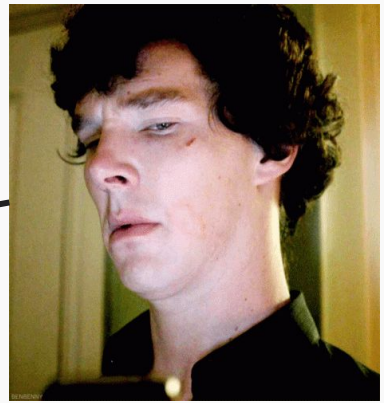
- Podemos ter variáveis do tipo ponteiro para estruturas.

```
typedef struct ponto {
    float x_value;
    float y_value;
}Ponto;


int main() {
    Ponto ponto_ini;
    Ponto *ponto_ptr;
    ponto_ptr = &ponto_ini;

    ponto_ptr.x_value = 7;

    ...
}
```



Acessando os membros do ponteiro p/ struct

- Podemos ter variáveis do tipo ponteiro para estruturas.

```
typedef struct ponto {
     float x_value;
     float y_value;
}Ponto;


int main() {
     Ponto ponto_ini;
     Ponto *ponto_ptr;
     ponto_ptr = &ponto_ini;

     ponto_ptr.x_value = 7;

     ...
}
```

Não podemos acessar membros de uma estrutura via ponteiro desta forma!

Acessando os membros do ponteiro p/ struct

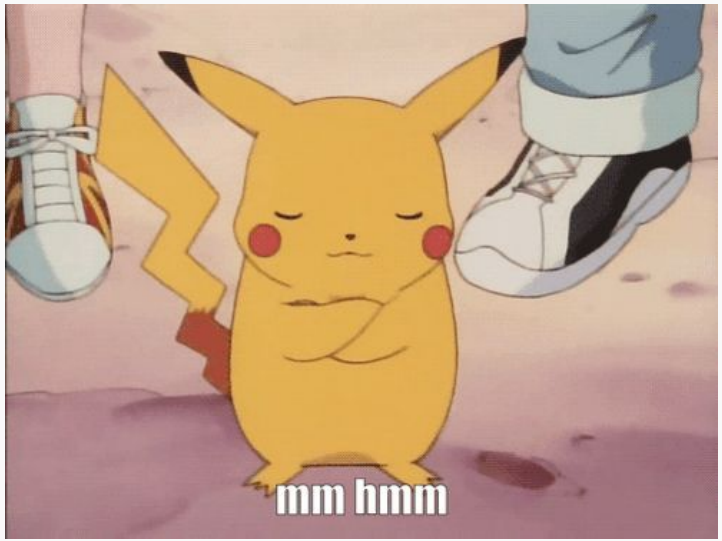
- Podemos ter variáveis do tipo ponteiro para estruturas.

```
typedef struct ponto {
    float x_value;
    float y_value;
}Ponto;


int main() {
    Ponto ponto_ini;
    Ponto *ponto_ptr;
    ponto_ptr = &ponto_ini;

    ponto_ptr -> x_value = 7;

    ...
}
```



Acessando os membros do ponteiro p/ struct

- Para acessar os membros de uma estrutura por meio de um ponteiro, existem 2 formas:
  - Usando o operador * seguido da variável dentro de parênteses:
    - `(*ponto_ptr).x_value = 7.0;`
  - Usando o operador ->
    - `ponto_ptr-> x_value = 7.0;`

Precisa de parênteses para variável **`ponto_ptr`**, senão o compilador entenderia como **`*(ponto_ptr.x_value)`**.

Isso está errado!

Alocação Dinâmica para Estruturas

- Podemos ter variáveis do tipo ponteiro para estruturas.

```
typedef struct pessoa {
     char nome[10];
     int idade;
     float peso;
}Pessoa;


int main() {
     Pessoa *pes_1;
     pes_1 = (Pessoa*) malloc(sizeof(Pessoa));
     strcpy(pes_1-> nome,"Ana");
     pes_1-> idade = 30;
     ...
}
```

Aloca espaço na memória para uma estrutura que contém um vetor de 10 caracteres (10 bytes), um inteiro (4 bytes) e um float (4 bytes)

O que retorna?

