Fundamentos da Programação

Memória dinâmica



Conteúdo

- Alocação de memória dinâmica
- Precauções
- Apontadores para apontadores
- Alocar memória em funções

Problema

- Nos programas vistos nos capítulos anteriores, todas as necessidades de memória foram determinadas antes da execução do programa, definindo as variáveis necessárias (alocação estática).
 - Quando é necessário alocar um array e não sabemos, à partida, quantos elementos vão ser quardados estabelecemos um tamanho "suficientemente grande":
 - Se não chegar, o programa é limitado...
 - Se for demasiado grande, existe um desperdício de memória....
- Podem existir casos em que as necessidades de memória de um programa só podem ser determinadas durante o tempo de execução.
 - Por exemplo, quando a memória necessária depende da entrada do utilizador. Nesses casos, os programas precisam **alocar memória dinamicamente**.

Alocação dinâmica de memória

	stack
???	0xffffcc20
???	0xffffcc1c
???	0xffffcc18
???	0xffffcc14
???	0xffffcc10
???	0xffffcc0c
???	0xffffcc08
???	0xffffcc04
•••	

• Nestas situações podemos alocar a memória dinamicamente na **heap**

??? ??? ### ### 0x6000cc04 0xffffcc08 0xffffcc0c 0xffffcc10 ### ??? ??? ??? 0x6000cc14 0xffffcc18 0xffffcc1c 0xffffcc20 ??? ??? ??? ??? 0x6000cc24 0xffffcc28 0xffffcc2c 0xffffcc30 ??? ### ??? 0x6000cc44 0xffffcc48 0xffffcc4c 0xffffcc50 ??? ### ### ### 0x6000cc54 0xffffcc58 0xffffcc5c 0xffffcc60				
### ??? ??? ??? 0x6000cc14	???	???	###	###
0x6000cc14 0xffffcc18 0xffffcc1c 0xffffcc20 ??? ??? ??? ??? 0x6000cc24 0xffffcc28 0xffffcc2c 0xffffcc30 ??? ??? ### ??? 0x6000cc44 0xffffcc48 0xffffcc4c 0xffffcc50 ??? ### ### ###	0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
??? ??? ??? ??? 0x6000cc24	###	???	???	???
0x6000cc24 0xffffcc28 0xffffcc2c 0xffffcc30 ??? ??? ### ??? 0x6000cc44 0xffffcc48 0xffffcc4c 0xffffcc50 ??? ### ### ###	0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
???	???	???	???	???
0x6000cc44 0xffffcc48 0xffffcc4c 0xffffcc50 ??? ### ### ###	0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
??? ### ###	???	???	###	???
	0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
0x6000cc54 0xffffcc58 0xffffcc5c 0xffffcc60	???	###	###	###
	0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60
···				

Stack vs Heap

Stack	Неар
Bloco contíguo de memória.	Memória não é necessariamente alocada sequencialmente. Pode existir fragmentação.
Limitada em termos de espaço.	Grande espaço de memória disponível para alocação de memória.
Mais rápida.	A heap é muito mais lenta que a stack.
Quando se pretende utilizar estruturas de maior dimensão, ou funções recursivas mais intrincadas, podem ocorrer erros do tipo Stack Overflow.	Memória alocada dinamicamente tem que ser explicitamente libertada. Caso contrário ocorre uma memory leak ou fuga de memória
Elementos estão empilhados, e apenas se pode inserir ou remover no topo da pilha. Quando um elemento da stack é removido, os seus recursos são automaticamente libertados. Insere-se no topo e remove-se no topo (LIFO – Last In, First Out).	Sistema operativo tem que gerir acessos à heap. Memória reservada existe durante toda a "vida" do programa, ou até que seja explicitamente removida

Alocação dinâmica de memória

- Processo de alocação de memória durante a execução de um programa.
- O header file **stdlib.h** fornece funções relacionadas com a alocação/libertação de memória:
 - void* malloc(size_t size)
 - Aloca um bloco de memória de **size** bytes.
 - void* calloc(size_t num, size_t size)
 - Aloca um bloco de memória de **num** * **size** bytes, e inicializa todos os seus valores a **0**.
 - void* realloc(void* ptr, size_t size)
 - Altera o tamanho do bloco para o qual **ptr** aponta, para **size** bytes.
 - void free(void* ptr)
 - Liberta o bloco de memória para o qual **ptr** aponta.

malloc

```
T *a = (T *) malloc(n * sizeof(T));
```

- T identifica o tipo de dados.
- A função **malloc** recebe um argumento que é o número de bytes de memória que pretendemos alocar.
 - Se quisermos guardar **n** inteiros temos de pedir **n** * o número de bytes que cada inteiro ocupa.
- Retorna o primeiro endereço do bloco de memória que foi reservado. Caso não seja possível fazer a alocação de memória, **malloc** devolve **NULL**.
 - É aconselhado verificar se a alocação de memória teve sucesso.
- É necessário o cast (T *) para o tipo apropriado, para se poder utilizar aritmética de apontadores.

malloc

Alocar um valor

```
#include <stdio.h>
#include <stdlib.h>
int main() {
 double dStack = 100.0;
 double *pDHeap = (double*) malloc(sizeof (double));
  *pDHeap = 200.0;
 printf("%lf %lf", dStack, *pDHeap);
 free(pDHeap);
  pDHeap = NULL;
  return 0;
```

100.000000 200.000000

	•••	stack
	???	0xffffcc20
	???	0xffffcc1c
	???	0xffffcc18
	???	0xffffcc14
	???	0xffffcc10
	???	0xffffcc0c
*pDHeap	0x6000cc04	0xffffcc08
dStack	100.0	0xffffcc04

heap

200	###	###	###
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
###	???	???	???
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
???	???	???	???
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
???	???	###	???
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
???	###	###	###
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60

8

malloc

Alocar um vetor

```
int i, tam;
double *pDHeap;
puts("Tamanho do vetor: ");
scanf("%d", &tam);
pDHeap = (double*) malloc(sizeof(double) * tam);
for (i = 0; i < tam; i++) {
   pDHeap[i] = 0; // equiv. a *(pDHeap + i) = 0;
   printf("%lf ", pDHeap[i]);
free(pDHeap);
pDHeap = NULL;
```

```
Tamanho do vetor:
5
0.000000 0.000000 0.000000 0.000000
```

		stack
	???	0xffffcc20
	???	0xffffcc1c
	???	0xffffcc18
	???	0xffffcc14
	???	0xffffcc10
	???	0xffffcc0c
*pDHeap	0xffffcc18	0xffffcc08
tamanho	5	0xffffcc04

