

DCC004 - Algoritmos e Estruturas de Dados II

Tipos Abstratos de Dados

Renato Martins

Email: renato.martins@dcc.ufmg.br

<https://www.dcc.ufmg.br/~renato.martins/courses/DCC004>

Material adaptado de PDS2 - Douglas Macharet e Flávio Figueiredo

Algoritmos e Estruturas de Dados

- Algoritmo:
 - Sequência de passos/ações
 - Trabalham em cima das estruturas de dados
- Estruturas de dados:
 - Abstração de uma situação real
 - “Dão suporte” aos algoritmos

Estruturas de Dados

- Dados podem ser representados de diversas formas
- A forma que representamos os dados é guiado pela operações que vamos fazer em cima deles
- Quais operações vamos realizar?
- Os dados para suportam as mesmas?

Como representar o coleções de objetos?

TADs e Operações

- Quais operações uma coleção suporta?
- Pense em um conjunto matemático

TADs e Operações

- Quais operações uma coleção suporta?
- Pense em um conjunto matemático
 - adicionar (união) elemento
 - remover (complemento) elemento
 - interseção
 - número de elementos
 - domínio
 - . . .

Tipos Abstratos de Dados (TADs)

- Modelo matemático, acompanhado das operações definidas sobre o modelo.
 - conjunto dos inteiros acompanhado das operações de adição, subtração e multiplicação.
- A implementação do algoritmo em uma linguagem de programação exige a representação do TAD em termos dos tipos de dados e dos operadores suportados.

Contrato

- TADs são contratos
- Funções que operam em cima da memória

Encapsulamento

- Conceito importante em TADs
- Usuário:
 - Enxerga a interface
 - Não se preocupa, em primeiro momento, como é o TAD por baixo

Então TADs são classes?

TADs vs Classes

Não!

- TADs são um conceito mais geral
- Existem em qualquer tipo de linguagem
- Em C++
 - Sim, mapeiam bem para classes
 - Ou para interfaces
 - Assuntos futuro
 - Vamos fazer uma abordagem bottom up

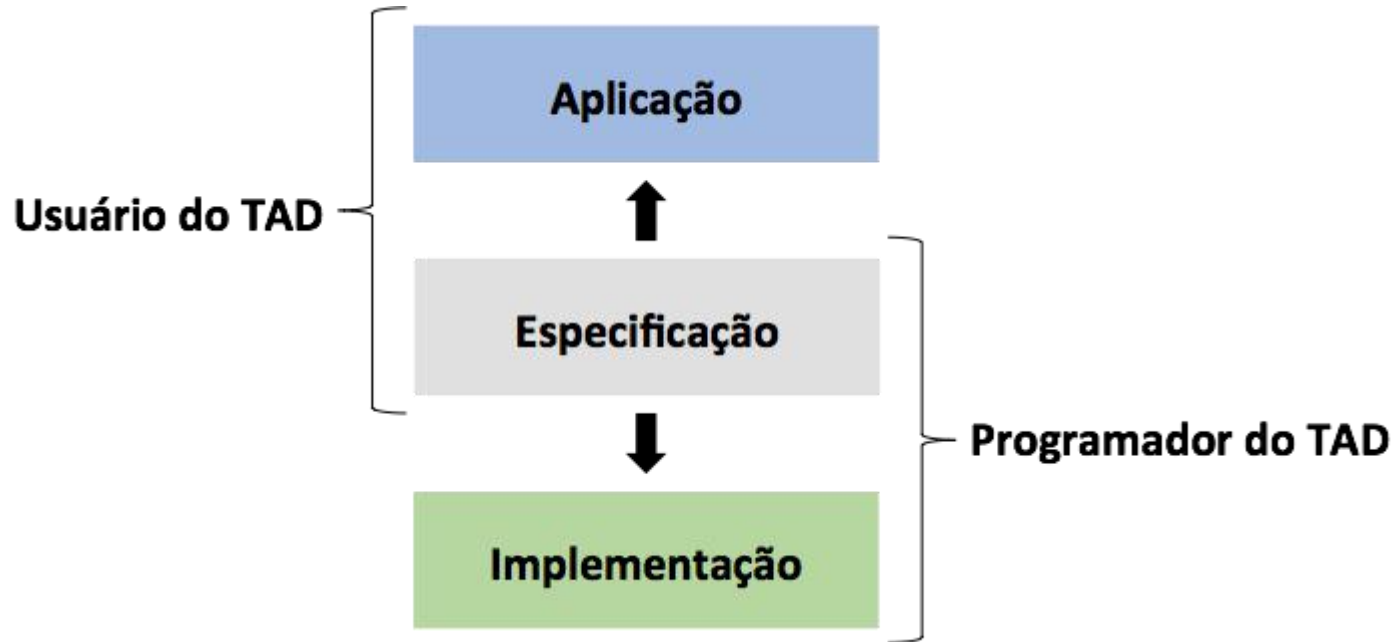
TADs vs Algoritmos

- Algoritmo
 - Sequência de ações executáveis
entrada \rightarrow saída
 - Exemplo: “Receita de Bolo”
 - Algoritmos usam TADs
- TADs
 - Contrato
+
 - Memória

Tipos Abstratos de Dados (TADs)

- Podemos considerar TADs como generalizações de tipos primitivos e procedimentos como generalizações de operações primitivas.
- O TAD encapsula tipos de dados. A definição do tipo e todas as operações ficam localizadas numa seção do programa.
- Os usuários do TAD só tem acesso a algumas operações disponibilizadas sobre esses dados

Tipos Abstratos de Dados (TADs)



Tipos Abstratos de Dados (TADs)

- TADs são um conceito de programação
- Vamos aprender como implementar os mesmos usando classes e objetos
- Outras linguagens
 - structs + funções (C)
 - traits (Rust)
 - duck typing (Python, Ruby)
 - **classes e interfaces (Java, C++)**

Listas

Listas Lineares

- Sequência de zero ou mais itens
 - $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$, na qual \mathbf{x}_i
- Posições relativas
 - Assumindo $n \geq 1$, \mathbf{x}_1 é o primeiro item da lista e \mathbf{x}_n é o último item da lista.
 - \mathbf{x}_i precede \mathbf{x}_{i+1} para $i = 1, 2, \dots, n - 1$
 - \mathbf{x}_i sucede \mathbf{x}_{i-1} para $i = 2, 3, \dots, n$
 - \mathbf{x}_i é dito estar na i -ésima posição da lista

Tipos Abstratos de Dados (TADs)

Lista de números inteiros

- Considere uma uma lista de inteiros.
Poderíamos definir TAD Lista, com as seguintes operações:
 - faça a lista vazia;
 - obtenha o primeiro elemento da lista; se a lista estiver vazia, então retorna nulo;
 - insira um elemento na lista.

Tipos Abstratos de Dados (TADs)

Lista de números inteiros

- Quais outras operações podem ser definidas?
 - Retirar o i -ésimo item.
 - Localizar o i -ésimo item
 - Fazer uma cópia da lista linear.
 - Ordenar os itens da lista em ordem ascendente ou descendente
 - Pesquisar a ocorrência de um item com um valor particular em algum componente.

Solução Zero

```
#ifndef PDS2_LISTA_VETOR_H
#define PDS2_LISTA_VETOR_H
#define TAMANHO 10
```

← Constante no .h. Em C++ também existe o const

```
class ListaVetorInteiros {
private:
    int *_elementos;
    int _num_elementos_inseridos;
public:
    // Construtor
    ListaVetorInteiros();
    // Destrutor
    ~ListaVetorInteiros();
    // Insere um inteiro na lista
    void inserir_elemento(int elemento);
    // Imprime a lista
    void imprimir();
};
#endif
```

← Vetor de elementos que será alocado dinamicamente (heap)

Como Implementar?

Construtor (e início do .h)

```
#include <iostream>
```

Em algum momento vamos imprimir a lista

```
#include "listavetor.h"
```

```
ListaVetorInteiros::ListaVetorInteiros() {  
    this->_elementos = new int[TAMANHO]();  
    this->_num_elementos_inseridos = 0;  
}
```

Construtor (e início do .h)

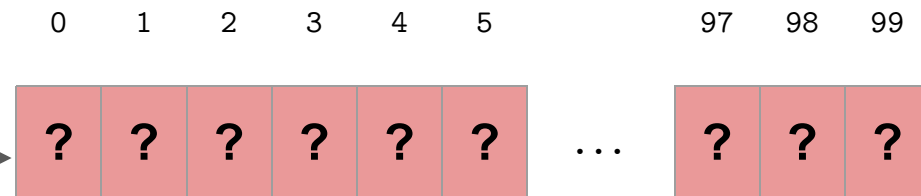
```
ListaVetorInteiros::ListaVetorInteiros() {  
    this->_elementos = new int[TAMANHO]();  
    this->_num_elementos_inseridos = 0;  
}
```

```
Objeto ListaVetorInteiros {  
    int *_elementos;  
    int _num_elementos_inseridos;  
}
```

Construtor (e início do .h)

```
ListaVetorInteiros::ListaVetorInteiros() {  
    this->_elementos = new int[TAMANHO]();  
    this->_num_elementos_inseridos = 0;  
}
```

Objeto ListaVetorInteiros {
 int *_elementos;
 int _num_elementos_inseridos;
}

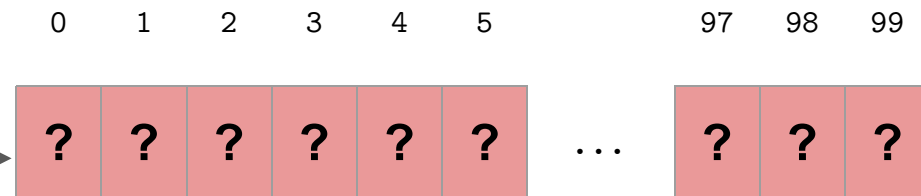


Construtor (e início do .h)

```
ListaVetorInteiros::ListaVetorInteiros() {  
    this->_elementos = new int[TAMANHO]();  
    this->_num_elementos_inseridos = 0;  
}
```



```
Objeto ListaVetorInteiros {  
    int *_elementos;  
    int _num_elementos_inseridos = 0;  
}
```

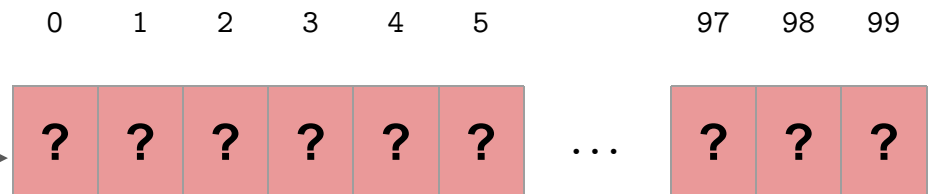


Adicionando elementos?

Adicionando elementos?

```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
if (this->_num_elementos_inseridos == TAMANHO) {  
    std::cerr << "Erro, lista cheia" << std::endl;  
    exit(1);  
}  
this->_elementos[this->_num_elementos_inseridos] = elemento;  
this->_num_elementos_inseridos++;  
}
```

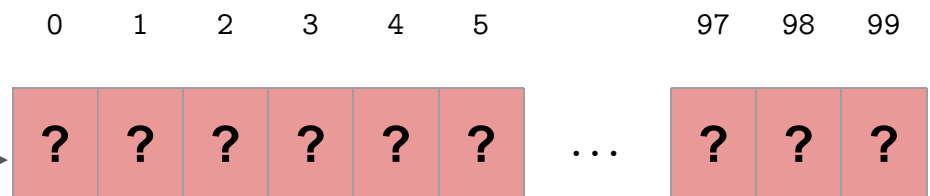
```
Objeto ListaVetorInteiros {  
    int *_elementos;  
    int _num_elementos_inseridos = 0;  
}
```



Adicionando elementos?

```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
    if (this->_num_elementos_inseridos == TAMANHO) {  
        std::cerr << "Erro, lista cheia" << std::endl;  
        exit(1);  
    }  
    this->_elementos[this->_num_elementos_inseridos] = elemento;  
    this->_num_elementos_inseridos++;  
}
```

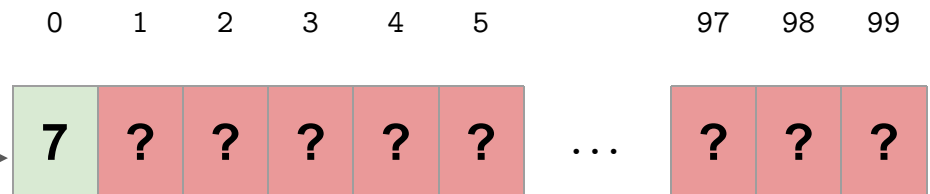
Objeto ListaVetorInteiros {
 int *_elementos;
 int _num_elementos_inseridos = 0;
}



Adicionando elementos?

```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
    if (this->_num_elementos_inseridos == TAMANHO) {  
        std::cerr << "Erro, lista cheia" << std::endl;  
        exit(1);  
    }  
    this->_elementos[this->_num_elementos_inseridos] = elemento;  
    this->_num_elementos_inseridos++;  
}
```

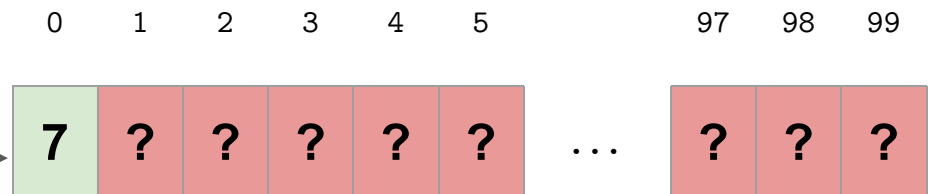
Objeto ListaVetorInteiros {
 int *_elementos;
 int _num_elementos_inseridos = 0;
}



Adicionando elementos?

```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
    if (this->_num_elementos_inseridos == TAMANHO) {  
        std::cerr << "Erro, lista cheia" << std::endl;  
        exit(1);  
    }  
    this->_elementos[this->_num_elementos_inseridos] = elemento;  
    this->_num_elementos_inseridos++;  
}
```

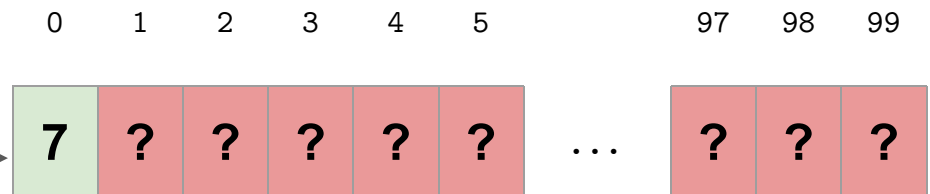
Objeto ListaVetorInteiros {
 int *_elementos;
 int _num_elementos_inseridos = 1;
}



Adicionando elementos?

```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
if (this->_num_elementos_inseridos == TAMANHO) {  
    std::cerr << "Erro, lista cheia" << std::endl;  
    exit(1);  
}  
this->_elementos[this->_num_elementos_inseridos] = elemento;  
this->_num_elementos_inseridos++;  
}
```

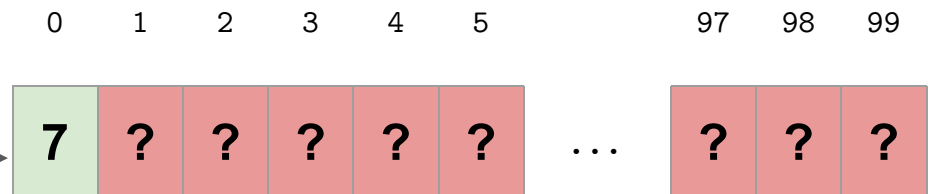
```
Objeto ListaVetorInteiros {  
    int *_elementos;  
    int _num_elementos_inseridos = 1;  
}
```



Adicionando elementos?

```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
    if (this->_num_elementos_inseridos == TAMANHO) {  
        std::cerr << "Erro, lista cheia" << std::endl;  
        exit(1);  
    }  
    this->_elementos[this->_num_elementos_inseridos] = elemento;  
    this->_num_elementos_inseridos++;  
}
```

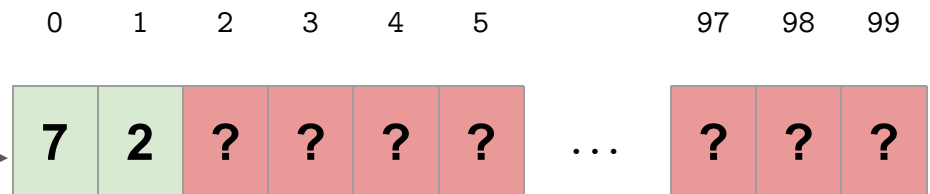
Objeto ListaVetorInteiros {
 int *_elementos;
 int _num_elementos_inseridos = 1;
}



Adicionando elementos?

```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
    if (this->_num_elementos_inseridos == TAMANHO) {  
        std::cerr << "Erro, lista cheia" << std::endl;  
        exit(1);  
    }  
    this->_elementos[this->_num_elementos_inseridos] = elemento;  
    this->_num_elementos_inseridos++;  
}
```

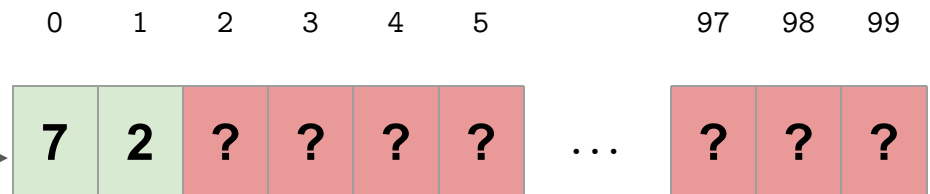
Objeto ListaVetorInteiros {
 int *_elementos;
 int _num_elementos_inseridos = 1;
}



Adicionando elementos?

```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
    if (this->_num_elementos_inseridos == TAMANHO) {  
        std::cerr << "Erro, lista cheia" << std::endl;  
        exit(1);  
    }  
    this->_elementos[this->_num_elementos_inseridos] = elemento;  
    this->_num_elementos_inseridos++;  
}
```

