BCC202 - Estruturas de Dados I

Aula 9: Listas (Parte I)

Pedro Silva

Universidade Federal de Ouro Preto, UFOP
Departamento de Computação, DECOM
Email: **silvap@edu.ufop.br**

2021

Conteúdo

Conteúdo

Listas

- Tipo Abstrato de Dado (TAD) para representar um conjunto de dados.
- Pode ser implementada de duas formas principais:
  - **a)** Utilização de *vetor*.
  - **b)** Utilização ponteiros (**lista encadeada)**.
- Dados podem ser **acessados**, **inseridos**, **localizados** ou **retirados**.

Listas

Outras Operações

- **Concatenação** de duas ou mais listas.
- **Partição** em duas ou mais listas.

Listas: Vetor vs Ponteiros

- Vantagens.
- Desvantagens.

Listas: Definição

- **Sequência de zero ou mais itens**.
  - $x_0, x_1, x_2, ..., x_{n-1}$, na qual x_i é de um determinado tipo e n representa o tamanho da lista.
- **Sua principal propriedade estrutural envolve as posições relativas dos itens em uma dimensão**.
  - Assumindo $n >= 0$, x_0 é o **primeiro** item da lista e x_{n-1} é o **último** item da lista.
    - x_i **precede** x_{i+1} para $i = 0, 1, 2, ..., n-2$.
    - x_i **sucede** x_{i-1} para $i = 1, 2, 3, ..., n-1$.
    - O elemento x_i é dito estar na i-ésima posição da lista.
    - Isto não implica que os dados estejam ordenados, estes conceitos remetem apenas à **posição** na lista.
  - A lista é **linear**.

Conteúdo

TAD Lista

O que o TAD Lista deveria conter?

- Representação do **tipo** da lista.
- Conjunto de **operações** que atuam sobre a lista.

Quais operações deveriam fazer parte da lista?

- **Depende de cada aplicação**.
- Mas, um conjunto *padrão* pode ser definido.

Conjunto Padrão de operações

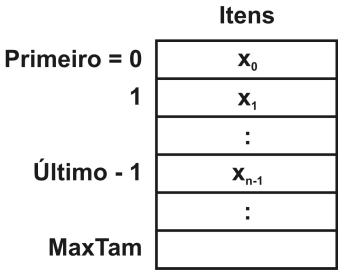
- Operações necessárias à grande maioria das aplicações:
  - **Criar** uma lista linear vazia.
  - **Inserir** um novo item.
  - **Retirar** um item.
  - **Localizar** um item para examinar ou alterar seu conteúdo.
  - **Combinar** duas ou mais listas em uma única lista.
  - **Dividir** uma lista em duas ou mais listas.
  - Fazer uma **cópia** da lista.
  - **Ordenar** os itens da lista de acordo com um de seus campos.
  - **Pesquisar** a ocorrência de um item com um valor específico.

Implementação de uma Lista

- Várias estruturas de dados podem ser usadas para representar listas lineares, cada uma com vantagens e desvantagens particulares.
- As duas representações mais utilizadas são:
  - Implementação por **arrays**.
  - Implementação por **ponteiros**.

Conteúdo

TAD Lista: Implementação por ARRAY

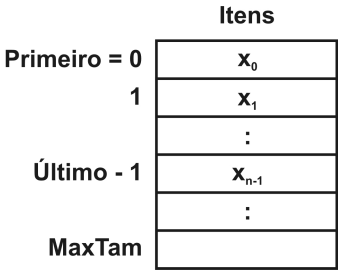


- Os itens são armazenados em posições contíguas de memória.
- A lista pode ser percorrida em qualquer direção.

TAD Lista: Implementação por ARRAY

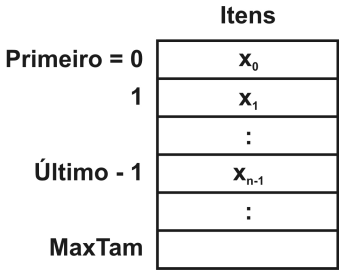
- A **inserção** de um novo item no **final** tem **Custo** $O(1)$.
- A **inserção** de um novo item no meio tem **Custo** $O(n)$.
- A **remoção** de um item no meio tem **Custo** $O(n)$.
- Os dois últimos requerem deslocamentos.

