



Ciência da Computação
Algoritmos e Estrutura de Dados 1

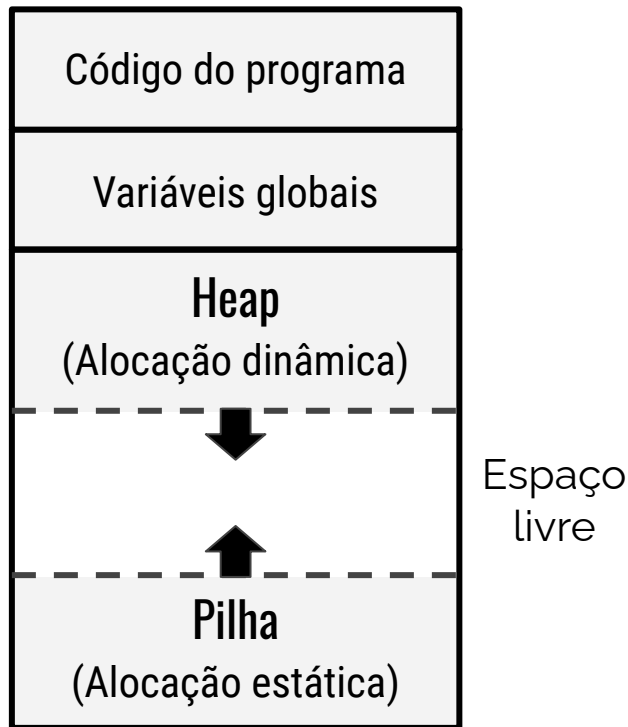
Memória e Alocação Dinâmica

Agenda

- Organização da memória
 - Pilha vs Heap
- Alocação Dinâmica
 - Funções da biblioteca stdlib.h
- Criando Vetores, Matrizes e Structs dinamicamente
 - Vetor
 - Matriz
 - Struct

Alocação de memória

- Esquema didático da distribuição da memória



Alocação de memória

■ Alocação Estática

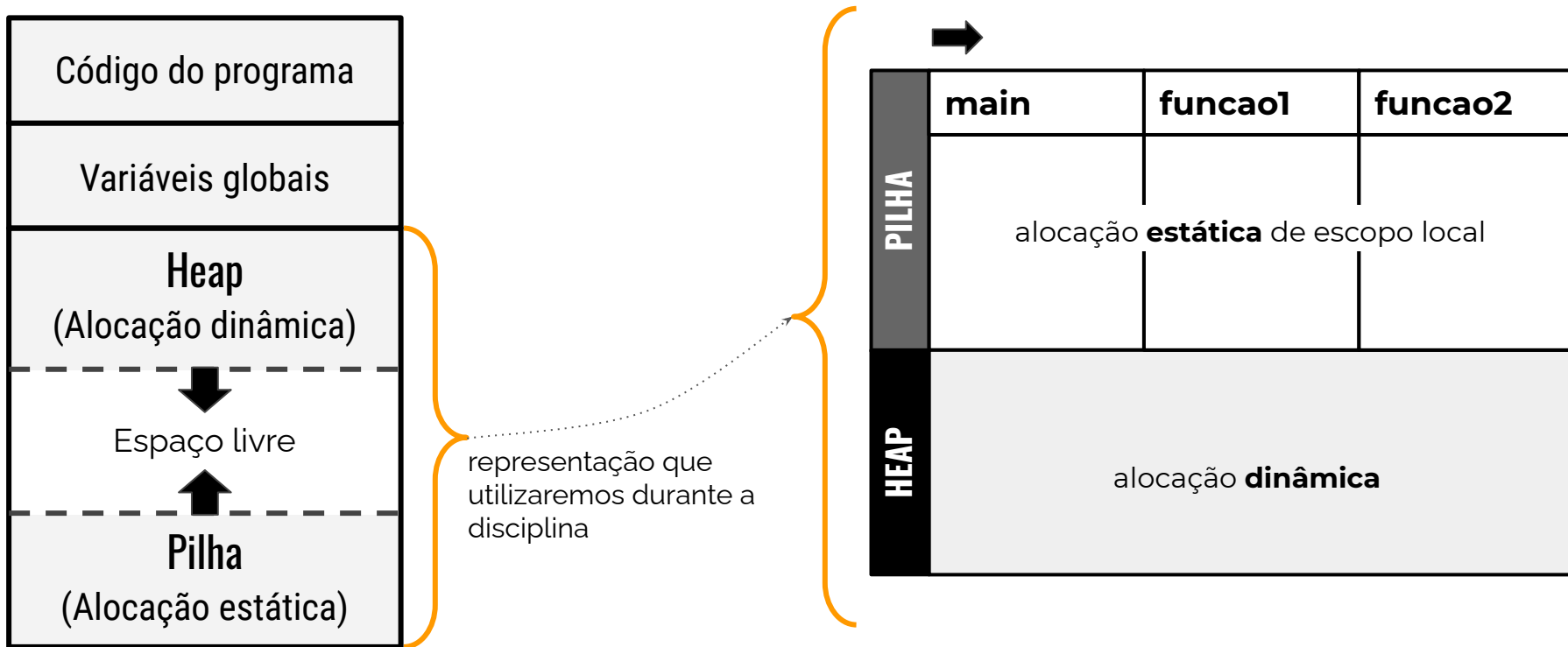
- O tamanho do espaço é definido durante a codificação (**tempo de compilação**)
- **Não precisamos nos preocupar com o gerenciamento do espaço alocado.** A vida útil do espaço é definido pelo escopo em que a variável foi declarada
- Escopo **local** ou **global**
 - variáveis locais
 - variáveis globais

■ Alocação Dinâmica

- O tamanho do espaço é definido durante a execução (**tempo de execução**)
- O gerenciamento do espaço alocado é de **responsabilidade do programador.**
- Escopo **global**

Alocação de memória

■ Esquema didático da distribuição da memória



Funções para Alocação Dinâmica

- `void *malloc(size_t tam)`
- `void *calloc(size_t qtde, size_t tam)`
- `void free(void *ptr)`

Alocação dinâmica Funções da biblioteca <stdlib.h>

■ **void** *malloc(**size_t** tam) [referência]

Aloca espaço na memória heap e devolve o endereço do primeiro byte.

Parâmetros

<tam> - tamanho em bytes do espaço a ser alocado.

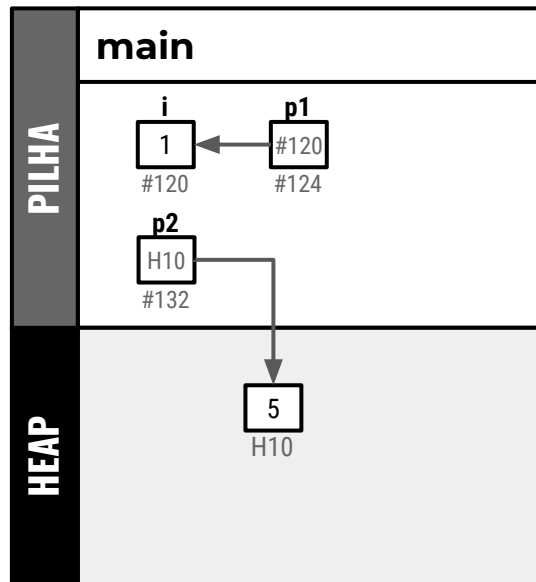
Retorno

void*

- V** endereço do primeiro byte do espaço alocado,
- F** ou endereço nulo (NULL) quando não houver espaço suficiente.

Exemplo

```
int i = 1;
int *p1 = &i;
int *p2;
p2 = (int*) malloc (sizeof(int));
*p2 = 5;
```



Alocação dinâmica Funções da biblioteca `<stdlib.h>`

■ `void *malloc(size_t tam)` [referência]

Aloca espaço na memória heap e devolve o endereço do primeiro byte.

Parâmetros

`< tam >` - tamanho em bytes do espaço a ser alocado.

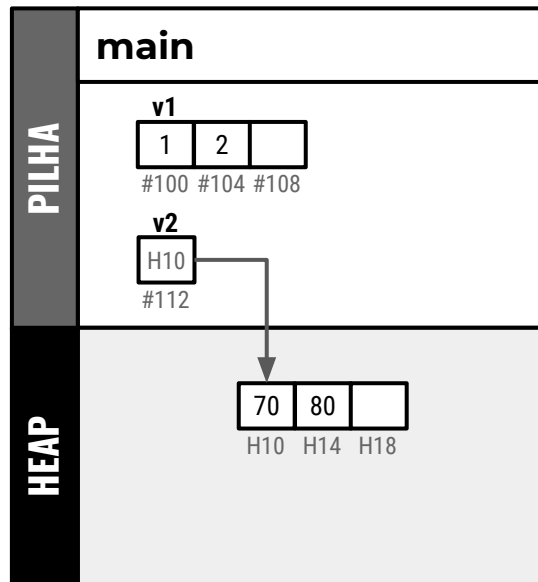
Retorno

`void*`

- V** endereço do primeiro byte do espaço alocado,
- F** ou endereço nulo (NULL) quando não houver espaço suficiente.

