

**2º Semestre**

# ***Técnicas de Programação***

**Luiz Fernando Carvalho**

luizfcarvalhoo@gmail.com

# Retornando um ponteiro a partir de uma função

- Além dos tipos primitivos de dados (int, float, char, etc) e de *structs*, as funções podem retornar um ponteiro;
- Qual a utilidade de retornar ponteiros?

# Retornando um ponteiro a partir de uma função

```
1  int *criarVet(int tam){
2      int *vet, i;
3      vet = (int *) malloc(tam * sizeof(int));
4      for(i=0;i<tam;i++)
5          vet[i] = i;
6
7      return vet;
8  }
9
10 int main(){
11     int tam, i, *v;
12
13     printf("Informe o tamanho do vetor:");
14     scanf("%d", &tam);
15
16     v = criarVet(tam);
17
18     printf("O vetor criado na funcao e': ");
19     for(i=0;i<tam;i++)
20         printf("%d ", v[i]);
21
22     return 0;
23 }
```

Faltou desalocar a memória usada pelo vetor!

E agora?!

Qual seria o comando para desalocar?

```

1  int *criarVet(int tam){
2      int *vet, i;
3      vet = (int *) malloc(tam * sizeof(int));
4
5      for(i=0;i<tam;i++)
6          vet[i] = i;
7      printf("\n Ponteiro vet: %p \n", vet);
8
9      return vet;
10 }
11
12 int main(){
13     int tam, i, *v;
14
15     printf("Informe o tamanho do vetor: ");
16     scanf("%d", &tam);
17
18     v = criarVet(tam);
19
20     printf("O vetor criado na funcao e': ");
21     for(i=0;i<tam;i++)
22         printf("%d ", v[i]);
23
24     printf("\n Ponteiro v: %p \n", v);
25     free(v);
26
27     return 0;
28 }

```

```
Informe o tamanho do vetor: 5
```

```

Ponteiro vet: 00D73210
O vetor criado na funcao e': 0 1 2 3 4
Ponteiro v: 00D73210

```

```

Process returned 0 (0x0)   execution time : 1.974 s
Press any key to continue.

```

Porque `free(v)` desaloca o espaço de memória que foi alocado para `vet`?

```

1 void criarVet(int tam){
2     int *vet, i;
3
4     vet = (int *) malloc(tam * sizeof(int));
5
6     for(i=0;i<tam;i++)
7         vet[i] = i;
8 }
9
10 int main(){
11     int tam, i;
12
13     printf("Informe o tamanho do vetor: ");
14     scanf("%d", &tam);
15
16     criarVet(tam);
17
18     printf("O vetor criado na funcao e': ");
19     for(i=0;i<tam;i++)
20         printf("%d ", vet[i]);
21
22     free(vet);
23
24     return 0;
25 }

```

Neste algoritmo o compilar apresenta o seguinte erro:  
 20 error: 'vet' undeclared

Porque isso ocorreu?

Lembre-se que o comando free apenas libera o espaço de memória para ser usado por outras aplicações. O ponteiro ainda continua apontando para esse espaço de memória. Isso pode ser resolvido fazendo o ponteiro "apontar para NULL":  
 ptr = NULL

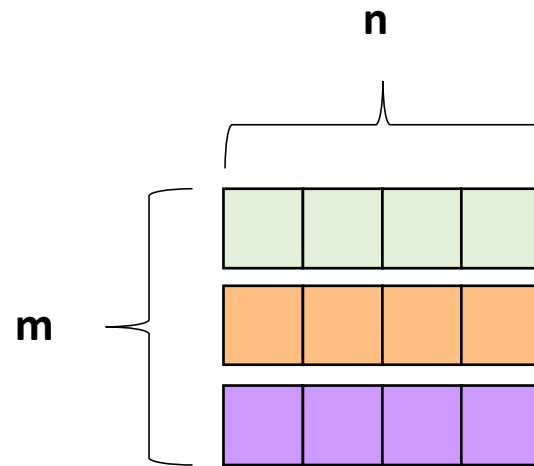
# Retornando um ponteiro a partir de uma função

- Conclusões

1. Como as variáveis alocadas dinamicamente ficam na HEAP, quando a função onde elas foram criadas são encerradas, as variáveis continuam existindo
  - Somente o comando `free` libera a memória alocada na HEAP
2. Retornando o endereço (por meio de um ponteiro) onde a variável está alocada, esta variável pode ser usada e até mesmo desalocada em outras funções;