TAD Lista: Implementação por ARRAY



- Os itens são armazenados em um array de tamanho suficiente para comportar todos os elementos da lista.
- O campo `Ultimo` aponta para a **posição seguinte** à do último elemento da lista.

TAD Lista: Implementação por ARRAY



- O *i-ésimo* item da lista está armazenado na *(i-1)-ésima* posição do array, $0 <= i < Ultimo$.
- A constante `MaxTam` define o tamanho máximo permitido para a lista.

TAD Lista: Implementação por ARRAY: Código Exemplo (.h)

```
1
2  #define INICIO 0
3  #define MAXTAM 1000
4
5  typedef struct{
6    int chave; /* outros campos */
7  } TItem;
8
9  typedef struct{
10   TItem item[MAXTAM];
11   int primeiro, ultimo;
12 } TLista;
13
14 ...
```

TAD Lista: Implementação por ARRAY: Código Exemplo (.h)

```
15 ...
16
17 /* procedimentos e funcoes da TAD */
18 void TLista_FazVazia(TLista *pLista);
19
20 int TLista_EhVazia(TLista *pLista);
21
22 int TLista_Insere(TLista *pLista, TItem x);
23
24 int TLista_Retira(TLista *pLista, int p, TItem *pX);
25
26 void TLista_Imprime(TLista *pLista);
27
28 int TLista_Get(TLista *pLista, int p, TItem *pX);
29
30 int TLista_Tamanho(TLista *pLista);
```

TAD Lista: Implementação por ARRAY: Código Exemplo (.c)

```c
#include "lista_vetor.h"

void TLista_FazVazia(TLista *pLista) {
    pLista->primeiro = INICIO;
    pLista->ultimo = pLista->primeiro;
}

int TLista_EhVazia(Tlista *pLista) {
    return (pLista->ultimo == pLista->primeiro);
}

int TLista_Insere(Tlista *pLista, TItem x) {
    if (pLista->ultimo == MaxTam)
        return 0; /* lista cheia */
    pLista->item[pLista->ultimo++] = x;
    return 1;
}

...
```

TAD Lista: Implementação por ARRAY: Código Exemplo (.c)

```
21 ...
22
23 int TLista_Retira(Tlista *pLista, int p, TItem *pX) {
24     if (LEhVazia(pLista) || p >= pLista->ultimo)
25         return 0;
26
27     int cont;
28     *pX = pLista->Item[p];
29     pLista->ultimo--;
30     for (cont = p+1; cont <= pLista->ultimo; cont++)
31         pLista->item[cont - 1] = pLista->item[cont];
32     return 1;
33 }
34
35 void TLista_Imprime(Tlista *pLista) {
36     int i;
37     for (i = pLista->primeiro; i < pLista->ultimo; i++)
38         printf("%d\n", pLista->item[i].chave);
39 }
```

TAD Lista: Implementação por ARRAY

Vantagens

- Economia de memória (apontadores implícitos).
- Acesso a qualquer elemento da lista é feito em tempo $O(1)$.

Desvantagens

- Custo para inserir itens da lista pode ser $O(n)$.
- Custo para retirar itens da lista pode ser $O(n)$.
- Quando não existe previsão sobre o crescimento da lista, o array que define a lista deve ser alocado de forma dinâmica.
- Pode ser necessária a realocação do array.

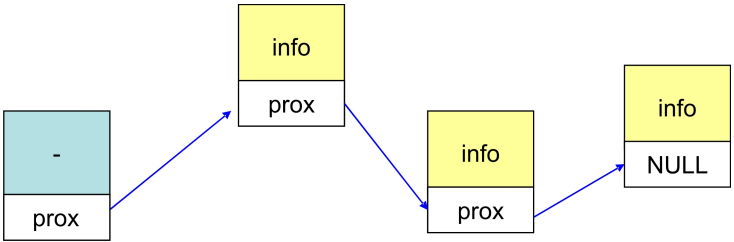
Conteúdo

TAD Lista: Implementação por PONTEIRO

- Qual o principal problema de utilizar array para implementar listas?
- E se a lista aumentar e depois diminuir drasticamente de tamanho?
- **Solução: Listas Encadeadas**!!!.
- O que é?
  - Implementação de uma lista utilizando apenas **ponteiros**.