Exemplo - Alocando um vetor

```
int v1[3];
int* v2 = (int*) malloc (3 * sizeof(int));
v1[0] = 1;
*(v1+1) = 2;
v2[0] = 70;
*(v2+1) = 80;
```



Alocação dinâmica Funções da biblioteca <stdlib.h>

■ `void *calloc(size_t qtde, size_t tam)` [referência]

Aloca espaço na memória heap e devolve o endereço do primeiro byte.

Atribui o byte 0 em todas as posições de memória alocadas.

Parâmetros

<qtde> - quantidade de itens a ser alocados..

<tam> - tamanho em bytes de cada item.

Retorno

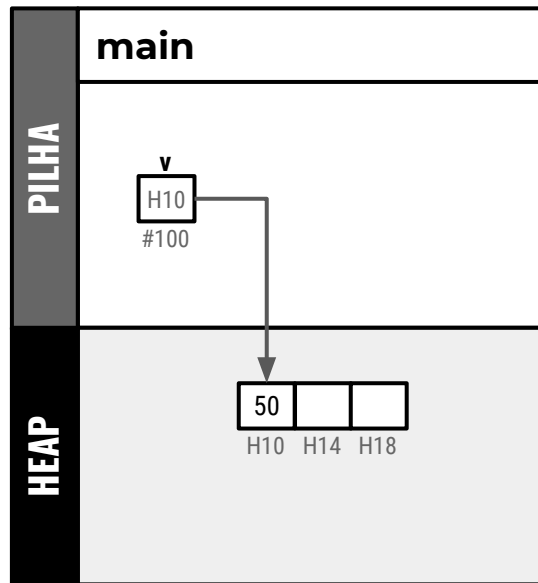
`void*`

V endereço do primeiro byte do espaço alocado,

F ou endereço nulo (NULL) quando não houver espaço suficiente.

Exemplo - Alocando um vetor

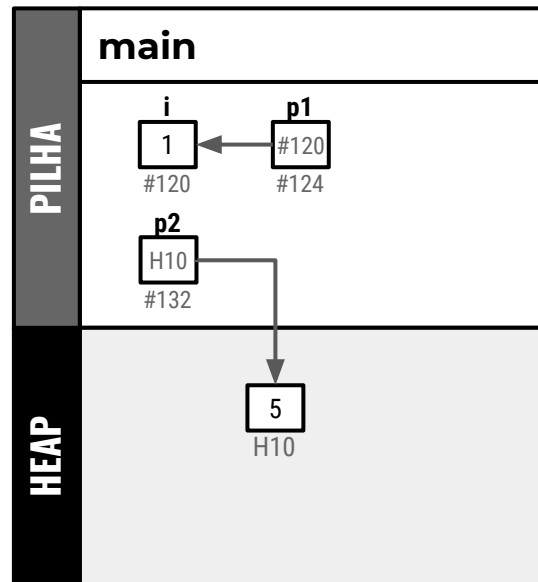
```
int *v;  
v = (int*) calloc (3, sizeof(int));  
v[0] = 50;
```



Alocação dinâmica Funções da biblioteca `<stdlib.h>`

■ Detalhes

- As funções reservam o espaço na HEAP e devolvem um ponteiro genérico **void***.
- É prudente fazermos a conversão explícita para o tipo específico para o qual alocamos o espaço. A conversão explícita é realizada pelo [operador de coerção \(cast\)](#)
- Perceba que os espaços alocados na HEAP não possuem rótulos. Isso significa que o programador é responsável por decidir quando o espaço será desalocado.
- Por este motivo, precisamos tomar cuidado para não perder o endereço de memória alocado na heap.



Alocação dinâmica Funções da biblioteca <stdlib.h>

■ `void free(void *ptr) [ref]`

Libera o espaço alocado na HEAP.

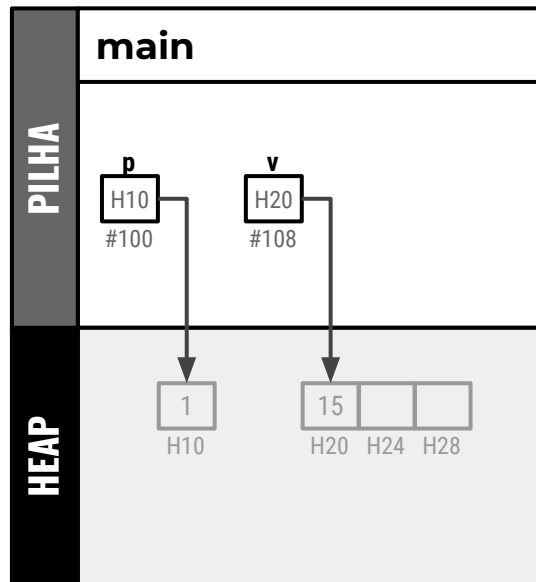
Parâmetros

<ptr> - endereço do espaço a ser liberado.

Exemplo

```
int *p = (int*) malloc(sizeof(int);
int *v1 = (int*) malloc (3 * sizeof(int));
*p = 1;
v1[0] = 15;

free(p);
free(v1);
```



Alocação dinâmica Funções da biblioteca <stdlib.h>

■ `void free(void *ptr) [ref]`

Libera o espaço alocado na HEAP.

Parâmetros

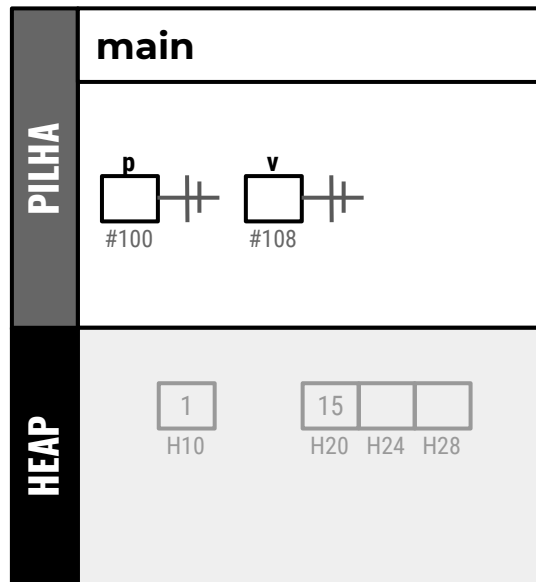
<ptr> - endereço do espaço a ser liberado.

Exemplo

```
int *p = (int*) malloc(sizeof(int);
int *v1 = (int*) malloc (3 * sizeof(int));
*p = 1;
v1[0] = 15;
```

```
free(p);
free(v1);
```

```
p = NULL;
v1 = NULL; } Boa prática
```



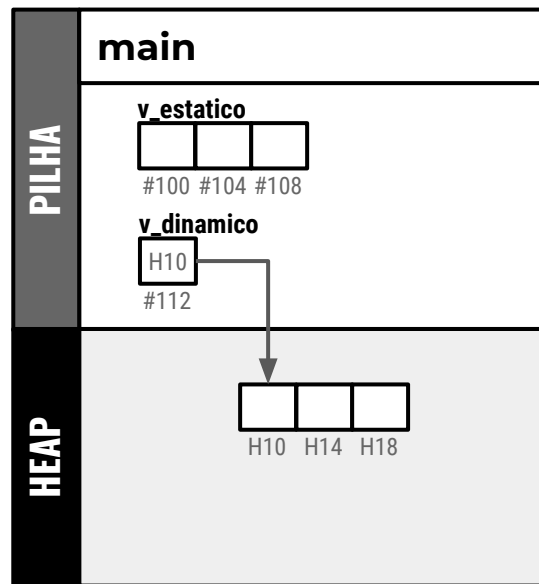
Criando vetores, matrizes e structs dinamicamente

Vetor



Independente da forma de alocação do vetor (estática ou dinâmica), a forma de manipulação dos dados se mantém a mesma (Notação de colchetes ou ponteiros).

