Universidade Estadual de Londrina

Técnicas de Programação

Introdução à lista

Estrutura de dados dinâmicos

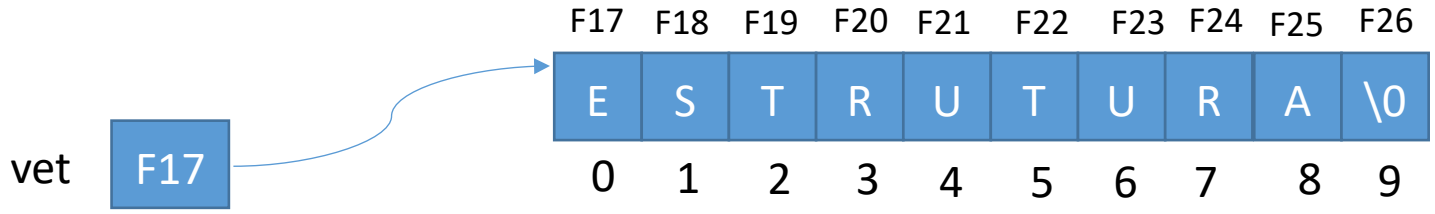
- Nesta aula
  - Revisão sobre ponteiros;
  - Diferenças entre estruturas de dados estáticas e dinâmicas;
  - Conceitos e implementação de listas encadeadas;
  - Manipulação dos elementos que compõe a lista encadeada.

Motivação

- **Estruturas de dados estáticas**
  - Ocupam um espaço contíguo de memória;
  - Sua localização na memória não se altera durante a execução;
  - Permite acesso randômico aos elementos;
  - Deve ser dimensionada com um número máximo de elementos;

- Exemplo: Vetor

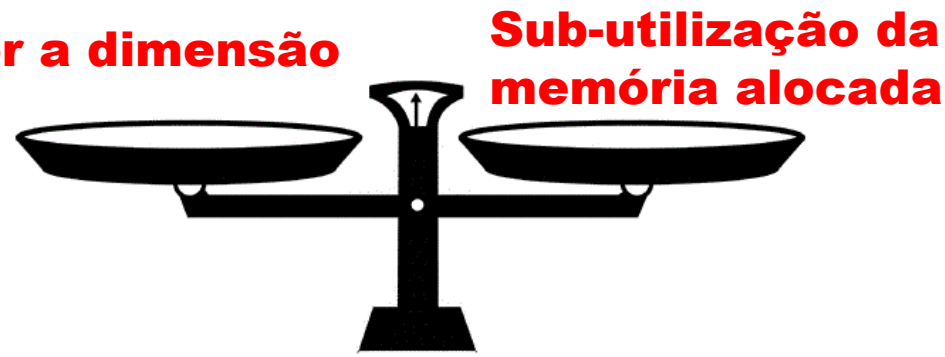
```
char vet[10];
```



O nome de um vetor nada mais é que um ponteiro para sua primeira posição

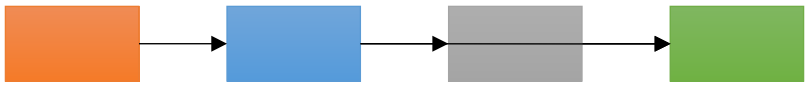
Motivação

- Estruturas estáticas não são muito flexíveis;
- Se o número de elementos que precisa ser armazenado exceder a dimensão da estrutura
  - Não há medida simples para alterar o tamanho da estrutura em tempo de execução. Vide a função *realloc*;
- Se o número de elementos for muito inferior à dimensão da estrutura, ocorrerá o desperdício de recursos



Motivação

- Solução é o uso de estruturas de dados dinâmicas
  - Armazenam cada um dos seus elementos usando alocação dinâmica.
- Estruturas de dados dinâmicas:
  - Crescem (ou decrescem) à medida que elementos são inseridos (ou removidos)
    - Listas encadeadas;
    - Filas;
    - Pilhas;
    - Árvores;