Alocação Dinâmica para Estruturas

- Podemos ter variáveis do tipo ponteiro para estruturas.

```
typedef struct pessoa {
     char nome[10];
     int idade;
     float peso;
}Pessoa;


int main() {
     Pessoa *pes_1;
     pes_1 = (Pessoa*) malloc(sizeof(Pessoa));
     strcpy(pes_1-> nome,"Ana");
     pes_1-> idade = 30;
     ...
}
```

Aloca espaço na memória para uma estrutura que contém um vetor de 10 caracteres (10 bytes), um inteiro (4 bytes) e um float (4 bytes)

O que retorna?

A posição da memória alocada

Alocação Dinâmica para Estruturas

- Podemos ter variáveis do tipo ponteiro para estruturas.

```
typedef struct pessoa {
      char nome[10];
      int idade;
      float peso;
}Pessoa;

typedef struct paciente {
      float* vetTemperatura;
      Pessoa individuo;
}Paciente;

int main() {
      Paciente *pac_1;
      pac_1 = (Paciente*) malloc(sizeof(Paciente));
      pac_1-> individuo.idade = 30;
      strcpy(pac_1-> individuo.nome,"Ana");
      pac_1-> vetTemperatura = (float*) malloc(5 * sizeof(float));
      pac_1-> vetTemperatura[0] = 37.5;
}
```

Aloca espaço na memória para um vetor com 5 inteiros

Aloca espaço na memória para uma estrutura que contém um ponteiro para float (4 bytes), e uma estrutura pessoa (18 bytes)