ALOCÇÃO ESTÁTICA ***VS*** ***ALOCÇÃO DINÂMICA***



Vetor

Alocando
memória

Manipulando
os dados

Liberando
memória

Passagem por
parâmetro

[\[Código no Repl.it\]](#)

■ Criando o vetor

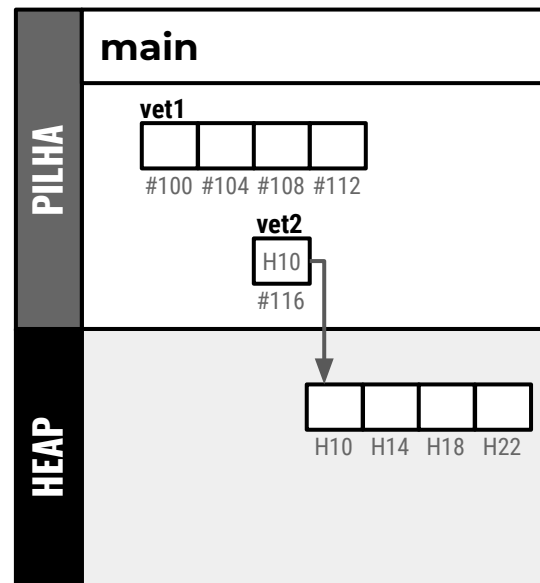
ALOCÇÃO ESTÁTICA

VS

ALOCÇÃO DINÂMICA

```
int vet1[4];
```

```
int* vet2;  
vet2 =(int*)calloc(4, sizeof(int));
```



Vetor

Alocando
memória

Manipulando
os dados

Liberando
memória

Passagem por
parâmetro

[\[Código no Repl.it\]](#)

■ Manipulando os dados

ALOCÇÃO ESTÁTICA

VS ALOCAÇÃO DINÂMICA

```
int vet1[4];
```

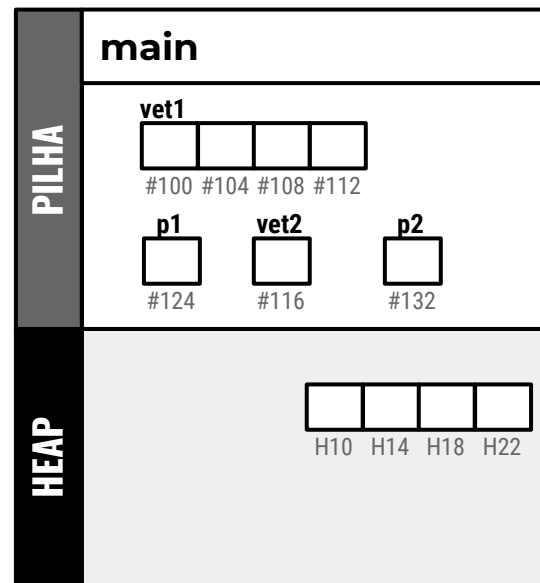
```
int* p1 = vet1;
```

```
vet1[0] = 10;  
*(vet1 + 1) = 20;  
p1[2] = 30;  
*(p1+3) = 40;
```

```
int* vet2;  
vet2 =(int*)calloc(4, sizeof(int));
```

```
int* p2 = vet2;
```

```
vet2[0] = 50;  
*(vet2 + 1) = 60;  
p2[2] = 70;  
*(p2+3) = 80;
```



Vetor

Alocando
memória

Manipulando
os dados

Liberando
memória

Passagem por
parâmetro

[\[Código no Repl.it\]](#)

Manipulando os dados

ALOCÇÃO ESTÁTICA

VS ALOCAÇÃO DINÂMICA

```
int vet1[4];
```

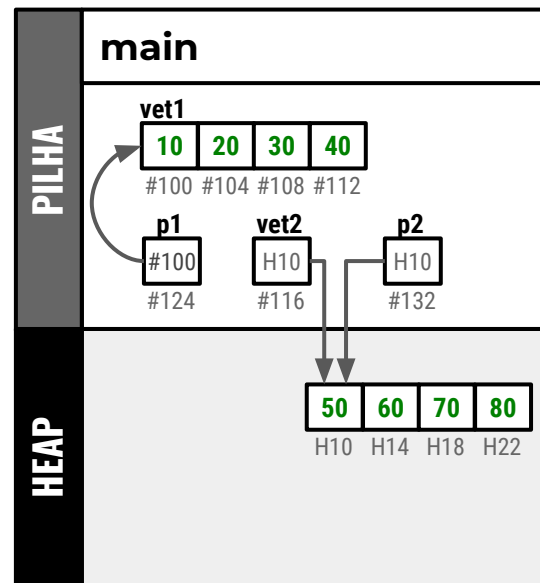
```
int* p1 = vet1;
```

```
vet1[0] = 10;  
*(vet1 + 1) = 20;  
p1[2] = 30;  
*(p1+3) = 40;
```

```
int* vet2;  
vet2 = (int*)calloc(4, sizeof(int));
```

```
int* p2 = vet2;
```

```
vet2[0] = 50;  
*(vet2 + 1) = 60;  
p2[2] = 70;  
*(p2+3) = 80;
```



Vetor

Alocando
memória

Manipulando
os dados

Liberando
memória

Passagem por
parâmetro

[\[Código no Repl.it\]](#)

Desalocando o vetor

ALOCAÇÃO ESTÁTICA

VS ALOCAÇÃO DINÂMICA

```
int vet1[4];
```

```
int* p1 = vet1;
```

```
vet1[0] = 10;  
*(vet1 + 1) = 20;  
p1[2] = 30;  
*(p1+3) = 40;
```

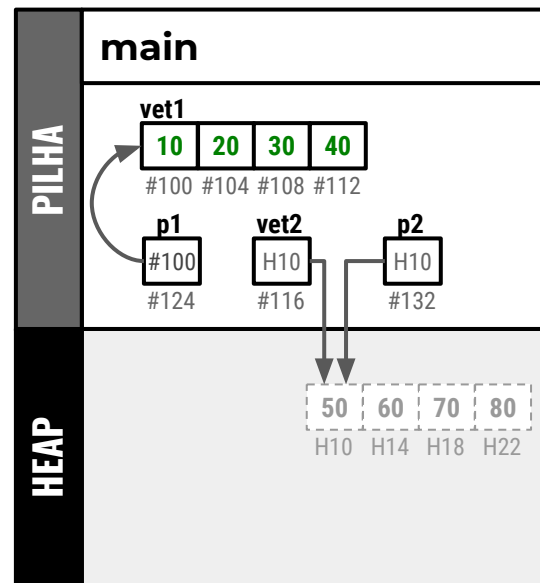
A liberação do espaço é realizada automaticamente de acordo com o escopo em que a variável foi declarada

```
int* vet2;  
vet2 = (int*)calloc(4, sizeof(int));
```

```
int* p2 = vet2;
```

```
vet2[0] = 50;  
*(vet2 + 1) = 60;  
p2[2] = 70;  
*(p2+3) = 80;
```

```
free(vet2);
```



Vetor

Alocando
memória

Manipulando
os dados

Liberando
memória

Passagem por
parâmetro

[\[Código no Repl.it\]](#)

Desalocando o vetor

ALOCAÇÃO ESTÁTICA

VS

ALOCAÇÃO DINÂMICA

```
int vet1[4];
```

```
int* p1 = vet1;
```

```
vet1[0] = 10;  
*(vet1 + 1) = 20;  
p1[2] = 30;  
*(p1+3) = 40;
```

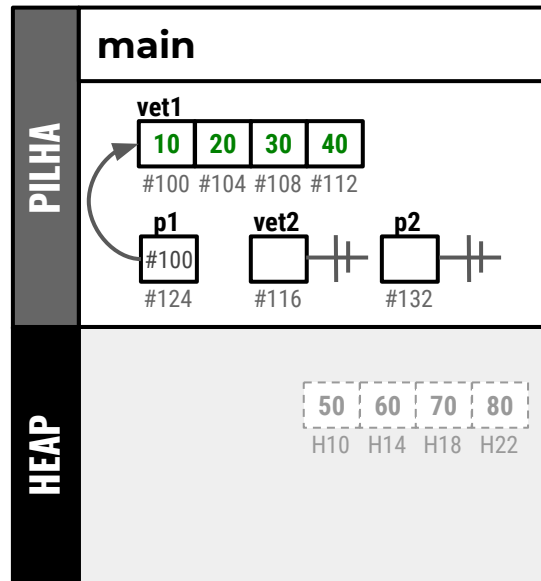
A liberação do espaço é realizada automaticamente de acordo com o escopo em que a variável foi declarada

```
int* vet2;  
vet2 = (int*)calloc(4, sizeof(int));
```

```
int* p2 = vet2;
```

```
vet2[0] = 50;  
*(vet2 + 1) = 60;  
p2[2] = 70;  
*(p2+3) = 80;
```

```
free(vet2);  
vet2 = p2 = NULL;
```



Vetor

Alocando
memória

Manipulando
os dados

Liberando
memória

Passagem por
parâmetro

[\[Código no Repl.it\]](#)

