

Alocação Dinâmica

Profa. Dra. Elisa Yumi Nakagawa
2. Semestre 2017

Sumário

- Funções para alocação de memória
 - malloc()
 - calloc()
 - realloc()
 - free()

Funções para alocação de memória

- malloc(), calloc(), realloc(), free()
- São funções utilizadas para trabalhar com alocação dinâmica (em tempo de execução) de memória.
- A memória é alocada a partir de uma área conhecida como **heap**.

Malloc

```
void *malloc(size_t size);
```

- size = tamanho do bloco de memória em **bytes**.
- size_t é um tipo pré-definido usado em stdlib.h que faz size_t ser equivalente ao tipo unsigned int.
- Retorna um ponteiro para o bloco de memória alocado.

Malloc

```
void *malloc(size_t size);
```

- Quando não consegue alocar memória, retorna um ponteiro nulo.
- A região alocada contém valores desconhecidos
- Sempre verifique o valor de retorno!

Malloc

char *str;

 type-casting: void para char

```
if((str = (char *)malloc(sizeof(char))) == NULL) {  
    printf("Espaco insuficiente para alocar buffer \n");  
    exit(0);  
}  
printf("Espaco alocado para str\n");
```

Malloc

```
int *num;  
if((num = (int *)malloc(50 * sizeof(int))) == NULL) {  
    printf("Espacio insuficiente para aloca buffer \n");  
    exit (0);  
}  
printf("Espaco alocado para num\n");
```

Malloc

```
#include <stdio.h>
#include <stdlib.h>
typedef struct {
    int dia, mes, ano;
} data;
int main() {
    data *d;
    if((d = (data *) malloc (sizeof (data))) == NULL) {
        printf("Espaco insuficiente para alocar buffer \n");
        exit(1);
    }
    d->dia = 13;
    d->mes = 06;
    d->ano = 2017;
    printf ("%d/%d/%d\n", d->dia, d->mes, d->ano);
}
```

Malloc

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

int main() {
    int *v;
    int n, i;
    scanf ("%d", &n);
    if((v = malloc (n * sizeof (int))) == NULL) {
        printf("Espaco insuficiente para alocar buffer \n");
        exit(1);
    }
    for (i = 0; i < n; ++i)
        scanf ("%d", &v[i]);
    for (i = 0; i < n; ++i)
        printf ("%d ", v[i]);
}
```

Calloc

```
void * calloc ( size_t num, size_t size );
```

- `calloc()` aloca um bloco de memória para um “array” de *num* elementos, sendo cada elemento de tamanho *size*.
- A função retorna um ponteiro para o primeiro byte
- Se não houver alocação, retorna um ponteiro nulo

Calloc

```
#include <stdlib.h>

unsigned int num;

int *ptr;

printf("Digite o numero de variaveis do tipo int.");
scanf("%d", &num);

if((ptr = (int *)calloc(num, sizeof(int))) == NULL) {
    printf("Espaco insuficiente para alocar \"num\" \n");
    exit(1);
}

printf("Espaco alocado com o calloc\n");
```

Realloc

```
void * realloc (void * ptr, size_t size );
```

- `realloc()` aumenta ou reduz o tamanho de um bloco de memória previamente alocado com `malloc()` ou `calloc()`
- *ptr* aponta para o bloco original de memória.
- *size* indica o novo tamanho desejado em bytes

Realloc

- Se houver espaço para expandir, a memória adicional é alocada e prt é retornado.
- Se não houver espaço suficiente para expandir o bloco atual, um novo bloco de tamanho size é alocado em outra região da memória.
- O conteúdo do bloco original é copiado para o novo bloco.
- O espaço de memória do bloco original é liberado e a função retorna um ponteiro para o novo bloco.

Realloc

- Se o argumento *size* for zero, a memória indicada por *ptr* é liberada e a função retorna NULL.
- Se não houver memória suficiente para a realocação (nem para um novo bloco), a função retorna NULL e o bloco original permanece inalterado.
- Se o argumento *ptr* for NULL, a função atua como malloc().

Exemplo: calloc seguido de realloc

```
unsigned int num; int *ptr;
printf("Digite o numero de variaveis do tipo int: ");
scanf("%d", &num);
if((ptr = (int *)calloc(num, sizeof(int))) == NULL){
    printf("Espaco insuficiente para alocar \"%num\" \n");
    exit(1);
}
//duplica o tamanho da região alocada para ptr
if((ptr = (int *)realloc(ptr, 2*num*sizeof(int))) == NULL){
    printf("Espaco insuficiente para alocar \"%num\" \n");
    exit(1);
}
printf("Novo espaço \"realocado\" com sucesso\n");
```

Free

```
void free ( void * ptr );
```

- O espaço alocado dinamicamente com `calloc()` ou `malloc()` não retorna ao sistema quando o fluxo de execução deixa uma função.
- A função `free()` “desaloca”/libera um espaço de memória previamente alocado usando *malloc*, *calloc* ou *realloc*.
- O espaço de memória fica disponível para uso futuro.

Free

```
void free ( void * ptr );
```

- A função deixa o valor de *ptr* inalterado, porém, apontando para uma região inválida.
- O ponteiro não se torna NULL.
- Se for passado um ponteiro nulo, nenhuma ação será realizada.

Exercício

Aponte os problemas que ocorrem nesse programa.

```
#include <stdlib.h>
void main () {
    int *p, *q;
    p = malloc (sizeof (int));
    *p = 123;
    q = malloc (sizeof (int));
    *q = *p;
    q = p;
    free (p);
    free (q);
}
```