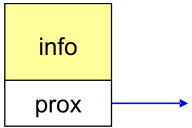
TAD Lista: Lista Encadeada

- Características:
  - Tamanho da lista não é pré-definido.
  - Cada elemento aponta para o próximo.
  - Elementos não estão contíguos na memória (a alocação é dinâmica).



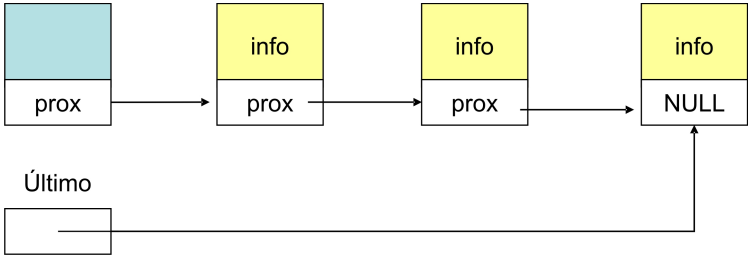
TAD Lista: Lista Encadeada: Elementos

- **Elemento da lista**:
  - Armazena as informações sobre cada elemento.
  - Aponta para o próximo elemento.
- Assim, é definido como uma estrutura que possui:
  - Campos representando as informações do elemento.
  - Ponteiro para o próximo elemento.



TAD Lista: Lista Encadeada: Elementos

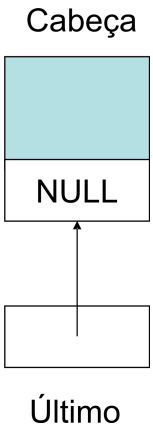
- Uma lista pode ter uma célula **cabeça**, antecedendo o primeiro elemento.
- Pode possuir também um apontador para o **último** elemento.



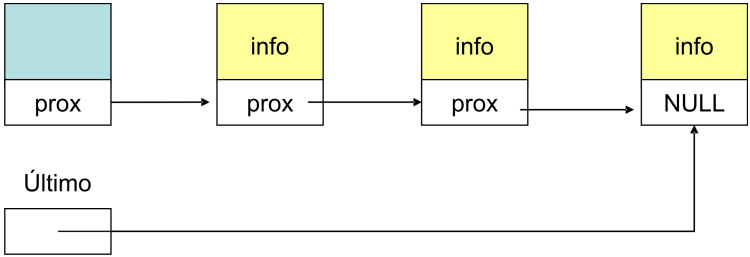
Lista Encadeada COM cabeça: Estruturas básicas

```
1  typedef struct {
2    /* Componentes de um item: "info" */
3  } TItem;
4
5  typedef struct celula {
6    struct celula *pProx;
7    TItem item;
8  } TCelula;
9
10 typedef struct {
11   TCelula *pPrimeiro, *pUltimo;
12 } TLista;
```

TAD Lista: Lista Encadeada: Criar Lista Vazia

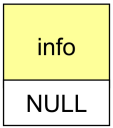


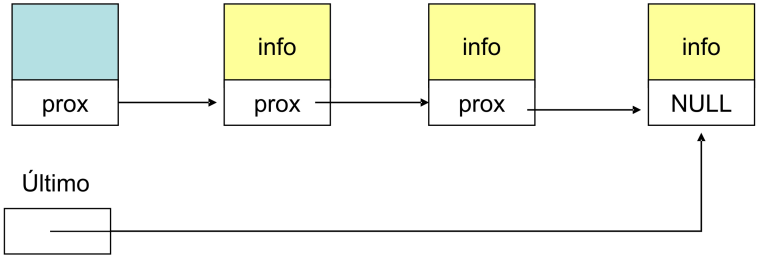
TAD Lista: Lista Encadeada: INSERÇÃO de Novos Elementos