Passando vetor por parâmetro

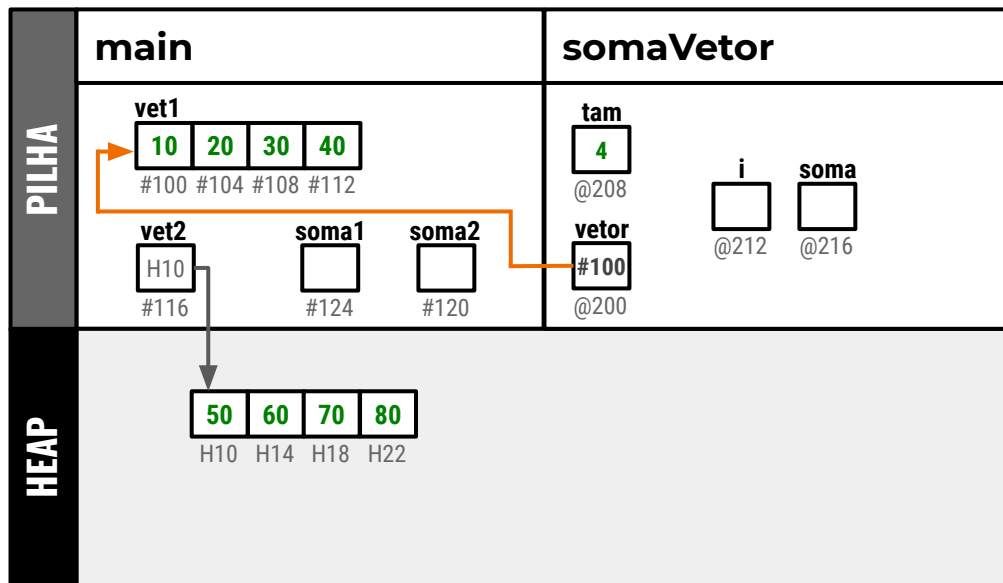
```
int main(){
    int vet1[4] = {10,20,30,40};

    int* vet2 =(int*)calloc(4, sizeof(int));
    vet2[0] = 50;
    vet2[1] = 60;
    vet2[2] = 70;
    vet2[3] = 80;

    int soma1 = somaVetor(vet1, 4);
    int soma2 = somaVetor(vet2, 4);
}
```

```
// Devolve a soma dos valores do vetor
int somaVetor(int* vetor, int tam){
    int i, soma = 0;;
    for(i=0; i<tam; i++){
        soma += vetor[i];
    }
    return soma;
}
```

```
int somaVetor(int* vetor , int tam);
int somaVetor(int vetor[], int tam);
```



Vetor

Alocando
memória

Manipulando
os dados

Liberando
memória

Passagem por
parâmetro

[\[Código no Repl.it\]](#)

Passando vetor por parâmetro

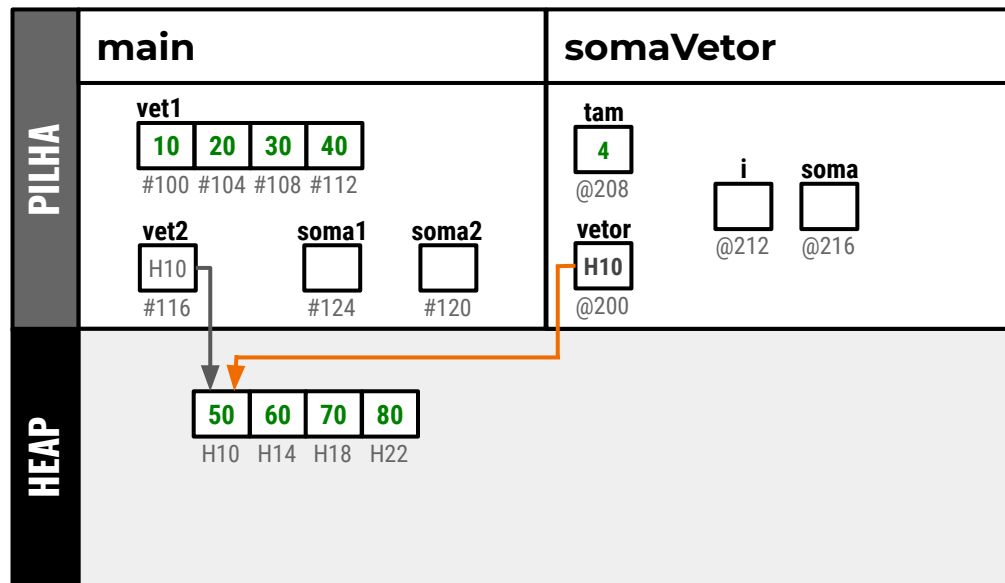
```
int main(){
    int vet1[4] = {10,20,30,40};

    int* vet2 =(int*)calloc(4, sizeof(int));
    vet2[0] = 50;
    vet2[1] = 60;
    vet2[2] = 70;
    vet2[3] = 80;

    int soma1 = somaVetor(vet1, 4);
    int soma2 = somaVetor(vet2, 4);
}
```

```
// Devolve a soma dos valores do vetor
int somaVetor(int* vetor, int tam){
    int i, soma = 0;;
    for(i=0; i<tam; i++){
        soma += vetor[i];
    }
    return soma;
}
```

```
int somaVetor(int* vetor , int tam);
int somaVetor(int vetor[], int tam);
```



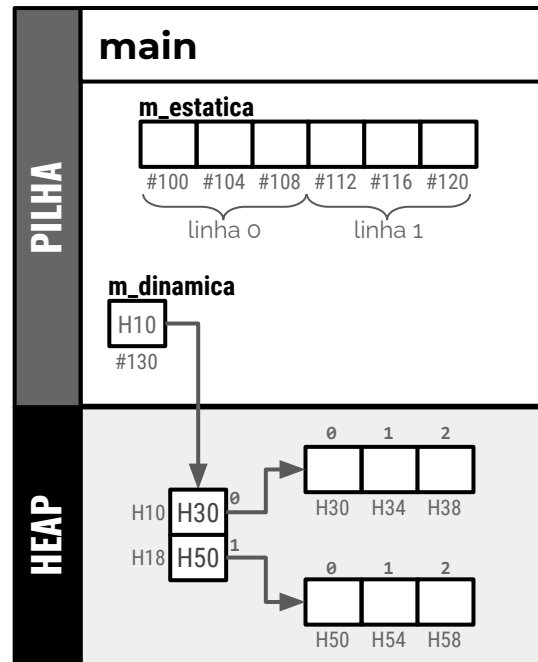
Matriz



A matriz alocada dinamicamente possui algumas diferenças em relação à alocada estaticamente. As principais diferenças são:

- organização dos dados na memória (vetor de vetores)
- forma de passagem de parâmetro

ALOCÇÃO ESTÁTICA
VS
ALOCÇÃO DINÂMICA



Matriz

Alocando
memória

Manipulando
os dados

Liberando
memória

Passagem por
parâmetro

[\[Código no Repl.it\]](#)

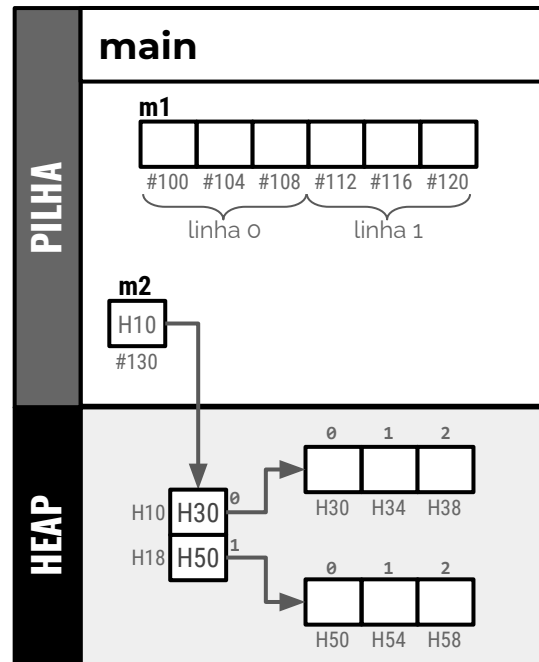
■ Criando a matriz

ALOCAÇÃO ESTÁTICA

VS ALOCAÇÃO DINÂMICA

```
int m1[2][3];
```

```
int** m2;  
// Primeira dimensão  
m2 = (int**)calloc(2, sizeof(int*));  
  
// Segunda dimensão  
m2[0] = (int*)calloc(3, sizeof(int));  
m2[1] = (int*)calloc(3, sizeof(int));
```



Matriz

Alocando
memória

Manipulando
os dados

Liberando
memória

Passagem por
parâmetro

[\[Código no Repl.it\]](#)

Manipulando os dados

ALOCAÇÃO ESTÁTICA

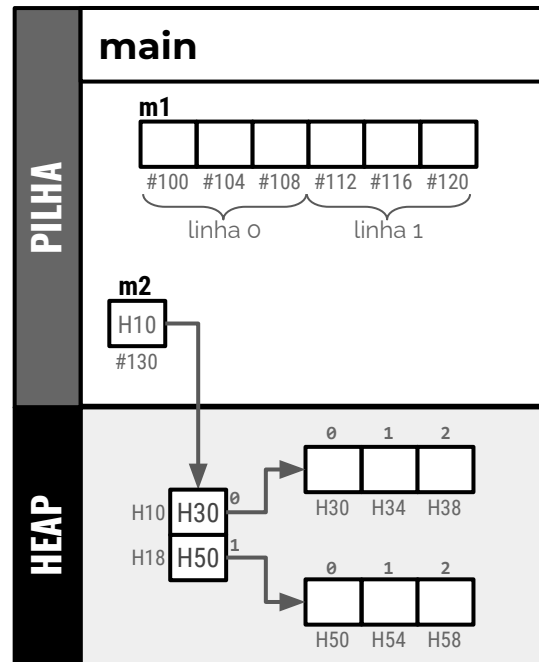
VS ALOCAÇÃO DINÂMICA

```
int m1[2][3];
```

```
m1[0][0] = 10;  
*(m1[1] + 2) = 60;
```

```
int** m2;  
// Primeira dimensão  
m2 = (int**)calloc(2, sizeof(int*));  
  
// Segunda dimensão  
m2[0] = (int*)calloc(3, sizeof(int));  
m2[1] = (int*)calloc(3, sizeof(int));
```

```
m1[0][0] = 10;  
*(m1[1] + 2) = 60;
```