???	###	###	###
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
###	0.000000	0.000000	0.000000
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
0.000000	0.000000	???	???
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
???	???	###	???
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
???	###	###	###
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60

calloc

```
T *a = (T *) calloc(n, sizeof(T));
```

• Semelhante ao malloc, mas os blocos são inicializados a 0.

calloc

Alocar um vetor

```
int i, tam;
double *pDHeap;
puts("Tamanho do vetor: ");
scanf("%d", &tam);
pDHeap = (double*) calloc(tam, sizeof(double));
for (i = 0; i < tam; i++) {
   printf("%lf ", pDHeap[i]);
free(pDHeap);
pDHeap = NULL;
```

```
Tamanho do vetor:
5
0.000000 0.000000 0.000000 0.000000
```

		stack
	???	0xffffcc20
	???	0xffffcc1c
	???	0xffffcc18
	???	0xffffcc14
	???	0xffffcc10
	???	0xffffcc0c
*pDHear	0xffffcc18	0xffffcc08
tamanho	5	0xffffcc04
]

???	###	###	###
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
###	0.000000	0.000000	0.000000
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
0.000000	0.000000	???	???
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
???	???	###	???
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
???	###	###	###
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60

```
ptr2 = (T *) realloc(ptr1, n);
```

- Realoca espaço previamente alocado com um novo tamanho (**n**).
 - A função recebe dois argumentos: o apontador (**ptr1**) que indica o atual bloco de memória e o novo tamanho (**n**) pretendido em bytes.
 - Permite aumentar ou diminuir tamanho.
 - Se o apontador passado como parâmetro for **NULL**, funcionamento é igual ao **malloc**.
- O **realloc** pode decidir:
 - Alocar um novo bloco de memória, copiar os dados e libertar o bloco original.
 - Expandir/contrair o bloco original "no local" sem alocar um novo.

```
ptr2 = (T *) realloc(ptr1, n);
```

- O bloco realocado **contém os dados existentes** no bloco antes da alteração.
 - Caso o novo bloco seja menor, apenas são mantidos os dados existentes até ao novo tamanho definido.
- Deve verificar-se se a realocação de memória teve sucesso.
 - Se sim, a função **retorna o apontador para a primeira posição** de memória alocada.
 - Caso não seja possível fazer a realocação de memória (p.e.. falta de memória), retorna
 NULL e mantém o anterior bloco e apontador (ptr1) é inalterado.
 - Por esta razão, é boa prática utilizar um apontador diferente do original na realocação, e fazer a atribuição ao original apenas em caso de sucesso na realocação.

1/4

```
int tam = 2;
double *pDTmp, *pDHeap;
pDHeap = (double*) calloc(tam, sizeof(double));
pDTmp = (double*)
        realloc(pDHeap, sizeof(double) * (tam * 2));
if (pDTmp != NULL) {
   tam *= 2;
   pDHeap = pDTmp;
free(pDHeap);
pDHeap = NULL;
```



		stack
	???	0xffffcc20
	???	0xffffcc1c
	???	0xffffcc18
	???	0xffffcc14
	???	0xffffcc10
ррнеар	???	0xffffcc0c
*pDTmp	???	0xffffcc08
tam	2	0xffffcc04
	· · · · · · · · · · · · · · · · · · ·	

???	###	###	###
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
###	???	???	???
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
111	???	???	???
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
???	???	###	???
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
???	###	###	###
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60

2/4

```
int tam = 2;
double *pDTmp, *pDHeap;
pDHeap = (double*) calloc(tam, sizeof(double));
pDTmp = (double*)
        realloc(pDHeap, sizeof(double) * (tam * 2));
if (pDTmp != NULL) {
   tam *= 2;
   pDHeap = pDTmp;
free(pDHeap);
pDHeap = NULL;
```



		stack
	•••	Stack
	333	0xffffcc20
	???	0xffffcc1c
	???	0xffffcc18
	???	0xffffcc14
	???	0xffffcc10
*pDHeap	0xffffcc18	0xffffcc0c
*pDTmp	???	0xffffcc08
tam	2	0xffffcc04

???	###	###	###
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
###	0	0	???
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
???	???	???	???
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
???	???	###	???
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
???	###	###	###
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60

3/4

```
int tam = 2;
double *pDTmp, *pDHeap;
pDHeap = (double*) calloc(tam, sizeof(double));
pDTmp = (double*)
        realloc(pDHeap, sizeof(double) * (tam * 2));
if (pDTmp != NULL) {
   tam *= 2;
   pDHeap = pDTmp;
free(pDHeap);
pDHeap = NULL;
```



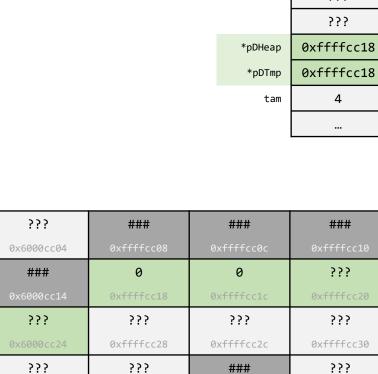
		stack
	???	0xffffcc20
	???	0xffffcc1c
	???	0xffffcc18
	???	0xffffcc14
	???	0xffffcc10
*pDHeap	0xffffcc18	0xffffcc0c
*pDTmp	0xffffcc18	0xffffcc08
tam	2	0xffffcc04

heap

???	###	###	###
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
###	0	0	???
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
???	???	???	???
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
???	???	###	???
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
???	###	###	###
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60

16

```
int tam = 2;
double *pDTmp, *pDHeap;
pDHeap = (double*) calloc(tam, sizeof(double));
pDTmp = (double*)
        realloc(pDHeap, sizeof(double) * (tam * 2));
if (pDTmp != NULL) {
   tam *= 2;
    pDHeap = pDTmp;
free(pDHeap);
pDHeap = NULL;
```



stack 0xffffcc20 0xffffcc1c 0xffffcc18 0xffffcc14 0xffffcc10 0xffffcc0c 0xffffcc08 0xffffcc04

???

