

# Ponteiros & Alocação

*Aula 14* | versão 1.2

**Marcos Silvano Almeida**

*[marcoossilvano@professores.utfpr.edu.br](mailto:marcoossilvano@professores.utfpr.edu.br)*

Departamento de Computação

UTFPR Campo Mourão

# Roteiro

- Layout de programa C na memória
- Tipos de alocação
  - Alocação automática
  - Alocação global/estática
  - Alocação manual de memória
- Exemplo: Alocação de vetores bidimensionais
- Retornando vetor de uma função

# Layout da memória de um programa C

- O código asm (assembly) gerado pelo compilador C organizar o código em seções. De forma simplificada, temos as seguintes seções:
  - Instruções
    - Código do programa
  - Variáveis globais e estáticas
    - Variáveis de escopo global ou estáticas locais
  - Heap
    - Área de alocação livre (alocação manual)
  - Stack
    - Pilha de chamada de funções
    - Aloca variáveis declaradas nas funções
- Até agora, temos utilizado somente variáveis de **alocação automática**

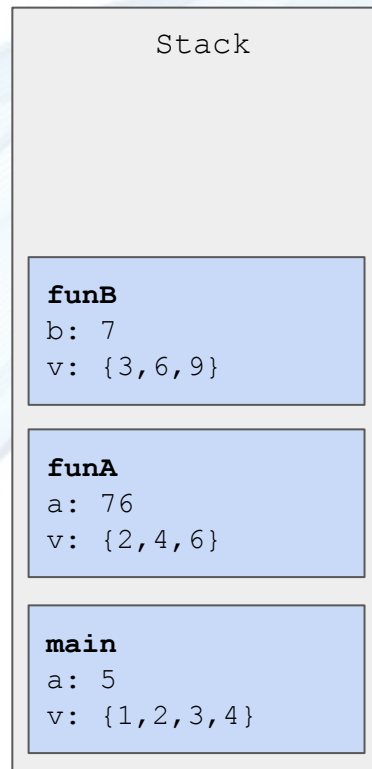


## Alocação Automática em Pilha

# Alocação: automática

- Variáveis alocadas e liberadas automaticamente dentro do frame/instância das funções, na pilha de chamadas

```
void funB() {  
    int b = 7;  
    int v = {3, 6, 9};  
}  
  
void funA() {  
    int a = 76;  
    int v = {2, 4, 6};  
    funB();  
}  
  
int main() {  
    int a = 5;  
    int v[] = {1, 2, 3, 4};  
    funA();  
}
```



## Alocação Global e Estática

# Alocação: global ou estática

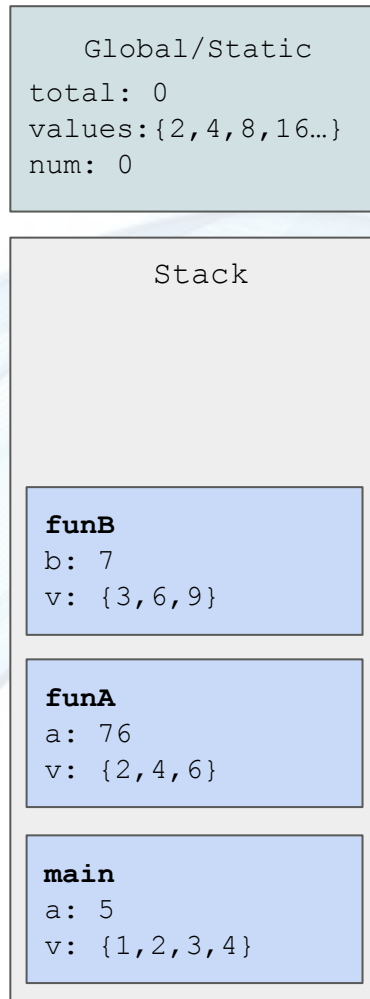
- Variáveis alocadas durante todo o programa

```
int total = 0;
int values[6] = {2,4,8,16,32,64};

void funB() {
    static num = 0;
    ...
}

void funA() { ... }

int main() {
    int a = 5;
    int v[] = {1,2,3,4};
    funA();
}
```



