Revisão de C: Alocação Dinâmica de Memória

Algoritmos e Estruturas de Dados 2
2017-1
Flavio Figueiredo (http://flaviovdf.github.io)

Vamos Finalizar a Revisão

- **Structs**
- Ponteiros (novamente)
- Alocação Dinâmica

Structs

- Combinam conjuntos de dados
- Campos são armazenados em sequência
- Acessado por um único ponteiro

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

struct NumeroComplexo {
  double real;
  double imag;
};

//... diversas funções que trabalham com números complexos

int main(void) {
  struct NumeroComplexo complexo = {3.0, 2.0};
  struct NumeroComplexo outroComplexo;
  outroComplexo.real = 7.0;
  outroComplexo.imag = 9.0;
  return 0;
}
```

Typedef Struct

- Typedefs simplificam o uso de struct
  - De qualquer tipo de dados por sinal
- Pseudonimo
- Devemos usar bastante na disciplina

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

struct NumeroComplexo {
  double real;
  double imag;
};
typedef struct NumeroComplexo complexo_t;

//... diversas funções que trabalham com números
complexos

int main(void) {
  complexo_t complexo = {3.0, 2.0};
  complexo_t outroComplexo;
  outroComplexo.real = 7.0;
  outroComplexo.imag = 9.0;
  return 0;
}
```

Typedef Struct

- Pode ser utilizado independente de struct
- Podemos criar typedefs para várias coisas
- Um tipo que é um ponteiro para função

```c
//Ponteiro para função
typedef int (*comparador)(int valor1, int valor2);
```

```c
//Ponteiro para int
typedef int *int_ptr;

//Ponteiro para double
typedef double *double_ptr;

int main() {
  int x = 2;
  double y = 3.2;

  int_ptr ponteiroParaX = &x;
  double_ptr ponteiroParaY = &y;
  return 0;
}
```

Vamos Finalizar a Revisão

- Structs
- **Ponteiros (novamente)**
- Alocação Dinâmica

Ponteiros

- Vários dos exemplos até agora já fizeram uso de ponteiros
- Pontos chave
  - Operadores & e *
  - & vai criar um ponteiro para uma variável
    - int *x_ptr
  - * Tem duas funções
    - Ao declarar o ponteiro
    - Acessar o valor para onde o ponteiro leva

```c
int main() {
  int x = 2;
  int *x_ptr = &x;
  int **x_ptr_ptr = &x_ptr;

  printf("Valor de x %d\n", x);
  printf("Valor de x_ptr %p\n", x_ptr);
  printf("Valor de x_ptr_ptr %p\n", x_ptr_ptr);

  int x_new = *x_ptr;
  printf("Valor de *(x_ptr) %d\n", *x_new);
  printf("Valor de *(x_ptr_ptr) %p\n", *x_ptr_ptr);
  return 0;
}
```

Ponteiros para Estruturas

- Declaração e inicialização igual

Variável	Posição	Valor
d1.dia	0x80	8
d1.mes	0x84	3
d1.ano	0x88	2012
ptr	0x8c	0x80
i	0x90	0

```c
struct data { int dia; int mes; int ano; };

int main(void) {
  struct data d1;
  struct data *ptr = &d1;
  int i = 0;

  (*ptr).dia = 8;
  (*ptr).mes = 3;
  (*ptr).ano = 2012;
}
```

Ponteiros para Funções

- Código de Exemplo aqui
- https://goo.gl/Y721pa

- Diversas utilidades
  - Compare 2 números usando método X
  - Mande um pacote pela rede com o método Y
  - ...