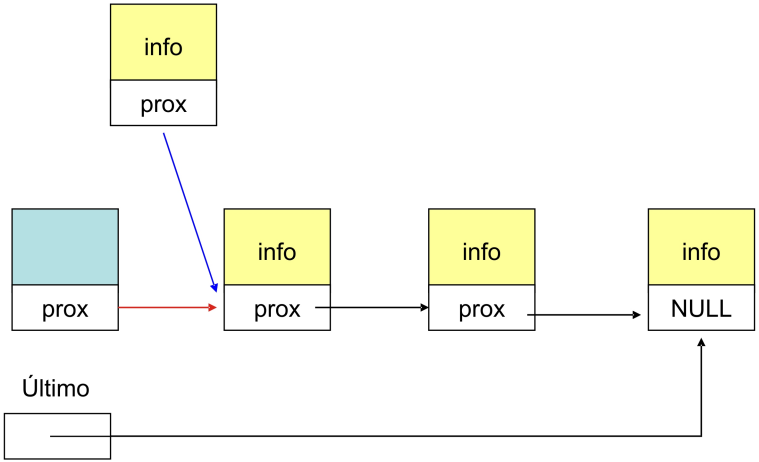
- 3 opções de posição onde se pode **inserir**:
  - Primeira posição.
  - Última posição.
  - Após um elemento `E`.

TAD Lista: Lista Encadeada: INSERÇÃO na Primeira Posição

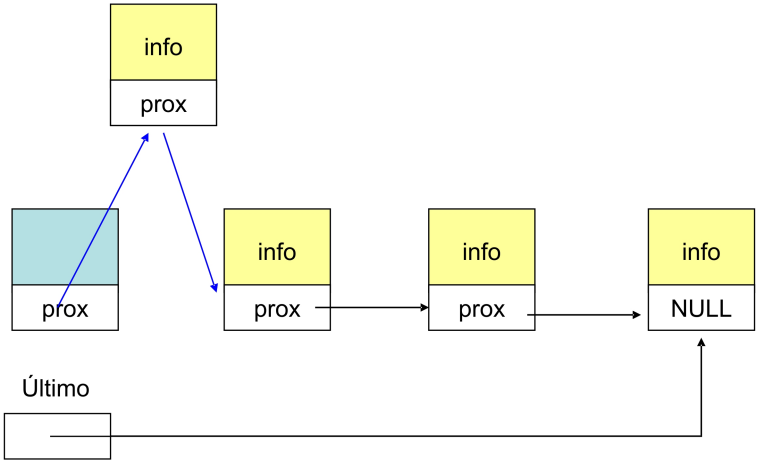




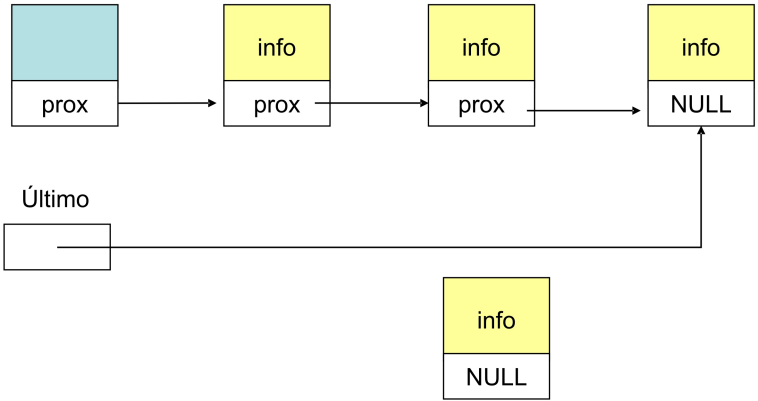
TAD Lista: Lista Encadeada: INSERÇÃO na Primeira Posição



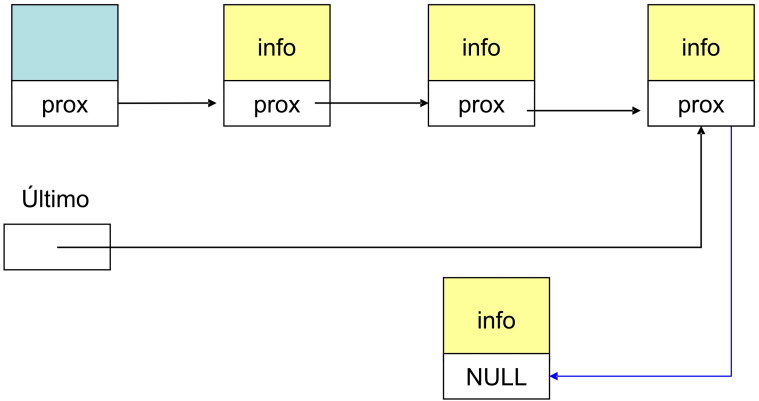
TAD Lista: Lista Encadeada: INSERÇÃO na Primeira Posição