## Alocação Manual em Heap





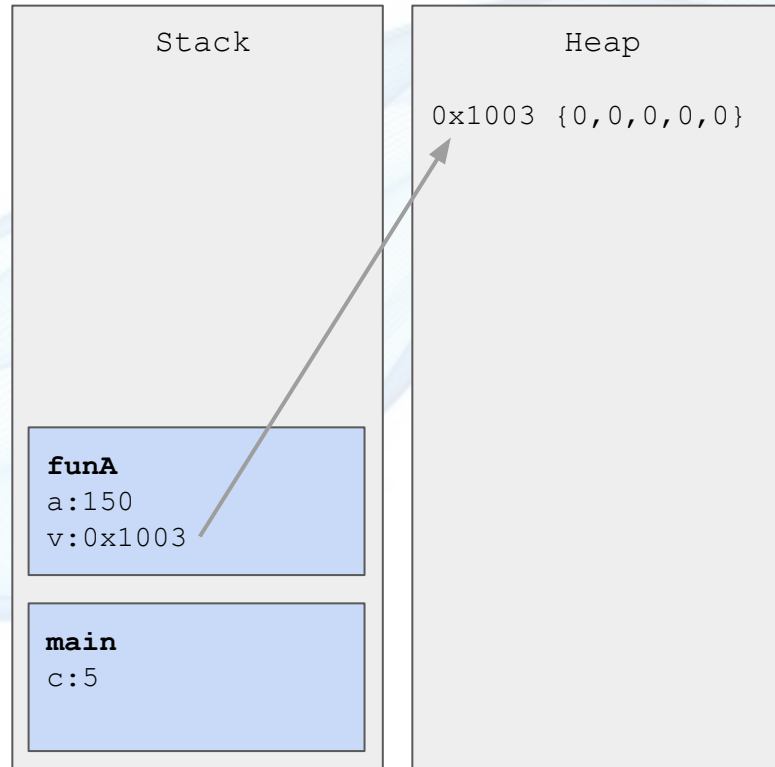
# Alocação: manual em heap

- Alocação e liberação é feita manualmente: malloc() e free()

```
#include <stdio.h>
#include <stdlib.h>

int funA() {
    int a = 150;
    int *v = malloc(5 * sizeof(int));
    // faz algo com o array...
    free(v);
}

int main() {
    int c = 5;
    funA();
    return 0;
}
```



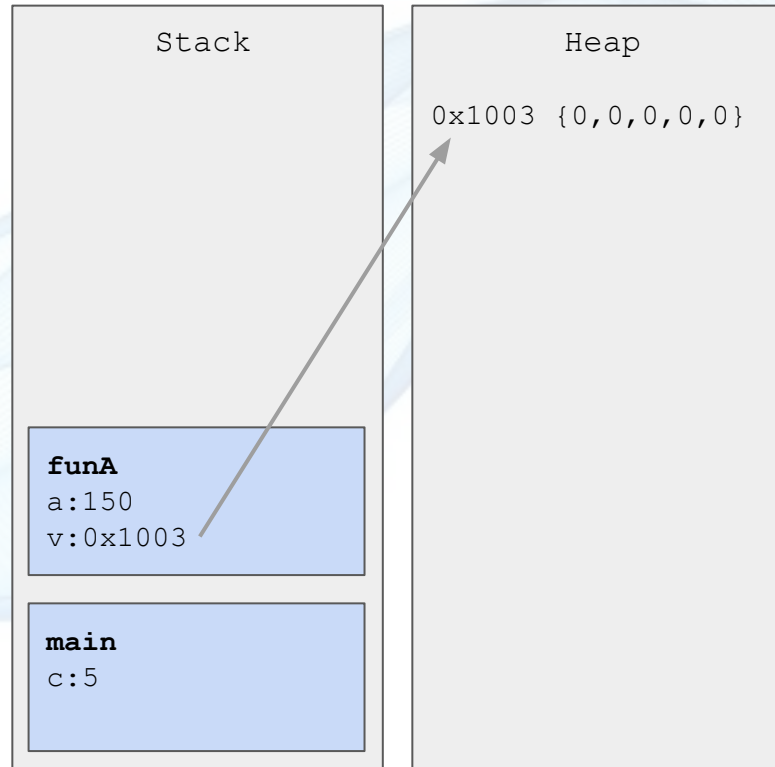
# Alocação: manual em heap

- Boas práticas em alocação e liberação

```
#include <stdio.h>
#include <stdlib.h>

int funA() {
    int a = 150;
    int *v = malloc(5 * sizeof(int));
    if (v == NULL) return; // conseguiu alocar?
    // faz algo com o array...
    free(v);
    v = NULL; // ponteiro nulo (inválido)
}

int main() {
    int c = 5;
    funA();
    return 0;
}
```