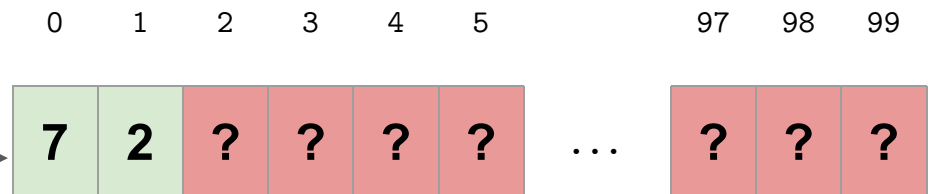
Objeto ListaVetorInteiros {
 int *_elementos;
 int _num_elementos_inseridos = 2;
}



Adicionando elementos?

```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
    if (this->_num_elementos_inseridos == TAMANHO) {  
        std::cerr << "Erro, lista cheia" << std::endl;  
        exit(1);  
    }  
    this->_elementos[this->_num_elementos_inseridos] = elemento;  
    this->_num_elementos_inseridos++;  
}
```

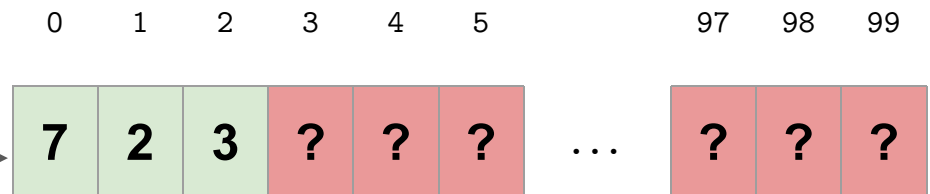
```
Objeto ListaVetorInteiros {  
    int *_elementos;  
    int _num_elementos_inseridos = 2;  
}
```



Adicionando elementos?

```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
    if (this->_num_elementos_inseridos == TAMANHO) {  
        std::cerr << "Erro, lista cheia" << std::endl;  
        exit(1);  
    }  
    this->_elementos[this->_num_elementos_inseridos] = elemento;  
    this->_num_elementos_inseridos++;  
}
```

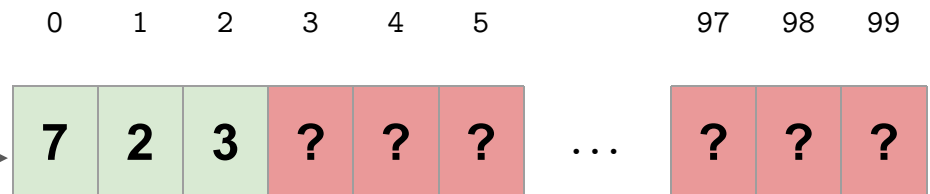
Objeto ListaVetorInteiros {
 int *_elementos;
 int _num_elementos_inseridos = 2;
}



E por aí vai...

```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
    if (this->_num_elementos_inseridos == TAMANHO) {  
        std::cerr << "Erro, lista cheia" << std::endl;  
        exit(1);  
    }  
    this->_elementos[this->_num_elementos_inseridos] = elemento;  
    this->_num_elementos_inseridos++;  
}
```

Objeto ListaVetorInteiros {
 int *_elementos;
 int _num_elementos_inseridos = 3;
}



Imprimir

Trivial

```
void ListaVetorInteiros::imprimir() {  
    for (int i = 0; i < this->_num_elementos_inseridos; i++)  
        std::cout << this->_elementos[i] << " ";  
    std::cout << std::endl;  
}
```

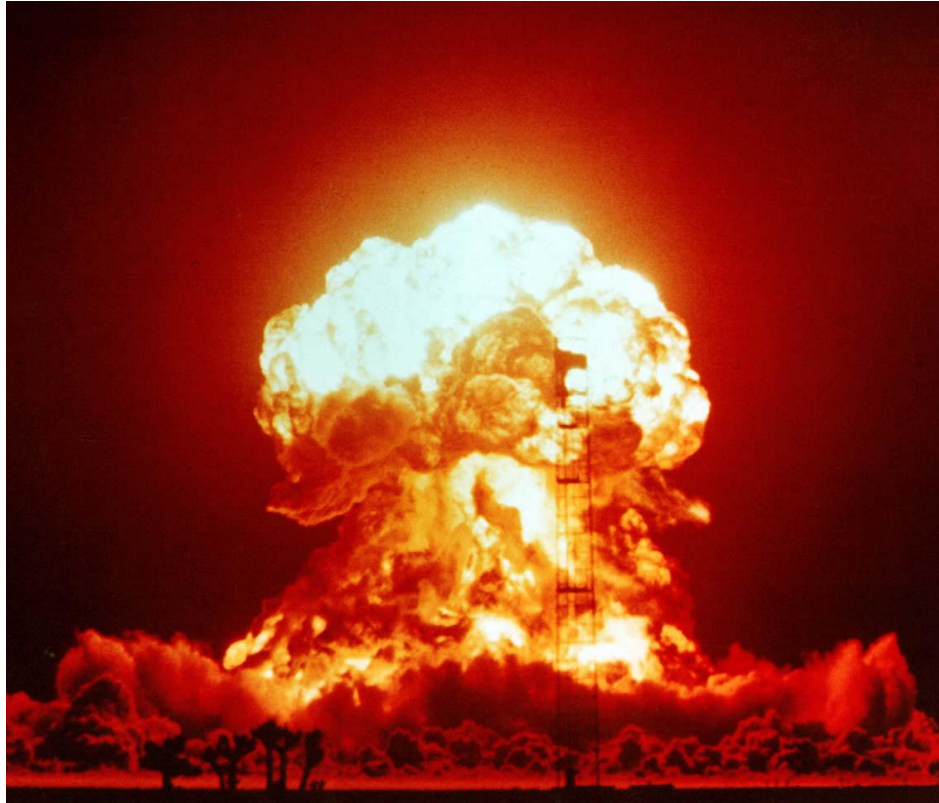
Destrutor

- Alocamos um vetor
- 1 **new**
- Precisamos de 1 **delete**
- Lembrando, cada **new** → 1 **delete**

```
ListaVetorInteiros::~~ListaVetorInteiros() {  
    delete[] this->_elementos;  
}
```

Mais de 100 elementos?

Mais de 100 elementos?



```
#ifndef PDS2_LISTA_VETOR_H
#define PDS2_LISTA_VETOR_H
```

```
#define TAMANHO_INICIAL 10
```

Tamanho inicial que será aumentado

```
class ListaVetorInteiros {
```

```
private:
```

```
    int *_elementos;
```

```
    int _num_elementos_inseridos;
```

```
    int _capacidade;
```

Tamanho atual que também será aumentado

```
public:
```

```
    // Construtor
```

```
    ListaVetorInteiros();
```

```
    // Destrutor
```

```
    ~ListaVetorInteiros();
```

```
    // Insere um inteiro na lista
```

```
    void inserir_elemento(int elemento);
```

```
    // Imprime a lista
```

```
    void imprimir();
```

```
};
```

```
#endif
```

Construtor

Construtor



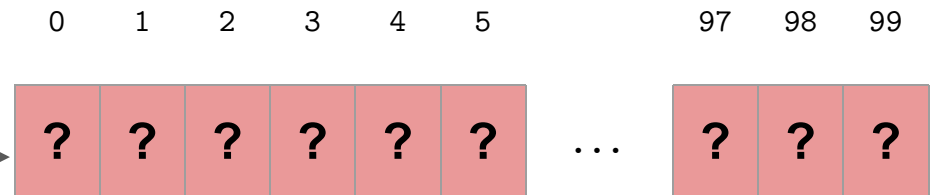
```
ListaVetorInteiros::ListaVetorInteiros() {  
    this->_elementos = new int[TAMANHO_INICIAL]();  
    this->_num_elementos_inseridos = 0;  
    this->_capacidade = TAMANHO_INICIAL;  
}
```

Construtor

```
ListaVetorInteiros::ListaVetorInteiros() {  
    this->_elementos = new int[TAMANHO_INICIAL]();  
    this->_num_elementos_inseridos = 0;  
    this->_capacidade = TAMANHO_INICIAL;  
}
```



```
Objeto ListaVetorInteiros {  
    int *_elementos;  
    int _num_elementos_inseridos = 0;  
    int _capacidade = 100;  
}
```



Métodos que não mudam

- Imprime
- Destruitor

Complicação

- Inserir elemento

Ideia

- Inserir elemento
- Caso o vetor fique cheio
 - Duplicar o mesmo
 - Copiar tudo para o novo
 - Aumentar a capacidade
- Estamos implementando o **vector**
 - Nome do container na **STL**

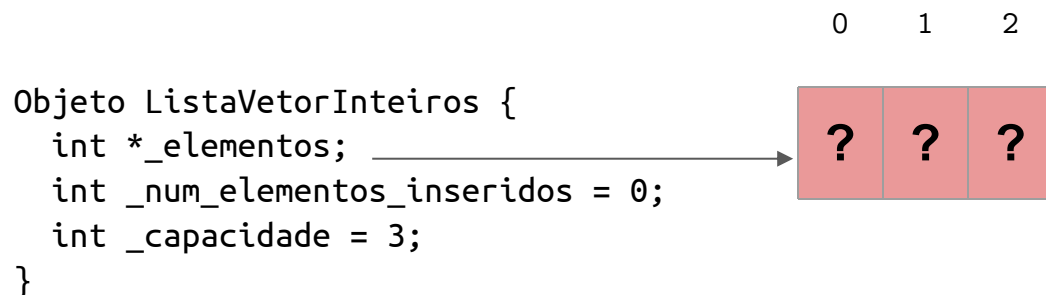
Inserir elemento

```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
    // ...  
    this->_elementos[this->_num_elementos_inseridos] = elemento;  
    this->_num_elementos_inseridos++;  
}
```

Passo a Passo

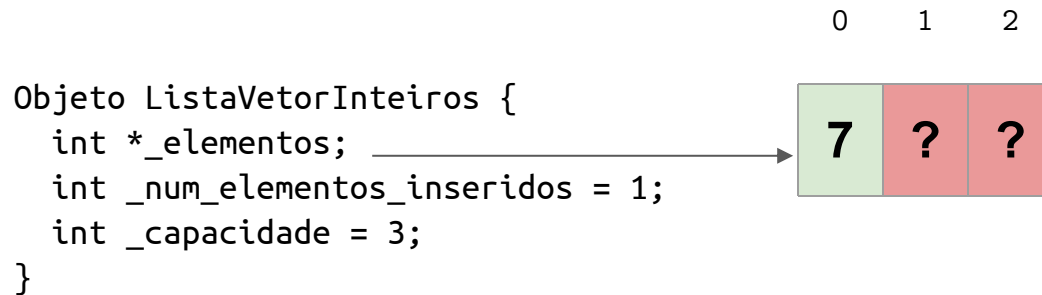
- Alocamos tamanho inicial.

Vamos supor que seja igual a 3



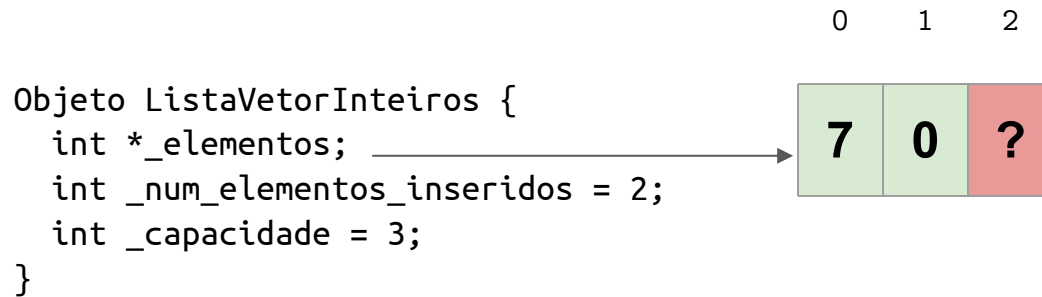
Passo a Passo

- Inserindo um elemento



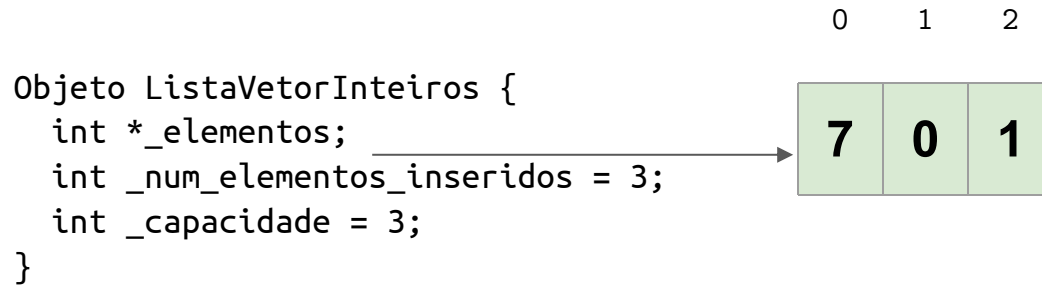
Passo a Passo

- Outro



Passo a Passo

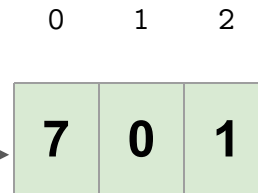
+1



Passo a Passo

+outro?!

```
Objeto ListaVetorInteiros {  
  int *_elementos;  
  int _num_elementos_inseridos = 3;  
  int _capacidade = 3;  
}
```



Passo a Passo

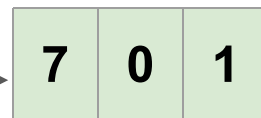
+alocamos
espaço

int *new_data;



0 1 2

```
Objeto ListaVetorInteiros {  
  int *_elementos;  
  int _num_elementos_inseridos = 0;  
  int _capacidade = 3;  
}
```

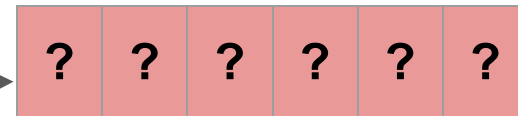


```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
  if (this->_num_elementos_inseridos == this->_capacidade) {  
    int *new_data = new int[this->_capacidade * 2];  
  
    for (int i = 0; i < this->_num_elementos_inseridos; i++)  
      new_data[i] = this->_elementos[i];  
  
    delete[] this->_elementos;  
    this->_elementos = new_data;  
    this->_capacidade = this->_capacidade * 2;  
  }  
  // ...  
}
```

Passo a Passo

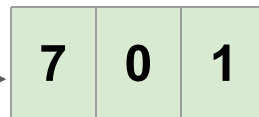
+ copiamos os dados

int *new_data;



0 1 2

```
Objeto ListaVetorInteiros {  
  int *_elementos;  
  int _num_elementos_inseridos = 3;  
  int _capacidade = 3;  
}
```

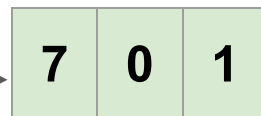
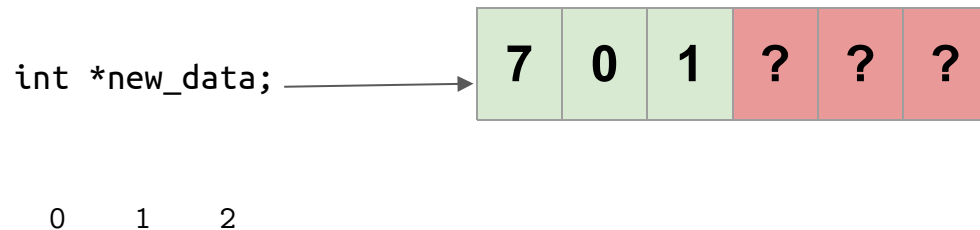


```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
  if (this->_num_elementos_inseridos == this->_capacidade) {  
    int *new_data = new int[this->_capacidade * 2];  
  
    for (int i = 0; i < this->_num_elementos_inseridos; i++)  
      new_data[i] = this->_elementos[i];  
  
    delete[] this->_elementos;  
    this->_elementos = new_data;  
    this->_capacidade = this->_capacidade * 2;  
  }  
  // ...  
}
```


Passo a Passo

+ copiamos os dados

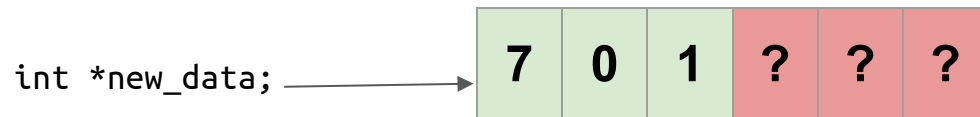
```
Objeto ListaVetorInteiros {  
  int *_elementos;  
  int _num_elementos_inseridos = 3;  
  int _capacidade = 3;  
}
```



```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
  if (this->_num_elementos_inseridos == this->_capacidade) {  
    int *new_data = new int[this->_capacidade * 2];  
  
    for (int i = 0; i < this->_num_elementos_inseridos; i++)  
      new_data[i] = this->_elementos[i];  
  
    delete[] this->_elementos;  
    this->_elementos = new_data;  
    this->_capacidade = this->_capacidade * 2;  
  }  
  // ...  
}
```

Passo a Passo

+apagamos os dados antigos



```
Objeto ListaVetorInteiros {  
    int *_elementos;  
    int _num_elementos_inseridos = 3;  
    int _capacidade = 3;  
}
```

```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
    if (this->_num_elementos_inseridos == this->_capacidade) {  
        int *new_data = new int[this->_capacidade * 2];  
  
        for (int i = 0; i < this->_num_elementos_inseridos; i++)  
            new_data[i] = this->_elementos[i];  
  
        delete[] this->_elementos;  
        this->_elementos = new_data;  
        this->_capacidade = this->_capacidade * 2;  
    }  
    // ...  
}
```

Passo a Passo

+ colocamos os
novos no local

```
Objeto ListaVetorInteiros {  
  int *_elementos;  
  int _num_elementos_inseridos = 3;  
  int _capacidade = 3;  
}
```

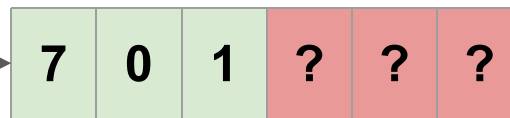


```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
  if (this->_num_elementos_inseridos == this->_capacidade) {  
    int *new_data = new int[this->_capacidade * 2];  
  
    for (int i = 0; i < this->_num_elementos_inseridos; i++)  
      new_data[i] = this->_elementos[i];  
  
    delete[] this->_elementos;  
    this->_elementos = new_data;  
    this->_capacidade = this->_capacidade * 2;  
  }  
  // ...  
}
```

Passo a Passo

+ aumentamos a
capacidade

```
Objeto ListaVetorInteiros {  
  int *_elementos;  
  int _num_elementos_inseridos = 3;  
  int _capacidade = 6;  
}
```

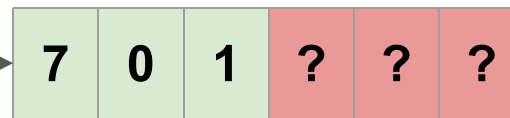


```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
  if (this->_num_elementos_inseridos == this->_capacidade) {  
    int *new_data = new int[this->_capacidade * 2];  
  
    for (int i = 0; i < this->_num_elementos_inseridos; i++)  
      new_data[i] = this->_elementos[i];  
  
    delete[] this->_elementos;  
    this->_elementos = new_data;  
    this->_capacidade = this->_capacidade * 2;  
  }  
  // ...  
}
```

Passo a Passo

Agora estamos igual a
antes

```
Objeto ListaVetorInteiros {  
  int *_elementos;  
  int _num_elementos_inseridos = 3;  
  int _capacidade = 6;  
}
```

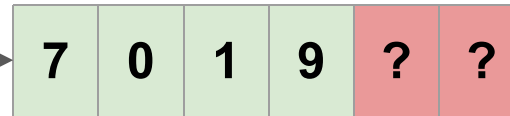


```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
  // ...  
  this->_elementos[this->_num_elementos_inserido] = elemento;  
  this->_num_elementos_inseridos++;  
}
```

Passo a Passo

Agora estamos igual a
antes

```
Objeto ListaVetorInteiros {  
  int *_elementos;  
  int _num_elementos_inseridos = 4;  
  int _capacidade = 6;  
}
```



```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
  // ...  
  this->_elementos[this->_num_elementos_inserido] = elemento;  
  this->_num_elementos_inseridos++;  
}
```

E para remover o último elemento?