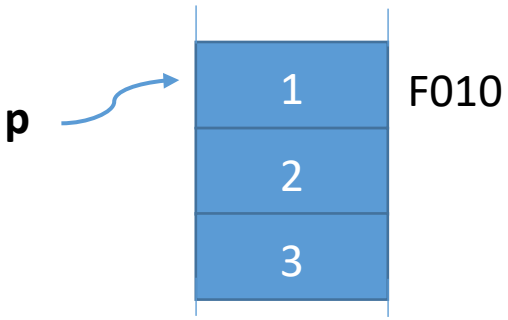
Ponteiros em C/C++

- Variáveis que **armazenam** o **endereço** de outras variáveis;

```
<tipo> * <variável>;

int *p;
```

- ***p*** é um ponteiro para **int**, isto é, uma variável que armazena o endereço de uma variável do tipo **int;**
- Supondo que ***p*** armazene o valor F010;



- Usamos o ponteiro **sem** *
  - Acessar o **endereço** que ele aponta na memória;

- Usamos o ponteiro **com** *
  - Acessar o **valor** do dado armazenado na posição de memória;

Lista Encadeada

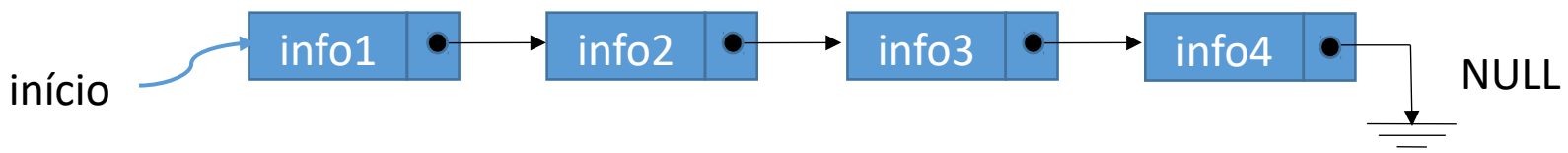
- Estrutura de representação de informação em que os elementos são mantidos de forma linear, ou seja, dispostos um após o outro
  - Exemplos: *listas de nomes, de valores, de pessoas, etc*.

- As listas encadeadas encontram-se entre as estruturas de dados dinâmicas mais simples e mais comuns;
- Constituem a base de implementação de estruturas de dados abstratos importantes
  - *Fila, pilha e árvores*.

Lista Encadeada

- Sequência encadeada de elementos “Nós”;
- O encadeamento permite percorrer a lista;
- Os nós apresentam dois tipos de dados
  - Informação armazenada;
  - Ponteiro para o próximo nó;

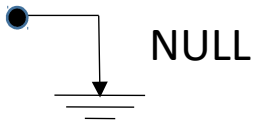
```
typedef struct elemento{
    int info;
    struct elemento *prox;
}No;
```



Lista Encadeada - Inicialização

- Como iniciar uma lista?

```
No *criar()
{
    return NULL;
}
```



Cria uma lista vazia, representada pelo ponteiro NULL. Na chamada da função...

```
int main(){
   No *inicio;

   inicio = criar(); // inicio recebe o ponteiro para No

   return 0;
}
```

Lista Encadeada

Operação Básica em Listas:

Busca

- Por um elemento através de uma “chave”
  - Ex. busca na lista telefônica: dado um nome, buscamos o telefone
- Inserção e Remoção de elementos
  - Implementação depende da organização da lista
    - manipulação distinta de listas ordenadas e não ordenadas