# Alocação manual em heap vs automática na stack

- Vetor alocado na heap vs stack

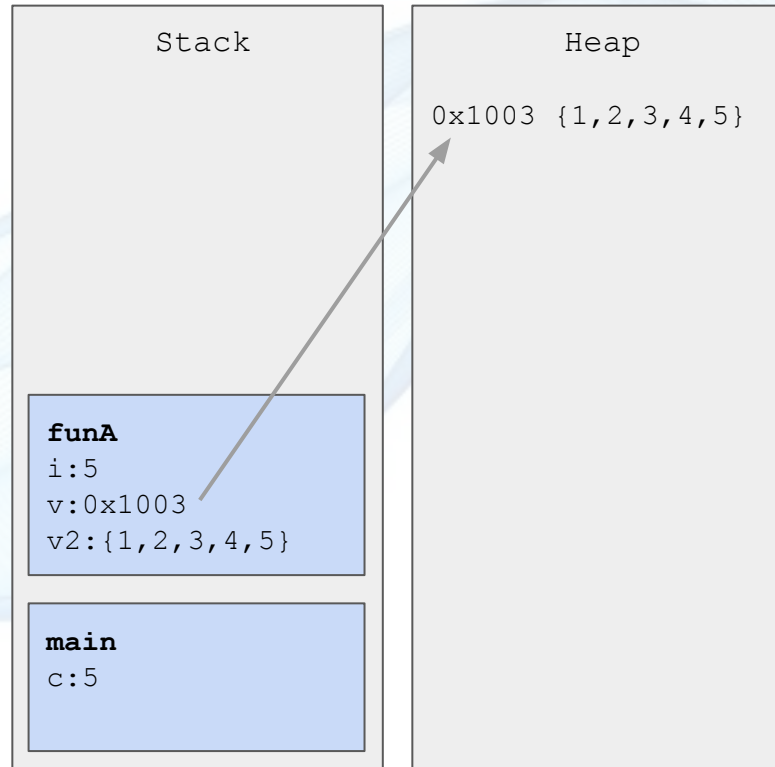
```
#include <stdio.h>
#include <stdlib.h>

int funA() {
    int *v1 = malloc(5 * sizeof(int));
    for (int i = 0; i < 5; i++) v1[i] = (i+1);

    int v2[5] = {1,2,3,4,5};

    free(v);
}

int main() {
    int c = 5;
    funA();
    return 0;
}
```



## Manipulação de vetor alocado em Heap

# Manipulando vetor alocado em heap



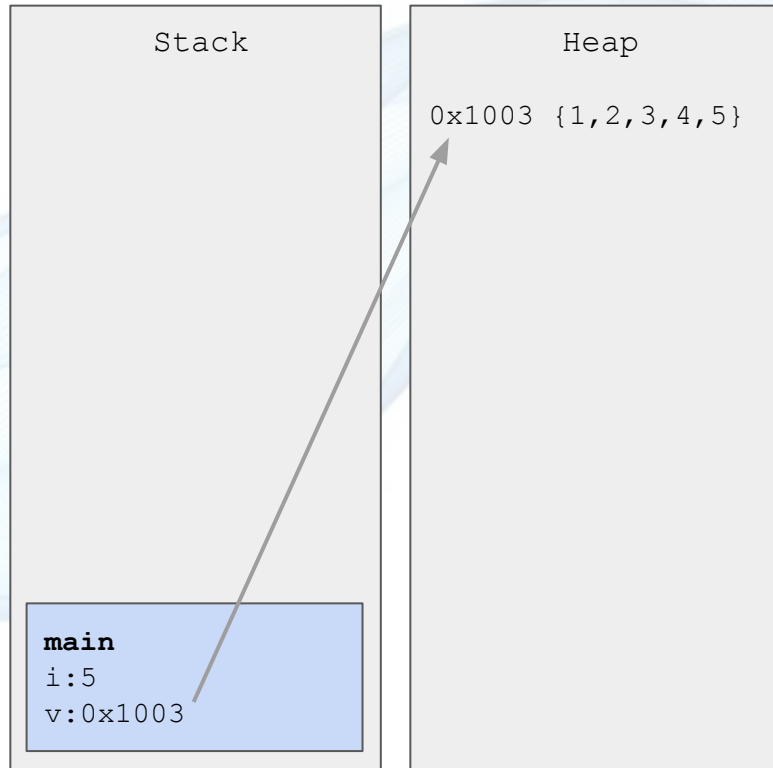
```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int *v = malloc(5 * sizeof(int));

    // acesso aos elementos do vetor ocorre
    // normalmente por índice ou ponteiro
    for (int i = 0; i < 5; i++) {
        v[i] = (i+1);
    }

    int *p = v;
    for (int i = 0; i < 5; i++) {
        printf("%d\n", *p);
        p++;
    }

    free(v);
    return 0;
}
```