Vamos Finalizar a Revisão

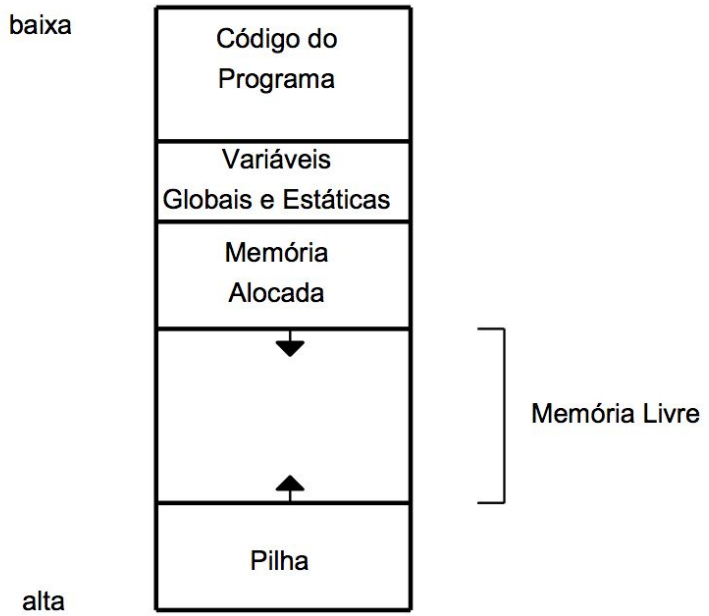
- Structs
- Ponteiros (novamente)
- **Alocação Dinâmica**

Alocação Dinâmica de Memória

- Preferível se não souber quanto de espaço vamos precisar
  - Será bastante utilizada nesta disciplina
- Alocação estática é fixa
  - char matrix[20]
  - Sempre 20 posições
- Alocação dinâmica tem maior flexibilidade

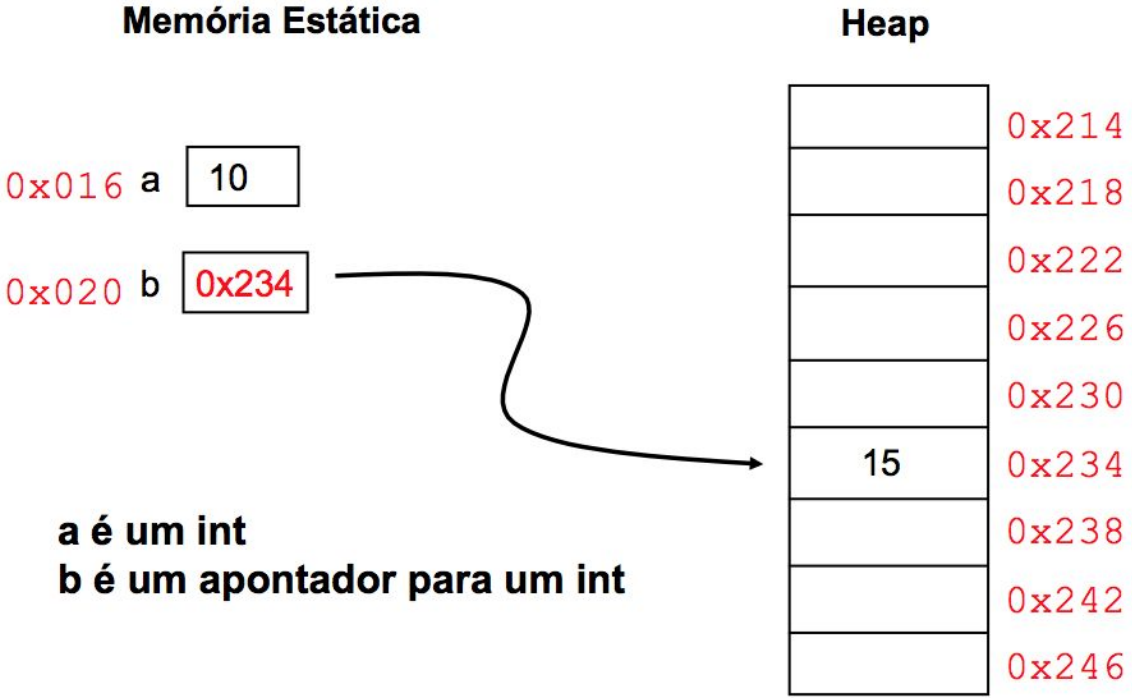
Seu programa na memória

- Pilha (Stack)
  - Chamadas de funções
  - Cada chamada um novo elemento na pilha
- Heap (Memória Alocada)
  - Suas variáveis