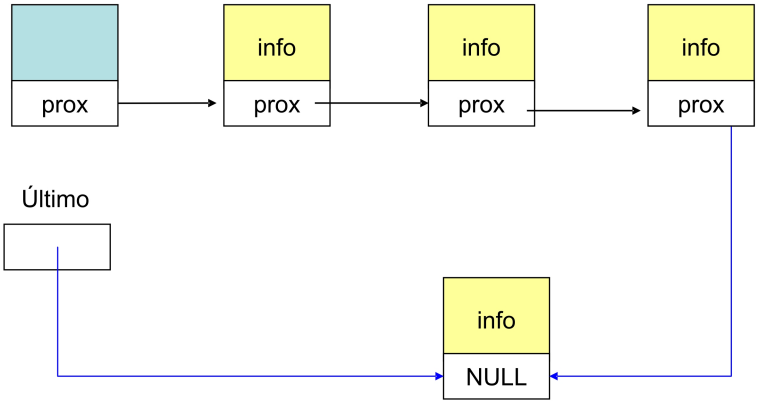
TAD Lista: Lista Encadeada: INSERÇÃO na Última Posição



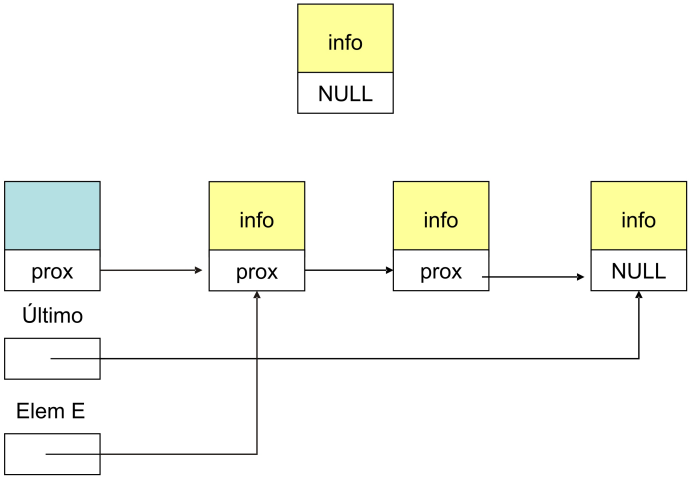
TAD Lista: Lista Encadeada: INSERÇÃO na Última Posição



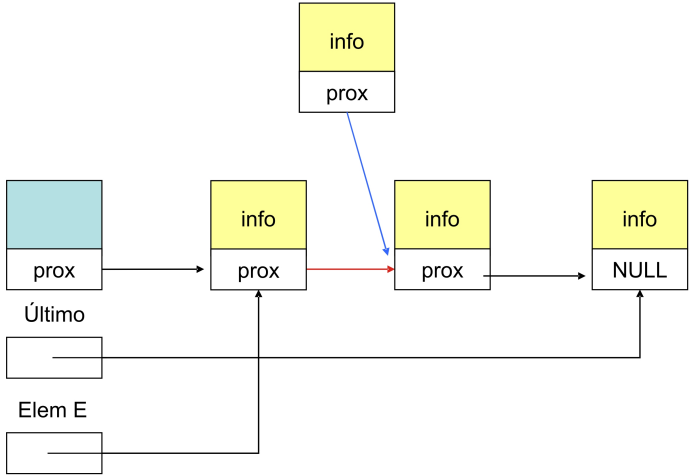
TAD Lista: Lista Encadeada: INSERÇÃO na Última Posição