???

???

???

???

4

???	###	###	###	
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10	
###	0	0	???	
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20	
???	???	???	???	
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30	•••
???	???	###	???	
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50	
???	###	###	###	
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60	

free(ptr)

- Liberta o bloco de memória apontado por **ptr**.
- Sempre que reservamos memória dinamicamente devemos libertar essa memória quando já não for necessária.
- Para libertarmos a memória alocada dinamicamente usamos a função **free**. Uma vez libertada a memória, esta pode ser reutilizada pelo sistema operativo.
- **free** não altera o valor do apontador. Este continua a apontar para o mesmo endereço, mas esse endereço agora é inválido.
 - Para indicar que a já não aponta para nada, é boa prática atribuir **NULL** ao apontador.

free

free

1/3

```
int main() {
    double *pDHeap1 = (double*) malloc(sizeof(double));
    double *pDHeap2 = (double*) malloc(sizeof(double));
    *pDHeap1 = 200.0;
    *pDHeap2 = 200.0;
    free(pDHeap1);
    pDHeap1 = NULL;
    free(pDHeap2);
    printf("%p %p %lf", pDHeap1, pDHeap2, *pDHeap2);
    return 0;
```



		stack
	???	0xffffcc20
	???	0xffffcc1c
	???	0xffffcc18
	???	0xffffcc14
	???	0xffffcc10
	???	0xffffcc0c
*pDHeap2	0xffffcc18	0xffffcc08
*pDHeap1	0x6000cc04	0xffffcc04

200.000000	###	###	###
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
###	200.000000	???	???
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
???	???	???	???
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
???	???	###	???
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
???	###	###	###
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60

free

2/3

```
int main() {
    double *pDHeap1 = (double*) malloc(sizeof(double));
    double *pDHeap2 = (double*) malloc(sizeof(double));
    *pDHeap1 = 200.0;
    *pDHeap2 = 200.0;
    free(pDHeap1);
    pDHeap1 = NULL;
    free(pDHeap2);
    printf("%p %p %lf", pDHeap1, pDHeap2, *pDHeap2);
    return 0;
```



		stack
	???	0xffffcc20
	???	0xffffcc1c
	???	0xffffcc18
	???	0xffffcc14
	???	0xffffcc10
	???	0xffffcc0c
*pDHeap2	0xffffcc18	0xffffcc08
*pDHeap1	0x0	0xffffcc04

200.000000	###	###	###
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
###	200.000000	???	???
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
???	???	???	???
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
???	???	###	???
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
???	###	###	###
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60
9X1111CC3C 9X1111CC3C			

free

3/3

```
int main() {
    double *pDHeap1 = (double*) malloc(sizeof(double));
    double *pDHeap2 = (double*) malloc(sizeof(double));
    *pDHeap1 = 200.0;
    *pDHeap2 = 200.0;
    free(pDHeap1);
    pDHeap1 = NULL;
    free(pDHeap2);
    printf("%p %p %lf", pDHeap1, pDHeap2, *pDHeap2);
    return 0;
```

```
0x0 0xffffcc18 200.000000
```

		stack
	???	0xffffcc20
	???	0xffffcc1c
	???	0xffffcc18
	???	0xffffcc14
	???	0xffffcc10
	???	0xffffcc0c
pDHeap2	0xffffcc18	0xffffcc08
pDHeap1	0x0	0xffffcc04

200.000000	###	###	###
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
###	200.000000	???	???
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
???	???	???	???
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
???	???	###	???
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
???	###	###	###
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60

Precauções

Fugas de memória 1/3

```
<u>^</u>
```

```
#include <stdlib.h>
int main() {
   int *p = (int*) malloc(sizeof(int));
   *p = 5;
   p = (int*) malloc(sizeof(int));
   *p = 8;
   free(p);
   p = NULL;
   return 0;
```



		stack
	???	0xffffcc20
	???	0xffffcc1c
	???	0xffffcc18
	???	0xffffcc14
	???	0xffffcc10
	???	0xffffcc0c
	???	0xffffcc08
*р	0x6000cc04	0xffffcc04

5	###	###	###
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
###	???	???	???
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
???	???	???	???
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
???	???	###	???
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
???	###	###	###
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60

Fugas de memória 2/3

```
#include <stdlib.h>
int main() {
   int *p = (int*) malloc(sizeof(int));
   *p = 5;
   p = (int*) malloc(sizeof(int));
   *p = 8;
   free(p);
   p = NULL;
   return 0;
```



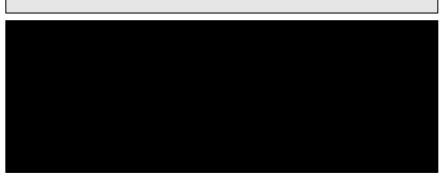
		stack
	???	0xffffcc26
	???	0xffffcc1
	???	0xffffcc18
	???	0xffffcc14
	???	0xffffcc10
	???	0xffffcc0
	???	0xffffcc08
*p	0xffffcc18	0xffffcc04
	<u> </u>	•

5	###	###	###
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
###	8	???	???
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
???	???	???	???
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
???	???	###	???
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
???	###	###	###
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60

Fugas de memória 3/3



```
#include <stdlib.h>
int main() {
   int *p = (int*) malloc(sizeof(int));
   *p = 5;
   p = (int*) malloc(sizeof(int));
   *p = 8;
   free(p);
   p = NULL;
   return 0;
```



		stack
	???	0xffffcc20
	???	0xffffcc1c
	???	0xffffcc18
	???	0xffffcc14
	???	0xffffcc10
	???	0xffffcc0c
	???	0xffffcc08
р	0x0	0xffffcc04

Memória alocada não libertada

				1100
♦ 5	###	###	###	
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10	
###	8	???	???	
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20	
???	???	???	???	
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30	•••
???	???	###	???	
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50	
???	###	###	###	
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60	

Referências inválidas 1/5

```
int *q, *p = (int*) malloc(sizeof(int));
q = p;
*q = 3;
free(q);
q = NULL;
printf("%d %d", *p, *q);
```