Matriz

Alocando
memória

Manipulando
os dados

Liberando
memória

Passagem por
parâmetro

[\[Código no Repl.it\]](#)

Manipulando os dados

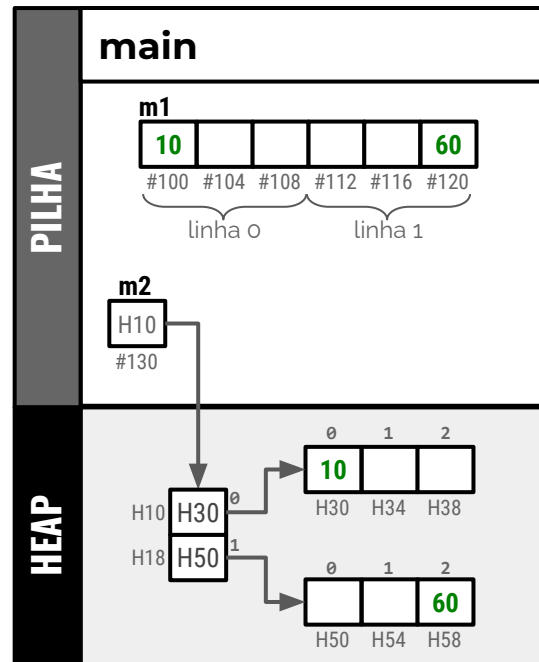
ALOCAÇÃO ESTÁTICA

VS ALOCAÇÃO DINÂMICA

```
int m1[2][3];
```

```
m1[0][0] = 10;  
*(m1[1] + 2) = 60;
```

```
int** m2;  
// Primeira dimensão  
m2 = (int**)calloc(2, sizeof(int*));  
  
// Segunda dimensão  
m2[0] = (int*)calloc(3, sizeof(int));  
m2[1] = (int*)calloc(3, sizeof(int));  
  
m1[0][0] = 10;  
*(m1[1] + 2) = 60;
```



Matriz

Alocando
memória

Manipulando
os dados

Liberando
memória

Passagem por
parâmetro

[\[Código no Repl.it\]](#)

Desalocando a matriz

ALOCAÇÃO ESTÁTICA

VS ALOCAÇÃO DINÂMICA

```
int m1[2][3];
```

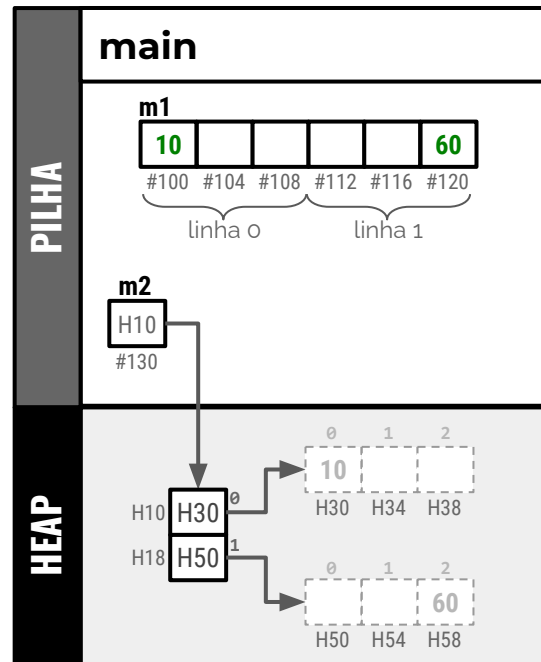
```
m1[0][0] = 10;  
*(m1[1] + 2) = 60;
```

A liberação do espaço é realizada automaticamente de acordo com o escopo em que a variável foi declarada

```
int** m2;  
// Primeira dimensão  
m2 = (int**)calloc(2, sizeof(int*));  
  
// Segunda dimensão  
m2[0] = (int*)calloc(3, sizeof(int));  
m2[1] = (int*)calloc(3, sizeof(int));
```

```
m1[0][0] = 10;  
*(m1[1] + 2) = 60;
```

```
free(m2[0]); Endereço H30  
free(m2[1]); Endereço H50
```



Matriz

Alocando
memória

Manipulando
os dados

Liberando
memória

Passagem por
parâmetro

[\[Código no Repl.it\]](#)

Desalocando a matriz

ALOCAÇÃO ESTÁTICA

VS ALOCAÇÃO DINÂMICA

```
int m1[2][3];
```

```
m1[0][0] = 10;  
*(m1[1] + 2) = 60;
```

A liberação do espaço é realizada automaticamente de acordo com o escopo em que a variável foi declarada

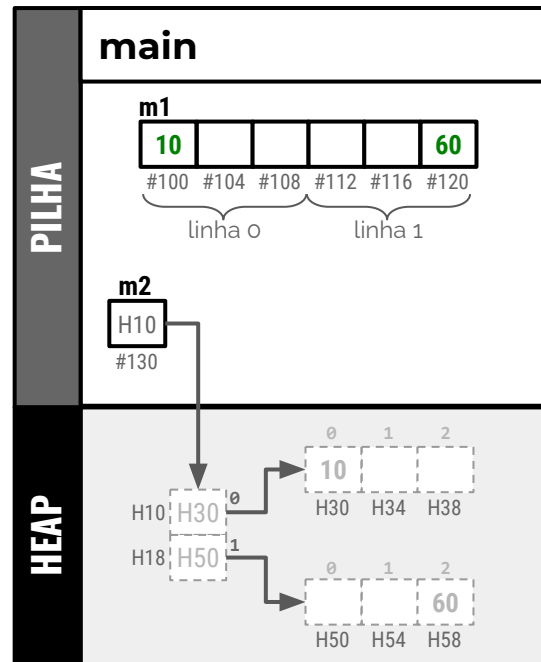
```
int** m2;  
// Primeira dimensão  
m2 = (int**)calloc(2, sizeof(int*));
```

```
// Segunda dimensão  
m2[0] = (int*)calloc(3, sizeof(int));  
m2[1] = (int*)calloc(3, sizeof(int));
```

```
m1[0][0] = 10;  
*(m1[1] + 2) = 60;
```

```
free(m2[0]); Endereço H30  
free(m2[1]); Endereço H50
```

```
free(m2); Endereço H10  
m2 = NULL;
```



Matriz

Alocando
memória

Manipulando
os dados

Liberando
memória

Passagem por
parâmetro

[\[Código no Repl.it\]](#)

Desalocando a matriz

ALOCAÇÃO ESTÁTICA

VS ALOCAÇÃO DINÂMICA

```
int m1[2][3];
```

```
m1[0][0] = 10;  
*(m1[1] + 2) = 60;
```

A liberação do espaço é realizada automaticamente de acordo com o escopo em que a variável foi declarada

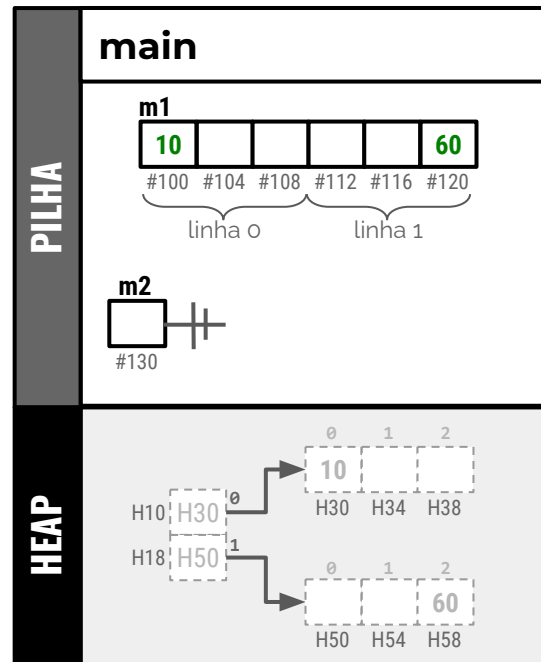
```
int** m2;  
// Primeira dimensão  
m2 = (int**)calloc(2, sizeof(int*));
```

```
// Segunda dimensão  
m2[0] = (int*)calloc(3, sizeof(int));  
m2[1] = (int*)calloc(3, sizeof(int));
```

```
m1[0][0] = 10;  
*(m1[1] + 2) = 60;
```

```
free(m2[0]); Endereço H30  
free(m2[1]); Endereço H50
```

```
free(m2); Endereço H10  
m2 = NULL;
```



Matriz

Alocando
memória

Manipulando
os dados

Liberando
memória

Passagem por
parâmetro

[\[Código no Repl.it\]](#)

Passando a matriz por parâmetro

```
int main(){
    int m1[2][3];

    int** m2=(int**)calloc(2, sizeof(int*));
    m2[0] = (int*)calloc(3, sizeof(int));
    m2[1] = (int*)calloc(3, sizeof(int));

    imprimeEst((int*)m1, 2, 3);
    imprimeDin(m2, 2, 3);
}
```

```
// imprime os valores da matriz
int imprimeEST(int* m, int lin, int col){
    int i,j;
    for(i=0; i<lin; i++){
        for(j=0; j<col; j++){
            printf("\t%d", m[i * col + j]);
        }
        printf("\n");
    }
}
```

