L I S T A S

→Pilha: inserções e remoções pelo topo, ordenação LIFO.

→Fila: inserção pela cauda, remoção pela frente, ordenação FIFO.

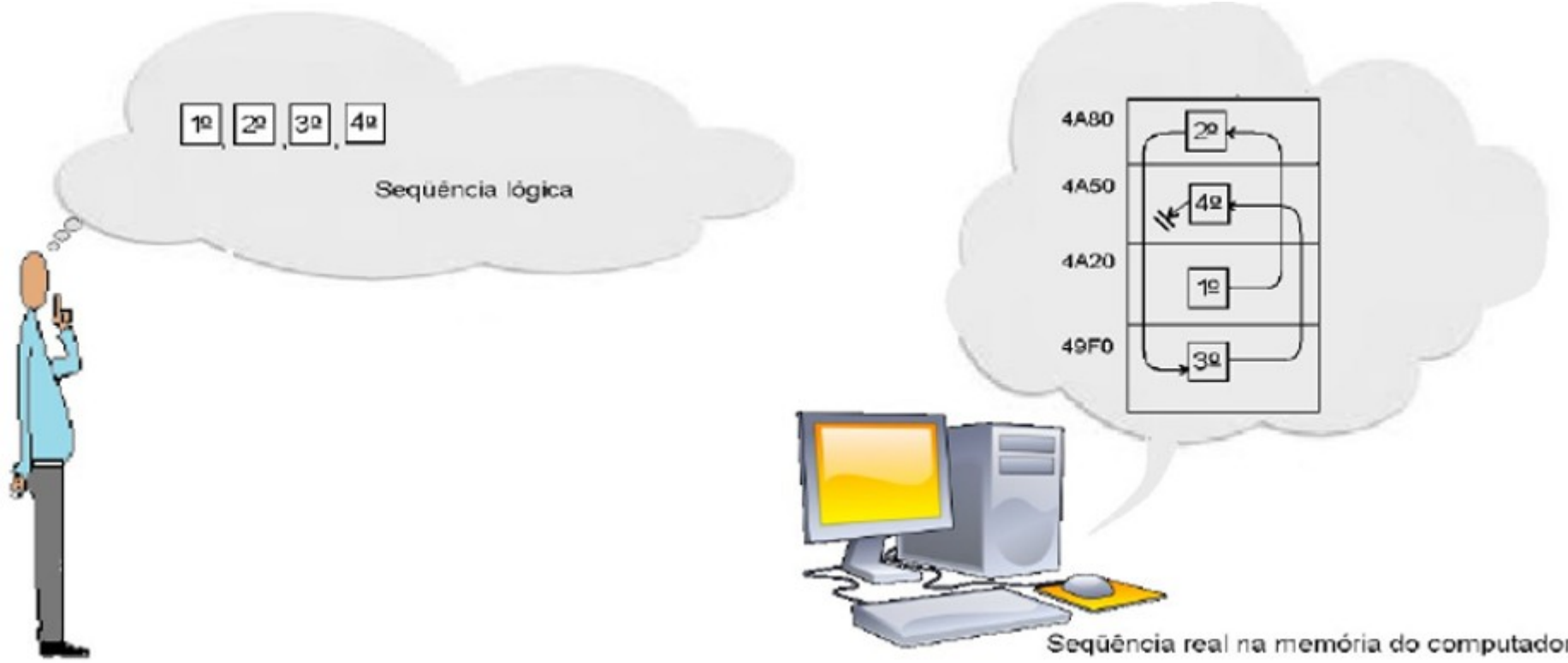
Pilhas e filas são especializações do conceito de listas onde foram feitas restrições quanto à inserção, remoção e pesquisa dos elementos.

Das estruturas até aqui discutidas a lista é a que menos restringe o acesso a seus itens.

- Listas podem ser manipuladas em qualquer ponto da sua sequência de itens.

- Uma lista é simplesmente uma sequência lógica de elementos cuja ordenação – caso exista – é definida na sua aplicação.

• A lista consiste em uma sequência lógica de elementos posicionados;
• O conceito de sequência lógica está associado a uma visão abstrata da lista;
• O programador de aplicação “enxerga” a lista por essa abstração (associada ou não a uma ordenação);
• É possível efetuar operações de busca, remoção ou inserção em qualquer posição dessa sequência lógica.