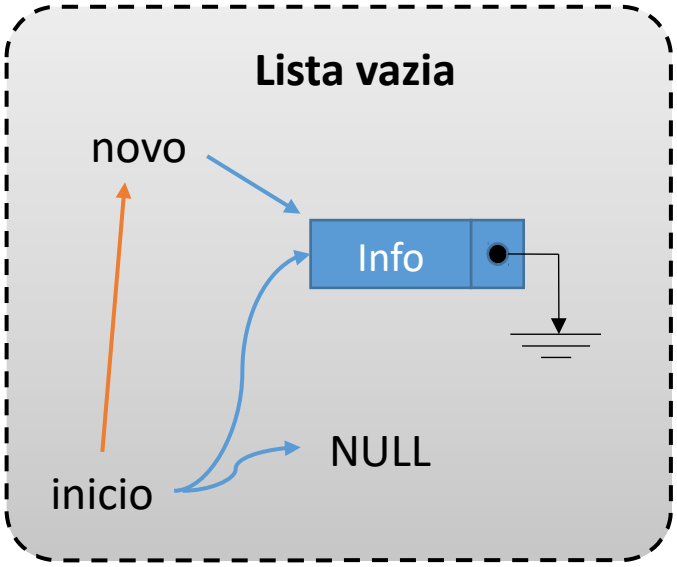
Lista Encadeada - Inserção

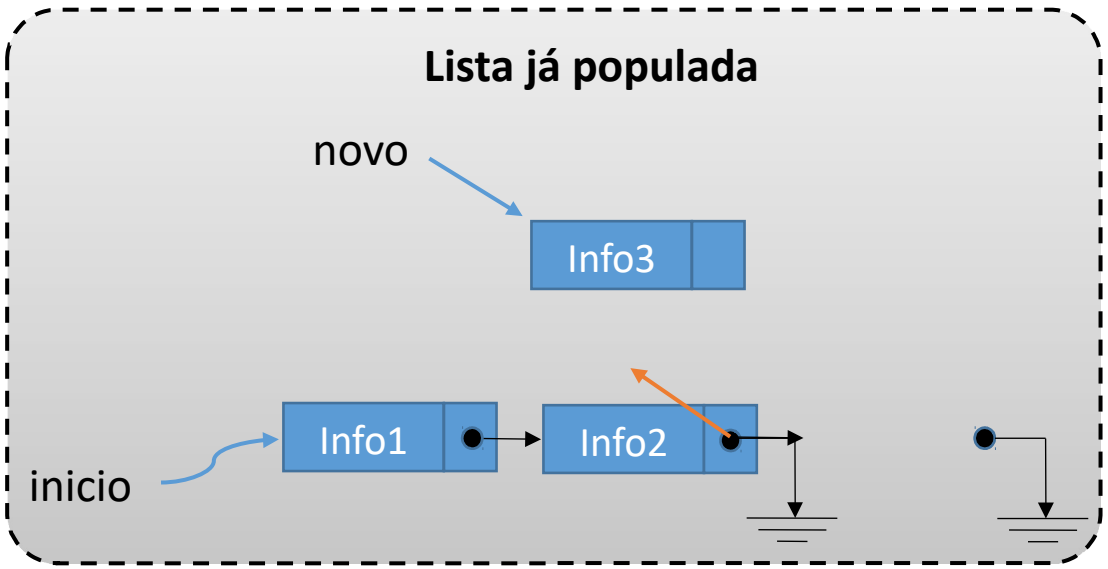
1. Aloca memória para armazenar o elemento

```
No *novo;
novo = (No *) malloc(sizeof(No));
```

- Aloca memória para um novo elemento nó
- Atribui um ponteiro a esse nó

2. Encadeia o elemento na lista existente





Lista Encadeada - Inserção

```
1  No *inserir(No *inicio, int num){
2      No *aux, *aux2;
3
4      if(inicio == NULL){
5            inicio = (No *) malloc(sizeof(No));
6            inicio->info = num;
7            inicio->prox = NULL;
8      }
9      else{
10           aux = inicio;
11           while(aux->prox != NULL)
12              aux = aux->prox;
13
14           aux2 = (No *) malloc(sizeof(No));
15           aux2->info = num;
16           aux2->prox = NULL;
17           aux->prox = aux2;
18     }
19     return inicio;
20 }
```