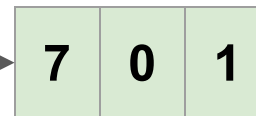
- Quero uma nova operação no meu TAD
 - Atualizar .h
 - Implementar no .cpp
- Remover o último elemento

E para remover o último elemento?

- Quero uma nova operação no meu TAD
- Remover o último elemento

```
void ListaVetorInteiros::remover_ultimo() {  
    this->_num_elementos_inseridos--;  
}
```

```
Objeto ListaVetorInteiros {  
    int *_elementos;  
    int _num_elementos_inseridos = 3;  
    int _capacidade = 3;  
}
```

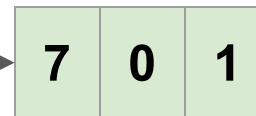


E para remover o último elemento?

- Quero uma nova operação no meu TAD
- Remover o último elemento

```
void ListaVetorInteiros::remover_ultimo() {  
    this->_num_elementos_inseridos--;  
}
```


```
Objeto ListaVetorInteiros {  
    int *_elementos;  
    int _num_elementos_inseridos = 3;  
    int _capacidade = 3;  
}
```



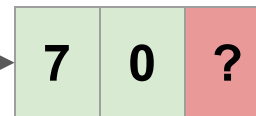
E para remover o último elemento?

- Quero uma nova operação no meu TAD
- Remover o último elemento

```
void ListaVetorInteiros::remover_ultimo() {  
    this->_num_elementos_inseridos--;  
}
```



```
Objeto ListaVetorInteiros {  
    int *_elementos;   
    int _num_elementos_inseridos = 3;  
    int _capacidade = 3;  
}
```



Removendo o Primeiro Elemento

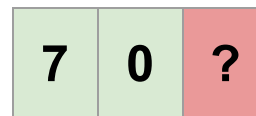
- Podemos copiar a ideia anterior
- Guardar um índice para o início
- Funciona bem?

Removendo o Primeiro Elemento

- Podemos copiar a ideia anterior
- Guardar um índice para o início
- Chato não desperdiçar memória

```
void ListaVetorInteiros::remover_ultimo() {  
    this->_inicio++;  
    this->_num_elementos_inseridos--;  
}
```


```
Objeto ListaVetorInteiros {  
    int *_elementos;  
    int _num_elementos_inseridos = 2;  
    int _capacidade = 3;  
    int _inicio = 0;  
}
```



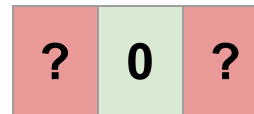
Removendo o Primeiro Elemento

- Podemos copiar a ideia anterior
- Guardar um índice para o início
- Chato não desperdiçar memória

```
void ListaVetorInteiros::remover_ultimo() {  
    this->_inicio++;  
    this->_num_elementos_inseridos--;  
}
```




```
Objeto ListaVetorInteiros {  
    int *_elementos;  
    int _num_elementos_inseridos = 2;  
    int _capacidade = 3;  
    int _inicio = 1;  
}
```



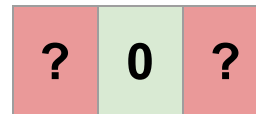
Removendo o Primeiro Elemento

- Podemos copiar a ideia anterior
- Guardar um índice para o início
- Chato não desperdiçar memória

```
void ListaVetorInteiros::remover_ultimo() {  
    this->_inicio++;  
    this->_num_elementos_inseridos--;  
}
```

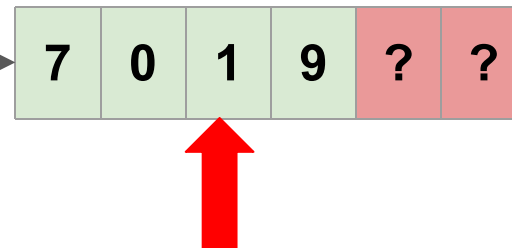


```
Objeto ListaVetorInteiros {  
    int *_elementos;  
    int _num_elementos_inseridos = 1;  
    int _capacidade = 3;  
    int _inicio = 1;  
}
```



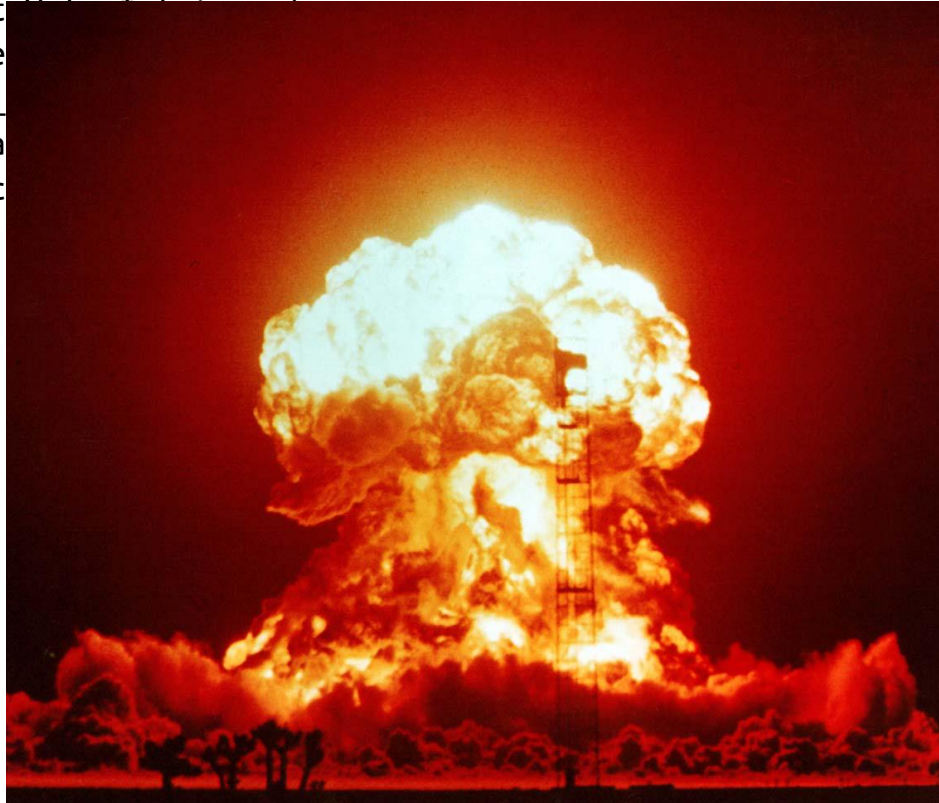
Remover o n-ésimo elemento?

```
Objeto ListaVetorInteiros {  
  int *_elementos;  
  int _num_elementos_inseridos = 4;  
  int _capacidade = 6;  
  int _inicio = 0;  
}
```



Remover o n-ésimo elemento?

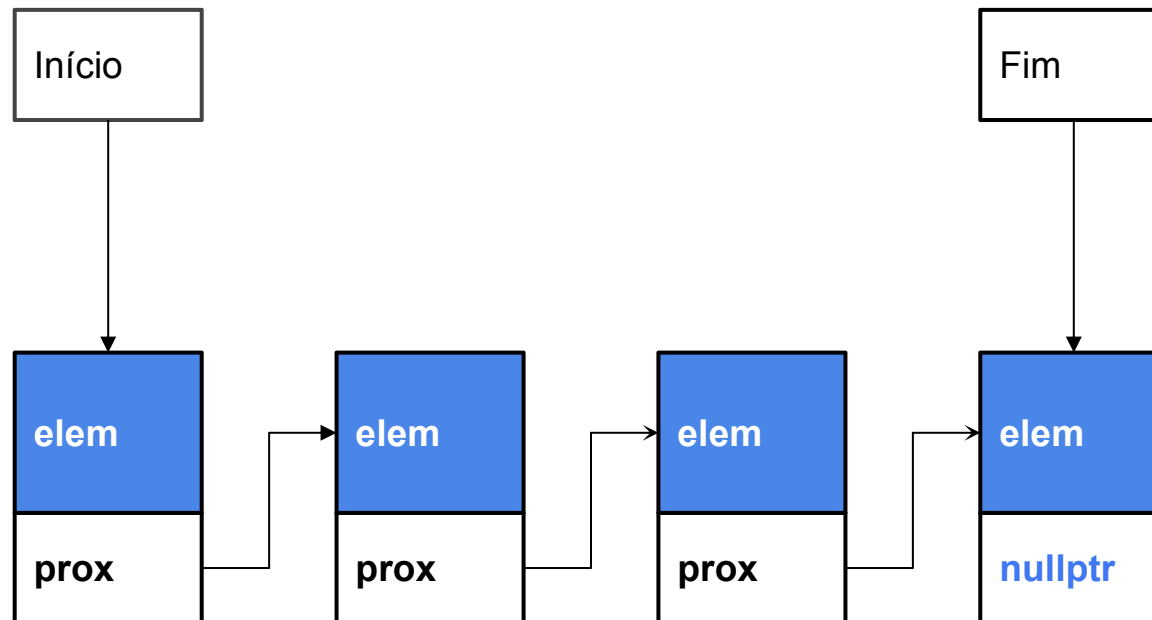
```
Objeto Lista  
int *_ele  
int _num_  
int _capa  
int _inic  
}
```



Listas com Ponteiros

Lista com Ponteiros

- Para casos complicados com arrays
- Podemos explorar ponteiros



```
#ifndef PDS2_NODE_H  
#define PDS2_NODE_H
```

```
struct node_t {  
    int elemento;  
    node_t *proximo;  
};
```

← Struct sem métodos. Representa um elemento

```
class ListaSimplesmenteEncadeada {
```

```
private:
```

```
    node_t *__inicio;  
    node_t *__fim;
```

← Início e fim da lista

```
    int __num_elementos_inseridos;
```

```
public:
```

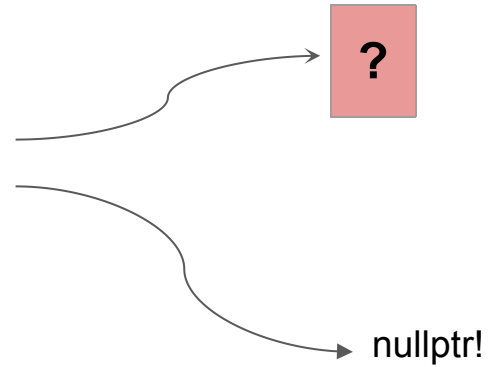
```
    ListaSimplesmenteEncadeada();  
    ~ListaSimplesmenteEncadeada();  
    void inserir_elemento(int elemento);  
    void imprimir();
```

```
};
```

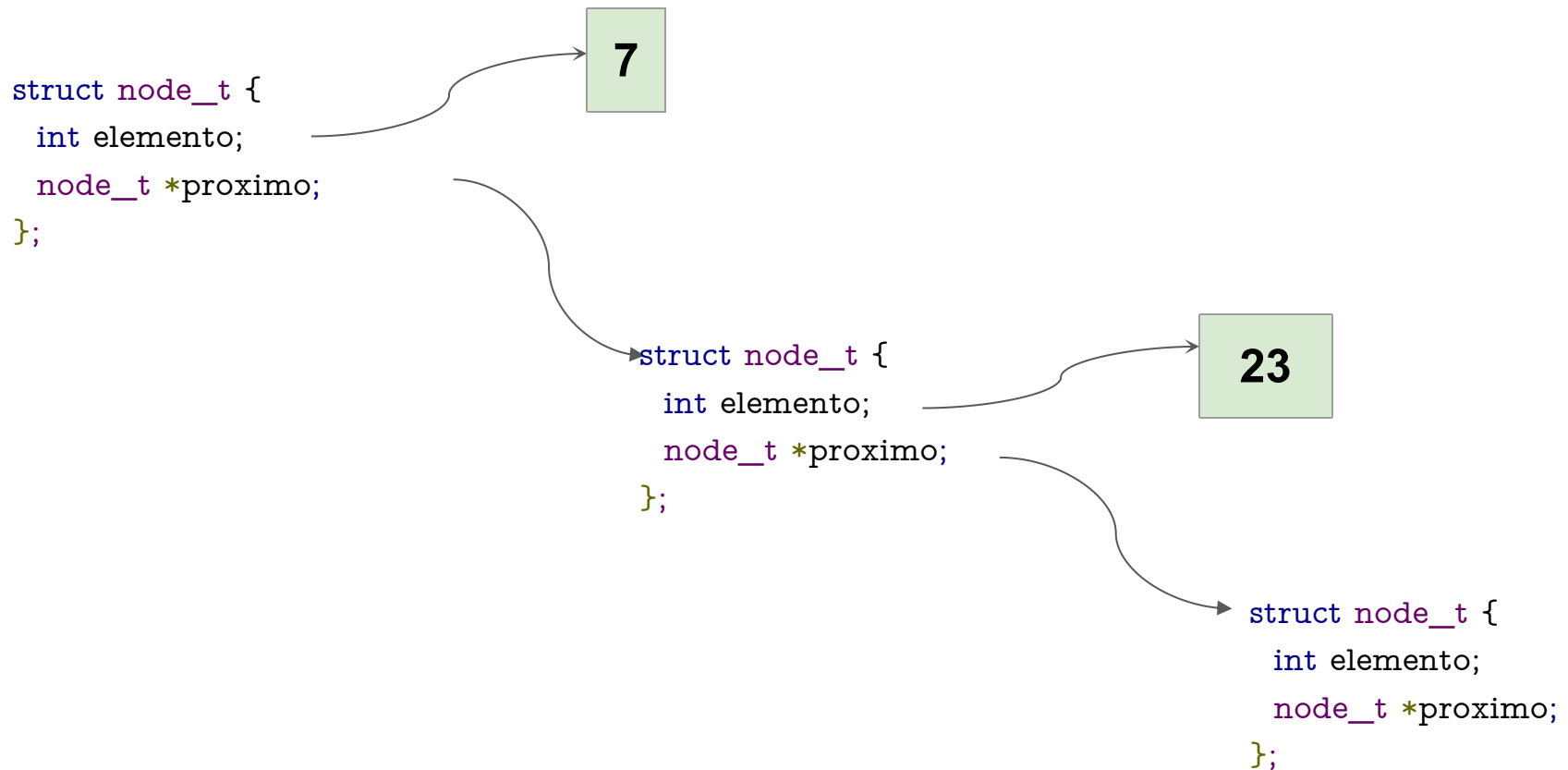
```
#endif
```

Iniciamos Assim

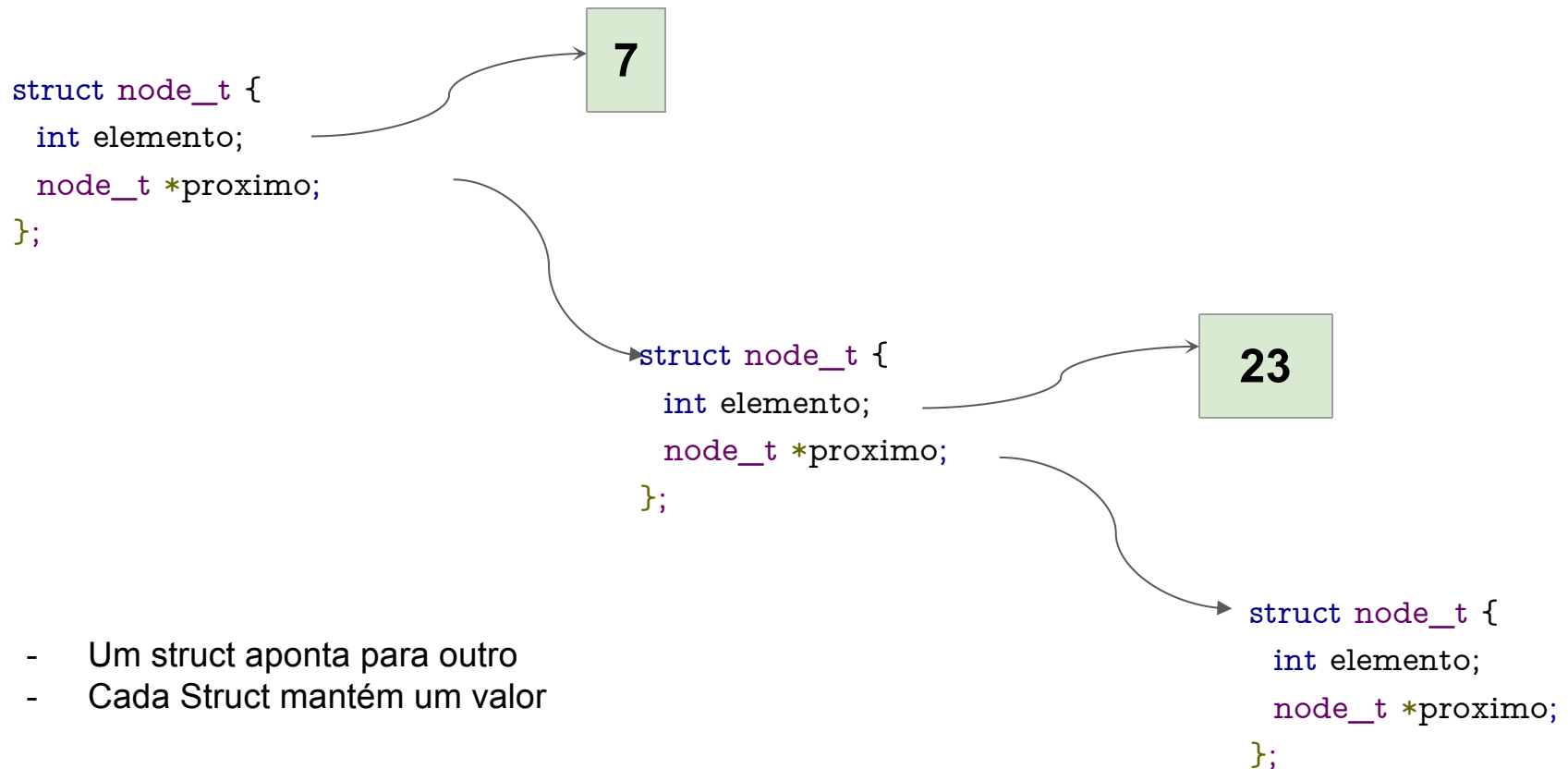
```
struct node_t {  
    int elemento;  
    node_t *next;  
};
```