	•••	stack
	???	0xffffcc20
	???	0xffffcc1c
	???	0xffffcc18
	???	0xffffcc14
	???	0xffffcc10
	???	0xffffcc0c
*q	???	0xffffcc08
*р	0x6000cc04	0xffffcc04

???	###	###	###
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
###	???	???	???
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
???	???	???	???
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
???	???	###	???
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
???	###	###	###
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60

Referências inválidas 2/5

```
int *q, *p = (int*) malloc(sizeof(int));
q = p;
*q = 3;
free(q);
q = NULL;
printf("%d %d", *p, *q);
```



		stac
	???	0xffffcc2
	???	0xffffcc1
	???	0xffffcc0
*q	0x6000cc04	0xffffcc0
*р	0x6000cc04	0xffffcc0
	•••	

???	###	###	###
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
###	???	???	???
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
???	???	???	???
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
???	???	###	???
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
???	###	###	###
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60

Referências inválidas 3/5

```
int *q, *p = (int*) malloc(sizeof(int));
q = p;
*q = 3;
free(q);
q = NULL;
printf("%d %d", *p, *q);
```



		stack
	???	0xffffcc20
	???	0xffffcc1c
	???	0xffffcc18
	???	0xffffcc14
	???	0xffffcc10
	???	0xffffcc0c
*q	0x6000cc04	0xffffcc08
*p	0x6000cc04	0xffffcc04

3	###	###	###	
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10	
###	???	???	???	
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20	
???	333	???	???	
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30	•••
???	???	###	???	
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50	
???	###	###	###	
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60	

Referências inválidas 4/5

```
int *q, *p = (int*) malloc(sizeof(int));
q = p;
*q = 3;
free(q);
q = NULL;
printf("%d %d", *p, *q);
```



		stack
	???	0xffffcc20
	???	0xffffcc1c
	???	0xffffcc18
	???	0xffffcc14
	???	0xffffcc10
	???	0xffffcc0c
*q	0x0	0xffffcc08
*р	0x6000cc04	0xffffcc04
	•••	
	·	

3	###	###	###
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
###	???	???	???
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
???	???	???	???
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
???	???	###	???
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
???	###	###	###
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60

Referências inválidas 5/5



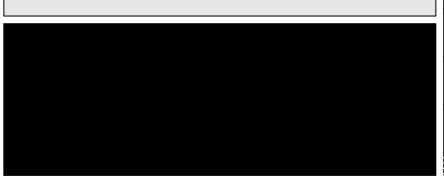
```
stack
int *q, *p = (int*) malloc(sizeof(int));
                                                                                                           ???
                                                                                                                     0xffffcc20
                                                                                                           ???
                                                                                                                     0xffffcc1c
q = p;
                                                                                                           ???
                                                                                                                     0xffffcc18
                                                                                                           ???
                                                                                                                     0xffffcc14
*q = 3;
                                                                                                                     0xffffcc10
                                                                                                           ???
free(q);
                                                                                                           ???
                                                                                                                     0xffffcc0c
q = NULL;
                                                                                                           0x0
                                                                                                                     0xffffcc08
                                                                                                       0x6000cc04
                                                                                                                     0xffffcc04
                                                             q aponta para NULL
printf("%d %d", *p, *q);
                                                                                                                     heap
```

RUN FAILED (exit value 1, total time: 100ms)

0x6000cc04 0xffffcc08 0xffffcc08 ### ??? ??? 0x6000cc14 0xffffcc18 0xffffcc1 ??? ??? ???	???
0x6000cc14 0xffffcc18 0xffffcc1	
	lc 0xffffcc20
??? ??? ???	
	333
0x6000cc24 0xffffcc28 0xffffcc2	2c 0xffffcc30
??? ??? ###	???
0x6000cc44 0xffffcc48 0xffffcc4	4c 0xffffcc50
??? ### ###	###
0x6000cc54 0xffffcc58 0xffffcc5	5c 0xffffcc60

Libertar memória 1/5

```
int *q, *p = (int*) malloc(sizeof(int));
q = p;
*q = 3;
free(q);
q = NULL;
free(p);
p = NULL;
```

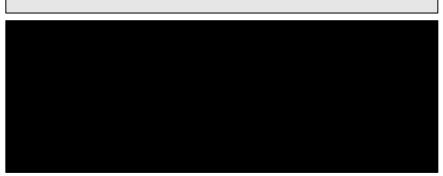


		stack
	???	0xffffcc20
	???	0xffffcc1c
	???	0xffffcc18
	???	0xffffcc14
	???	0xffffcc10
	???	0xffffcc0c
*q	???	0xffffcc08
*р	0x6000cc04	0xffffcc04
	•••	
	·	

???	###	###	###	
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10	
###	???	???	???	
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20	
???	???	???	???	
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30	
???	???	###	???	
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50	
???	###	###	###	
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60	

Libertar memória 2/5

```
int *q, *p = (int*) malloc(sizeof(int));
q = p;
*q = 3;
free(q);
q = NULL;
free(p);
p = NULL;
```



		stack
	???	0xffffcc20
	???	0xffffcc1c
	???	0xffffcc18
	???	0xffffcc14
	???	0xffffcc10
	???	0xffffcc0c
*q	0x6000cc04	0xffffcc08
*p	0x6000cc04	0xffffcc04
	•••	
	·	

???	###	###	###	
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10	
###	???	???	???	
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20	
???	???	???	???	
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30	
???	???	###	???	
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50	
???	###	###	###	
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60	

Libertar memória 3/5

```
<u>^</u>
```

```
int *q, *p = (int*) malloc(sizeof(int));
q = p;
*q = 3;
free(q);
q = NULL;
free(p);
p = NULL;
```



		stack
	???	0xffffcc20
	???	0xffffcc1c
	???	0xffffcc18
	???	0xffffcc14
	???	0xffffcc10
	???	0xffffcc0c
*q	0x6000cc04	0xffffcc08
*p	0x6000cc04	0xffffcc04

3	###	###	###
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
###	???	???	???
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
???	???	???	???
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
???	???	###	???
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
???	###	###	###
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60

