# Alocação Dinâmica de Memória

Professor: Jó Ueyama
Estagiário PAE: Heitor Freitas

Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo

SSC 300 - Linguagem de Programação e Aplicações

São Carlos, 06 de Outubro de 2014

# Sumário

# Sumário

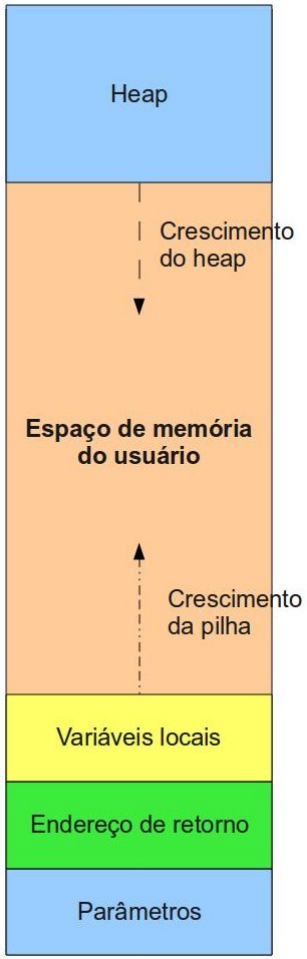
# Funções para Alocação de Memória

- Malloc( );
- Calloc( );
- Realloc( ) e;
- Free( ).

São funções utilizadas para trabalhar com alocação dinâmica **(em tempo de exeução)** de memória e estão na biblioteca *stdlib*.

# Funções para Alocação de Memória

A memória é alocada a partir de uma área conhecida como **heap**.



# Sumário

# Malloc( )

A função **malloc** (abreviatura de *memory allocation*) aloca um bloco de bytes consecutivos na memória do computador e devolve o endereço desse bloco.

# Malloc( )

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

- **size** é o tamanho do bloco de mémoria em *bytes*;
- Retorna um ponteiro para o bloco de memória alocado;
- Quando não conseguir alocar a memória, retorna um ponteiro nulo;
- A região alocada contém valores desconhecidos;
- Sempre verifique o valor do retorno!!!.

# Exemplo malloc( )

```c
#include <stdlib.h>

char *str;

if ((str = (char *)malloc(100)) == NULL)
{
    printf("Espaço insuficiente para alocar buffer \n");
    exit(1);
}

printf("Espaço alocado para str \n");

```

# Exemplo malloc( )

```c
#include <stdlib.h>

int *num;

if ((num = (int *)malloc(50 * sizeof(int))) == NULL)
{
    printf("Espaço insuficiente para alocar buffer \n");
    exit(1);
}

printf("Espaço alocado para str \n");

```

# Sumário

# Calloc( )

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

- A função **calloc()** aloca um bloco de memória para um *array* de **num** elementos, sendo cada elemento de tamanho **size**;
- A região alocada da memória é inicializada com o valor zero;
- A função retorna um ponteiro para o primeiro *byte*;
- Se não houver alocação, retorna um ponteiro nulo.

# Exemplo calloc( )

```c
#include <stdlib.h>

unsigned int num;
int *ptr;

printf("Digite um número de variáveis do tipo int:");
scanf("&d", &num);

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

printf("Espaço alocado com calloc \n");
```

# Sumário

# Realloc( )

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

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

# Retornos do Realloc( )

- Se houver espaço para expandir, a memória adicional é alocada e retorna **ptr**;
- 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 um **malloc()**.

# Retornos do Realloc( )

- Se o argumento **size** for zero, a memória indicada por **ptr** é liberada e a função retorna NULL;
- Se não houver espaço suficiente para expandir o bloco atual, um novo bloco de tamanho size é alocado numa outra região da memória e o conteúdo do bloco original é copiado para o novo. O espaço de memória do bloco original é liberado e a função retorna um ponteiro para o novo bloco;

# Exemplo realloc( )

```c
#include <stdlib.h>

unsigned int num, *ptr;

printf("Digite um número de variáveis do tipo int:");
scanf("&d", &num);

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

if ((ptr = (int *)realloc(ptr, 2 * num * sizeof(int))) == NULL)
{
    printf("Espaço insuficiente para alocar num \n");
    exit(1);
}

printf("Novo espaço realocado com sucesso. \n");
```

# Sumário

# Free( )

**free(ptr);**

- "Desaloca" /libera um espaço de memória previamente alocado usando **malloc**, **calloc** ou **realloc**, tornando-o disponível para uso futuro;
- A função deixa o valor de **ptr** inalterado, porém apontando para uma região inválida (note que o ponteiro não se torna NULL);
- Se for passado um ponteiro nulo, nenhuma ação será realizada.

# Problemas da utilização de memória dinâmica

O uso de memória dinâmica administrada pelo próprio programador abre a possibilidade para a ocorrência de problemas relativos ao mau uso da memória que muitas vezes geram vulnerabilidades de segurança e erros de difícil localização.

# Problemas na utilização de memória dinâmica

Entre os problemas mais comuns estão:

- Vazamento de memória: ter uma região de memória alocada porém sem ponteiros para ela o que impede sua libertação;
- *Buffer overflow*: escrever na memória para além do limite da área alocada;
- Se for passado um ponteiro nulo, nenhuma ação será realizada;
- Uso de áreas liberadas: escrever numa região da memória que já foi libertada pela função **free** e;
- Falha do malloc: se não houver memória disponível, o malloc irá falhar.

# Exercício

- Escreva um programa que aloque a memória para cada caractere digitado pelo usuário e imprima no final o texto completo;

- Implemente um código capaz de realizar a soma de duas matrizes, sendo os tamanhos e os conteúdos das mesmas informado pelo usuário.