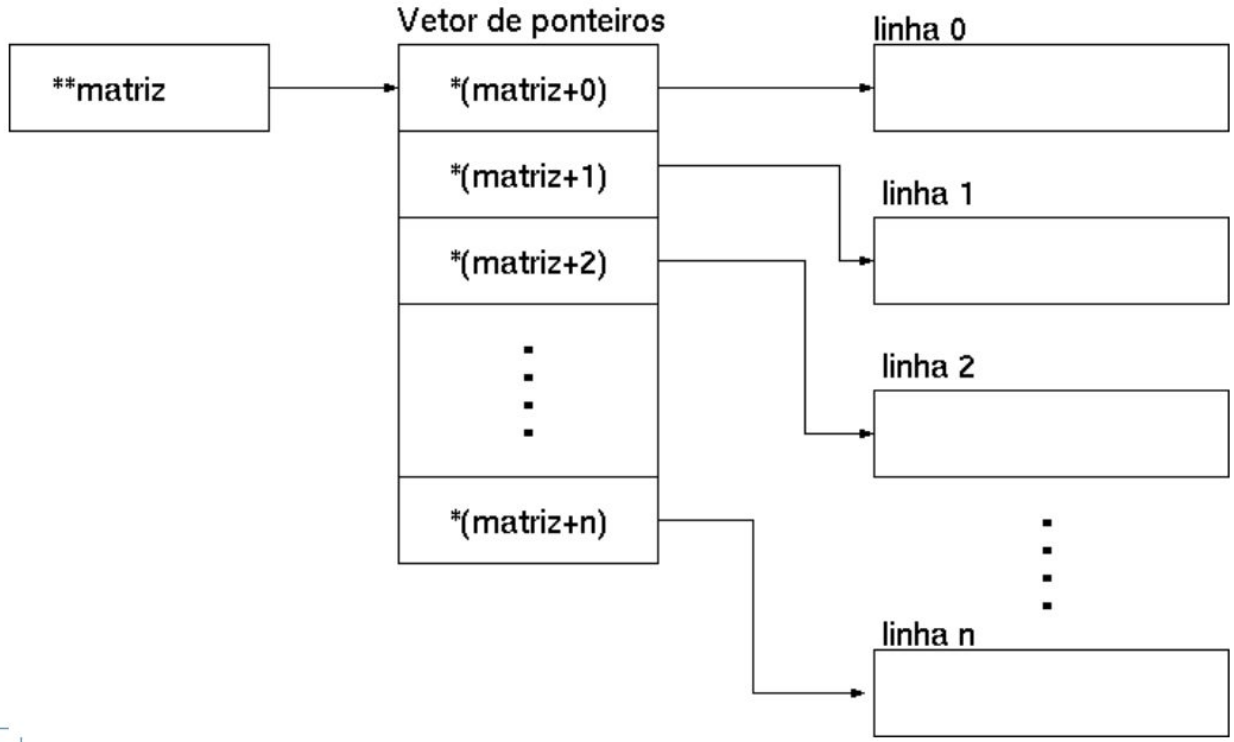
Matrizes Dinamicamente Alocadas

- Deve-se usar um ponteiro para ponteiro
  - `int** matriz;`
- Pode-se pensar em uma matriz dinâmica como um vetor de ponteiros
  - Cada elemento do vetor contém o endereço inicial de uma linha da matriz (vetor-linha)
  - Para alocar uma matriz com **malloc**, é preciso fazer a alocação do vetor de ponteiros e, em seguida, de cada vetor-linha
  - Da mesma forma, a liberação da memória é feita em partes
- Em um ponteiro para ponteiro, cada nível do ponteiro permite criar uma nova dimensão no array.

Ponteiro para ponteiro...

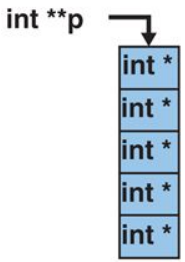
MEMÓRIA		
ENDEREÇO	VARIÁVEL	CONTEÚDO
119		
120	char ***ptrPtr	#122
121		
122	char **ptrPtrChar	#124
123		
124	char *ptrChar	#126
125		
126	char letra	‘a’
127		

Alocando linhas e colunas

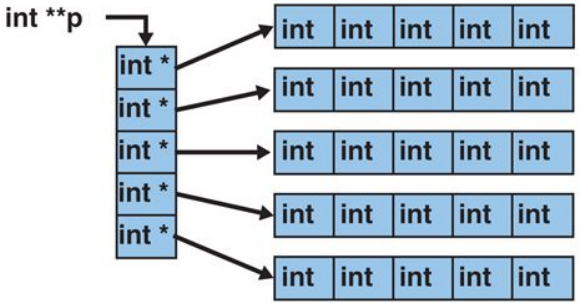


Alocando linhas e colunas

1º malloc
Cria as linhas da matriz



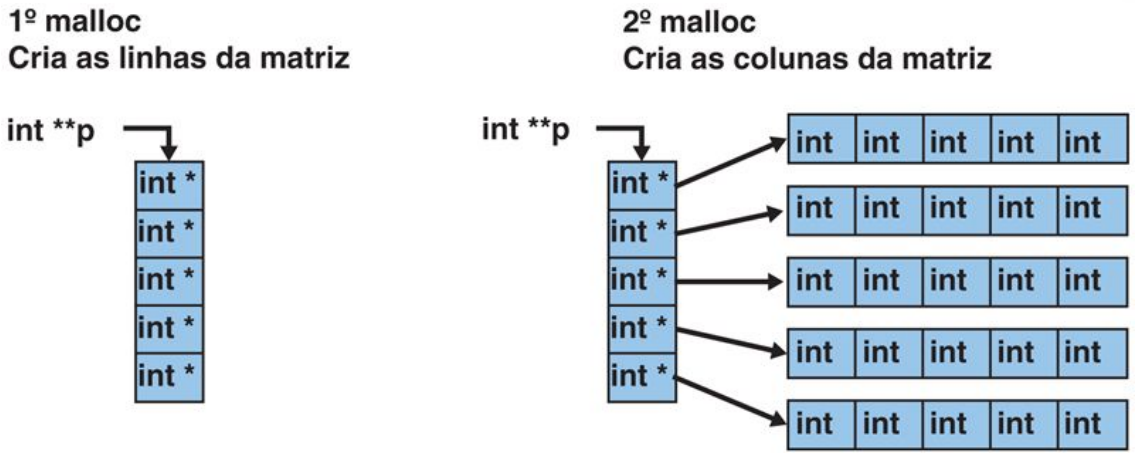
2º malloc
Cria as colunas da matriz



MEMÓRIA		
#	VAR	CONTEÚDO
119	int** p	#121
120		
121	p [0]	#124
122	p [1]	#127
123		
124	p [0][0]	69
125	p [0][1]	74
126		
127	p [1][0]	14
128	p [1][1]	31
129		

Alocando linhas e colunas

- Sempre que se aloca memória, os dados alocados possuem um nível a menos que o do ponteiro usado na alocação:
  - Ponteiro para inteiro (int *), aloca-se um array de inteiros (int).
  - Ponteiro para ponteiro para inteiro (int **), aloca-se um array de ponteiros para inteiros (int *).
  - Ponteiro para ponteiro para ponteiro para inteiro (int ***), aloca-se um array de inteiros (int **).