Alocação de matriz (vetor 2D)

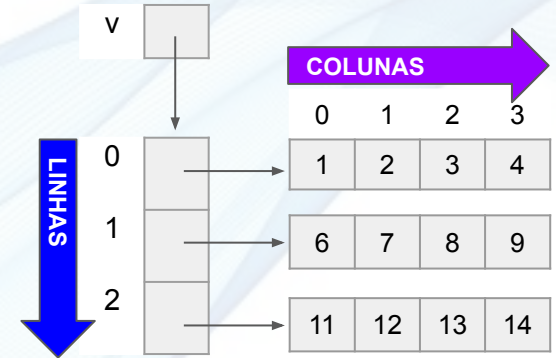
# Alocando matriz 3x4

- Para alocar vetores de 2 ou mais dimensões, é preciso usar ponteiros de ponteiros

```
// Passo 1: alocar array com 3 ponteiros
// (Apontará para os vetores de inteiros das linhas)
int** v = malloc(3 * sizeof(int*));

int k = 1;
// Passo 2: alocar cada vetor das 3 linhas
for (int i = 0; i < 3; i++) {
    // Cada linha é um vetor de 4 colunas
    v[i] = malloc(4 * sizeof(int));

    // Neste exemplo, preenche linhas com contagem
    for (int j = 0; j < 4; j++) {
        v[i][j] = k;
        k++;
    }
}
```



# Liberando matriz 3x4

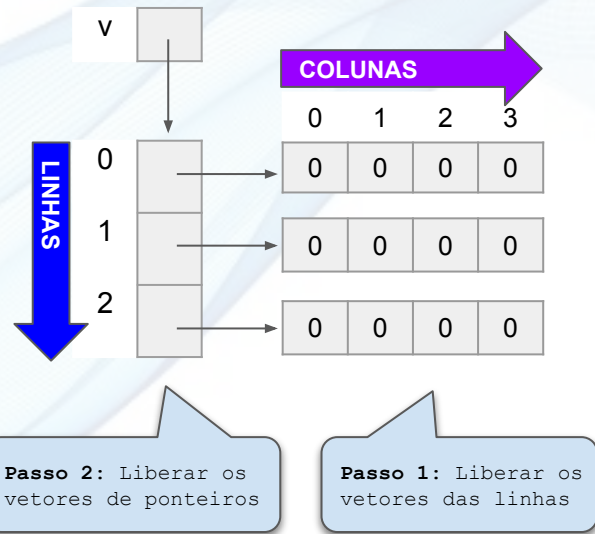
- Para liberar matriz, é preciso seguir na ordem contrária à alocação

```
// ##### ALOCAR MATRIZ 3x4 #####  
// Passo 1: alocar array com 3 ponteiros  
// (Apontará para os vetores de inteiros das linhas)  
int** v = malloc(3 * sizeof(int*));
```

```
// Passo 2: alocar cada vetor das 3 linhas  
for (int i = 0; i < 3; i++) {  
    // Cada linha é um vetor de 4 colunas  
    v[i] = malloc(4 * sizeof(int));  
}
```

```
// ##### LIBERAR MATRIZ 3x4 #####  
// Passo 1: liberar cada vetor das 3 linhas  
for (int i = 0; i < 3; i++) {  
    free(v[i]);  
}
```

```
// Passo 2: liberar o vetor principal de 3 ponteiros  
free(v);
```