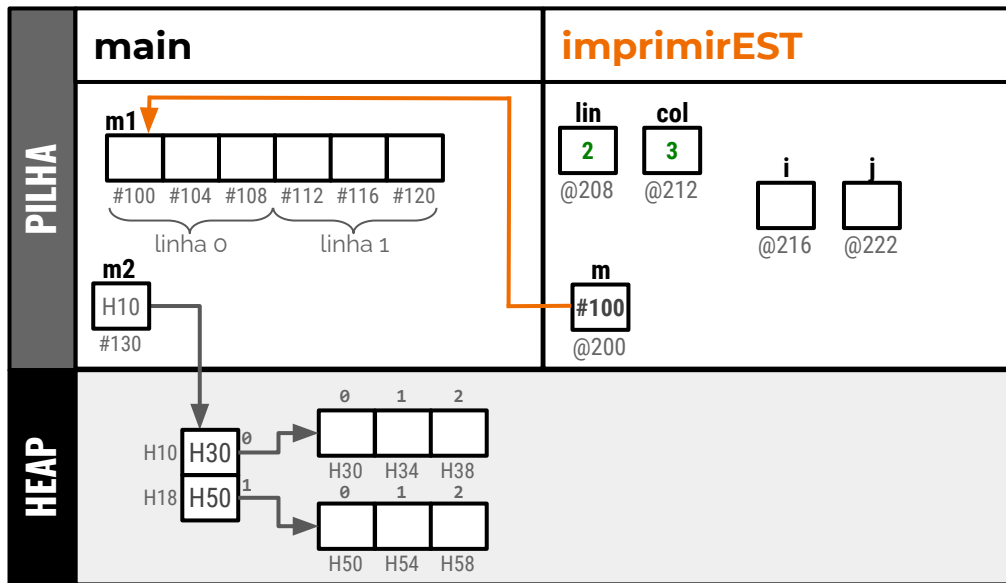
Alocação Estática

```
void imprimeEST(int* m, int lin, int col)
void imprimeEST(int m[][COL], int lin, int col)
```

Alocação Dinâmica

```
void imprimirDIN(int** m, int lin, int col)
```

Como declarar os parâmetros?



Matriz

Alocando
memória

Manipulando
os dados

Liberando
memória

Passagem por
parâmetro

[\[Código no Repl.it\]](#)

Passando a matriz por parâmetro

```
int main(){
    int m1[2][3];

    int** m2=(int**)calloc(2, sizeof(int*));
    m2[0] = (int*)calloc(3, sizeof(int));
    m2[1] = (int*)calloc(3, sizeof(int));

    imprimeEst((int*)m1, 2, 3);
    imprimeDin(m2, 2, 3);
}
```

```
// imprime os valores da matriz
int imprimeDIN(int** m, int lin, int col){
    int i,j;
    for(i=0; i<lin; i++){
        for(j=0; j<col; j++){
            printf("\t%d", m[i][j]);
        }
        printf("\n");
    }
}
```

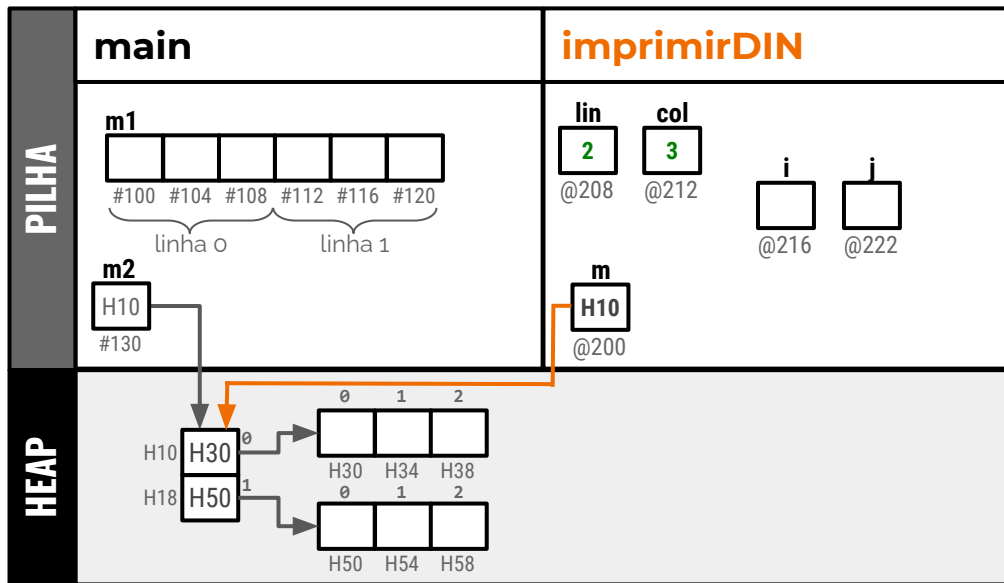
Alocação Estática

```
void imprimeEST(int* m, int lin, int col)
void imprimeEST(int m[][COL], int lin, int col)
```

Alocação Dinâmica

```
void imprimirDIN(int** m, int lin, int col)
```

Como declarar os parâmetros?



Struct

Alocando
memória

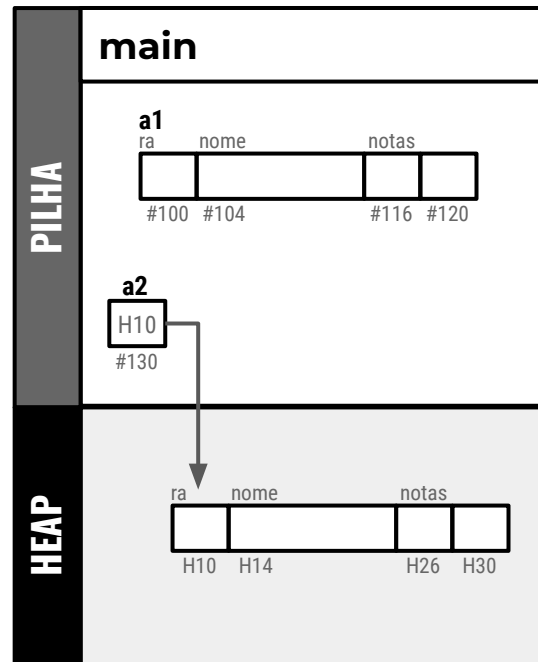
Manipulando
os dados

Liberando
memória

Passagem por
parâmetro

```
typedef struct aluno{  
    int ra;  
    char nome[12];  
    float notas[2];  
} Aluno;
```

ALOCAÇÃO ESTÁTICA
VS
ALOCAÇÃO DINÂMICA



Struct

Alocando
memória

Manipulando
os dados

Liberando
memória

Passagem por
parâmetro

[\[Código no Repl.it\]](#)

■ Criando a struct

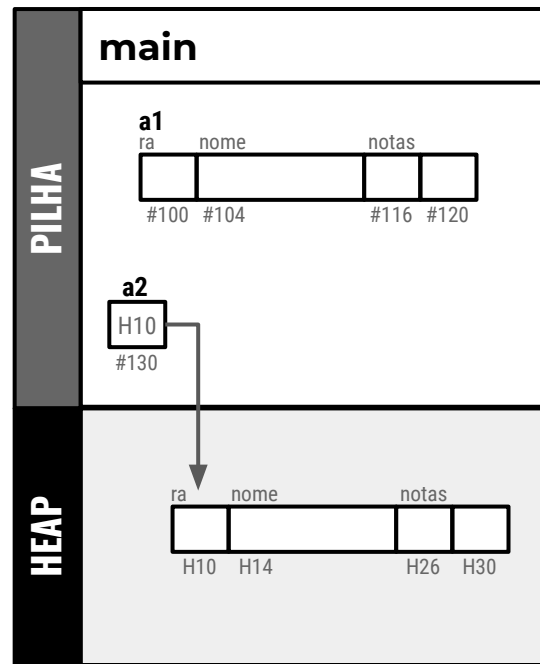
ALOCAÇÃO ESTÁTICA

VS

ALOCAÇÃO DINÂMICA

```
Aluno a1;
```

```
Aluno* a2;  
a2 = (Aluno*)malloc(sizeof(Aluno));
```



Struct

Alocando
memória

Manipulando
os dados

Liberando
memória

Passagem por
parâmetro

[\[Código no Repl.it\]](#)