Lista vazia

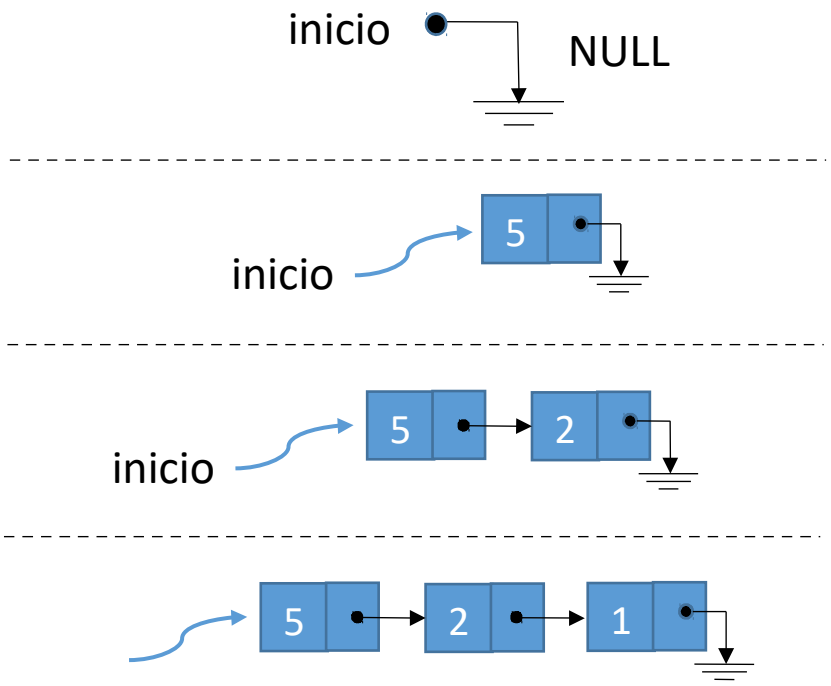
Lista já populada

Lista Encadeada - Inserção

- Exemplo 1:

```
int main()
{
    No *inicio;

    inicio = iniciar();
    inicio = inserir(inicio, 5);
    inicio = inserir(inicio, 2);
    inicio = inserir(inicio, 1);
}
```

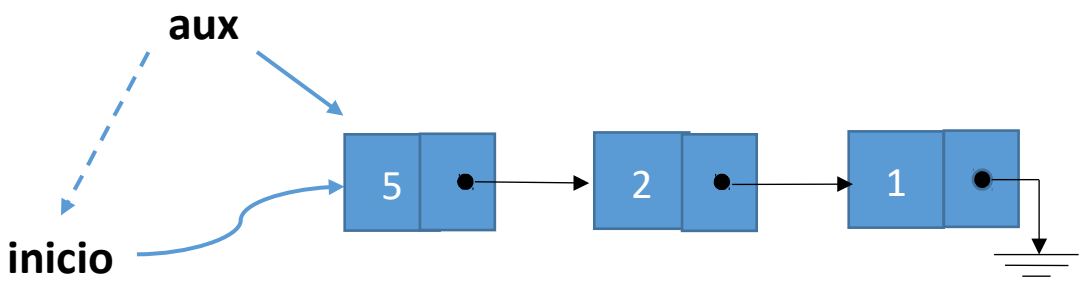


Deve-se atualizar a variável que representa o início lista a cada inserção de um novo elemento.

Lista Encadeada - Impressão

```
void imprimir(No *inicio)
{
        No *aux;
        aux = inicio;
        while(aux != NULL){
            printf(" %d ", aux->info);
            aux = aux->prox;
         }
}
```