# Matriz: alocação manual em heap vs automática em stack

- Alocação na stack é contígua, mas na heap, não.

```
int v0[3][4] = { // array 2D alocado
    {1, 2, 3, 4}, // automaticamente na stack
    {5, 6, 7, 8},
    {9,10,11,12}
};
```

v0

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

```
int** v1 = malloc(3 * sizeof(int*)); // array de ponteiros das linhas
int k = 1;
// arrays dos valores de cada linha (colunas)
for (int i = 0; i < 3; i++) {
    v1[i] = malloc(4 * sizeof(int));
    // Neste exemplo, preenche linhas com contagem
    for (int j = 0; j < 4; j++) {
        v1[i][j] = k;
        k++;
    }
}
```

v1

	0	1	2	3
0	1	2	3	4
1	6	7	8	9
2	11	12	13	14

**COLONAS** →

**LINHAS** ↓

Criando e devolvendo vetor de função



# Criando e devolvendo vetor de função

- Para devolvermos um vetor criado em uma função, o mesmo precisa ser alocado na heap.
  - Se for alocado automaticamente na pilha, deixará de existir ao término da função.

```
int* create_vector(int n, int value) {  
    int* vec = malloc(sizeof(int) * n); // n * 4 bytes (tamanho do inteiro)  
  
    for (int i = 0; i < n; i++) {  
        vec[i] = value;  
    }  
    return vec; // retorna endereço do vetor  
}  
  
int main() {  
    int* v = create_vector(5, 0);  
    // faz algo com o vetor...  
    free(v); // libera  
    return 0;  
}
```

# Criando e devolvendo matriz de função

- Podemos criar duas funções, para alocar matriz e para liberar matriz

```
/*  EXEMPLO MATRIZ 3x4
    STACK (função)      HEAP
    v -----> [ ]----> [1, 2, 3, 4]
                                [ ]----> [5, 6, 7, 8]
                                [ ]----> [9,10,11,12]
*/
int** create_vector2D(int rows, int cols, int value) {
    int** v = malloc(rows * sizeof(int*));
    for (int i = 0; i < rows; i++) {
        v[i] = malloc(cols * sizeof(int));
    }
    return v;
}

int** free_vector2D(int rows, int** v) {
    for (int i = 0; i < rows; i++) free(v[i]);
    free(v);
    return NULL;
}
```

```
// Utilizando as funções para
// alocar e liberar matriz
int main() {
    int** v = create_vector2D(3,4);

    // faz algo com a matriz...

    v = free_vector2D(3);
    return 0;
}
```

# Criando e devolvendo vetor de structs em função

- O processo é o mesmo para vetores de qualquer outro elemento

```
#include <stdio.h>
#include <stdlib.h>

struct item {
    int id;
    char name[51];
};

typedef struct item Item;

Item* create_items(int n) {
    Item* items = malloc(n * sizeof(Item));
    for (int i = 0; i < n; i++) {
        items[i].id = i+1;
        snprintf(items[i].name, 51, "Item %d", i+1);
    }
    return items;
}
```

```
void print_items(int n, Item* v) {
    for (int i = 0; i < n; i++) {
        printf("%d, %s\n", v->id, v->name);
        v++;
    }
    //Alternativa:
    //printf("%d,%s\n",v[i].id ,v[i].name);
}

int main() {
    Item* items_list = create_items(5);
    print_items(5, items_list);
    free(items_list);

    return 0;
}
```

# Referências

- Algoritmos e Programação
  - Marcela Gonçalves dos Santos
  - Disponível pelo Moodle
- Estruturas de Dados, Waldemar Celes e José Lucas Rangel
  - PUC-RIO - Curso de Engenharia
  - Disponível pelo Moodle
- Linguagem C, Silvio do Lago Pereira
  - USP - Instituto de Matemática e Estatística
  - Disponível pelo Moodle
- Curso Interativo da Linguagem C
  - <https://www.tutorialspoint.com/cprogramming>