Ficamos Assim

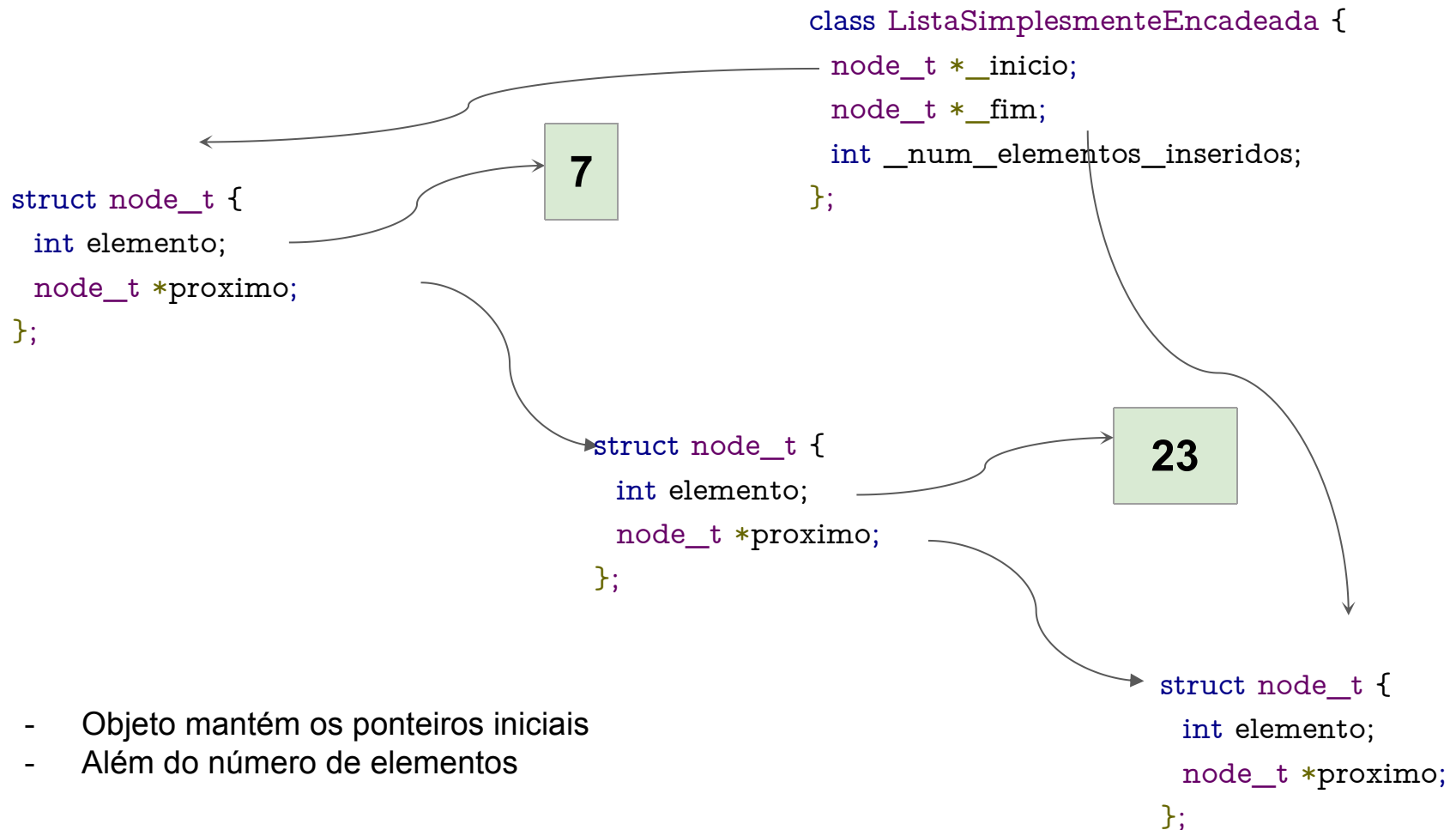


Ficamos Assim



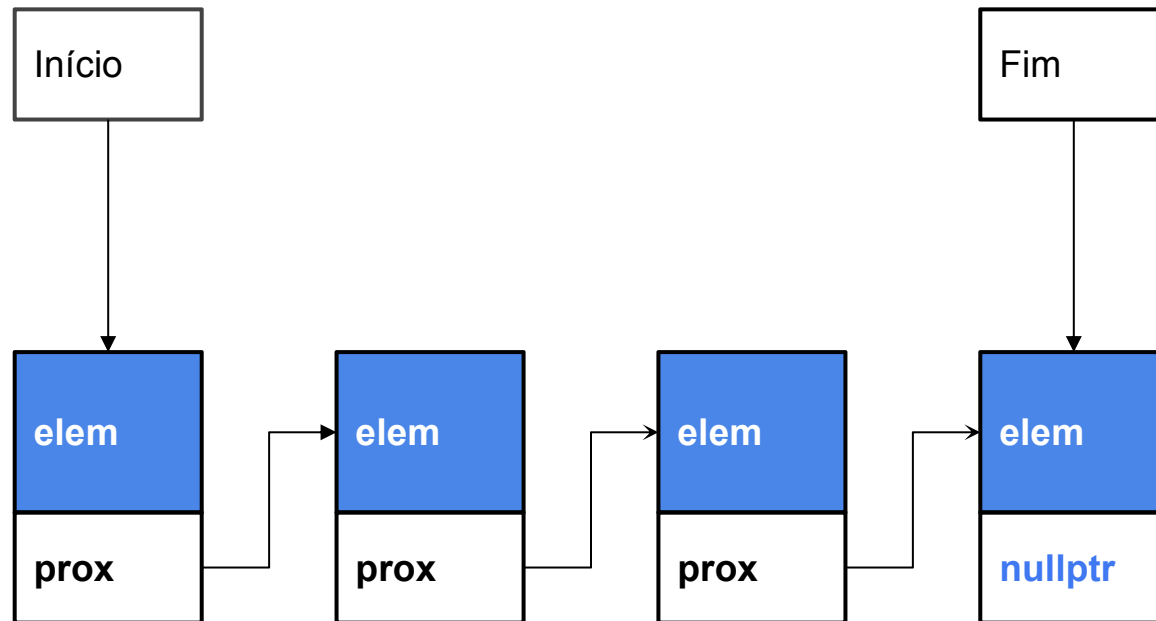
- Um struct aponta para outro
- Cada Struct mantém um valor

Ficamos Assim



- Objeto mantém os ponteiros iniciais
- Além do número de elementos

Mais Abstrato



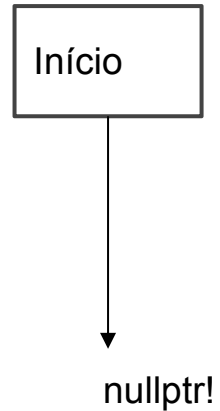
Vamos Implementar?

Construtor

```
ListaSimplesmenteEncadeada::ListaSimplesmenteEncadeada() {  
→ this->_inicio = nullptr;  
  this->_fim = nullptr;  
  this->_num_elementos_inseridos = 0;  
}
```

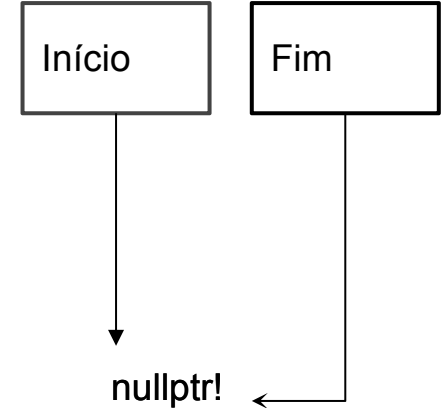
Construtor

```
ListaSimplesmenteEncadeada::ListaSimplesmenteEncadeada() {  
    this->_inicio = nullptr;  
    this->_fim = nullptr;  
    this->_num_elementos_inseridos = 0;  
}
```



Construtor

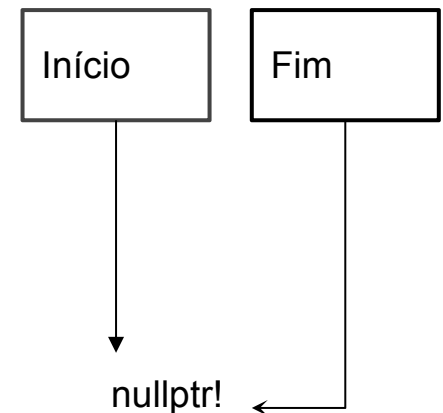
```
ListaSimplesmenteEncadeada::ListaSimplesmenteEncadeada() {  
    this->_inicio = nullptr;  
    this->_fim = nullptr;  
    this->_num_elementos_inseridos = 0;  
}
```



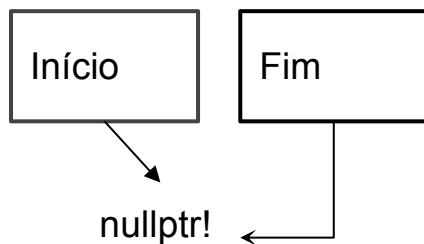
Construtor

```
ListaSimplesmenteEncadeada::ListaSimplesmenteEncadeada() {  
    this->_inicio = nullptr;  
    this->_fim = nullptr;  
    this->_num_elementos_inseridos = 0;  
}
```

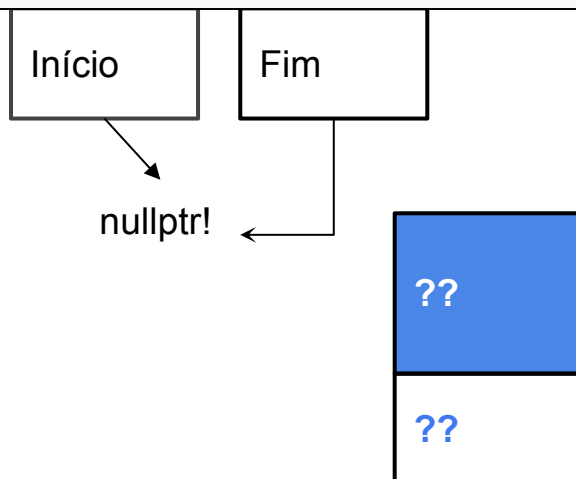
```
Objeto ListaVetorInteiros {  
    node_t *inicio = nullptr;  
    node_t *fim = nullptr;  
    int _num_elementos_inseridos = 0;  
}
```



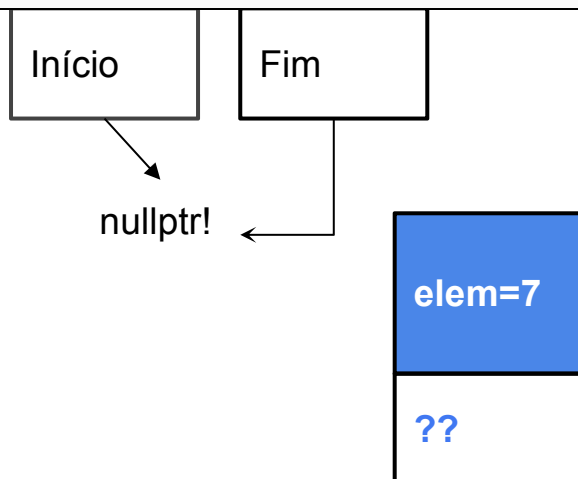
```
void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {  
    node_t *novo = new node_t();  
    novo->elemento = elemento;  
    novo->proximo = nullptr;  
    if (this->_inicio == nullptr) {  
        this->_inicio = novo;  
        this->_fim = novo;  
    } else {  
        this->_fim->proximo = novo;  
        this->_fim = novo;  
    }  
    this->_num_elementos_inseridos++;  
}
```



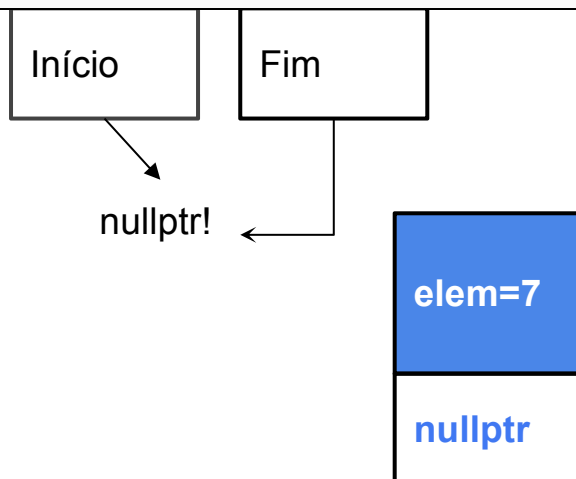
```
void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {  
    node_t *novo = new node_t();  
    novo->elemento = elemento;  
    novo->proximo = nullptr;  
    if (this->_inicio == nullptr) {  
        this->_inicio = novo;  
        this->_fim = novo;  
    } else {  
        this->_fim->proximo = novo;  
        this->_fim = novo;  
    }  
    this->_num_elementos_inseridos++;  
}
```



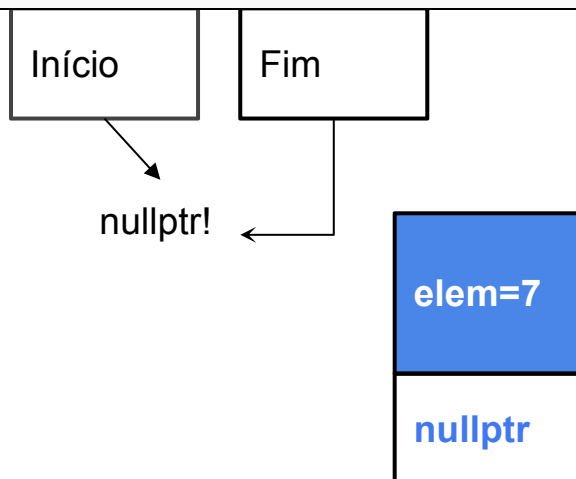
```
void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {  
    node_t *novo = new node_t();  
    novo->elemento = elemento;  
    novo->proximo = nullptr;  
    if (this->_inicio == nullptr) {  
        this->_inicio = novo;  
        this->_fim = novo;  
    } else {  
        this->_fim->proximo = novo;  
        this->_fim = novo;  
    }  
    this->_num_elementos_inseridos++;  
}
```



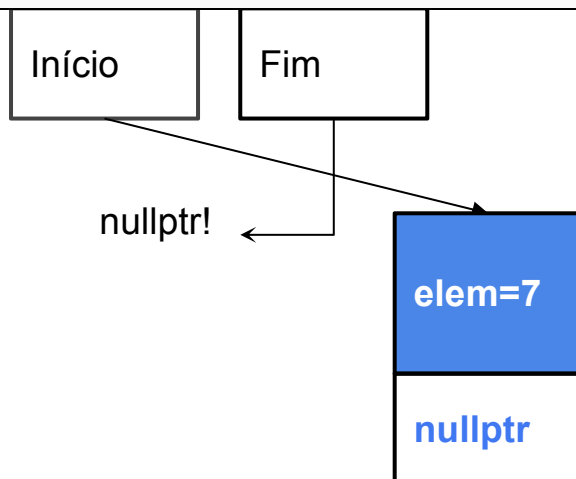

```
void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {  
    node_t *novo = new node_t();  
    novo->elemento = elemento;  
    novo->proximo = nullptr;  
    if (this->_inicio == nullptr) {  
        this->_inicio = novo;  
        this->_fim = novo;  
    } else {  
        this->_fim->proximo = novo;  
        this->_fim = novo;  
    }  
    this->_num_elementos_inseridos++;  
}
```



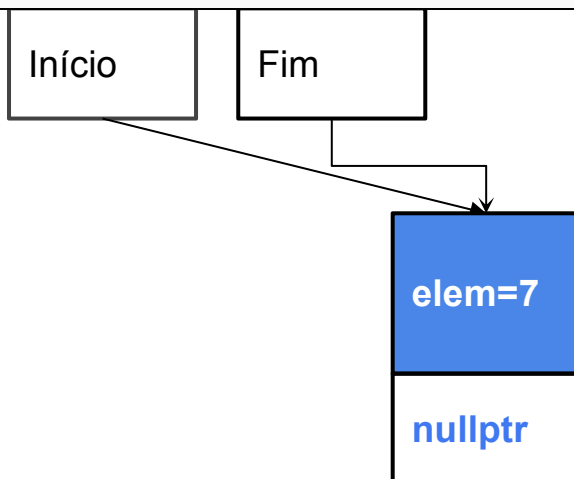
```
void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {  
    node_t *novo = new node_t();  
    novo->elemento = elemento;  
    novo->proximo = nullptr;  
    if (this->_inicio == nullptr) {  
        this->_inicio = novo;  
        this->_fim = novo;  
    } else {  
        this->_fim->proximo = novo;  
        this->_fim = novo;  
    }  
    this->_num_elementos_inseridos++;  
}
```



```
void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {  
    node_t *novo = new node_t();  
    novo->elemento = elemento;  
    novo->proximo = nullptr;  
    if (this->_inicio == nullptr) {  
        this->_inicio = novo;  
        this->_fim = novo;  
    } else {  
        this->_fim->proximo = novo;  
        this->_fim = novo;  
    }  
    this->_num_elementos_inseridos++;  
}
```