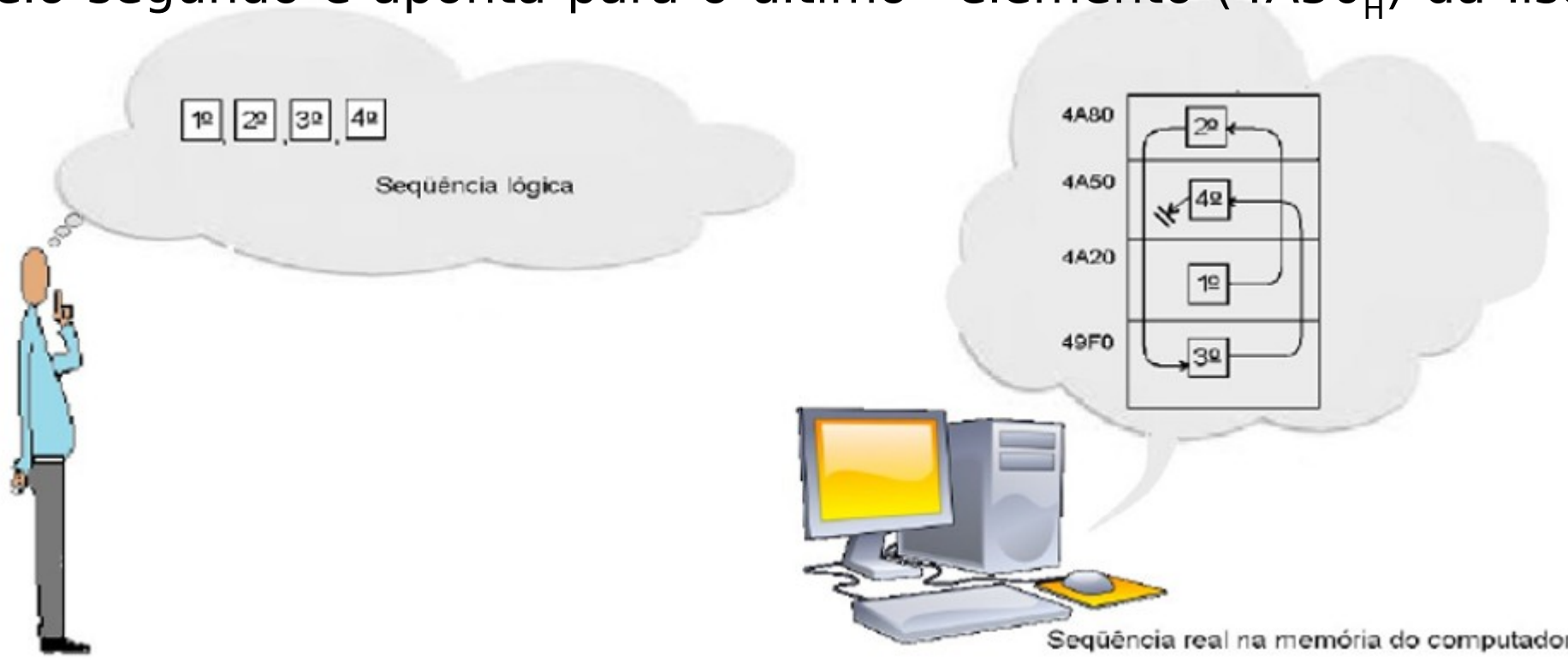


Sequência física não necessariamente corresponde à sequência lógica da lista encadeada.

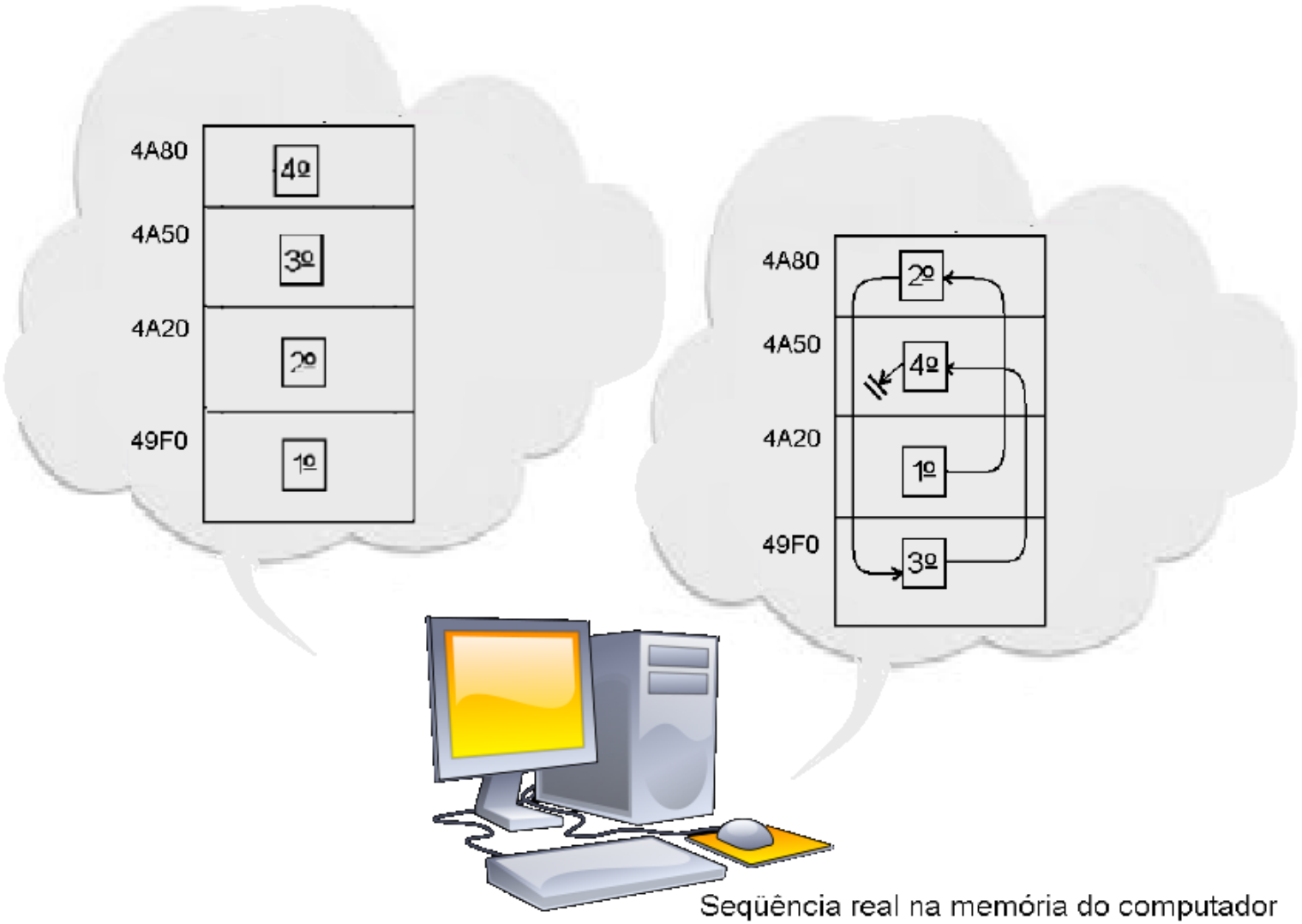
Na figura:

