Aula 4: Tipos de Alocação de Memória

DCC302-Estrutura de Dados I

Prof. Me. Acauan C. Ribeiro

Passagem por Valor

É feito uma cópia do **argumento/valor** (ou **variável**), que pode ser usada e alterada dentro da função **sem afetar** a variável da qual ela foi gerada.

```c
#include <stdio.h>

int soma(int x, int y) {
  int z = x + y;
  return z;
}

int main() {
  int a = 10;
  int b = 20;
  int c = soma(a, b);
  printf("Resultado: %d\n", c);
}
```

Memória RAM		
Endereço	Valor	Nome

Passagem por Valor

É feito uma cópia do **argumento/valor** (ou **variável**), que pode ser usada e alterada dentro da função **sem afetar** a variável da qual ela foi gerada.

```c
#include <stdio.h>

int soma(int x, int y) {
  int z = x + y;
  return z;
}

int main() {
  int a = 10;
  int b = 20;
  int c = soma(a, b);
  printf("Resultado: %d\n", c);
}
```

Depois que a função termina seus valores e nomes de variáveis são apagados da memória, liberando aquele espaço para uso futuro.

Memória RAM

Endereço	Valor	Nome
0x8248	30	z
0x8244	20	y
0x8240	10	x
...	...	...
0x3010	30	c
0x3006	20	b
0x3002	10	a

Passagem por Referência

É passado a **referência** (leia-se endereço de memória) de uma **variável** (ponteiro) para uma função, possibilitando alterar uma variável que é externa a uma função.

```c
#include <stdio.h>

void soma(int x, int y, int *z)
    *z = x + y;
}

int main() {
    int a = 30;
    int b = 20;
    int c;
    soma(a, b, &c);
    printf("Resultado: %d\n", c);
}
```

Memória RAM		
Endereço	Valor	Nome
0x8248	30	z
0x8244	20	y
0x8240	10	x
...	...	...
0x3010	30	c
0x3006	20	b
0x3002	10	a

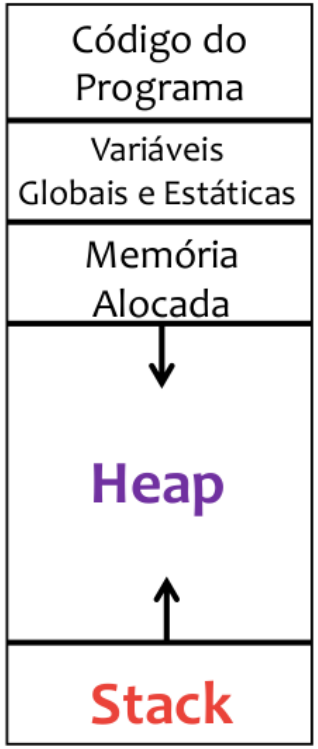
Passagem por Referência

É passado a **referência** (leia-se endereço de memória) de uma **variável** (ponteiro) para uma função, possibilitando alterar uma variável que é externa a uma função.

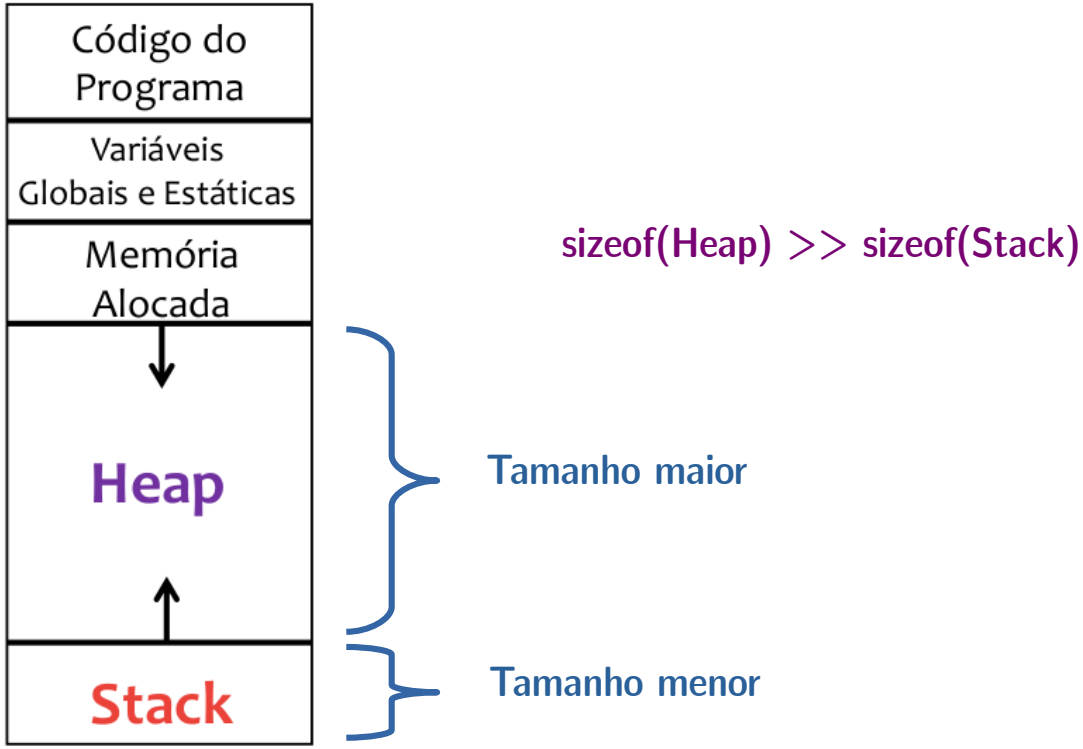
```c
#include <stdio.h>

void soma(int x, int y, int *z)
  *z = x + y;
}

int main() {
  int a = 30;
  int b = 20;
  int c;
  soma(a, b, &c);
  printf("Resultado: %d\n", c);
}
```