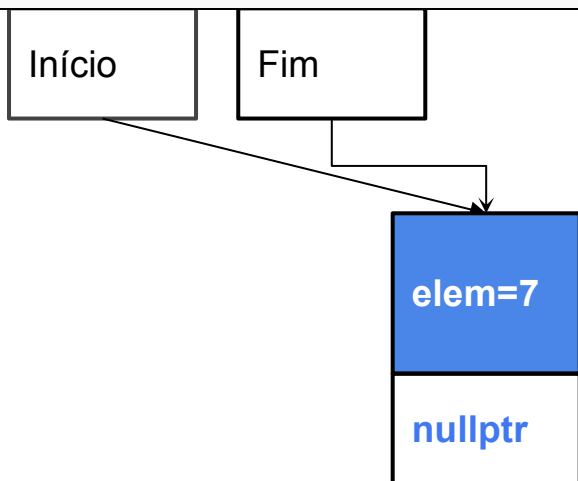


```
void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {  
    node_t *novo = new node_t();  
    novo->elemento = elemento;  
    novo->proximo = nullptr;  
    if (this->_inicio == nullptr) {  
        this->_inicio = novo;  
        this->_fim = novo;  
    } else {  
        this->_fim->proximo = novo;  
        this->_fim = novo;  
    }  
    this->_num_elementos_inseridos++;  
}
```

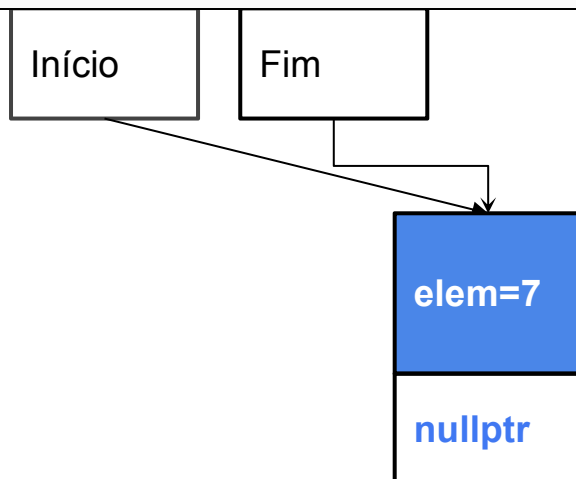




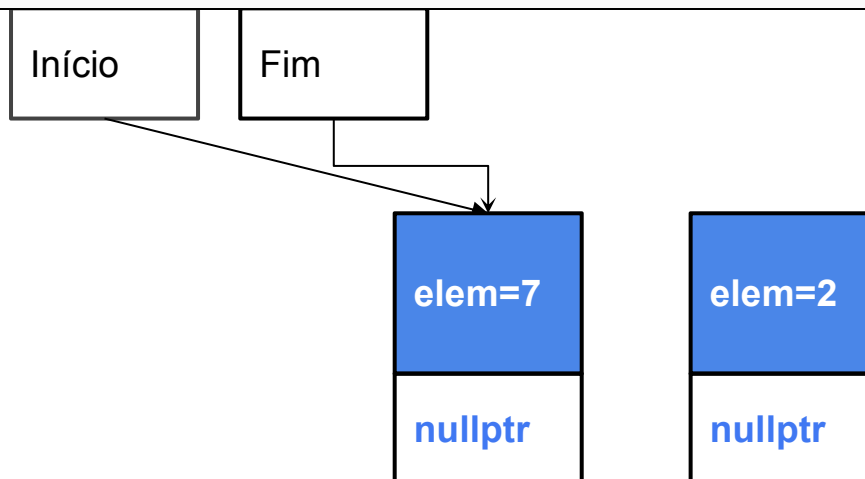
```
void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {  
    node_t *novo = new node_t();  
    novo->elemento = elemento;  
    novo->proximo = nullptr;  
    if (this->_inicio == nullptr) {  
        this->_inicio = novo;  
        this->_fim = novo;  
    } else {  
        this->_fim->proximo = novo;  
        this->_fim = novo;  
    }  
    this->_num_elementos_inseridos++;  
}
```



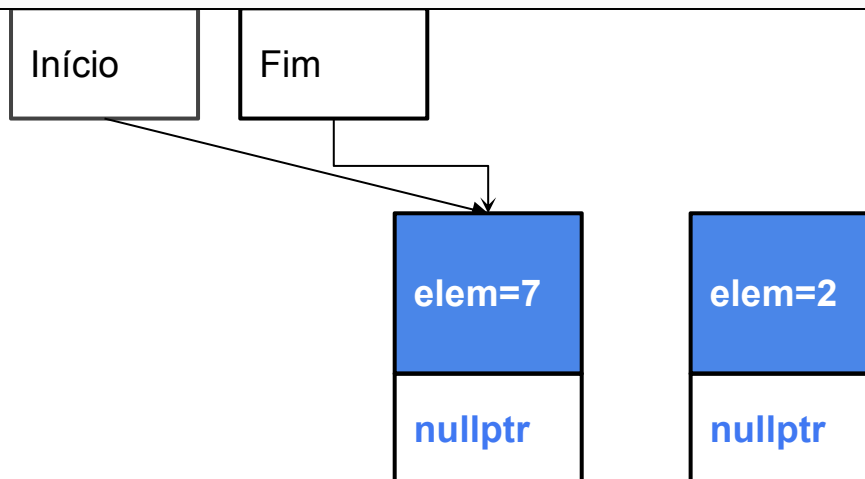
```
void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {  
node_t *novo = new node_t();  
novo->elemento = elemento;  
novo->proximo = nullptr;  
if (this->_inicio == nullptr) {  
this->_inicio = novo;  
this->_fim = novo;  
} else {  
this->_fim->proximo = novo;  
this->_fim = novo;  
}  
this->_num_elementos_inseridos++;  
}
```



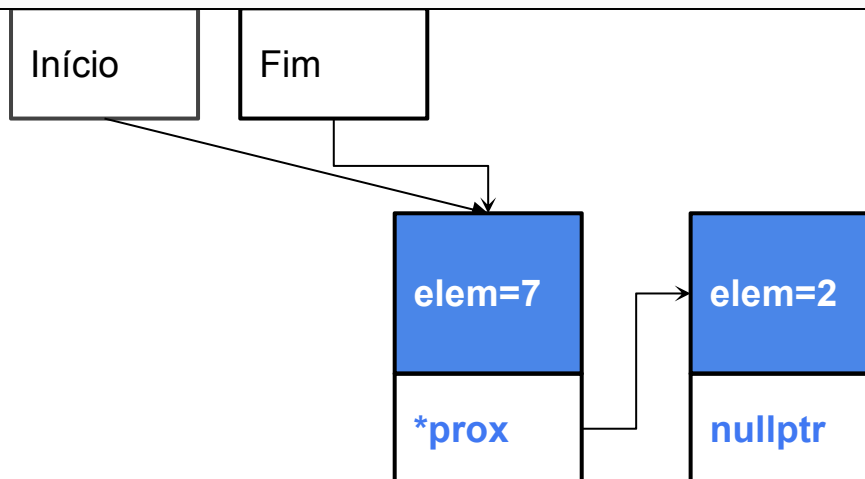
```
void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {  
    node_t *novo = new node_t();  
    novo->elemento = elemento;  
    novo->proximo = nullptr;  
    if (this->_inicio == nullptr) {  
        this->_inicio = novo;  
        this->_fim = novo;  
    } else {  
        this->_fim->proximo = novo;  
        this->_fim = novo;  
    }  
    this->_num_elementos_inseridos++;  
}
```



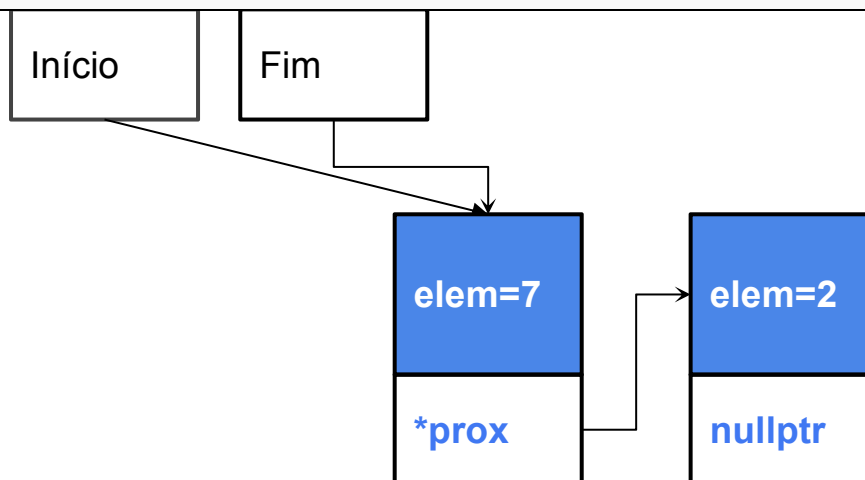
```
void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {  
    node_t *novo = new node_t();  
    novo->elemento = elemento;  
    novo->proximo = nullptr;  
    if (this->_inicio == nullptr) {  
        this->_inicio = novo;  
        this->_fim = novo;  
    } else {  
        this->_fim->proximo = novo;  
        this->_fim = novo;  
    }  
    this->_num_elementos_inseridos++;  
}
```



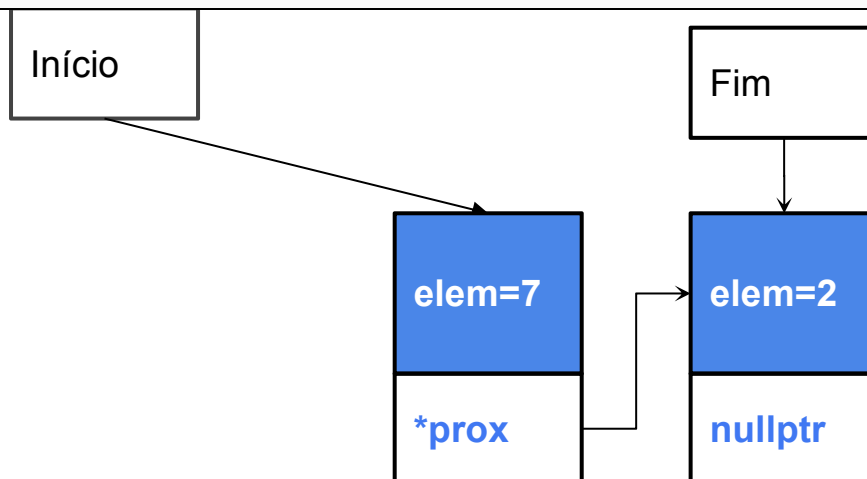

```
void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {  
    node_t *novo = new node_t();  
    novo->elemento = elemento;  
    novo->proximo = nullptr;  
    if (this->_inicio == nullptr) {  
        this->_inicio = novo;  
        this->_fim = novo;  
    } else {  
        this->_fim->proximo = novo;  
        this->_fim = novo;  
    }  
    this->_num_elementos_inseridos++;  
}
```



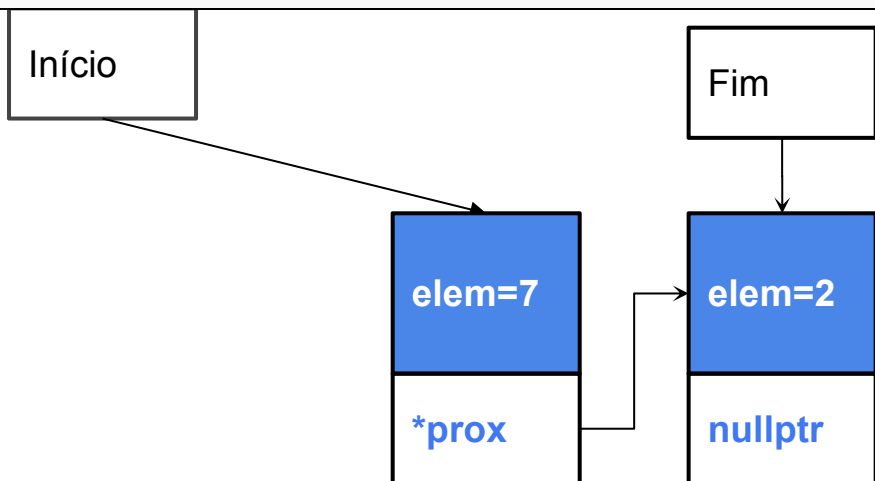
```
void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {  
    node_t *novo = new node_t();  
    novo->elemento = elemento;  
    novo->proximo = nullptr;  
    if (this->_inicio == nullptr) {  
        this->_inicio = novo;  
        this->_fim = novo;  
    } else {  
        this->_fim->proximo = novo;  
        this->_fim = novo;  
    }  
    this->_num_elementos_inseridos++;  
}
```



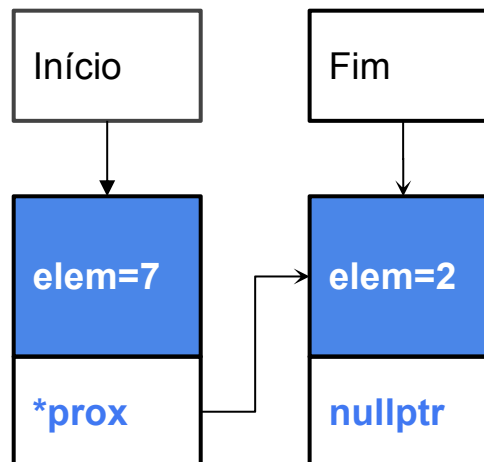
```
void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {  
    node_t *novo = new node_t();  
    novo->elemento = elemento;  
    novo->proximo = nullptr;  
    if (this->_inicio == nullptr) {  
        this->_inicio = novo;  
        this->_fim = novo;  
    } else {  
        this->_fim->proximo = novo;  
        this->_fim = novo;  
    }  
    this->_num_elementos_inseridos++;  
}
```



```
void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {  
    node_t *novo = new node_t();  
    novo->elemento = elemento;  
    novo->proximo = nullptr;  
    if (this->_inicio == nullptr) {  
        this->_inicio = novo;  
        this->_fim = novo;  
    } else {  
        this->_fim->proximo = novo;  
        this->_fim = novo;  
    }  
    this->_num_elementos_inseridos++;  
}
```