Libertar memória 4/5

```
int *q, *p = (int*) malloc(sizeof(int));
q = p;
*q = 3;
free(q);
q = NULL;
free(p);
p = NULL;
```



	•••	stack
	???	0xffffcc20
	???	0xffffcc1c
	???	0xffffcc18
	???	0xffffcc14
	???	0xffffcc10
	???	0xffffcc0c
*q	0x0	0xffffcc08
*p	0x6000cc04	0xffffcc04

3	###	###	###	
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10	
###	???	???	???	
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20	
???	???	???	???	
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30	•••
???	???	###	???	
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50	
???	###	###	###	
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60	

Libertar memória 5/5



```
int *q, *p = (int*) malloc(sizeof(int));
q = p;
*q = 3;
free(q);
q = NULL;
free(p);
p = NULL;
```

RUN FAILED (exit value 1, total time: 147ms)

		stack
	???	0xffffcc20
	???	0xffffcc1c
	???	0xffffcc18
	???	0xffffcc14
	???	0xffffcc10
	???	0xffffcc0c
*q	0x0	0xffffcc08
*p	0x6000cc04	0xffffcc04

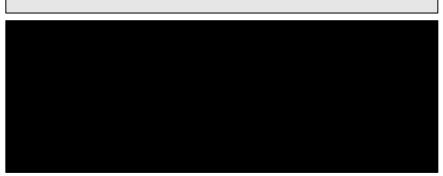
Memória já libertada

♦ 3	###	###	###	
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10	
###	???	???	???	
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20	
???	???	???	???	
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30	
???	???	###	???	
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50	
???	###	###	###	
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60	

Não libertar variáveis estáticas



```
int *p = NULL, *q = NULL, a = 5;
p = (int*) malloc(sizeof (int));
*p = 3;
q = &a;
free(p);
p = NULL;
free(q);
q = NULL;
```



		stack
	???	0xffffcc20
	???	0xffffcc1c
	???	0xffffcc18
	???	0xffffcc14
	???	0xffffcc10
a	5	0xffffcc0c
q	0x0	0xffffcc08
р	0x0	0xffffcc04

???	###	###	###
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
###	???	???	???
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
???	???	???	???
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
???	???	###	???
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
???	###	###	###
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60

Não libertar variáveis estáticas 2/5



```
int *p = NULL, *q = NULL, a = 5;
p = (int*) malloc(sizeof (int));
*p = 3;
q = &a;
free(p);
p = NULL;
free(q);
q = NULL;
```



		stack
	???	0xffffcc20
	???	0xffffcc1c
	???	0xffffcc18
	???	0xffffcc14
	???	0xffffcc10
a	5	0xffffcc0c
*q	0x0	0xffffcc08
*p	0x6000cc04	0xffffcc04

3	###	###	###
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
###	???	???	???
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
???	???	???	???
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
???	???	###	???
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
???	###	###	###
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60

Não libertar variáveis estáticas 3/5



```
int *p = NULL, *q = NULL, a = 5;
p = (int*) malloc(sizeof (int));
*p = 3;
q = &a;
free(p);
p = NULL;
free(q);
q = NULL;
```



		stack
	???	0xffffcc20
	???	0xffffcc1c
	???	0xffffcc18
	???	0xffffcc14
	???	0xffffcc10
а	5	0xffffcc0c
*q	0xffffcc0c	0xffffcc08
*р	0x6000cc04	0xffffcc04

0x6000cc04 0xffffcc08 0xffffcc0c 0xffffcc10 ### ??? ??? ??? 0x6000cc14 0xffffcc18 0xffffcc1c 0xffffcc20 ??? ??? ??? ??? 0x6000cc24 0xffffcc28 0xffffcc2c 0xffffcc30 ??? ??? ### ???
0x6000cc14 0xffffcc18 0xffffcc1c 0xffffcc20 ??? ??? ??? ??? 0x6000cc24 0xffffcc28 0xffffcc2c 0xffffcc30 ??? ??? ### ???
??? ??? ??? 0x6000cc24 0xffffcc28 0xffffcc2c 0xffffcc30 ??? ??? ### ???
0x6000cc24 0xffffcc28 0xffffcc2c 0xffffcc30 ??? ??? ### ???
??? ??? ### ???
0x6000cc440xffffcc480xffffcc4c0xffffcc50
??? ### ###
0x6000cc54 0xffffcc58 0xffffcc5c 0xffffcc60

Não libertar variáveis estáticas 4/5

```
int *p = NULL, *q = NULL, a = 5;
p = (int*) malloc(sizeof (int));
*p = 3;
q = &a;
free(p);
p = NULL;
free(q);
q = NULL;
```



		stack
	???	0xffffcc20
	???	0xffffcc1c
	???	0xffffcc18
	???	0xffffcc14
	???	0xffffcc10
а	5	0xffffcc0c
q	0xffffcc0c	0xffffcc08
р	0x0	0xffffcc04
		•

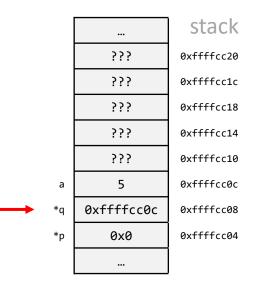
3	###	###	###	
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10	
###	???	???	???	
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20	
???	???	???	???	
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30	•••
???	???	###	???	
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50	
???	###	###	###	
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60	
				•

Não libertar variáveis estáticas 5/5



```
int *p = NULL, *q = NULL, a = 5;
p = (int*) malloc(sizeof (int));
*p = 3;
q = &a;
free(p);
p = NULL;
free(q);
q = NULL;
```

RUN FAILED (exit value 1, total time: 144ms)



heap

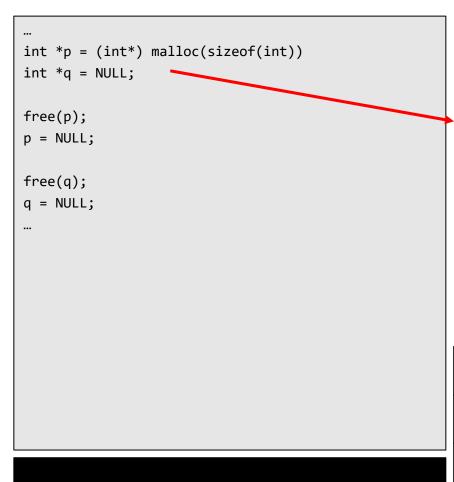
3	###	###	###
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
###	???	???	???
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
???	???	???	???
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
???	???	###	???
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
???	###	###	###
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60

Tentativa de libertação de

memória estática.