Memória RAM		
Endereço	Valor	Nome
0x8248	0x3010	*z
0x8244	20	y
0x8240	30	x
...	...	...
0x3010	50	c
0x3006	20	b
0x3002	30	a

Esquema de Memória

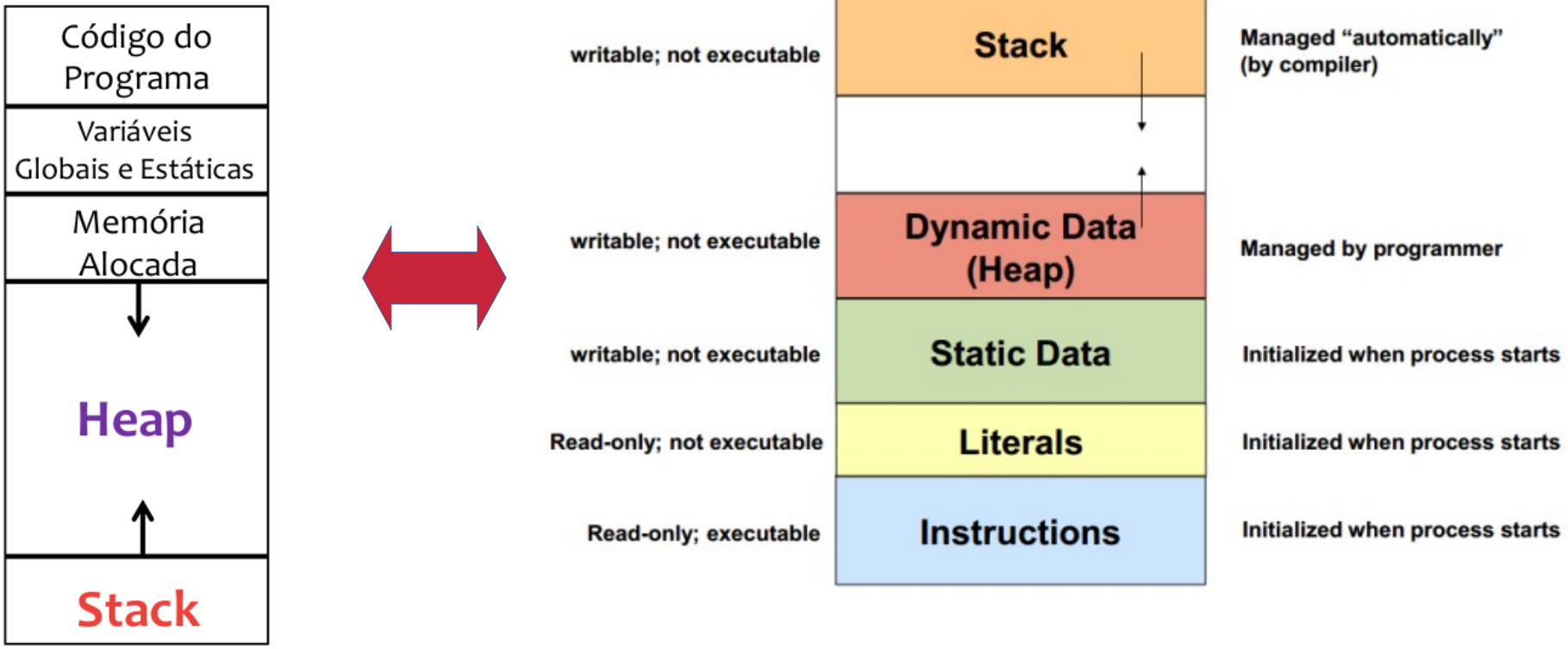


Esquema de Memória



sizeof(Heap) >> sizeof(Stack)