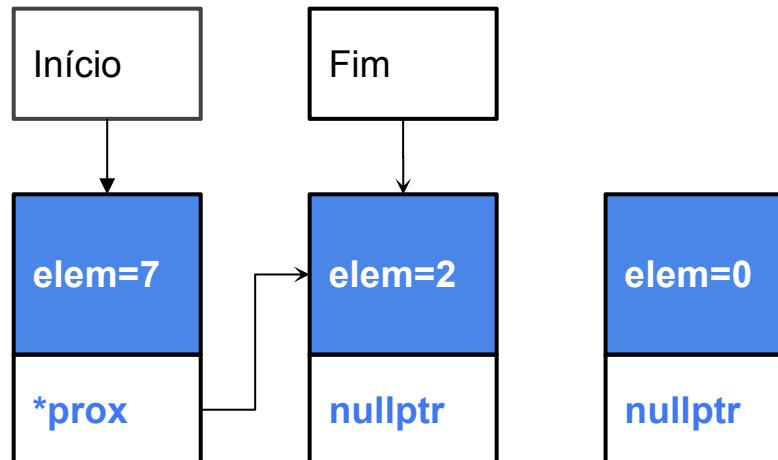
```
void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {  
    node_t *novo = new node_t();  
    novo->elemento = elemento;  
    novo->proximo = nullptr;  
    if (this->_inicio == nullptr) {  
        this->_inicio = novo;  
        this->_fim = novo;  
    } else {  
        this->_fim->proximo = novo;  
        this->_fim = novo;  
    }  
    this->_num_elementos_inseridos++;  
}
```



```

void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {
    node_t *novo = new node_t();
    novo->elemento = elemento;
    novo->proximo = nullptr;
    if (this->_inicio == nullptr) {
        this->_inicio = novo;
        this->_fim = novo;
    } else {
        this->_fim->proximo = novo;
        this->_fim = novo;
    }
    this->_num_elementos_inseridos++;
}

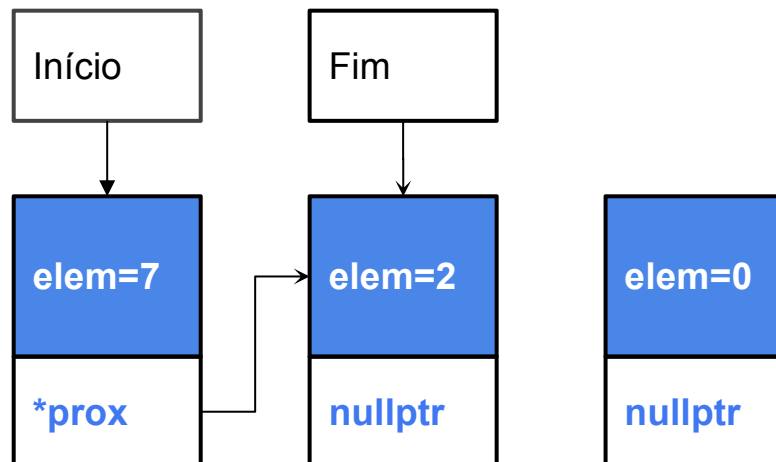
```



```

void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {
    node_t *novo = new node_t();
    novo->elemento = elemento;
    novo->proximo = nullptr;
    if (this->_inicio == nullptr) {
        this->_inicio = novo;
        this->_fim = novo;
    } else {
        this->_fim->proximo = novo;
        this->_fim = novo;
    }
    this->_num_elementos_inseridos++;
}

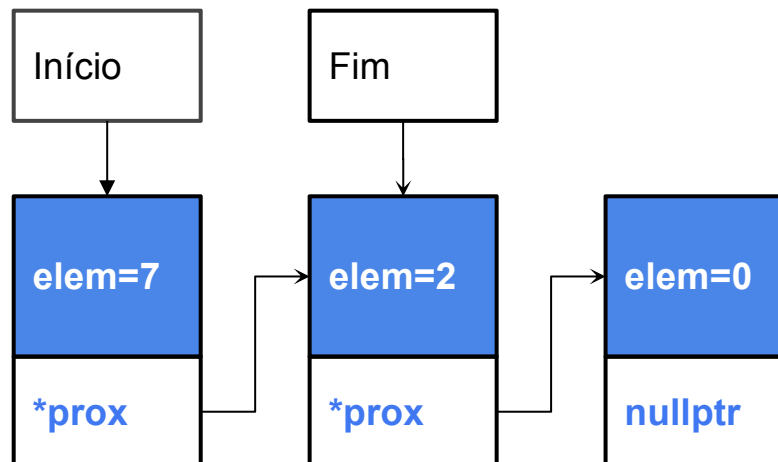
```



```

void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {
    node_t *novo = new node_t();
    novo->elemento = elemento;
    novo->proximo = nullptr;
    if (this->_inicio == nullptr) {
        this->_inicio = novo;
        this->_fim = novo;
    } else {
        this->_fim->proximo = novo;
        this->_fim = novo;
    }
    this->_num_elementos_inseridos++;
}

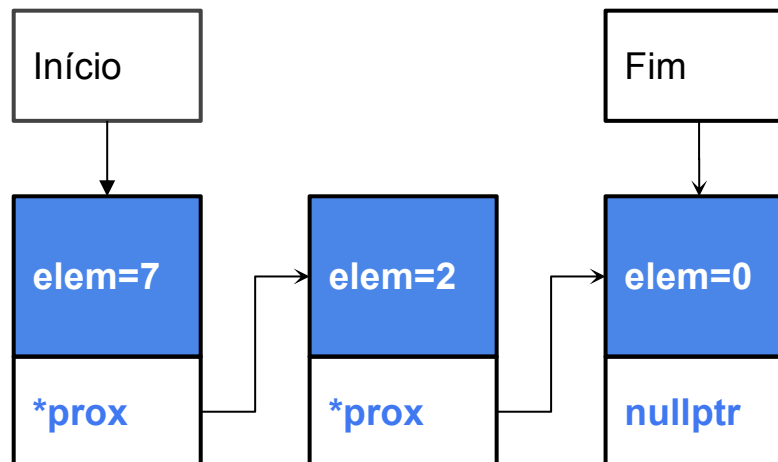
```




```

void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {
    node_t *novo = new node_t();
    novo->elemento = elemento;
    novo->proximo = nullptr;
    if (this->_inicio == nullptr) {
        this->_inicio = novo;
        this->_fim = novo;
    } else {
        this->_fim->proximo = novo;
        this->_fim = novo;
    }
    this->_num_elementos_inseridos++;
}

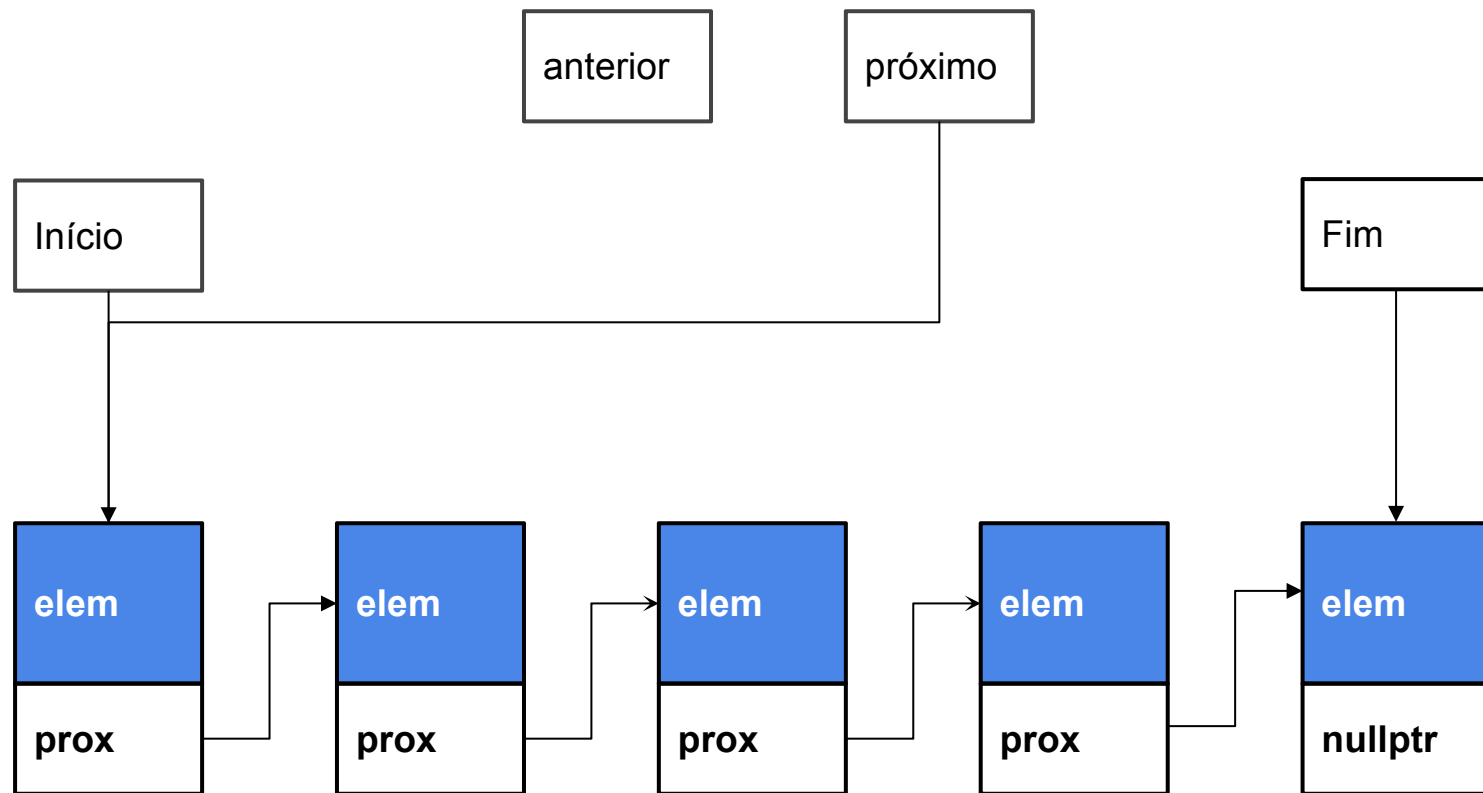
```



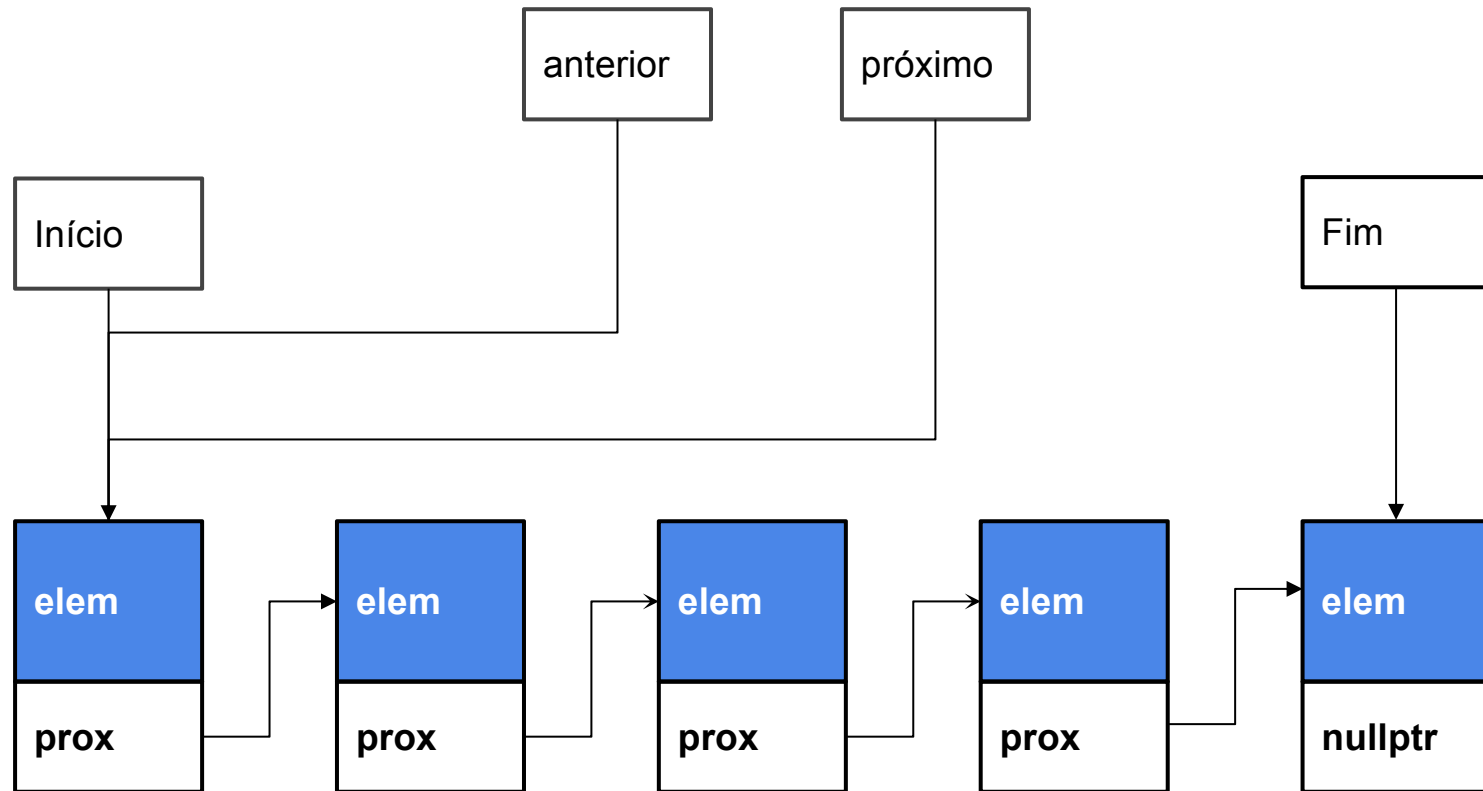
Qual a ideia?

- Criar um novo elemento
- Setar o valor
- Atualizar o ponteiro do último
- Atualizar o fim da lista!