Não libertar apontadores indefinidos





Para evitar problemas inesperados, deve sempre inicializar um apontador com a alocação de memória ou com o valor **NULL**.

		stack
	???	0xffffcc20
	???	0xffffcc1c
	???	0xffffcc18
	???	0xffffcc14
	???	0xffffcc10
	???	0xffffcc0c
q	0x0	0xffffcc08
р	0x6000cc04	0xffffcc04

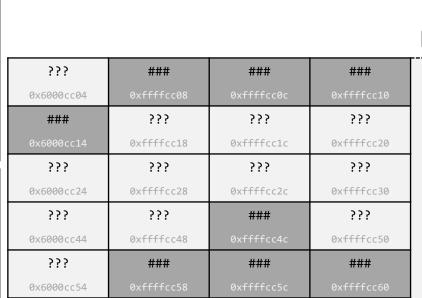
heap

???	###	###	###
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
###	???	???	???
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
???	???	???	???
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
???	???	###	???
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
???	###	###	###
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60

Apontadores para apontadores

Apontadores para Apontadores 1/4

```
int num = 5;
int *p1 = #
int **p2 = &p1;
printf("%p %d\n", &num, num);
printf("%p %p %d\n", &p1, p1, *p1);
printf("\n%p %p %p %d", &p2, p2, *p2, **p2);
```



stack ??? 0xffffcc20 0xffffcc1c 333 ??? 0xffffcc18 ??? 0xffffcc14 ??? 0xffffcc10 **p2 0xffffcc08 0xffffcc0c 0xffffcc04 0xffffcc08 *p1 5 0xffffcc04 num

Apontadores para Apontadores 2/4

```
int num = 5;
int *p1 = #
int **p2 = &p1;
printf("%p %d\n", &num, num);
printf("%p %p %d\n", &p1, p1, *p1);
printf("\n%p %p %p %d", &p2, p2, *p2, **p2);
```

```
0xffffcc04 5
```

		stac
	???	0xffffcc2
	???	0xffffcc1
*p2	0xffffcc08	0xffffcc0
*p1	0xffffcc04	0xffffcc0
num	5	0xffffcc0

???	###	###	###
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
###	???	???	???
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
???	???	???	???
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
???	???	###	???
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
???	###	###	###
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60

Apontadores para Apontadores 3/4

```
int num = 5;
int *p1 = #
int **p2 = &p1;
printf("%p %d\n", &num, num);
printf("%p %p %d\n", &p1, p1, *p1);
printf("\n%p %p %p %d", &p2, p2, *p2, **p2);
```

```
0xffffcc04 5
0xffffcc08 0xffffcc04 5
```

		stack
	???	0xffffcc20
	???	0xffffcc1c
	???	0xffffcc18
	???	0xffffcc14
	???	0xffffcc10
*p2	0xffffcc08	0xffffcc0c
*p1	0xffffcc04	0xffffcc08
num	5	0xffffcc04

???	###	###	###
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
###	???	???	???
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
???	???	???	???
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
???	???	###	???
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
???	###	###	###
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60

Apontadores para Apontadores 4/4

```
int num = 5;
int *p1 = #
int **p2 = &p1;
printf("%p %d\n", &num, num);
printf("%p %p %d\n", &p1, p1, *p1);
printf("\n%p %p %p %d", &p2, p2, *p2, **p2);
```

```
0xffffcc04 5
0xffffcc08 0xffffcc04 5
0xffffcc0c 0xffffcc08 0xffffcc04 5
```

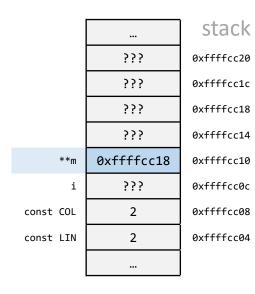
		stack
	???	0xffffcc20
	???	0xffffcc1c
	???	0xffffcc18
	???	0xffffcc14
	???	0xffffcc10
*p2	0xffffcc08	0xffffcc0c
*p1	0xffffcc04	0xffffcc08
num	5	0xffffcc04

???	###	###	###
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
###	???	???	???
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
???	???	???	???
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
???	???	###	???
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
???	###	###	###
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60

Matrizes

Matrizes 1/5

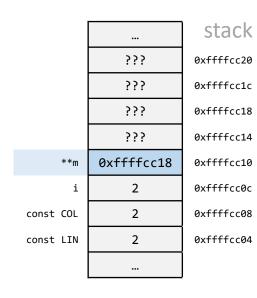
```
int i, **m = (int **) malloc(LIN * sizeof(int *));
for (i = 0; i < LIN; ++i) {
   m[i] = (int *) malloc(COL * sizeof(int));
m[0][0] = 10;
printf("%d", m[0][0]);
for (i = 0; i < LIN; ++i) {
   free(m[i]);
   m[i] = NULL;
free(m);
m = NULL;
```



???	###	###	###
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
###	???	???	???
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
???	???	???	???
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
???	???	###	???
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
???	###	###	###
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60

Matrizes 2/5

```
int i, **m = (int **) malloc(LIN * sizeof(int *));
for (i = 0; i < LIN; ++i) {
   m[i] = (int *) malloc(COL * sizeof(int));
m[0][0] = 10;
printf("%d", m[0][0]);
for (i = 0; i < LIN; ++i) {
   free(m[i]);
   m[i] = NULL;
free(m);
m = NULL;
```



???	###	###	###	
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10	
###	0xffffcc20	0xffffcc28	???	
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20	
???	???	333	???	
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30	•••
???	???	###	???	
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50	
???	###	###	###	
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60	