Esquema de Memória



Alocação Estática

- O espaço para as variáveis é reservado no **início da execução**;
- Cada variável tem seu **endereço** fixado e a área de memória ocupada por ela se **mantém constante** durante **toda a execução**;
- São alocadas na **Stack** da Memória Ram;
- **Liberação** de memória feita; **automaticamente** pelo **compilador/SO.**

```
int a;
float b;
char c;
int a[10];
float *p;
```

Alocação Estática

- O espaço para as variáveis é reservado no **início da execução**;
- Cada variável tem seu **endereço** fixado e a área de memória ocupada por ela se **mantém constante** durante **toda a execução**;
- São alocadas na **Stack** da Memória Ram;
- **Liberação** de memória feita; **automaticamente** pelo **compilador/SO.**

```
int a;
float b;
char c;
int a[10];
float *p;
```

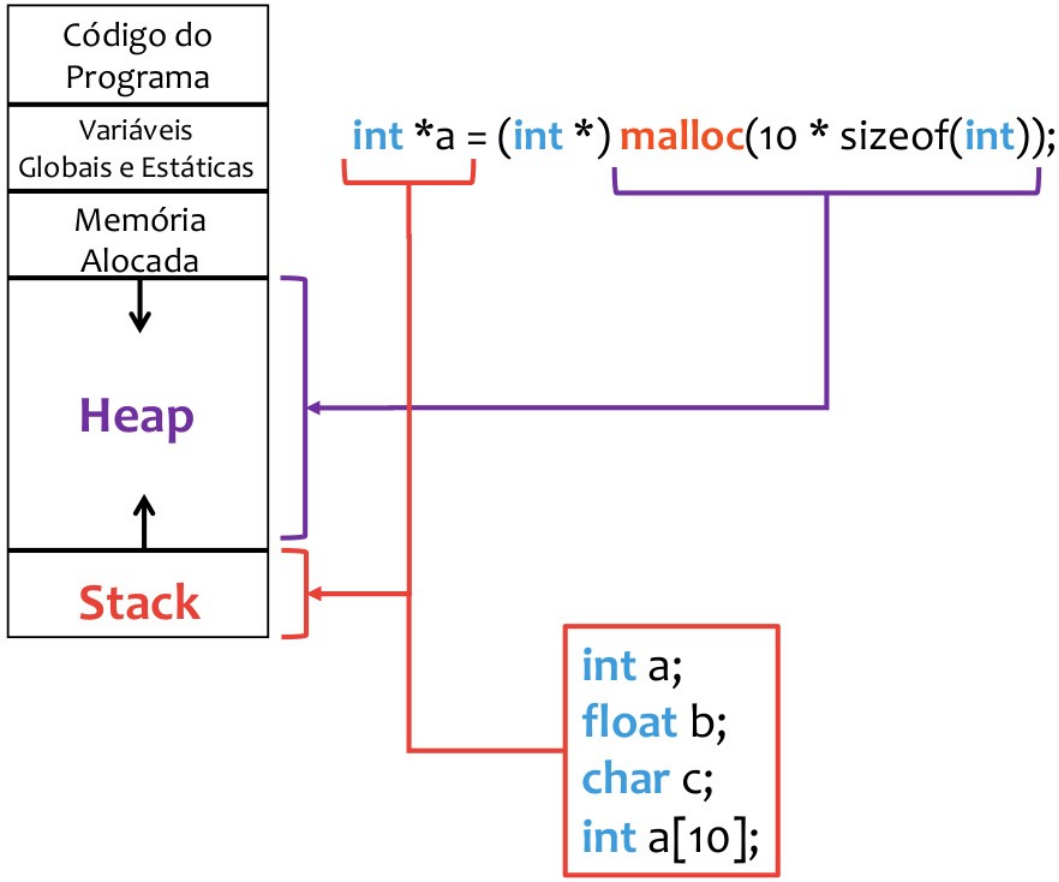
"Toda variável é alocada na memória **stack**"

Alocação Dinâmica

- O **espaço** é alocado dinamicamente **durante a execução do programa**;
- Pode ser criada ou eliminada durante a execução do programa, ocupando espaço na memória penas enquanto está sendo utilizada;
- São alocadas na **Heap (free store)** da Memória Ram;
- **Liberação** de memória feita manualmente pelo **programador** (Perigo!)

```
int *a = malloc(10 * sizeof(int));
float *b = calloc(5, sizeof(float));
free(a);
free(b);
```

Esquema de Memória



Alocação Dinâmica

Por que usar?