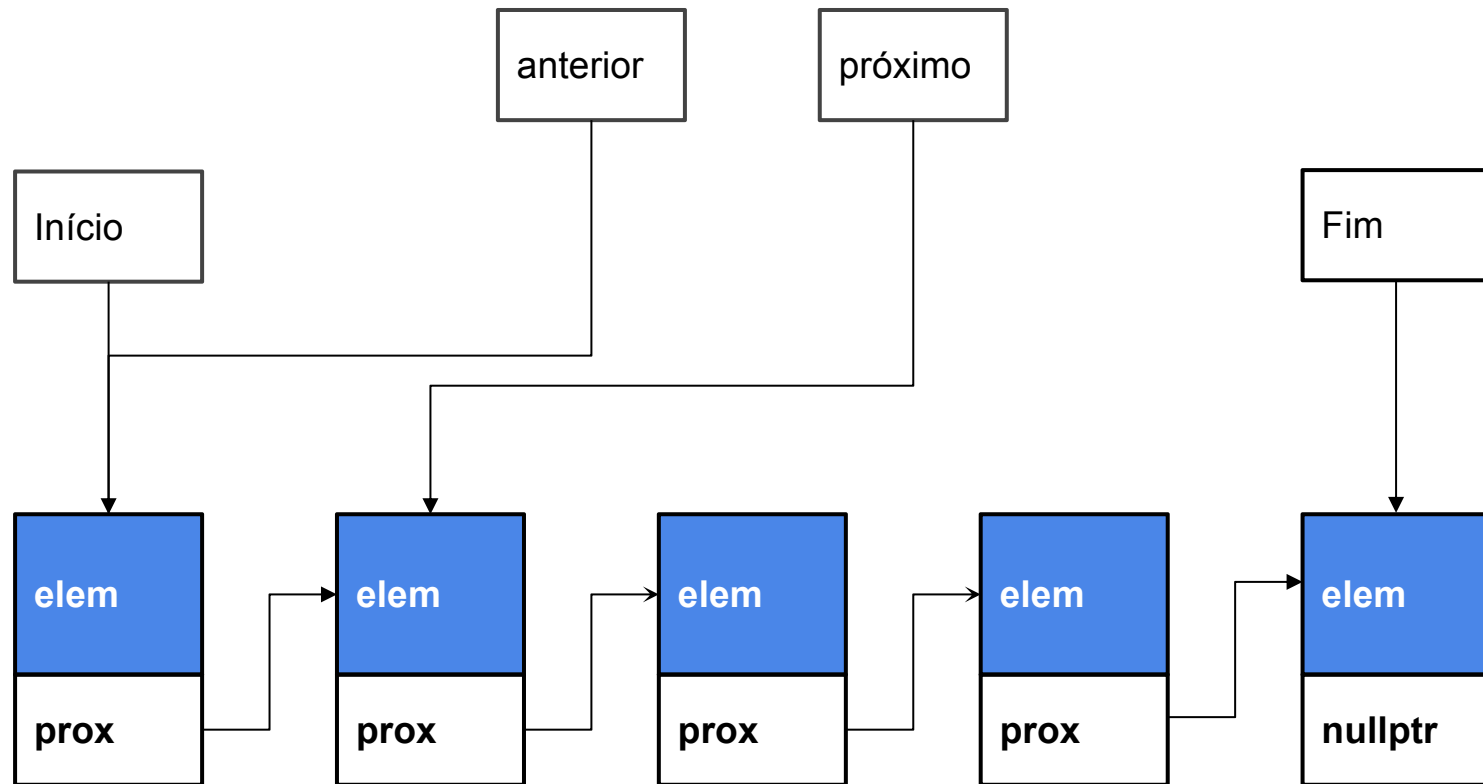
Destrutor



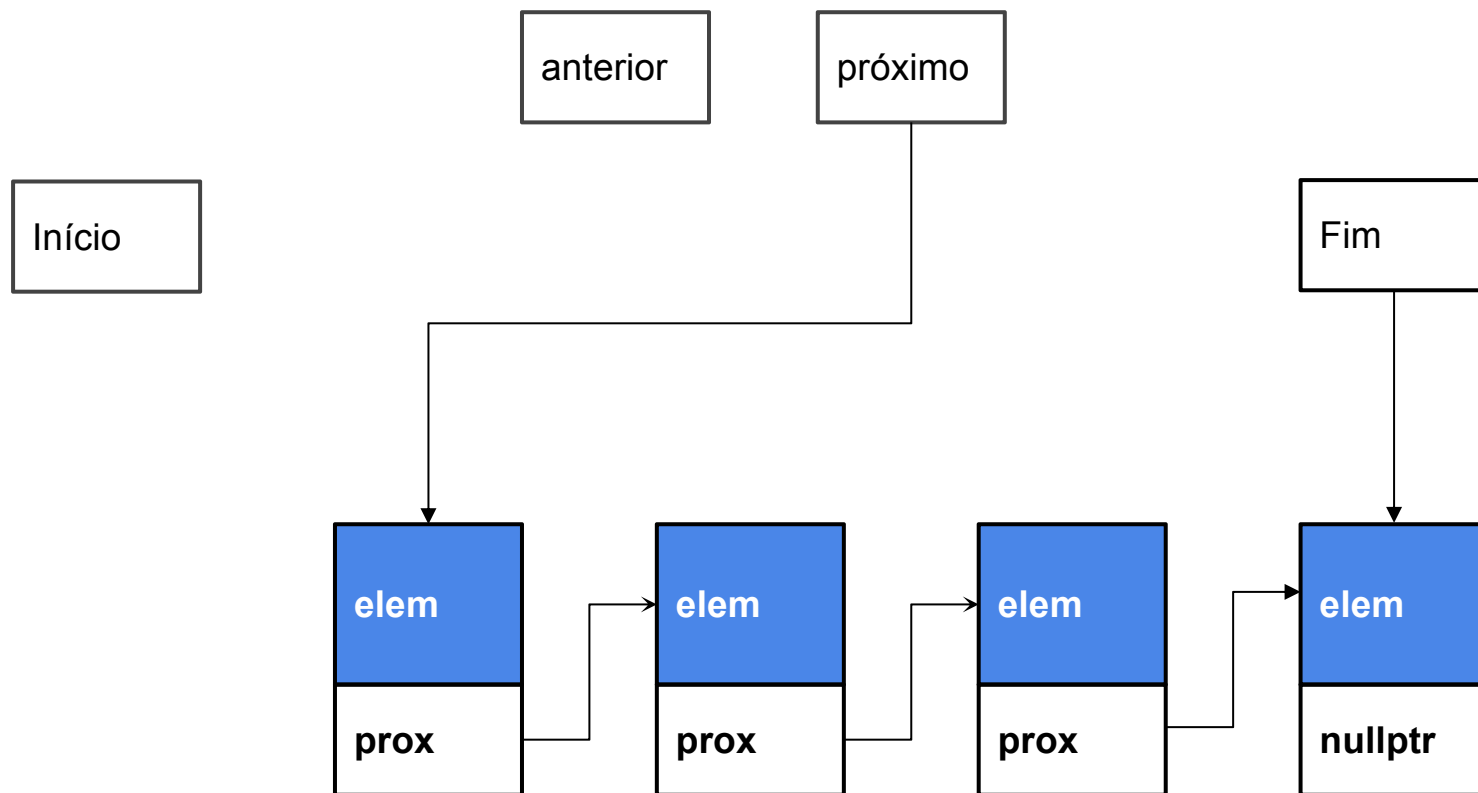
Destrutor



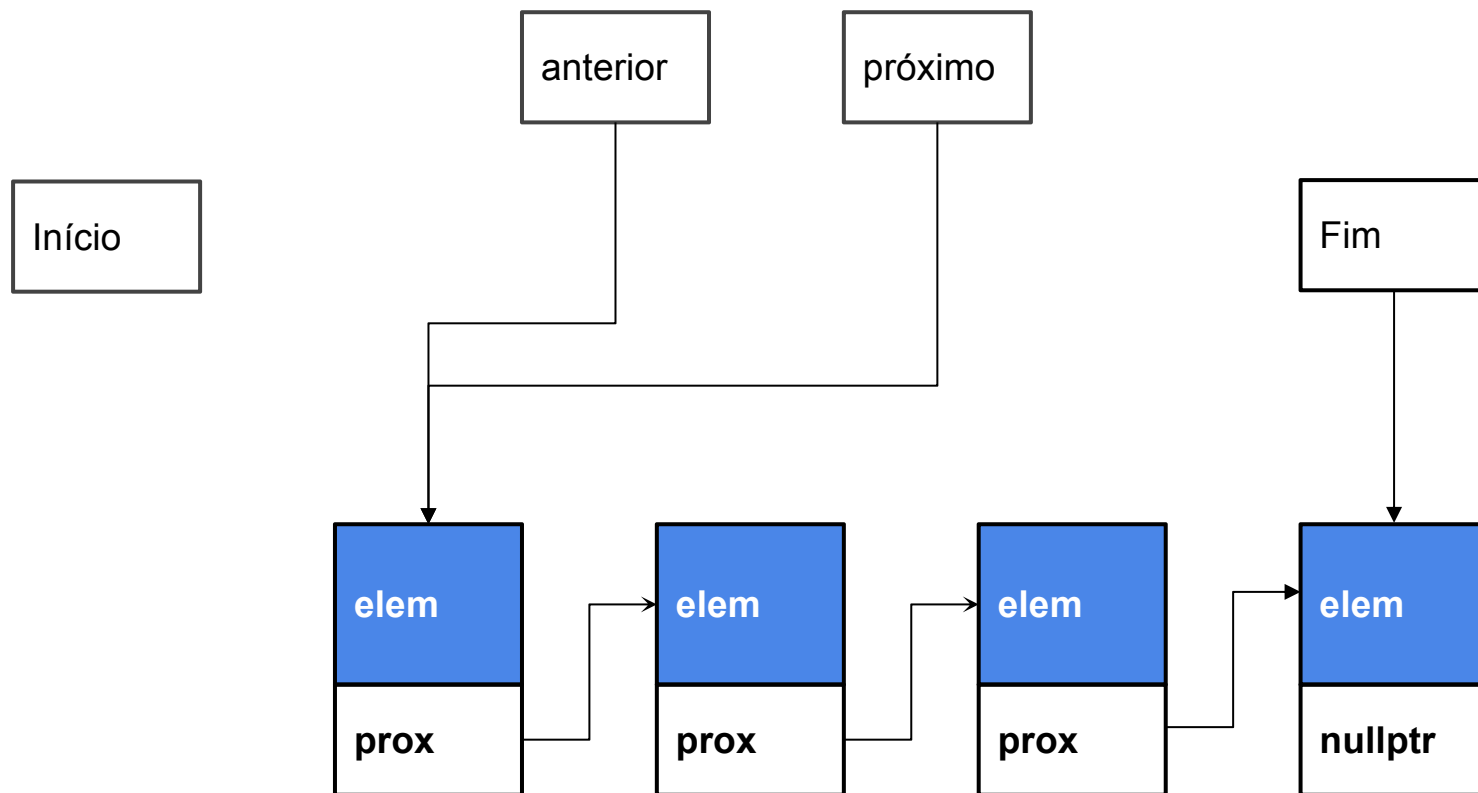
Destrutor



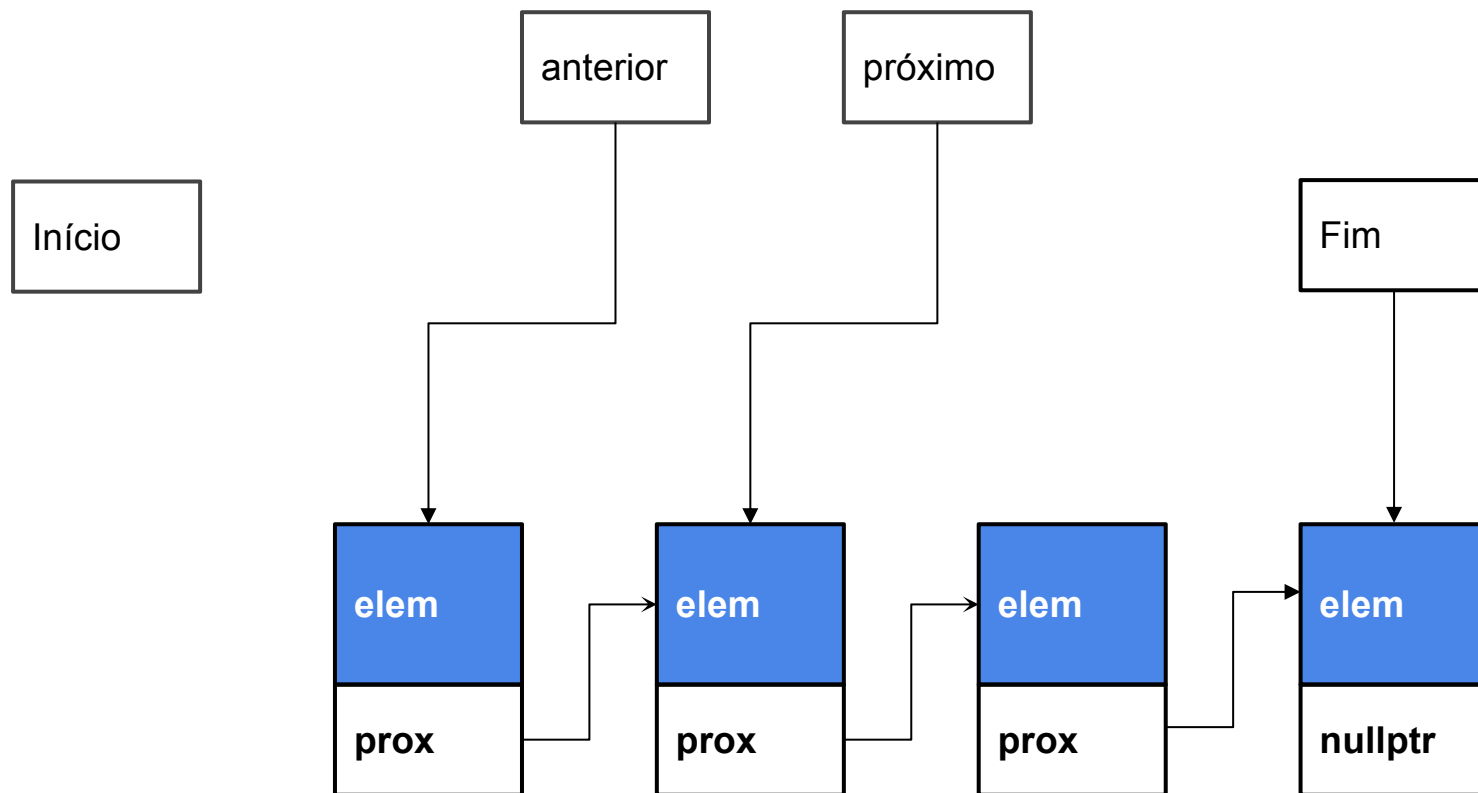
Destrutor



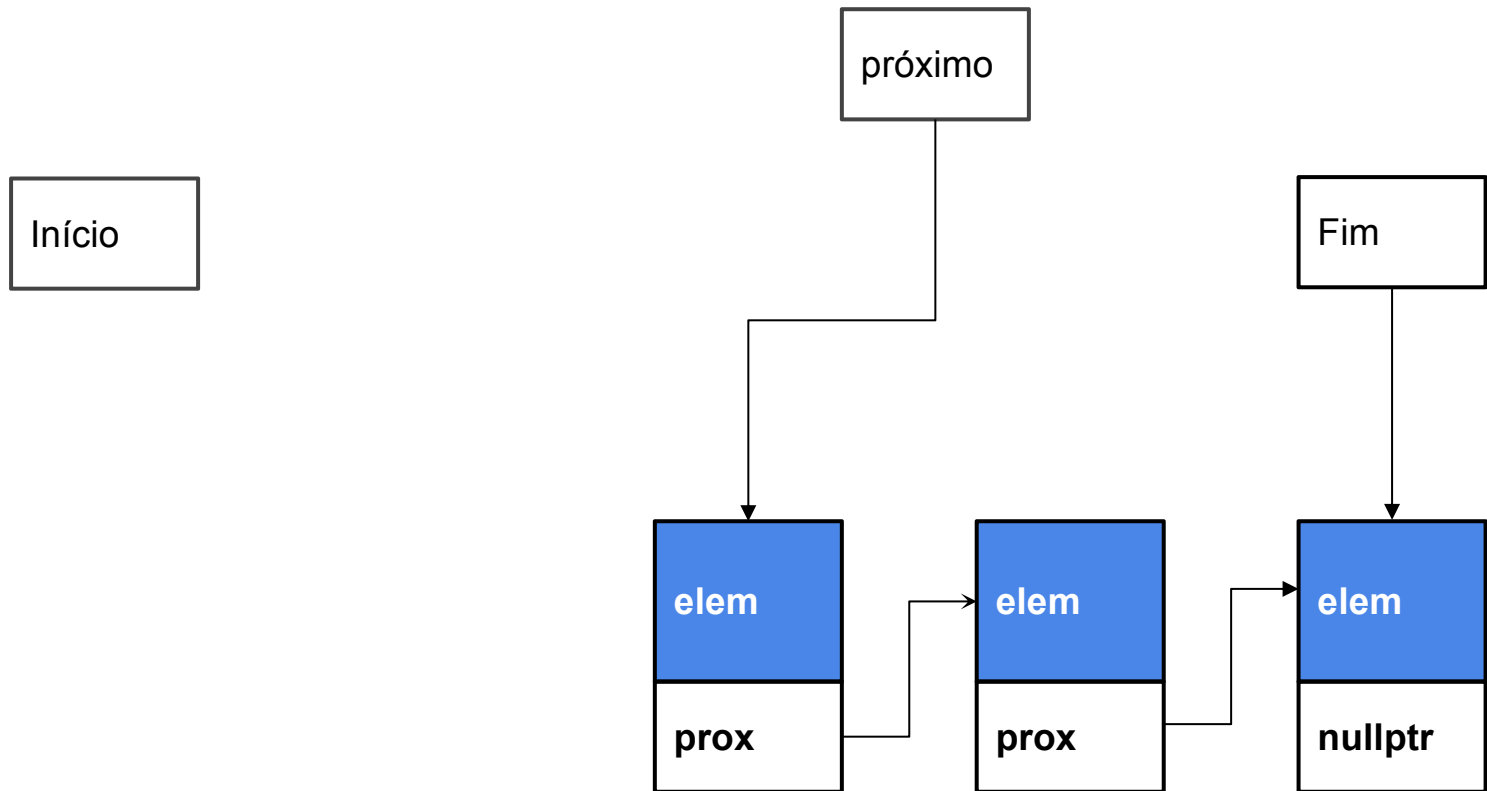
Destrutor



Destrutor

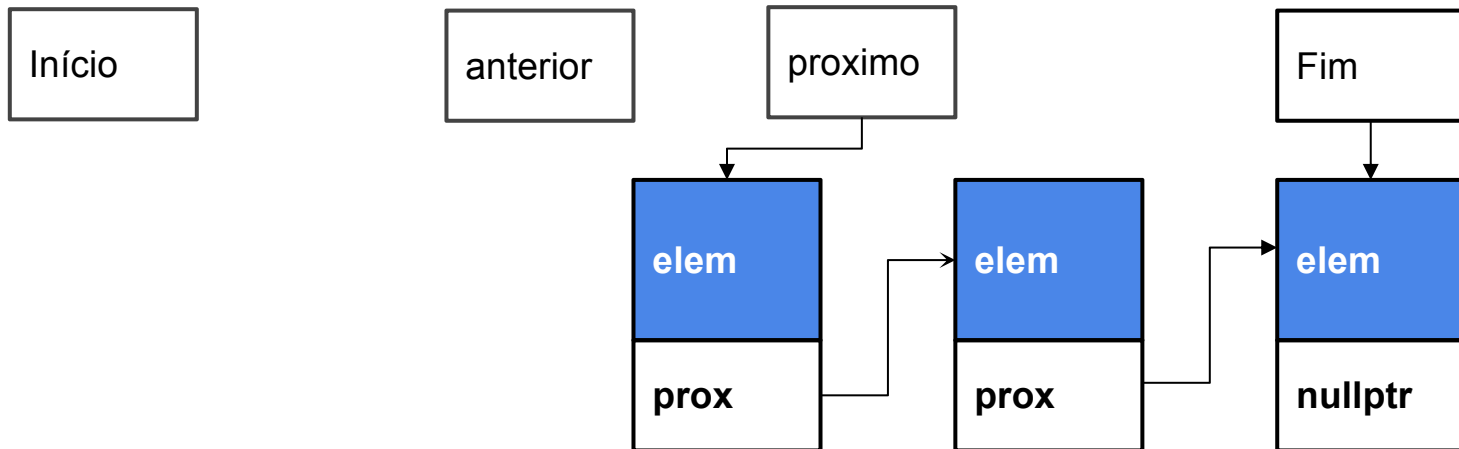


Destrutor



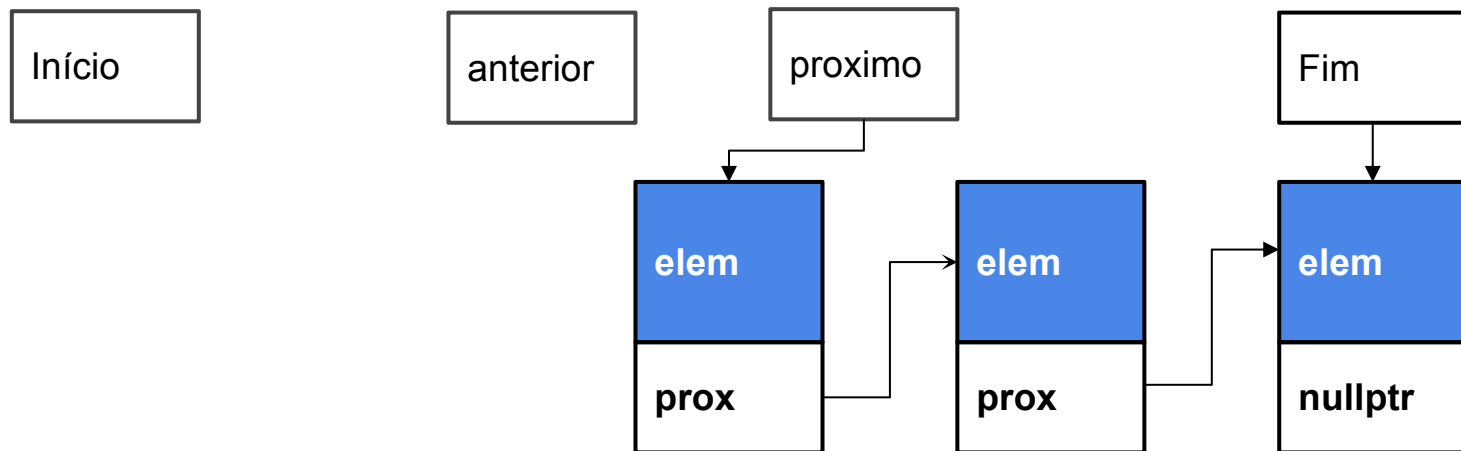
Destruitor

```
ListaSimplesmenteEncadeada::~ListaSimplesmenteEncadeada() {  
    node_t *anterior = nullptr;  
    node_t *proximo = this->_inicio;  
    while (proximo != nullptr) {  
        anterior = proximo;  
        proximo = proximo->proximo;  
        delete anterior;  
    }  
}
```



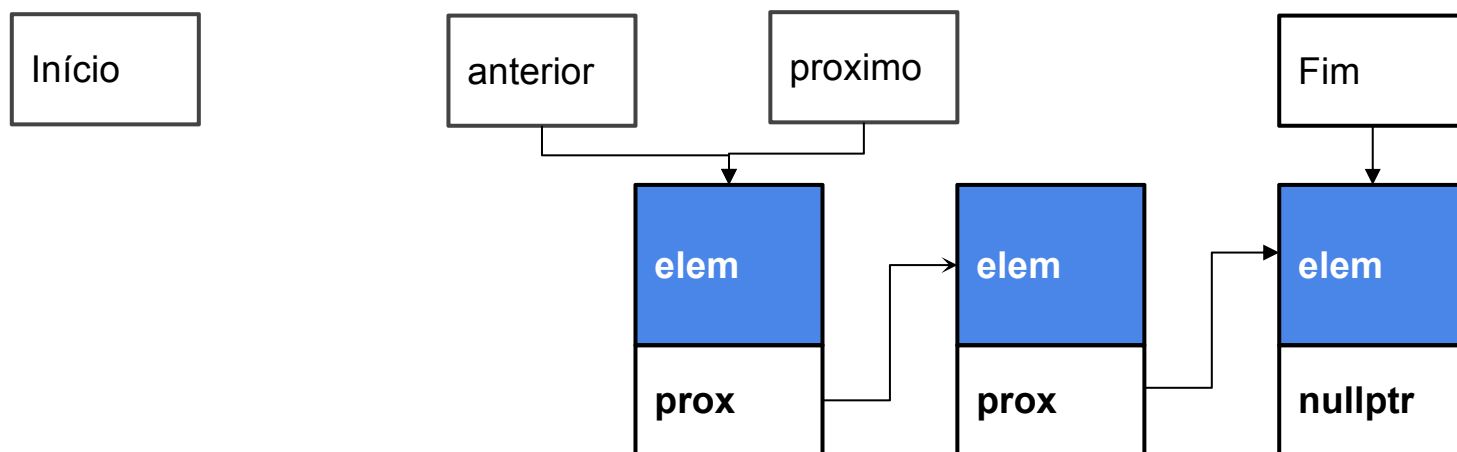
Destruitor

```
ListaSimplesmenteEncadeada::~ListaSimplesmenteEncadeada() {  
    node_t *anterior = nullptr;  
    node_t *proximo = this->_inicio;  
    while (proximo != nullptr) {  
        anterior = proximo;  
        proximo = proximo->proximo;  
        delete anterior;  
    }  
}
```