Faz o ponteiro *aux* percorrer a lista



Lista Encadeada - Busca

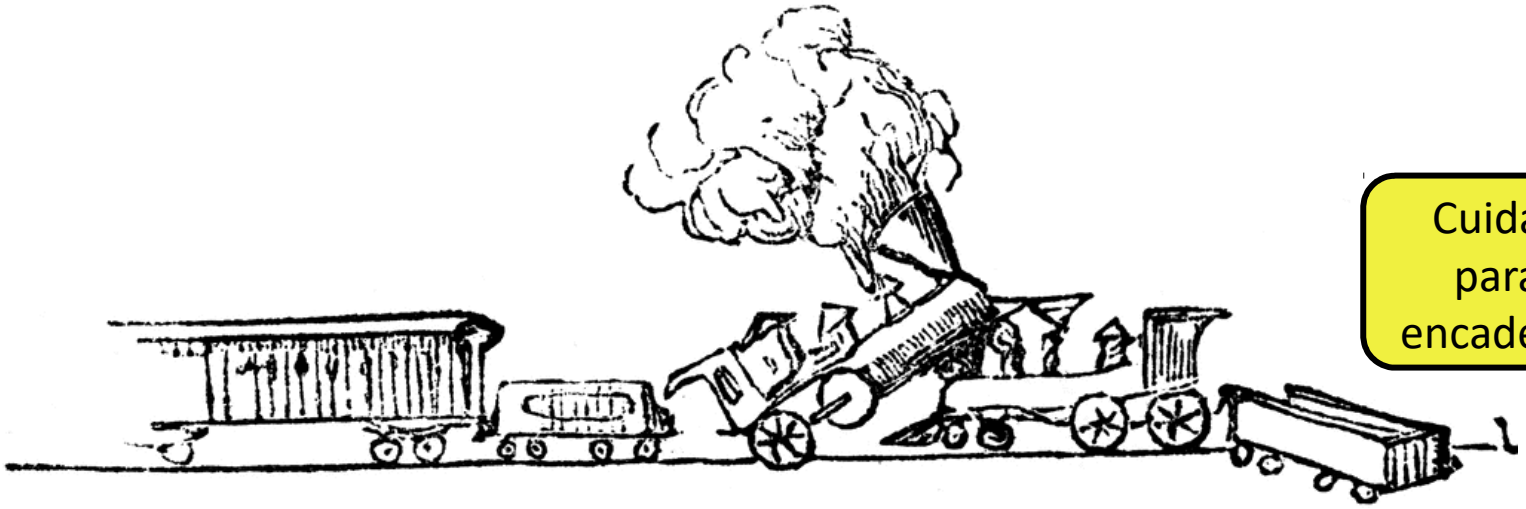
1. Recebe a informação referente ao elemento a pesquisar
2. Retorna o ponteiro do nó que representa o elemento, ou NULL, caso o elemento não seja encontrado na lista

```
No *buscar(No *inicio, int num)
{
    No *aux;
    aux = inicio;

    while(aux != NULL ){
         if(aux->info == num){
            printf("elemento encontrado");
            return aux;
         }
         aux = aux->prox;
    }
   return NULL;
}
```

Lista Encadeada - Exclusão

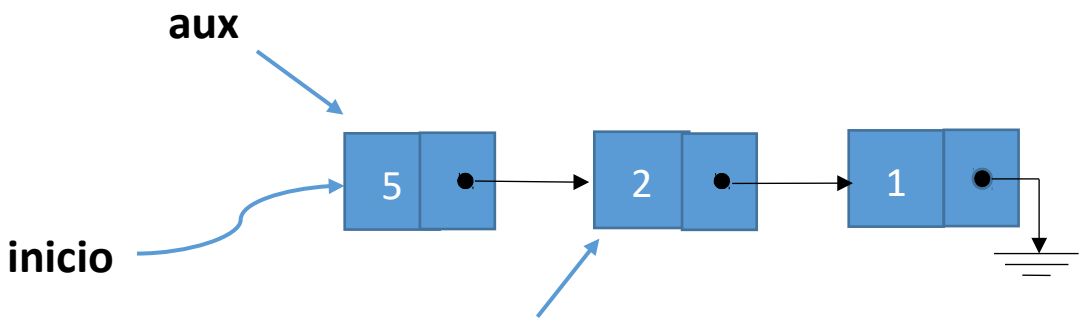
1. Recebe como entrada a lista e o valor do elemento a retirar;
2. Encontra o elemento a ser excluído;
3. Atualiza o valor da lista, se o elemento removido for o primeiro;
   - Caso contrário, apenas remove o elemento da lista