- A **alocação dinâmica** é o processo que aloca memória em tempo de execução;
- Ela é utilizada quando não se sabe ao certo quanto de memória será necessário para o armazenamento dos elementos;
- Assim, o tamanho de memória é determinado conforme necessidade;
- Dessa forma evita-se o desperdíciio de memória;

Além disso, **size(Heap)** >> **size(Stack)**

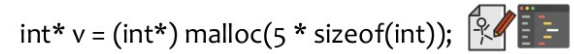
Alocação Dinâmica

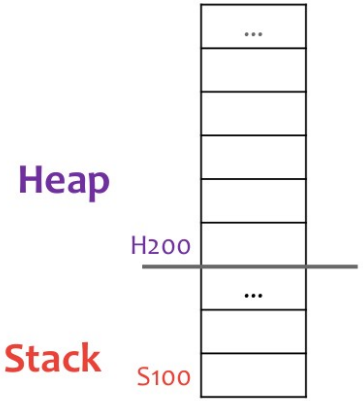
malloc

- Aloca um **bloco de bytes consecutivos** na memória **heap** e devolve o endereço desse bloco.

```
tipo* v = (tipo*) malloc(n * sizeof(tipo));
          opcional
```

```
int* v = (int*) malloc(5 * sizeof(int));
```





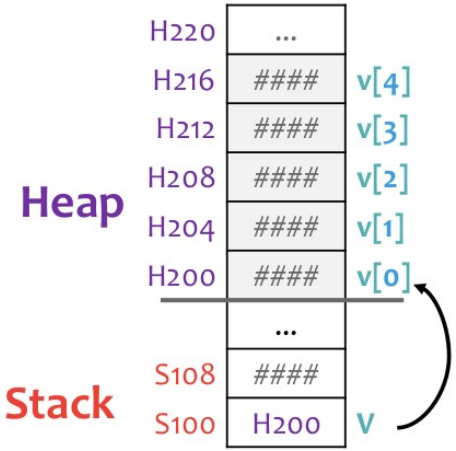
Alocação Dinâmica

malloc

- Aloca um **bloco de bytes consecutivos** na memória **heap** e devolve o endereço desse bloco.

```
tipo* v = (tipo*) malloc(n * sizeof(tipo));
          opcional
```

```
int* v = (int*) malloc(5 * sizeof(int));
```



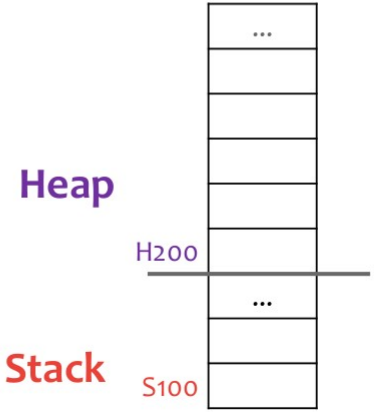
Alocação Dinâmica

calloc

- Aloca um **bloco de bytes consecutivos** na **memória heap** e inicializa todos os valores com **0** (**NULL** para ponteiros).

```
tipo* v = (tipo*) calloc(n, sizeof(tipo));
          opcional
```

```
int* v = (int*) calloc(5, sizeof(int));
```



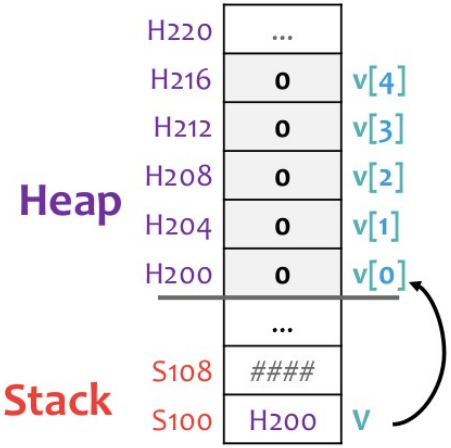
Alocação Dinâmica

calloc

- Aloca um **bloco de bytes consecutivos** na **memória heap** e inicializa todos os valores com **0** (**NULL** para ponteiros).

```
tipo* v = (tipo*) calloc(n, sizeof(tipo));
          opcional
```

```
int* v = (int*) calloc(5, sizeof(int));
```



Alocação Dinâmica

Let's code...

Alocação Dinâmica

free

- Libera a porção de **memória Heap** alocada por **malloc** ou **calloc**, no qual é apontada por um ponteiro.

```
free(v);
```

BOA PRÁTICA DE PROGRAMAÇÃO:
Sempre após dar o **free** em um **ponteiro,** atribua **NULL**

```
free(v);
v = NULL;
```

Por quê?

Exercícios

- Fazer exemplo com htop (memory leak)

  1) Crie uma função que calcule o **mínimo** e o **máximo** de um vetor de inteiros e retorne os valores em duas variáveis diferentes.

Referências

- Aulas Estrutura de Dados I – Prof. Samuel Martins – IFSP