Esquema da memória do sistema

Alocação Dinâmica

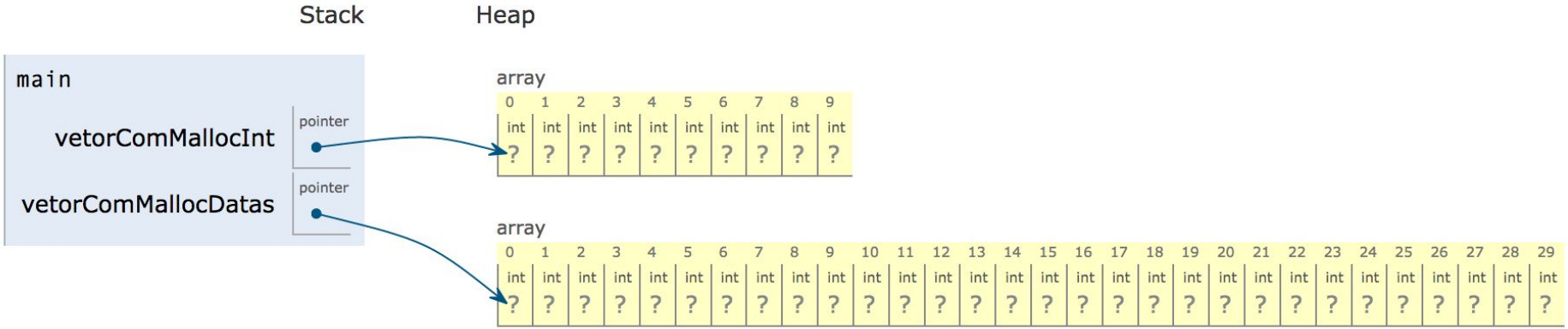


Mallocs

- Aloca um espaço contínuo de memória
- Não inicializado

```c
struct data { int dia; int mes; int ano; };
typedef struct data data_t;

int main(void) {
  int *vetorComMallocInt = malloc(10 * sizeof(int));
  int *vetorComMallocDatas = malloc(10 * sizeof(data_t));
  free(vetorComMallocInt);
  free(vetorComMallocDatas);
}
```

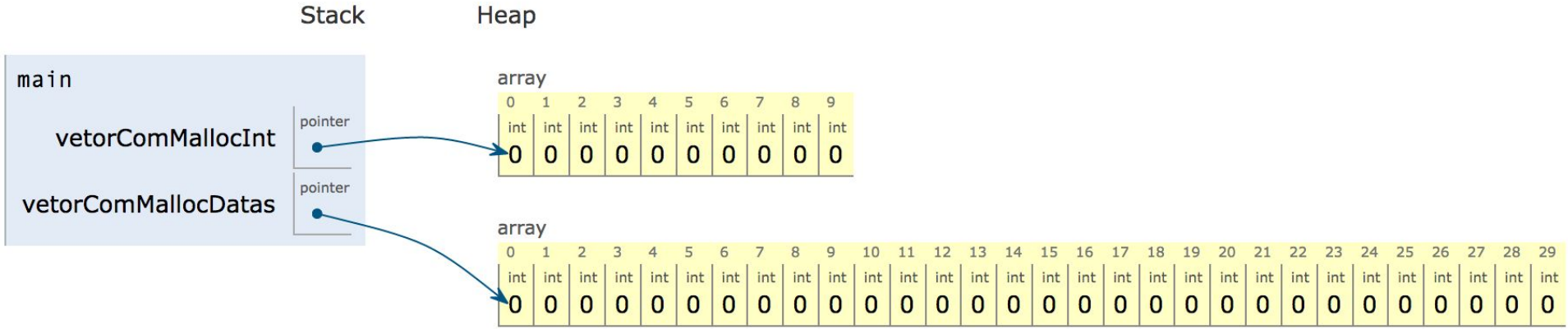


Callocs

- Aloca um espaço contínuo de memória
- **Inicializado com 0**
- Chamada um pouco diferente

```c
struct data { int dia; int mes; int ano; };
typedef struct data data_t;

int main(void) {
  int *vetorComMallocInt = calloc(10, sizeof(int));
  int *vetorComMallocDatas = calloc(10, sizeof(data_t));
  free(vetorComMallocInt);
  free(vetorComMallocDatas);
}
```



Free!

- Libere a memória do seu programa
- Free the mallocs!
- Função
  - free()
- Memory Leaks
  - Memória que não foi liberada
  - Seu programa é mais pesado do que deveria ser

Ponteiros para Nulo

- Ao importar stdlib.h você pode usar o NULL
- NULL é nulo
  - nada vazio
- Geralmente usado para indicar que o ponteiro não tem um "destino"