Lista Encadeada - Exclusão

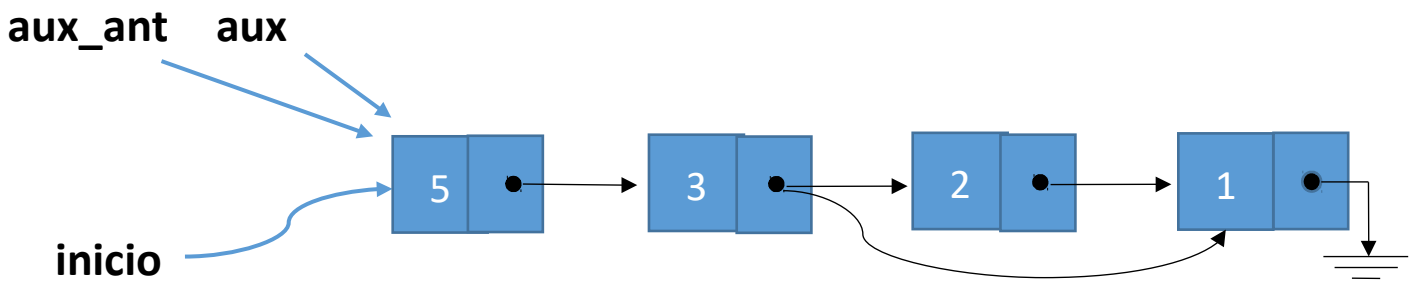
- Excluir nó inicial

```
free(aux);
```



Lista Encadeada - Exclusão

- Excluir um nó qualquer (não inicial)



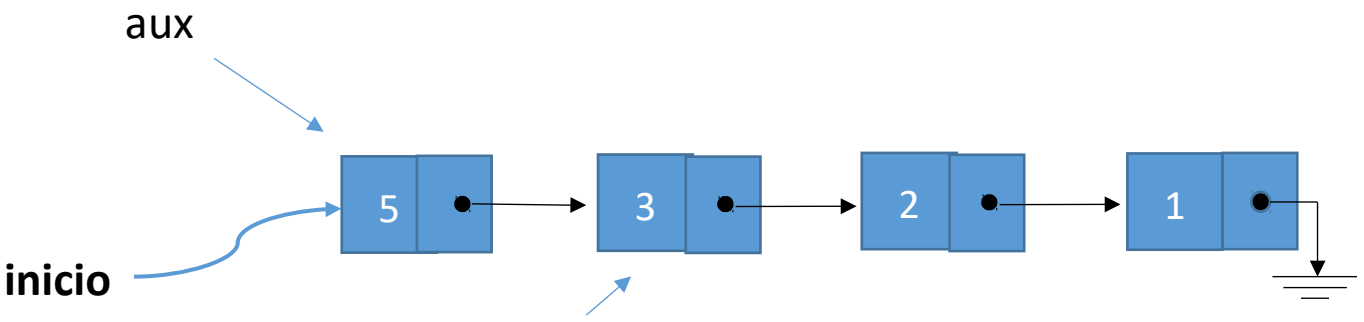
Lista Encadeada - Exclusão

```
1  No *excluir(No *inicio, int num){
2      No *aux, *aux_ant;
3
4      aux = inicio;
5      aux_ant = NULL;
6
7      while(aux != NULL && aux->info != num){ //procurar nó a ser excluído
8          aux_ant = aux;
9          aux = aux->prox;
10     }
11     if(aux == NULL)
12       return inicio; //nao encontrou o elemento. Início permanece o mesmo
13
14     if(aux_ant == NULL){  //Excluir primeiro No
15       aux = aux->prox;
16       return aux;
17     }
18     else{ //Excluir No que não seja o início
19          aux_ant->prox = aux->prox;
20     }
21     free(aux);
22     return inicio; //se chegou aqui nao alterou o No inicio
23 }
```