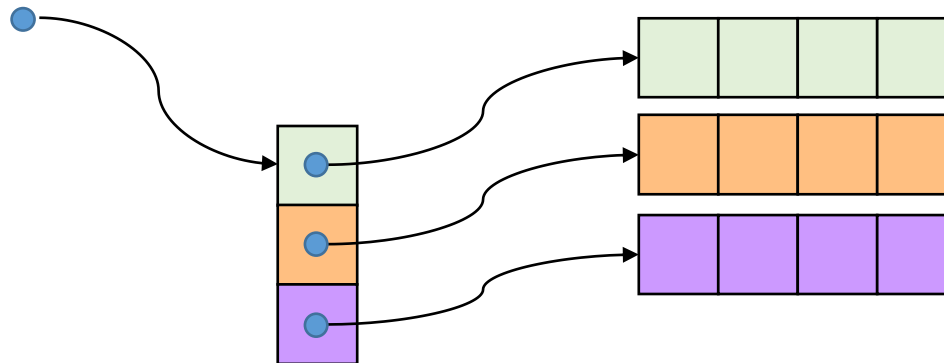
# Alocação dinâmica de matrizes

- Como visto anteriormente, matrizes são implementadas como vetores de vetores;
- Uma matriz com **m** linhas e **n** colunas é um vetor de m elementos cada um sendo um vetor de n elementos



# Alocação dinâmica de matrizes

- Temos então um **vetor de ponteiros**
  - Cada elemento do vetor aponta para o primeiro elemento de outro vetor;
  - Portanto, temos um ponteiro que aponta para um ponteiro que aponta para um valor final
    - O nome disso é indireção múltipla;
    - A indireção pode levar ser levada a qualquer nível.



Usamos o jargão “ponteiro para ponteiro” para indicar indireções



# Alocação dinâmica de matrizes

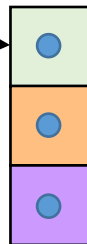
```
1  int i, L, C, **A; → Ponteiro para ponteiro
2
3  printf("Informe a quantidade de linhas: ");
4  scanf("%d", &L);
5  printf("Informe a quantidade de colunas: ");
6  scanf("%d", &C);
7
8  A = (int **) malloc(L * sizeof(int *)); // Aloca as L linhas da matriz A
9  for (i = 0; i < L; i++)
10     A[i] = (int *) malloc(C * sizeof(int)); // Aloca as C colunas de cada linha
```

Linha 1

A

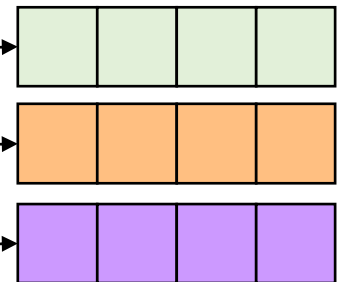
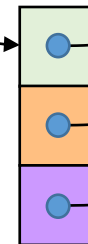
Linha 8