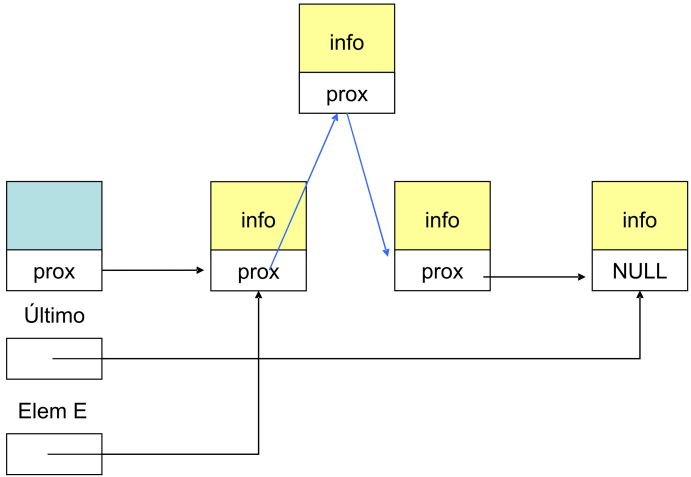
TAD Lista: Lista Encadeada: INSERÇÃO Após o Elemento E



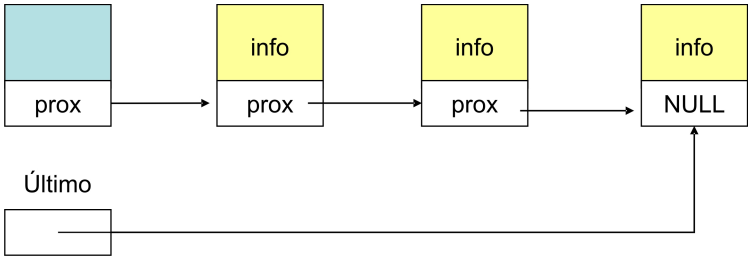
TAD Lista: Lista Encadeada: INSERÇÃO Após o Elemento E



TAD Lista: Lista Encadeada: INSERÇÃO Após o Elemento E

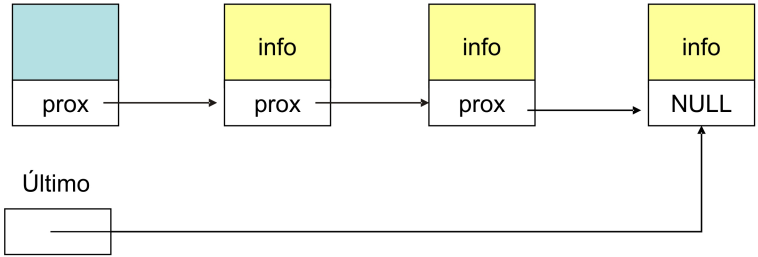


TAD Lista: Lista Encadeada: RETIRADA de Elementos

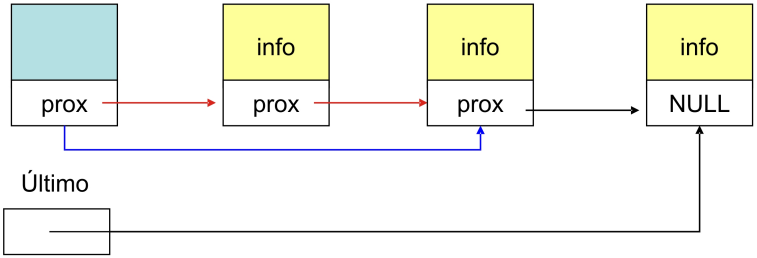


- 3 opções de posição onde se pode **retirar**:
  - Primeira posição.
  - Última posição.
  - Um elemento E.

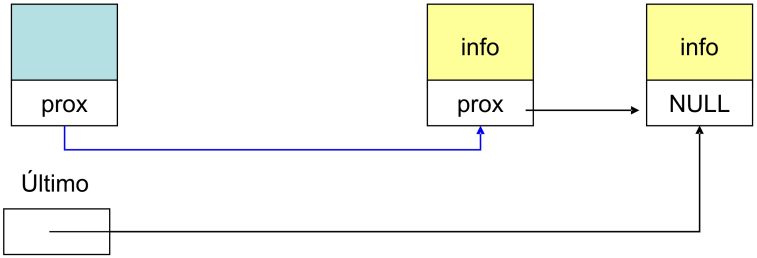
TAD Lista: Lista Encadeada: RETIRADA na Primeira Posição