■ Manipulando os dados

ALOCÇÃO ESTÁTICA

VS

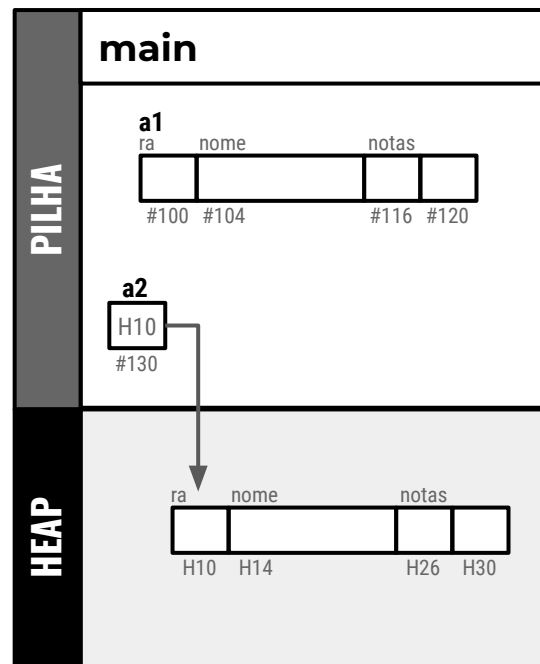
ALOCÇÃO DINÂMICA

```
Aluno a1;
```

```
a1.ra = 1;  
strcpy(a1.nome, "Joao");  
a1.notas[0] = 7.0;  
*(a1.notas+1) = 8.0;
```

```
Aluno* a2;  
a2 = (Aluno*)malloc(sizeof(Aluno));
```

```
(*a2).ra = 2;  
strcpy(a2->nome, "Maria");  
a2->notas[0] = 8.0;  
*(a2->notas + 1) = 9.0;
```



Struct

Alocando
memória

Manipulando
os dados

Liberando
memória

Passagem por
parâmetro

[\[Código no Repl.it\]](#)

■ Manipulando os dados

ALOCÇÃO ESTÁTICA

VS

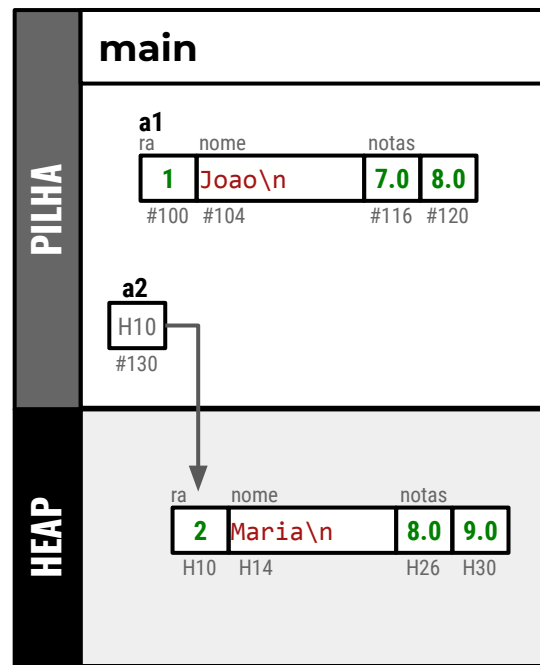
ALOCÇÃO DINÂMICA

```
Aluno a1;
```

```
a1.ra = 1;  
strcpy(a1.nome, "Joao");  
a1.notas[0] = 7.0;  
*(a1.notas+1) = 8.0;
```

```
Aluno* a2;  
a2 = (Aluno*)malloc(sizeof(Aluno));
```

```
(*a2).ra = 2;  
strcpy(a2->nome, "Maria");  
a2->notas[0] = 8.0;  
*(a2->notas + 1) = 9.0;
```



Struct

Alocando
memória

Manipulando
os dados

Liberando
memória

Passagem por
parâmetro

[\[Código no Repl.it\]](#)

Desalocando a struct

ALOCÇÃO ESTÁTICA

VS

ALOCÇÃO DINÂMICA

```
Aluno a1;
```

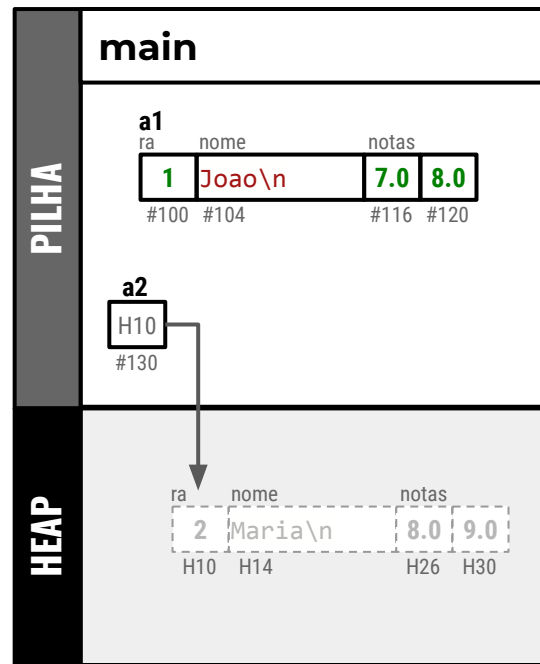
```
a1.ra = 1;  
strcpy(a1.nome, "Joao");  
a1.notas[0] = 7.0;  
*(a1.notas+1) = 8.0;
```

A liberação do espaço é realizada automaticamente de acordo com o escopo em que a variável foi declarada

```
Aluno* a2;  
a2 = (Aluno*)malloc(sizeof(Aluno));
```

```
(*a2).ra = 2;  
strcpy(a2->nome, "Maria");  
a2->notas[0] = 8.0;  
*(a2->notas + 1) = 9.0;
```

```
free(a2);
```



Struct

Alocando
memória

Manipulando
os dados

Liberando
memória

Passagem por
parâmetro

[\[Código no Repl.it\]](#)

Desalocando a struct

ALOCÇÃO ESTÁTICA

VS

ALOCÇÃO DINÂMICA

```
Aluno a1;
```

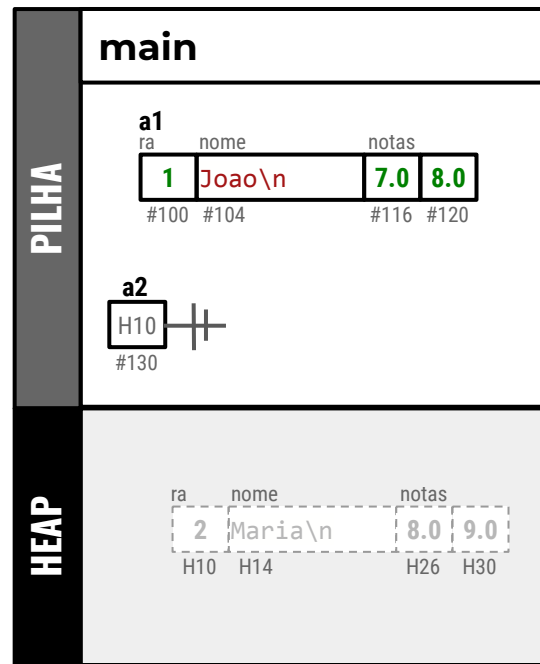
```
a1.ra = 1;  
strcpy(a1.nome, "Joao");  
a1.notas[0] = 7.0;  
*(a1.notas+1) = 8.0;
```

A liberação do espaço é realizada automaticamente de acordo com o escopo em que a variável foi declarada

```
Aluno* a2;  
a2 = (Aluno*)malloc(sizeof(Aluno));
```

```
(*a2).ra = 2;  
strcpy(a2->nome, "Maria");  
a2->notas[0] = 8.0;  
*(a2->notas + 1) = 9.0;
```

```
free(a2);  
a2 = NULL;
```



Struct

Alocando
memória

Manipulando
os dados

Liberando
memória

Passagem por
parâmetro

[\[Código no Repl.it\]](#)

```
int main(){
    Aluno a1;
    Aluno* a2 = (Aluno*)malloc(sizeof(Aluno));

    a1.ra = 1;
    strcpy(a1.nome, "Joao");
    a1.notas[0] = 7.0;
    *(a1.notas+1) = 8.0;

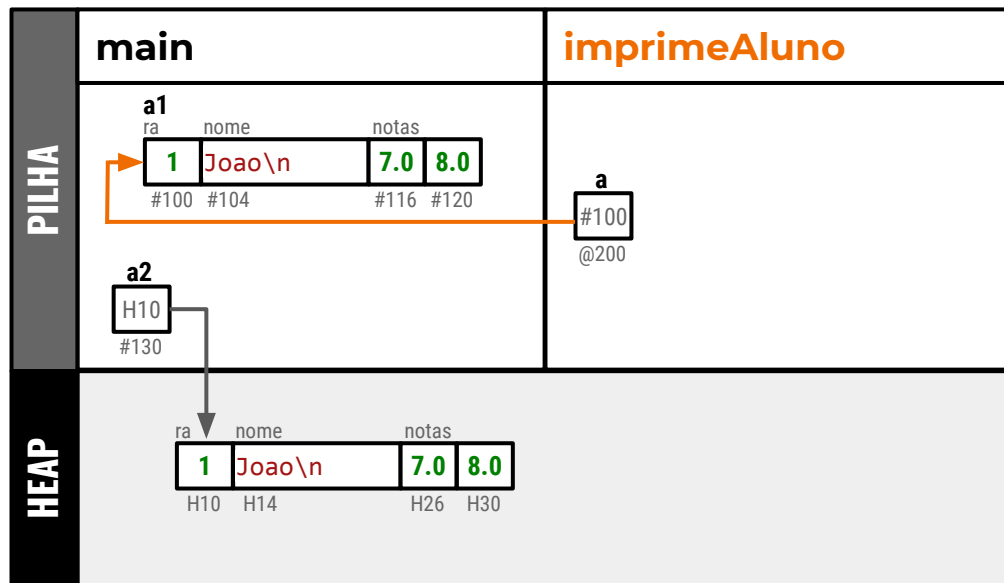
    *a2 = a1;

    imprimeAluno(&a1);
    imprimeAluno(a2);

    free(a2);
}
```

```
void imprimeAluno(Aluno* a){
    printf("RA      : %d\n", (*a).ra);
    printf("Nome    : %s\n", a->nome);
    printf("Nota 1: %.2f\n", a->notas[0]);
    printf("Nota 2: %.2f\n", a->notas[1]);
    printf("\n");
}
```