Alocando linhas e colunas

- Sempre que se aloca memória, os dados alocados possuem um nível a menos que o do ponteiro usado na alocação:

Diferentemente dos arrays de uma dimensão, para liberar da memória um array com mais de uma dimensão é preciso liberar a memória alocada em cada uma de suas dimensões, na ordem inversa da que foi alocada.

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04    int **p; //2 "*" = 2 níveis = 2 dimensões
05    int i, j, N = 2;
06    p = (int **) malloc(N*sizeof(int *));
07    for (i = 0; i < N; i++){
08        p[i] = (int *) malloc(N*sizeof(int));
09        for (j = 0; j < N; j++)
10             scanf("%d",&p[i][j]);
11    }
12    for (i = 0; i < N; i++){
13        free(p[i]);
14    }
15    free(p);
16    system("pause");
17    return 0;
18  }
```

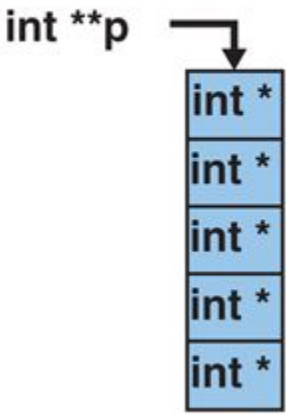
Alocando linhas e colunas, com N != M

- Mas e se quisermos criar linhas com colunas variáveis?

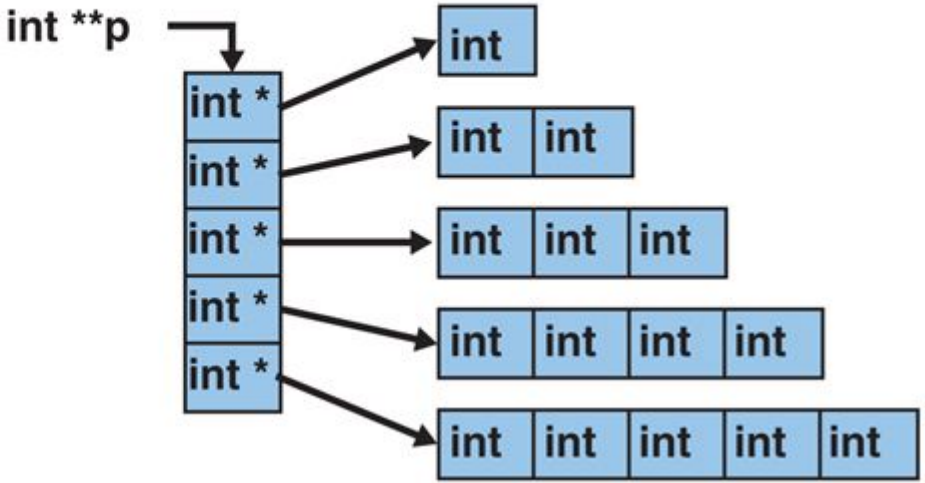
Alocando linhas e colunas, com N != M

-

1º malloc
Cria as linhas da matriz



2º malloc
Cria as colunas da matriz



Alocando linhas e colunas, com N != M

- Sejam N a quantidade de linhas e M, colunas. M = [i .. N-1]

Esse tipo de alocação, usando ponteiro para ponteiro, permite criar matrizes que não sejam quadradas ou retangulares.

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04    int **p; //2 "*" = 2 níveis = 2 dimensões
05    int i, j, N = 3;
06    p = (int **) malloc(N*sizeof(int *));
07    for (i = 0; i < N; i++){
08        p[i] = (int *) malloc((i+1)*sizeof(int));
09        for (j = 0; j < (i+1); j++)
10             scanf("%d",&p[i][j]);
11    }
12    for (i = 0; i < N; i++){
13        free(p[i]);
14    }
15    free(p);
16    system("pause");
17    return 0;
18  }
```

Referências

- SARMENTO, Adriano Augusto de Moraes. **Ponteiros para Estruturas, Outros Tipos de Estruturas**. Recife: Ufpe, 2011. Color.
- Backes, undefined A. **Linguagem C - Completa e Descomplicada**. Disponível em: Grupo GEN, (2nd edição). Grupo GEN, 2018.