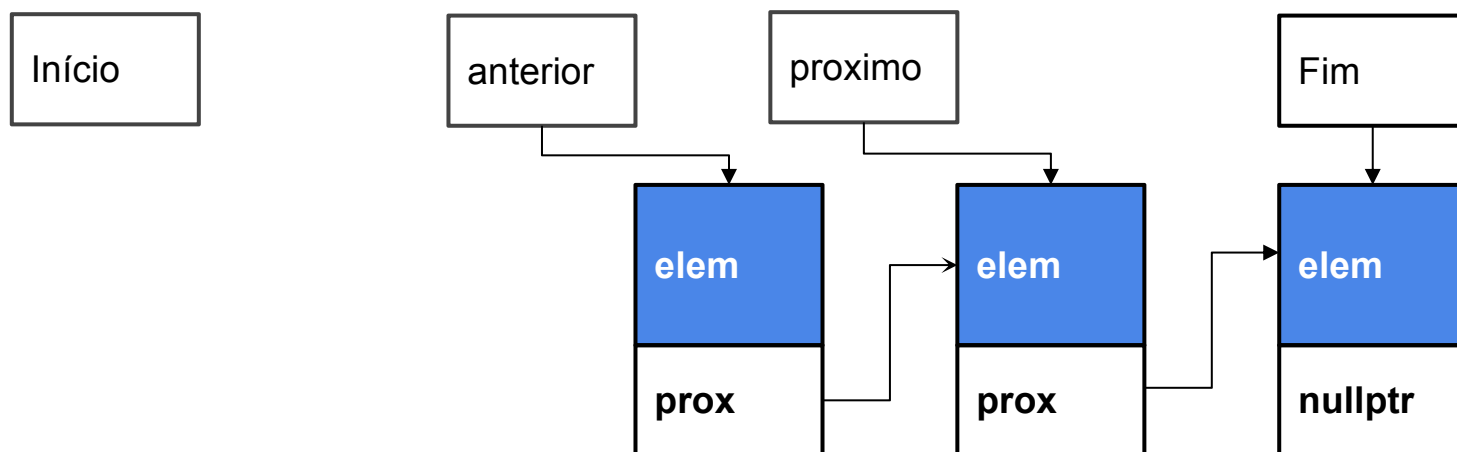
Destruitor

```
ListaSimplesmenteEncadeada::~ListaSimplesmenteEncadeada() {  
    node_t *anterior = nullptr;  
    node_t *proximo = this->_inicio;  
    while (proximo != nullptr) {  
        anterior = proximo;  
        proximo = proximo->proximo;  
        delete anterior;  
    }  
}
```



Destruitor

```
ListaSimplesmenteEncadeada::~ListaSimplesmenteEncadeada() {  
    node_t *anterior = nullptr;  
    node_t *proximo = this->_inicio;  
    while (proximo != nullptr) {  
        anterior = proximo;  
        proximo = proximo->proximo;  
        delete anterior;  
    }  
}
```



Destrutor

```
ListaSimplesmenteEncadeada::~ListaSimplesmenteEncadeada() {  
    node_t *anterior = nullptr;  
    node_t *proximo = this->_inicio;  
    while (proximo != nullptr) {  
        anterior = proximo;  
        proximo = proximo->proximo;  
        delete anterior;  
    }  
}
```

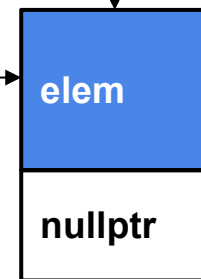


Início

anterior

proximo

Fim



Destrutor

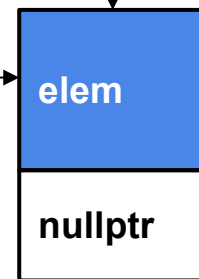
```
ListaSimplesmenteEncadeada::~ListaSimplesmenteEncadeada() {  
    node_t *anterior = nullptr;  
    node_t *proximo = this->_inicio;  
    while (proximo != nullptr) {  
        anterior = proximo;  
        proximo = proximo->proximo;  
        delete anterior;  
    }  
}
```

Início

anterior

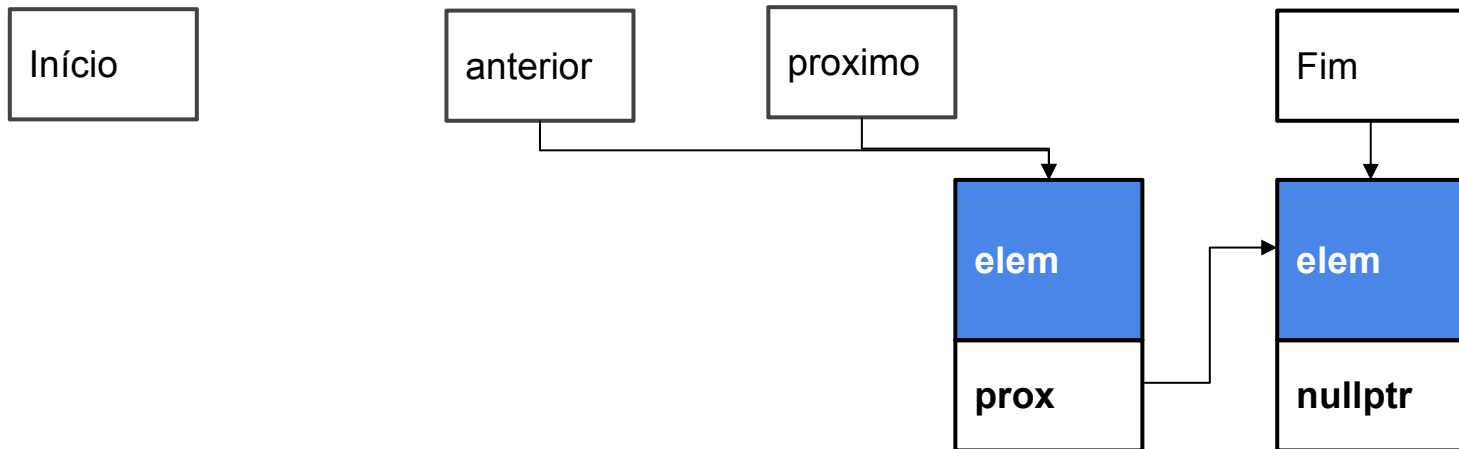
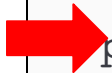
proximo

Fim



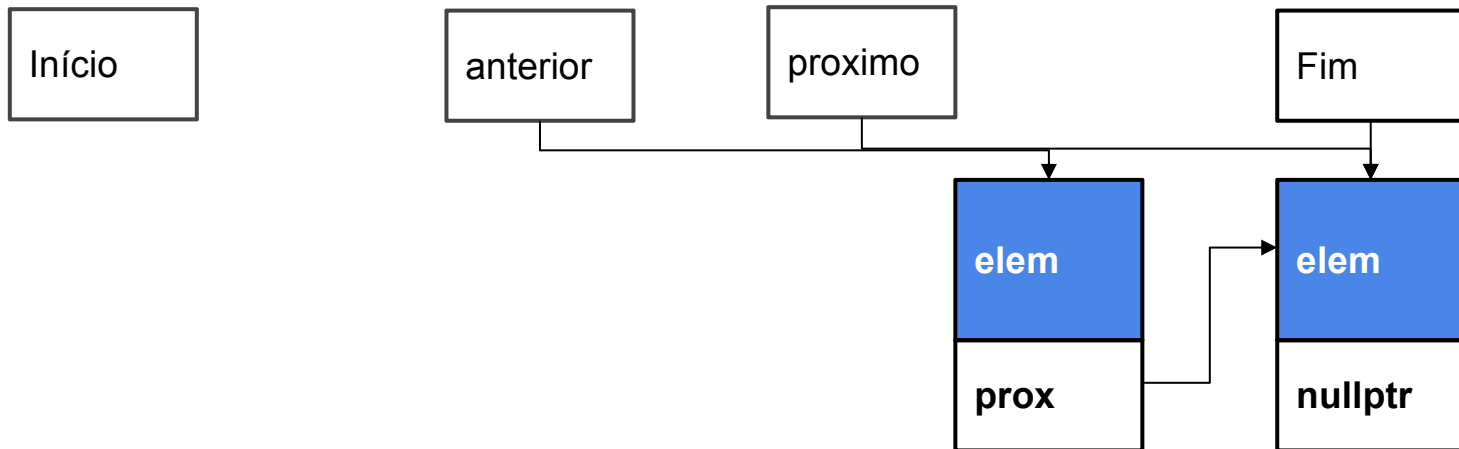
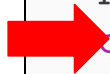
Destrutor

```
ListaSimplesmenteEncadeada::~ListaSimplesmenteEncadeada() {  
    node_t *anterior = nullptr;  
    node_t *proximo = this->_inicio;  
    while (proximo != nullptr) {  
        anterior = proximo;  
        proximo = proximo->proximo;  
        delete anterior;  
    }  
}
```



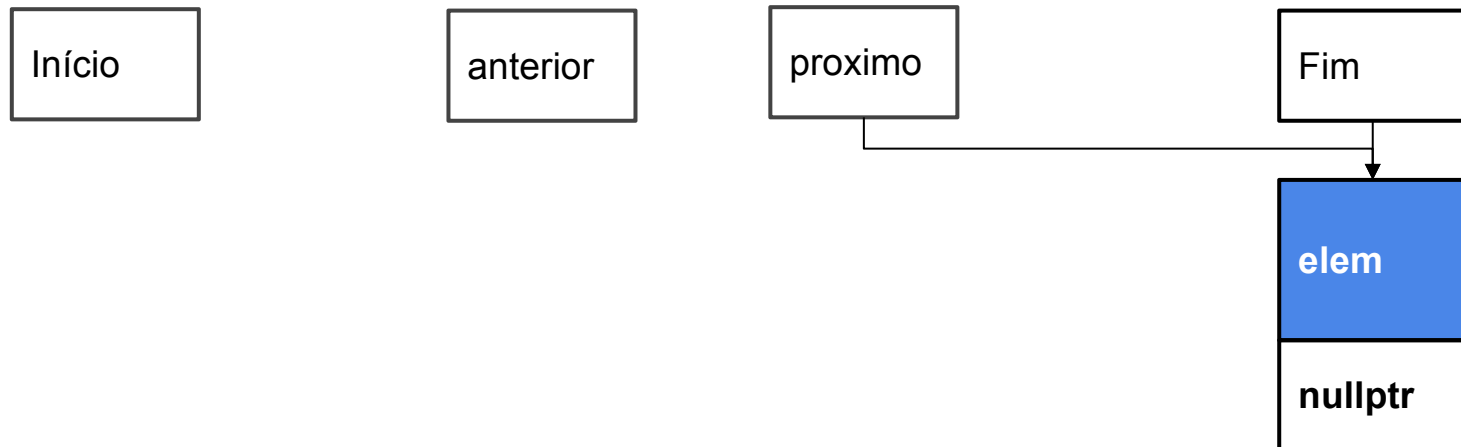
Destrutor

```
ListaSimplesmenteEncadeada::~ListaSimplesmenteEncadeada() {  
    node_t *anterior = nullptr;  
    node_t *proximo = this->_inicio;  
    while (proximo != nullptr) {  
        anterior = proximo;  
        proximo = proximo->proximo;  
        delete anterior;  
    }  
}
```



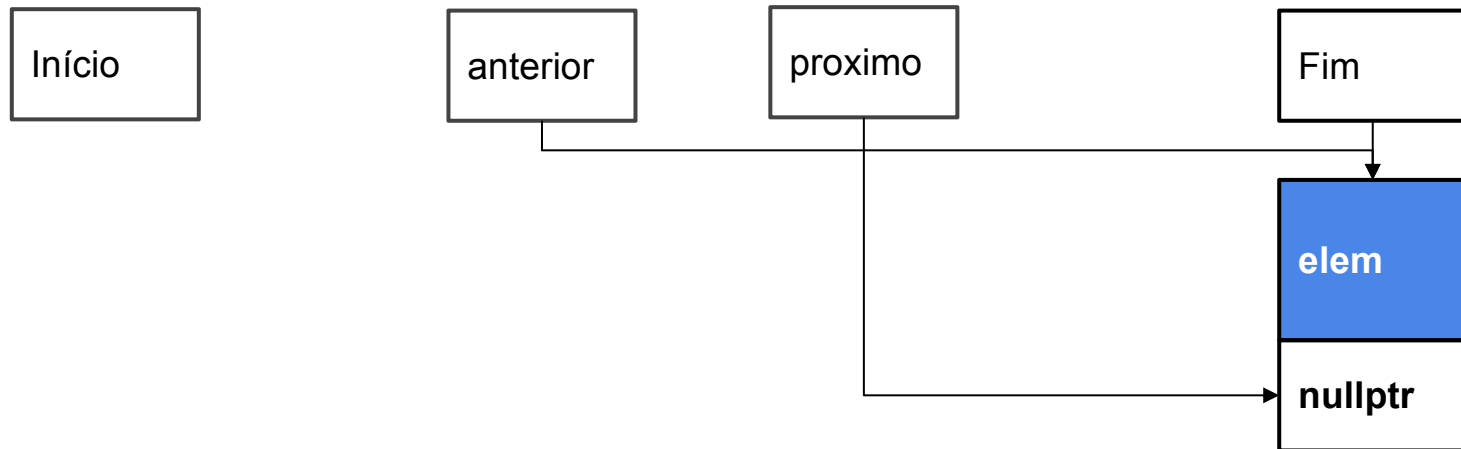
Destrutor

```
ListaSimplesmenteEncadeada::~ListaSimplesmenteEncadeada() {  
    node_t *anterior = nullptr;  
    node_t *proximo = this->_inicio;  
    while (proximo != nullptr) {  
        anterior = proximo;  
        proximo = proximo->proximo;  
        delete anterior;  
    }  
}
```



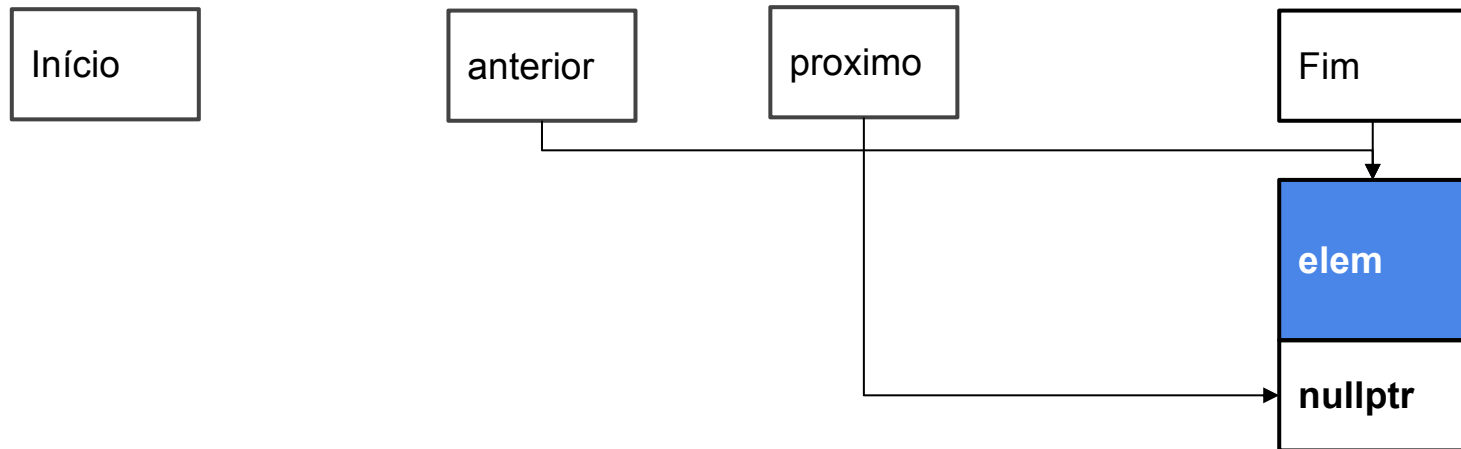
Neste momento próximo é null

```
ListaSimplesmenteEncadeada::~ListaSimplesmenteEncadeada() {  
    node_t *anterior = nullptr;  
    node_t *proximo = this->_inicio;  
    while (proximo != nullptr) {  
        anterior = proximo;  
        proximo = proximo->proximo;  
        delete anterior;  
    }  
}
```



Vai terminar o laço

```
ListaSimplesmenteEncadeada::~ListaSimplesmenteEncadeada() {  
    node_t *anterior = nullptr;  
    node_t *proximo = this->_inicio;  
    while (proximo != nullptr) {  
        anterior = proximo;  
        proximo = proximo->proximo;  
        delete anterior;  
    }  
}
```



Vai terminar o laço

```
ListaSimplesmenteEncadeada::~ListaSimplesmenteEncadeada() {  
    node_t *anterior = nullptr;  
    node_t *proximo = this->_inicio;  
    while (proximo != nullptr) {  
        anterior = proximo;  
        proximo = proximo->proximo;  
        delete anterior;  
    }  
}
```

Início

anterior

proximo

Fim

Fora do laço, ficam os campos sem **new**

```
ListaSimplesmenteEncadeada::~ListaSimplesmenteEncadeada() {  
    node_t *anterior = nullptr;  
    node_t *proximo = this->_inicio;  
    while (proximo != nullptr) {  
        anterior = proximo;  
        proximo = proximo->proximo;  
        delete anterior;  
    }  
}
```



Início


anterior

proximo

Fim

Morrem com o objeto.

```
ListaSimplesmenteEncadeada::~ListaSimplesmenteEncadeada() {  
    node_t *anterior = nullptr;  
    node_t *proximo = this->_inicio;  
    while (proximo != nullptr) {  
        anterior = proximo;  
        proximo = proximo->proximo;  
        delete anterior;  
    }  
}
```



Ideia da aula

- Temos duas implementações de um mesmo TAD
- Uma com vetor, outra com ponteiros
- Podemos implementar na lista encadeada
 - Remover i -ésimo elemento
 - Imprimir
 - . . .

TADs

- Os dois TADs listas suportam
 - inserção no início/fim
 - remoção no início/fim
 - imprimir elementos
- Apenas a lista encadeada (ponteiros)
 - remover o i -ésimo
 - seria possível fazer no vetor, mas chato

Hands on

- Remover i -ésimo elemento
- Imprimir

Futuro . . .

- Dois TADs com métodos iguais
 - Semanticamente
- No futuro vamos usar interfaces
 - Polimorfismo
 - Os métodos em comum viram um tipo único
- Podemos usar templates também
 - Listas de qualquer coisa
 - Ver no git (aulas futuras)