Desaloca o No apontado por aux

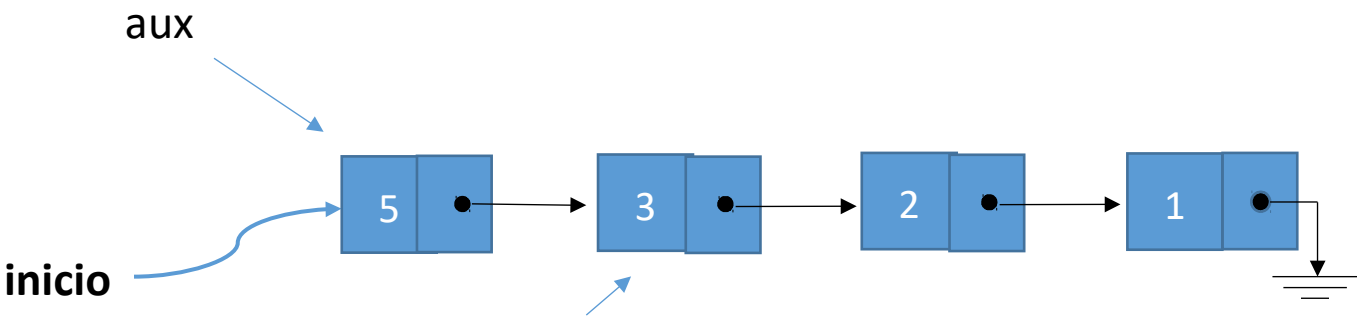
Lista Encadeada: Liberar memória alocada

```
1  void desalocar(No *inicio){
2       No* aux;
3       aux = inicio;
4       while (aux != NULL){
5            inicio = inicio->prox;
6            free(aux);
7            aux = inicio;
8       }
9  }
```



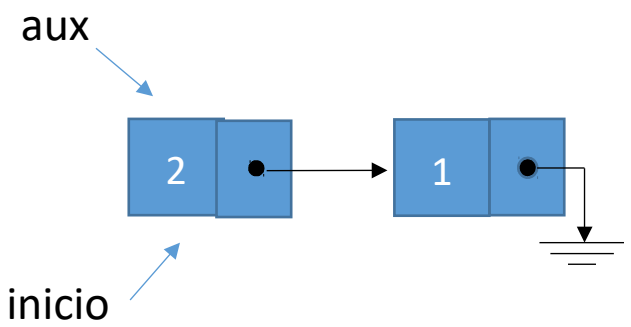
Lista Encadeada: Liberar memória alocada

```
1  void desalocar(No *inicio){
2       No* aux;
3       aux = inicio;
4       while (aux != NULL){
5            inicio = inicio->prox;
6            free(aux);
7            aux = inicio;
8       }
9  }
```



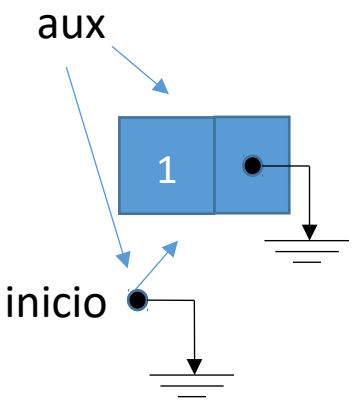
Lista Encadeada: Liberar memória alocada

```
1  void desalocar(No *inicio){
2       No* aux;
3       aux = inicio;
4       while (aux != NULL){
5            inicio = inicio->prox;
6            free(aux);
7            aux = inicio;
8       }
9  }
```



Lista Encadeada: Liberar memória alocada

```
1  void desalocar(No *inicio){
2       No* aux;
3       aux = inicio;
4       while (aux != NULL){
5            inicio = inicio->prox;
6            free(aux);
7            aux = inicio;
8       }
9  }
```

