

# *Linguagens de Programação 1*

**Francisco Sant'Anna**

**Sala 6020-B**

**francisco@ime.uerj.br**

**<http://github.com/fsantanna-uerj/LP1>**

# Listas Encadeadas

# Exercício 9.1

- Criar um vetor `vet` de 5 posições
- Ler 5 números e guardá-los em `vet`
- Exibir todos os números de `vet`
- Ler um outro número `I`
- Remover o valor de `vet` no índice `I`
  - Manter o vetor sem buracos
  - As posições vazias devem ser preenchidas com 0
  - Exemplo:

- | 90 | 55 | 15 | 70 | 10 |

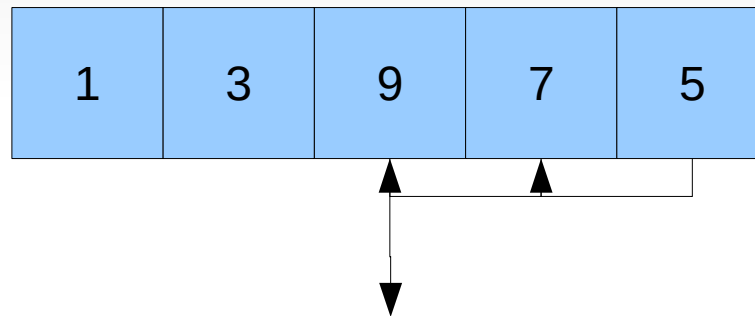
Após remover `I=2`:

- | 90 | 55 | 70 | 10 | 0 |

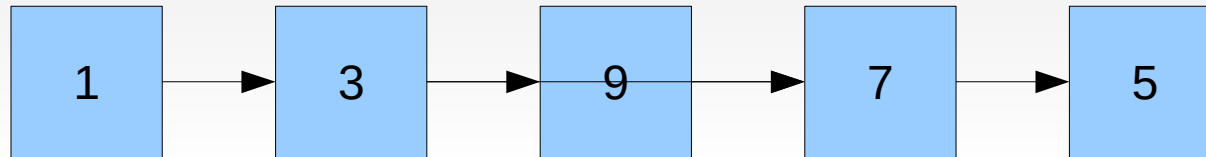
# Exercício 9.1 - Problemas?

- Precisamos re-ajustar as posições o tempo todo.
- E se eu precisar de mais um elemento?

# Vetor - Rígido



# Lista Encadeada - Flexível



```
struct Caixa {  
    int valor;  
    struct Caixa* prox;  
};
```

## Exercício 9.2

- [illegible]

## Exercício 9.3

- Qual é o `prox` da última caixa?
  - Tem que ser um endereço que “não existe”
  - `NULL == 0`
- Crie uma função que receba um ponteiro para uma caixa e percorra todas as caixas (até o `NULL`), exibindo todos os valores
- `void exhibe (struct Caixa* caixa);`



## Exercício 9.4

- Altere o Ex. 9.2 para remover a caixa no índice 2
- Exiba o valor das caixas (usando `exibe`)
- Insira a caixa removida, no início da lista
- Exiba o valor das caixas (usando `exibe`)
- Mantenha um novo ponteiro `cabeca` que sempre aponta para a caixa inicial
  - `struct Caixa* cabeca =`

# **Alocação Dinâmica**

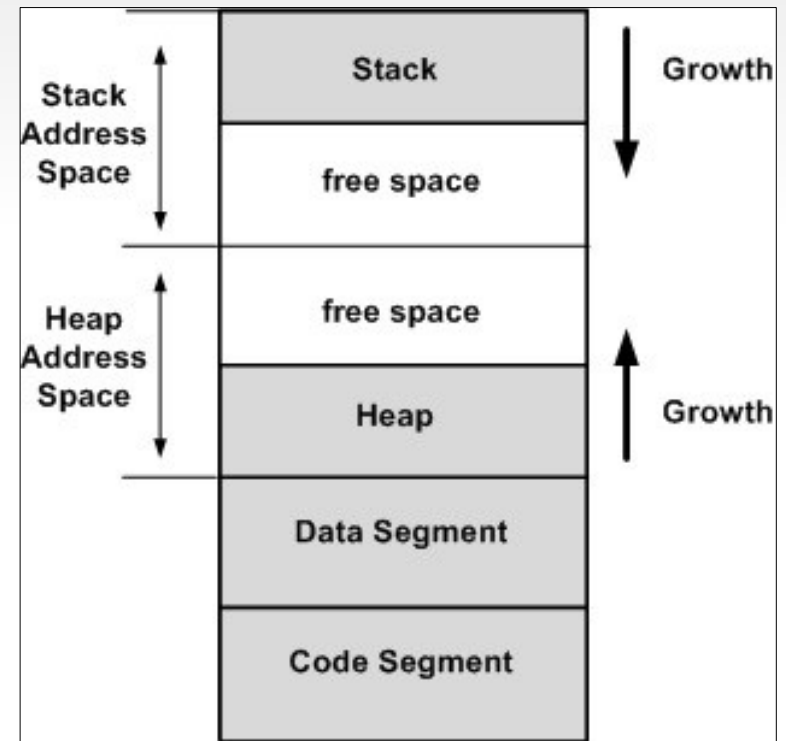
# Alocação de Memória

```
static int v[5];

void f (int i, int x) {
    if (i == 5) {
        return;
    }

    v[i] = x;
    f(i+1, x*2);
    printf("%d %d\n", i, x);
}

int main (void) {
    f(0, 10);
    return 0;
}
```



# Exercício 9.1 (revisão)

- Criar um vetor `vet` de 5 posições
- Ler 5 números e guardá-los em `vet`
- Exibir todos os números de `vet`
- Ler um outro número `I`
- Remover o valor de `vet` no índice `I`
  - Manter o vetor sem buracos

# Exemplo 9.1 - Problemas?

- Precisamos re-ajustar as posições o tempo todo.
- E se eu precisar de mais um elemento?

# Alocação Dinâmica

- Necessário quando é impossível prever o uso total de não locais antes de executar o programa.
  - `malloc`: aloca um bloco de memória na heap
  - `free`: desaloca o bloco de memória

```
typedef struct Caixa {  
    ...  
} Caixa;  
  
int main (void) {  
    ...  
    Caixa* p = (Caixa*) malloc(sizeof(Caixa));  
    ...  
    free(p);  
    ...  
}
```

# Alocação Dinâmica

- Começando de uma lista vazia (`cabeca=NULL`), ler e incluir vários valores na lista até que seja digitado `-1`.

## Exercício 9.6

- Crie uma função que receba um ponteiro para uma lista e um valor inteiro e retorne se a lista contém esse valor.
- ```
int contem (struct Caixa* caixa,  
            int valor);
```
- E para remover um elemento?



## Exercício 9.7

- Crie uma função que recebe um ponteiro para uma lista e um valor inteiro e, se a lista contém esse valor, retira-o.
- ```
int retira (struct Caixa* caixa,  
           int valor);
```