■ Passando a struct como parâmetro por referência



Struct

Alocando
memória

Manipulando
os dados

Liberando
memória

Passagem por
parâmetro

[\[Código no Repl.it\]](#)

```
int main(){
    Aluno a1;
    Aluno* a2 = (Aluno*)malloc(sizeof(Aluno));

    a1.ra = 1;
    strcpy(a1.nome, "Joao");
    a1.notas[0] = 7.0;
    *(a1.notas+1) = 8.0;

    *a2 = a1;

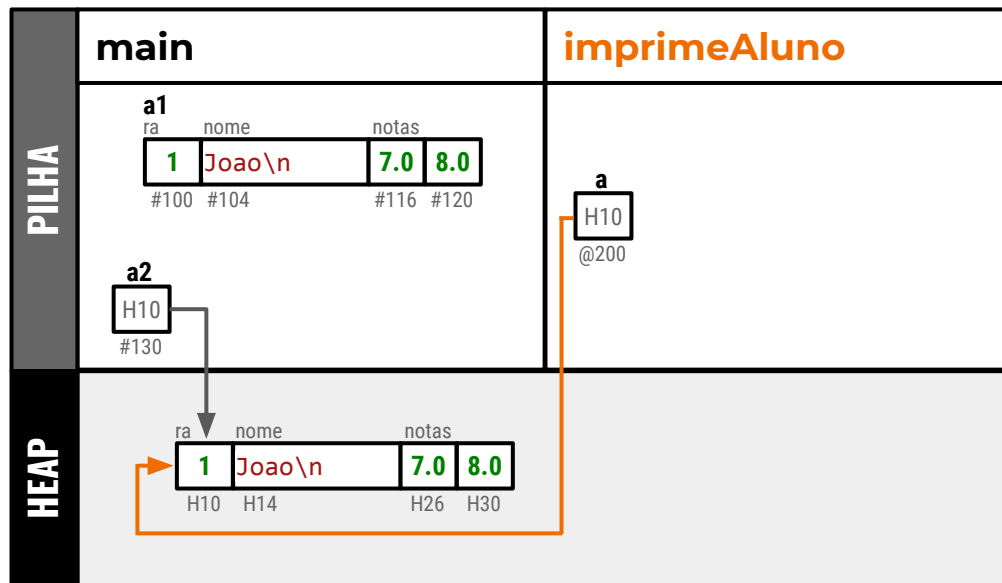
    imprimeAluno(&a1);
    imprimeAluno(a2);

    free(a2);
}
```



```
void imprimeAluno(Aluno* a){
    printf("RA      : %d\n", a->ra);
    printf("Nome    : %s\n", a->nome);
    printf("Nota 1: %.2f\n", a->notas[0]);
    printf("Nota 2: %.2f\n", a->notas[1]);
    printf("\n");
}
```

■ Passando a struct como parâmetro por referência



Struct

Alocando
memória

Manipulando
os dados

Liberando
memória

Passagem por
parâmetro

[\[Código no Repl.it\]](#)

```
int main(){
    Aluno a1;
    Aluno* a2 = (Aluno*)malloc(sizeof(Aluno));

    a1.ra = 1;
    strcpy(a1.nome, "Joao");
    a1.notas[0] = 7.0;
    *(a1.notas+1) = 8.0;

    *a2 = a1;

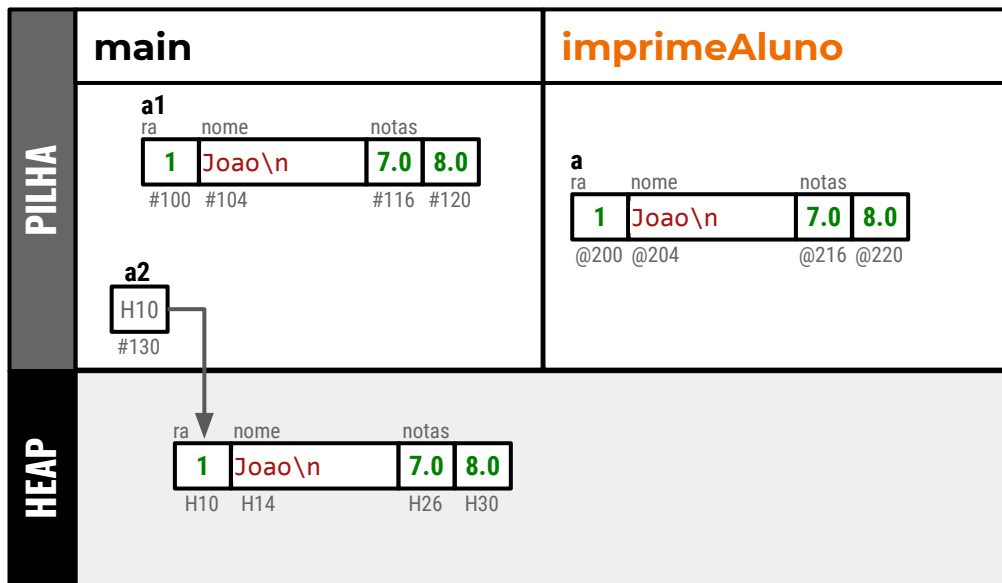
    imprimeAluno(a1);
    imprimeAluno(*a2);

    free(a2);
}
```



```
void imprimeAluno(Aluno a){
    printf("RA      : %d\n", a.ra);
    printf("Nome    : %s\n", a.nome);
    printf("Nota 1: %.2f\n", a.notas[0]);
    printf("Nota 2: %.2f\n", a.notas[1]);
    printf("\n");
}
```

■ Passando a struct como parâmetro por valor



Struct

Alocando
memória

Manipulando
os dados

Liberando
memória

Passagem por
parâmetro

[\[Código no Repl.it\]](#)

```
int main(){
    Aluno a1;
    Aluno* a2 = (Aluno*)malloc(sizeof(Aluno));

    a1.ra = 1;
    strcpy(a1.nome, "Joao");
    a1.notas[0] = 7.0;
    *(a1.notas+1) = 8.0;

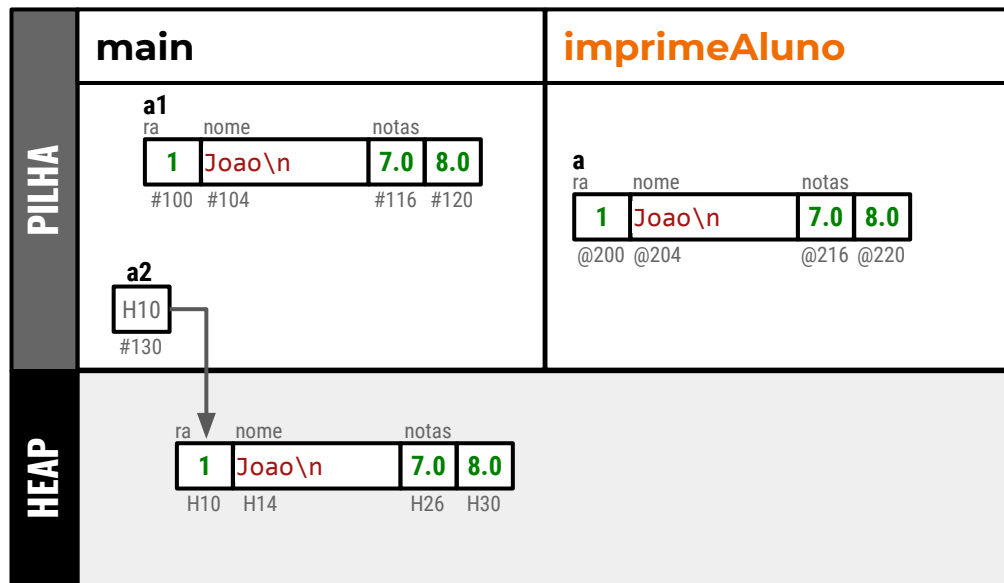
    *a2 = a1;

    imprimeAluno(a1);
    imprimeAluno(*a2); ←

    free(a2);
}
```

```
void imprimeAluno(Aluno a){
    printf("RA      : %d\n", a.ra);
    printf("Nome    : %s\n", a.nome);
    printf("Nota 1: %.2f\n", a.notas[0]);
    printf("Nota 2: %.2f\n", a.notas[1]);
    printf("\n");
}
```

■ Passando a struct como parâmetro por valor



Fim