Matrizes 3/5

```
int i, **m = (int **) malloc(LIN * sizeof(int *));
for (i = 0; i < LIN; ++i) {
   m[i] = (int *) malloc(COL * sizeof(int));
m[0][0] = 10;
printf("%d", m[0][0]);
for (i = 0; i < LIN; ++i) {
   free(m[i]);
   m[i] = NULL;
free(m);
m = NULL;
```

```
10
```

		stack
	???	0xffffcc20
	???	0xffffcc1c
	???	0xffffcc18
	???	0xffffcc14
**m	0xffffcc18	0xffffcc10
i	2	0xffffcc0c
i const COL	2 2	0xffffcc0c 0xffffcc08
_	_	
const COL	2	0xffffcc08

???	###	###	###
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
###	0xffffcc20	0xffffcc28	10
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
???	???	333	???
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
???	???	###	???
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
???	###	###	###
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60

Matrizes 4/5

```
int i, **m = (int **) malloc(LIN * sizeof(int *));
for (i = 0; i < LIN; ++i) {
   m[i] = (int *) malloc(COL * sizeof(int));
m[0][0] = 10;
printf("%d", m[0][0]);
for (i = 0; i < LIN; ++i) {
   free(m[i]);
   m[i] = NULL;
free(m);
m = NULL;
```



	ctack
	stack
???	0xffffcc20
???	0xffffcc1c
???	0xffffcc18
???	0xffffcc14
0xffffcc18	0xffffcc10
2	0xffffcc0c
2	0xffffcc08
2	0xffffcc04
•••	
	??? ??? ??? 0xffffcc18 2 2

???	###	###	###
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
###	0x0	0x0	10
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
???	???	???	???
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
???	???	###	???
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
???	###	###	###
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60

Matrizes 5/5

```
int i, **m = (int **) malloc(LIN * sizeof(int *));
for (i = 0; i < LIN; ++i) {
   m[i] = (int *) malloc(COL * sizeof(int));
m[0][0] = 10;
printf("%d", m[0][0]);
for (i = 0; i < LIN; ++i) {
   free(m[i]);
   m[i] = NULL;
free(m);
m = NULL;
```



		stack
	???	0xffffcc20
	???	0xffffcc1
	???	0xffffcc1
	???	0xffffcc1
**m	0x0	0xffffcc1
i	2	0xffffcc0
const COL	2	0xffffcc0
const LIN	2	0xffffcc0
	•••	

???	###	###	###
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
###	0x0	0x0	10
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
???	???	???	???
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
???	???	###	???
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
???	###	###	###
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60

Alocar memória em funções

Alocar memória em funções 1/4

```
int criarVetor(int *vetor) {
    int tamanho;
    scanf("%d", &tamanho);
    vetor = (int *) calloc(tamanho, sizeof (int));
    if (tamanho > 0 && vetor != NULL) {
        printf("\nt: %d p: %p", tamanho, vetor);
        return tamanho;
    return 0;
int main() {
    int tamanho, *vetor = NULL;
    if ((tamanho = criarVetor(vetor)) > 0) {
        printf("\nt: %d p: %p", tamanho, vetor);
       free(vetor);
       vetor = NULL;
   } else {
        puts("vetor nao criado");
    return 0;
```



		stack
	???	0xffffcc20
	???	0xffffcc1c
	???	0xffffcc18
	???	0xffffcc14
	???	0xffffcc10
	???	0xffffcc0c
*vetor	0x0	0xffffcc08
amanho	???	0xffffcc04

heap

### 0xffffcc08 ??? 0xffffcc18 ???	### 0xffffcc0c ??? 0xffffcc1c ???	### 0xffffcc10 ???? 0xffffcc20 ???	
??? 0xffffcc18	??? 0xfffffcc1c	??? 0xffffcc20	
0xffffcc18	0xffffcc1c	0xffffcc20	
???	???	???	
0xffffcc28	0xffffcc2c	0xffffcc30	
???	###	???	
0xffffcc48	0xffffcc4c	0xffffcc50	
###	###	###	
0xffffcc58	0xffffcc5c	0xffffcc60	
	??? 0xffffcc48 ###	??? ### 0xffffcc48 ### ###	??? ### ??? 0xffffcc48 0xffffcc4c 0xffffcc50 ### ###

54

Alocar memória em funções 2/4

```
int criarVetor(int *vetor) {
    int tamanho;
    scanf("%d", &tamanho);
    vetor = (int *) calloc(tamanho, sizeof (int));
    if (tamanho > 0 && vetor != NULL) {
        printf("\nt: %d p: %p", tamanho, vetor);
        return tamanho;
    return 0;
int main() {
    int tamanho, *vetor = NULL;
    if ((tamanho = criarVetor(vetor)) > 0) {
        printf("\nt: %d p: %p", tamanho, vetor);
       free(vetor);
       vetor = NULL;
   } else {
        puts("vetor nao criado");
    return 0;
```



		stack
	???	0xffffcc20
	???	0xffffcc1c
	???	0xffffcc18
	???	0xffffcc14
tamanho	4	0xffffcc10
*vetor	0x0	0xffffcc0c
*vetor	0x0	0xffffcc08
tamanho	???	0xffffcc04
	•••	

???	###	###	###
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
###	???	???	???
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
???	???	???	???
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
???	???	###	???
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
???	###	###	###
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60

Alocar memória em funções 3/4

```
int criarVetor(int *vetor) {
    int tamanho;
    scanf("%d", &tamanho);
    vetor = (int *) calloc(tamanho, sizeof (int));
    if (tamanho > 0 && vetor != NULL) {
        printf("\nt: %d p: %p", tamanho, vetor);
        return tamanho;
    return 0;
int main() {
    int tamanho, *vetor = NULL;
    if ((tamanho = criarVetor(vetor)) > 0) {
        printf("\nt: %d p: %p", tamanho, vetor);
       free(vetor);
       vetor = NULL;
   } else {
        puts("vetor nao criado");
    return 0;
```

```
t: 4 p: 0xffffcc18
```

		stack
	???	0xffffcc20
	???	0xffffcc1c
	???	0xffffcc18
	???	0xffffcc14
tamanho	4	0xffffcc10
*vetor	0xffffcc18	0xffffcc0c
*vetor	0x0	0xffffcc08
tamanho	???	0xffffcc04

???	###	###	###
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
###	0	0	0
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
0	???	???	???
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
???	???	###	???
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
???	###	###	###
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60