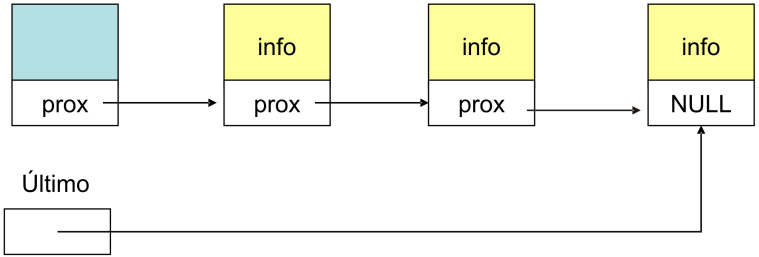
TAD Lista: Lista Encadeada: RETIRADA na Primeira Posição



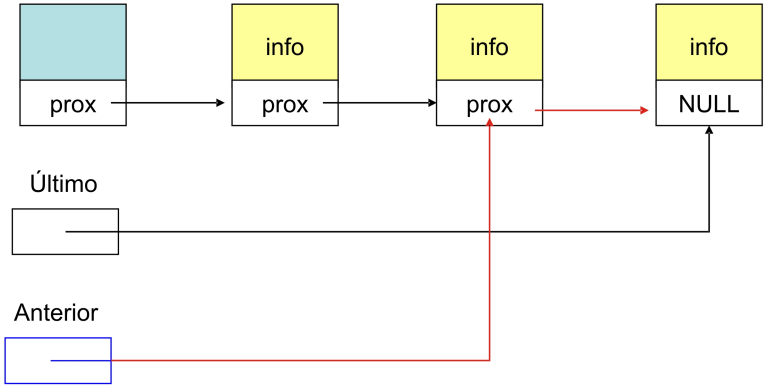
TAD Lista: Lista Encadeada: RETIRADA na Primeira Posição



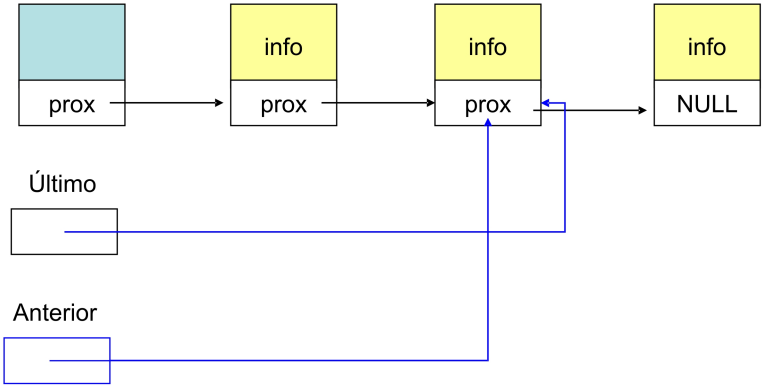
TAD Lista: Lista Encadeada: RETIRADA na Última Posição



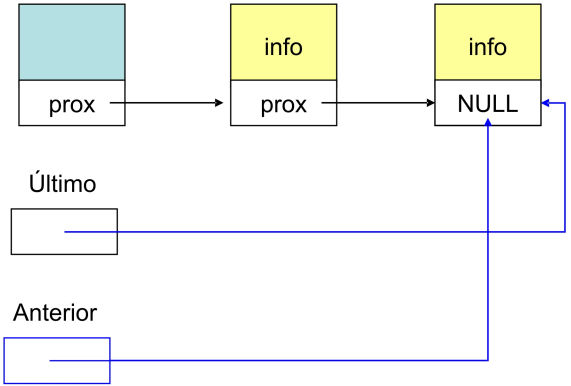
TAD Lista: Lista Encadeada: RETIRADA na Última Posição



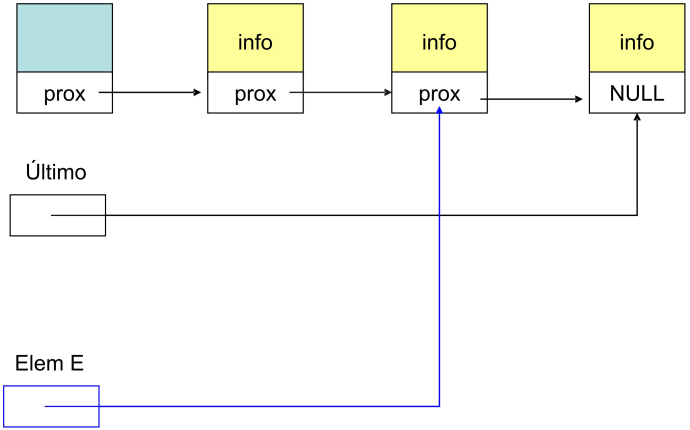
TAD Lista: Lista Encadeada: RETIRADA na Última Posição