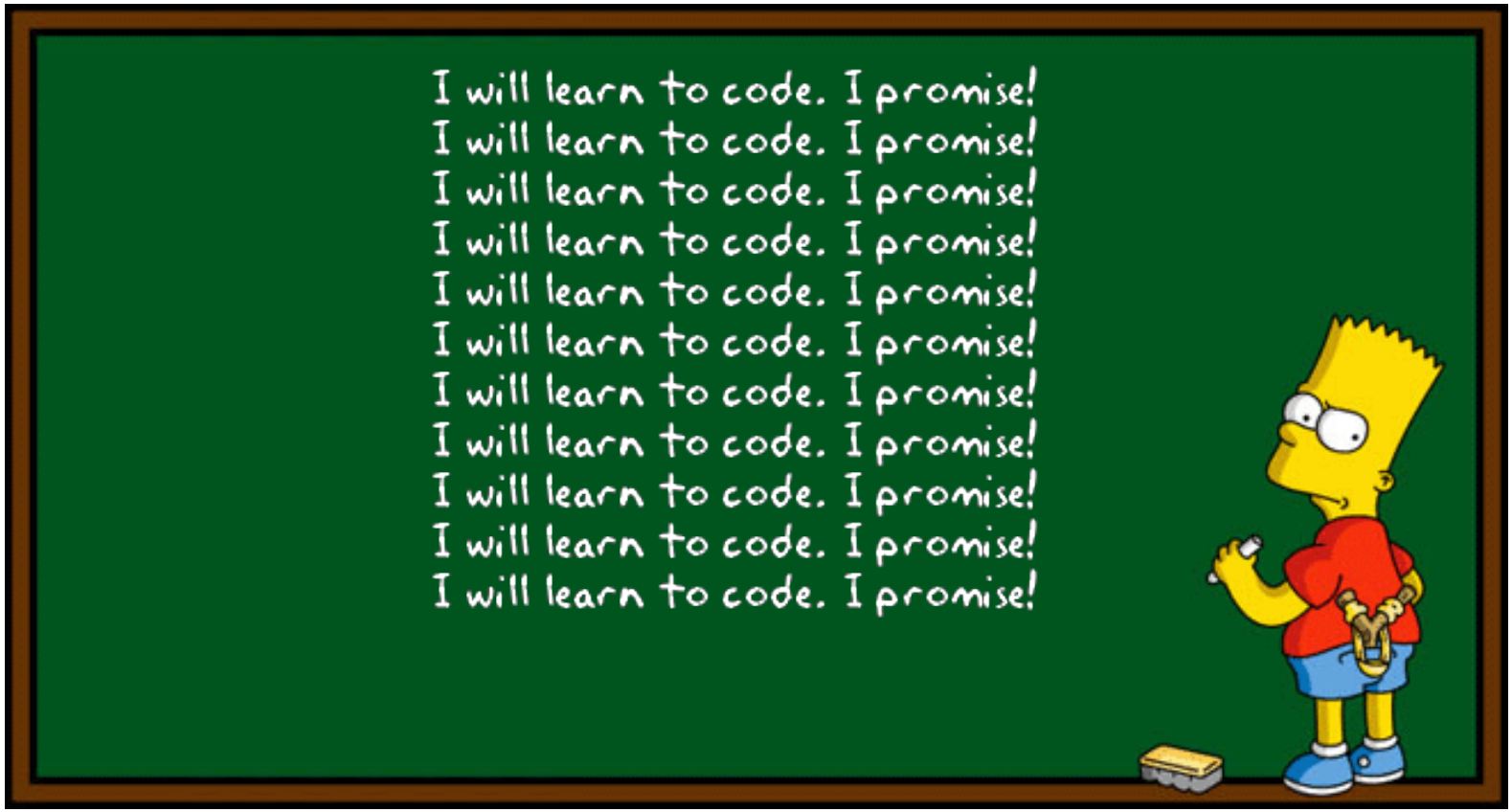


Revisão

- Comparação entre estruturas estática e dinâmicas

	Estática	Dinâmica
Tamanho	Fixo	Flexível ✓
Acesso aos elementos	Rápido ✓	Lento
Custo de inserção/remoção	Elevado	Baixo ✓

Exercícios


I will learn to code. I promise!
I will learn to code. I promise!
I will learn to code. I promise!
I will learn to code. I promise!
I will learn to code. I promise!
I will learn to code. I promise!
I will learn to code. I promise!
I will learn to code. I promise!
I will learn to code. I promise!
I will learn to code. I promise!
I will learn to code. I promise!

Exercício 1

- Construa uma lista encadeada, na qual cada elemento deverá conter as seguintes informações de alunos:
  - Matrícula;
  - Nome;
  - Curso;
  - Idade;
- Construa funções de manipulação:
  - Inserir
  - Imprimir
  - Excluir
  - Buscar;

Exercícios 2 e 3

- Exercício 2
  - Implemente uma lista de inteiros que não permita a inserção de números repetidos;

- Exercício 3
  - Usando a lista anterior, crie uma função que receba dois valores inteiros, *m* e *n.* A função deve trocar a ordem na lista dos elementos de índice *m* e *n.*

  Obs.: Considere o inicio da lista como índice 0.

Próxima Aula

- Lista duplamente encadeada
- Lista circular
- Lista Ordenada