Alocar memória em funções 4/4

```
int criarVetor(int *vetor) {
    int tamanho;
    scanf("%d", &tamanho);
    vetor = (int *) calloc(tamanho, sizeof (int));
    if (tamanho > 0 && vetor != NULL) {
        printf("\nt: %d p: %p", tamanho, vetor);
        return tamanho;
    return 0;
int main() {
    int tamanho, *vetor = NULL;
    if ((tamanho = criarVetor(vetor)) > 0) {
        printf("\nt: %d p: %p", tamanho, vetor);
       free(vetor);
       vetor = NULL;
   } else {
        puts("vetor nao criado");
    return 0;
```

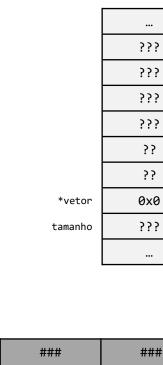
```
4
t: 4 p: 0xffffcc18
t: 4 p: 0x0
```

		stack
	???	0xffffcc20
	???	0xffffcc1c
	???	0xffffcc18
	???	0xffffcc14
	4	0xffffcc10
	0xffffcc18	0xffffcc0c
*vetor	0x0	0xffffcc08
tamanho	4	0xffffcc04

333	###	###	###
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
###	0	0	0
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
0	???	???	???
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
???	???	###	???
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
???	###	###	###
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60

Alocar memória em funções 1/4

```
int criarVetor(int **vetor) {
    int tamanho;
    scanf("%d", &tamanho);
    *vetor = (int *) calloc(tamanho, sizeof (int));
    if (tamanho > 0 && *vetor != NULL) {
        printf("\nt: %d p: %p", tamanho, *vetor);
        return tamanho;
    return 0;
int main() {
    int tamanho, *vetor = NULL;
   if ((tamanho = criarVetor(&vetor)) > 0) {
        printf("\nt: %d p: %p", tamanho, vetor);
        free(vetor);
        vetor = NULL;
   } else {
        puts("vetor nao criado");
    return 0;
```



stack

0xffffcc20

0xffffcc1c

0xffffcc18

0xffffcc14

0xffffcc10

0xffffcc0c

0xffffcc0c

???	###	###	###
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
###	???	???	???
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
???	???	???	???
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
???	???	###	???
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
???	###	###	###
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60

Alocar memória em funções 2/4

```
int criarVetor(int **vetor) {
    int tamanho;
    scanf("%d", &tamanho);
    *vetor = (int *) calloc(tamanho, sizeof (int));
    if (tamanho > 0 && *vetor != NULL) {
        printf("\nt: %d p: %p", tamanho, *vetor);
        return tamanho;
    return 0;
int main() {
    int tamanho, *vetor = NULL;
   if ((tamanho = criarVetor(&vetor)) > 0) {
        printf("\nt: %d p: %p", tamanho, vetor);
        free(vetor);
       vetor = NULL;
   } else {
        puts("vetor nao criado");
    return 0;
```



		stack
	???	0xffffcc20
	???	0xffffcc1c
	???	0xffffcc18
	???	0xffffcc14
tamanho	4	0xffffcc10
**vetor	0xffffcc08	0xffffcc0c
*vetor	0x0	0xffffcc08
tamanho	???	0xffffcc04

???	###	###	###
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
###	???	???	???
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
???	???	???	???
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
???	???	###	???
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
???	###	###	###
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60

Alocar memória em funções 3/4

```
int criarVetor(int **vetor) {
    int tamanho;
    scanf("%d", &tamanho);
    *vetor = (int *) calloc(tamanho, sizeof (int));
    if (tamanho > 0 && *vetor != NULL) {
        printf("\nt: %d p: %p", tamanho, *vetor);
        return tamanho;
    return 0;
int main() {
    int tamanho, *vetor = NULL;
   if ((tamanho = criarVetor(&vetor)) > 0) {
        printf("\nt: %d p: %p", tamanho, vetor);
       free(vetor);
       vetor = NULL;
   } else {
        puts("vetor nao criado");
    return 0;
```

```
t: 4 p: 0xffffcc18
```

		stack
	???	0xffffcc20
	???	0xffffcc1c
	???	0xffffcc18
	???	0xffffcc14
tamanho	4	0xffffcc10
*vetor	0xffffcc08	0xffffcc0c
*vetor	0xffffcc18	0xffffcc08
tamanho	???	0xffffcc04

???	###	###	###
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
###	0	0	0
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
0	???	???	???
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
???	???	###	???
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
???	###	###	###
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60

Alocar memória em funções 4/4

```
int criarVetor(int **vetor) {
    int tamanho;
    scanf("%d", &tamanho);
    *vetor = (int *) calloc(tamanho, sizeof (int));
    if (tamanho > 0 && *vetor != NULL) {
        printf("\nt: %d p: %p", tamanho, *vetor);
        return tamanho;
    return 0;
int main() {
    int tamanho, *vetor = NULL;
    if ((tamanho = criarVetor(&vetor)) > 0) {
        printf("\nt: %d p: %p", tamanho, vetor);
        free(vetor);
        vetor = NULL;
   } else {
        puts("vetor nao criado");
    return 0;
```

```
t: 4 p: 0xffffcc18
t: 4 p: 0xffffcc18
```

		stack	
	???	0xffffcc20	
	???	0xffffcc1c	
	???	0xffffcc18	
	???	0xffffcc14	
	4	0xffffcc10	
	0xffffcc08	0xffffcc0c	
*vetor	0xffffcc18	0xffffcc08	
tamanho	4	0xffffcc04	

???	###	###	###
0x6000cc04	0xffffcc08	0xffffcc0c	0xffffcc10
###	0	0	0
0x6000cc14	0xffffcc18	0xffffcc1c	0xffffcc20
0	???	???	???
0x6000cc24	0xffffcc28	0xffffcc2c	0xffffcc30
???	???	###	???
0x6000cc44	0xffffcc48	0xffffcc4c	0xffffcc50
???	###	###	###
0x6000cc54	0xffffcc58	0xffffcc5c	0xffffcc60

Leitura recomendada

• (Capítulo 12) Damas, L. Linguagem C; FCA – Editora de Informática, Lda, 1999; ISBN 9789727221561.