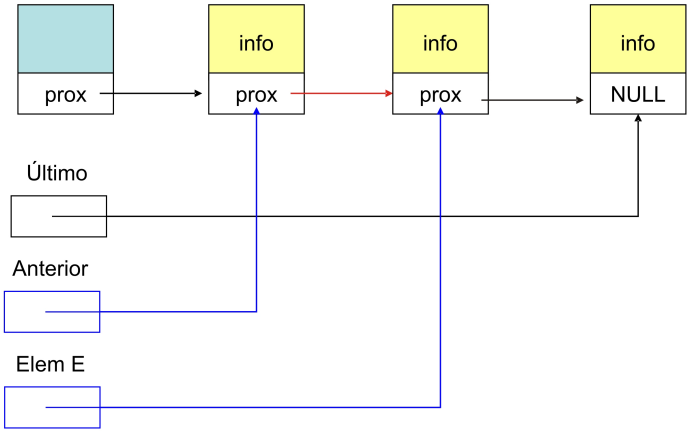
TAD Lista: Lista Encadeada: RETIRADA na Última Posição



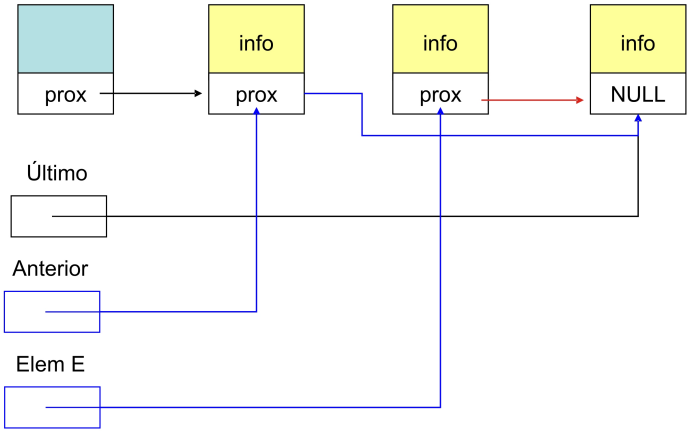
TAD Lista: Lista Encadeada: RETIRADA do Elemento E



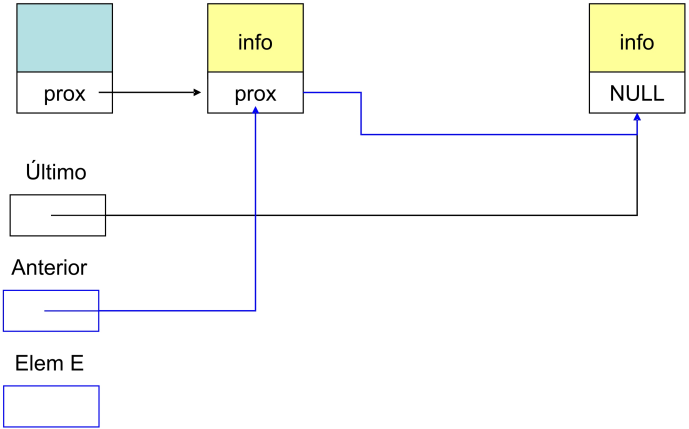
TAD Lista: Lista Encadeada: RETIRADA do Elemento E



TAD Lista: Lista Encadeada: RETIRADA do Elemento E



TAD Lista: Lista Encadeada: RETIRADA do Elemento E



Conteúdo

Conclusão

- Nesta aula tivemos o primeiro contato com a estrutura de dados **Lista**.
- Este tópico é muito importante para o entendimento das estruturas de dados que virão a seguir.
- *Próxima aula*: Lista (Parte 2) – Implementação de lista encadeada (por ponteiro).
- Dúvidas?

Conteúdo

Exercício 01

Implemente um TAD Lista utilizando Array, acrescentando as seguintes operações ao conjunto de operações vistos nesta aula:

- Concatenação de duas listas.
- Divisão de uma lista em duas em uma posição i.
- Cópia de uma lista.
- Pesquisa por um elemento na lista.