A



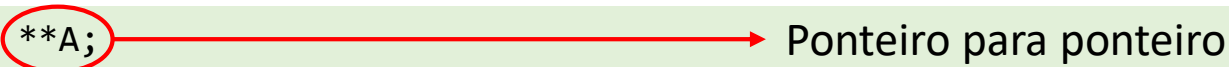
Linha 9 e 10

A



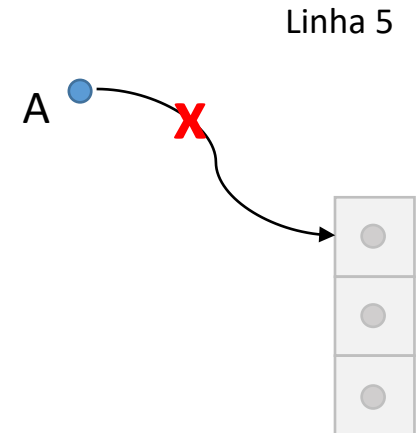
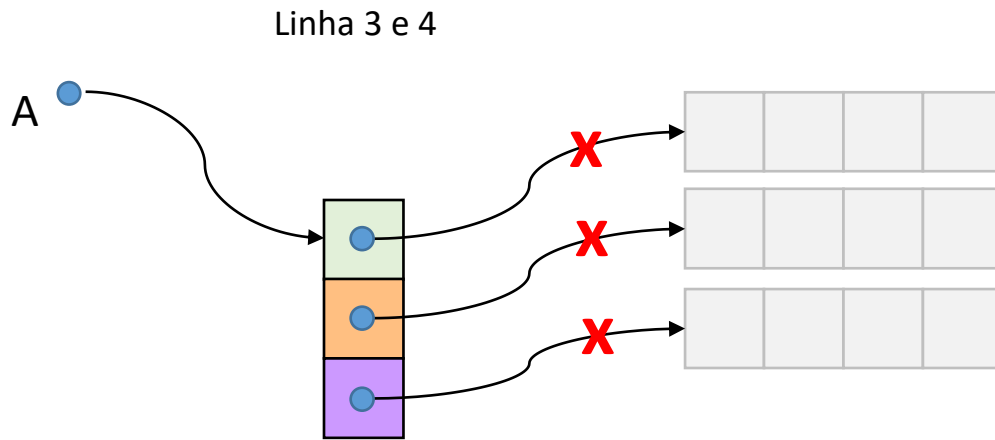
# Alocação dinâmica de matrizes

```
1  int i, L, C, **A;
2
3  printf("Informe a quantidade de linhas: ");
4  scanf("%d", &L);
5  printf("Informe a quantidade de colunas: ");
6  scanf("%d", &C);
7
8  A = (int **) malloc(L * sizeof(int *)); // Aloca as L linhas da matriz A
9  for (i = 0; i < L; i++)
10     A[i] = (int *) malloc(C * sizeof(int)); // Aloca as C colunas de cada linha
11     ...
12
13  for (i = 0; i < L; i++)
14     free(A[i]); // Libera as C colunas de cada linha
15  free(A);
```

 **Ponteiro para ponteiro**

# Alocação dinâmica de matrizes

```
1 ...  
2  
3 for (i = 0; i < L; i++)  
4     free(A[i]); // Libera as C colunas de cada linha  
5 free(A);
```



# Exemplo

```
1  int i, *p;
2
3  p = calloc(5, sizeof(int));
4
5  for(i = 0; i < 5; i++)
6      p[i] = i+1;
7  for(i = 0; i < 5; i++)
8      printf("%d ", p[i]);
9
10 printf("\n");
11 p = realloc(p, 3*sizeof(int)); //Diminui o tamanho do array
12 for(i = 0; i < 3; i++)
13     printf("%d ", p[i]);
14
15 printf("\n");
16 p = realloc(p, 10 * sizeof(int)); //Aumenta o tamanho do array
17 for(i = 0; i < 10; i++)
18     printf("%d ", p[i]);
```

# Exercícios

- Crie um algoritmo para realizar multiplicação de matrizes alocadas dinamicamente. O usuário deve fornecer os valores de ambas as matrizes e o seus valores;
- Modifique o programa anterior para ler as duas matrizes a partir de arquivos, uma de cada arquivo;

# Função calloc

- A função calloc também serve para alocar memória, mas possui um protótipo um pouco diferente

```
void *calloc(int numero_elementos, int tamanho_elemento)
```

- Essa função tem efeito parecido da malloc.
- A diferença é que são passados dois parâmetros
  - Numero de elementos a serem alocados;
  - Tamanho de cada elemento a ser alocado;
- Todos os elementos são iniciados como zero (todos os *bits* zero);

# Função calloc

```
void *calloc(int numero_elementos, int tamanho_elemento)
```

```
1 int *p;  
2 p = (int *) malloc(50*sizeof(int));  
3 if(p == NULL)  
4     printf("Erro: Memoria insuficiente");
```

```
1 int *p;  
2 p = (int *) calloc(50, sizeof(int));  
3 if(p == NULL)  
4     printf("Erro: Memoria insuficiente");
```

# Função realloc

- A função `realloc` serve para realocar memória e tem o seguinte protótipo

```
void *realloc(void *ptr, int tamanho)
```

- A função modifica o tamanho da memória previamente alocada e apontada por `*ptr`
- O tamanho pode ser maior ou menor que o original
  - Caso um bloco maior precisar ser alocado, o SO pode encontrar outro local na memória HEAP para armazenar a nova variável
    - Um ponteiro para o bloco é devolvido porque `realloc` pode precisar mover o bloco na memória;
    - O conteúdo do bloco antigo é copiado no novo bloco e nenhuma informação é perdida;
    - Se a função falhar em realocar memória, um ponteiro `NULL` é retornado.