A sequência física obedece a ordem dos endereços físicos em memória, por exemplo: $4AF0_H$, $4A20_H$, $4A50_H$, $4A80_H$

A sequência lógica é determinada pelo encadeamento: o primeiro elemento ($4A20_H$) é apontado por um descritor, o segundo ($4A80_H$) é apontado pelo primeiro, o terceiro ($49F0_H$) é apontado pelo segundo e aponta para o último elemento ($4A50_H$) da lista.



Lista encadeada ou lista não encadeada? Qual a melhor implementação ?

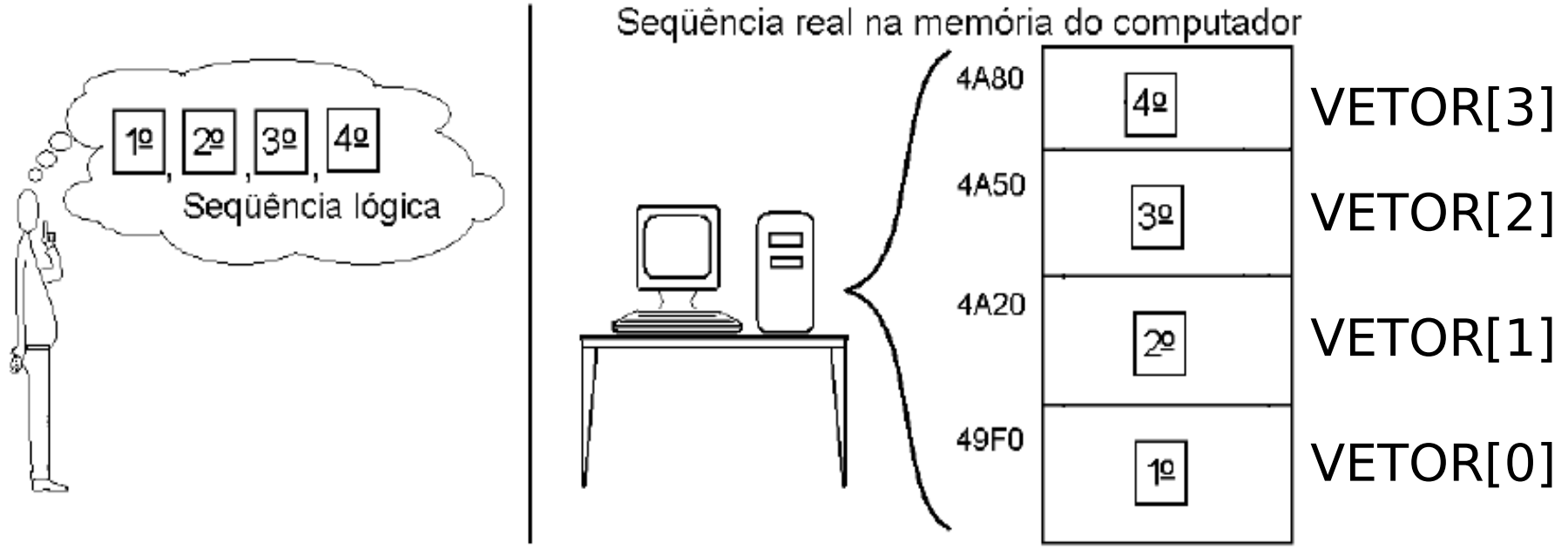


Seqüência real na memória do computador

Lista encadeada ou lista não encadeada? Qual a melhor implementação ?

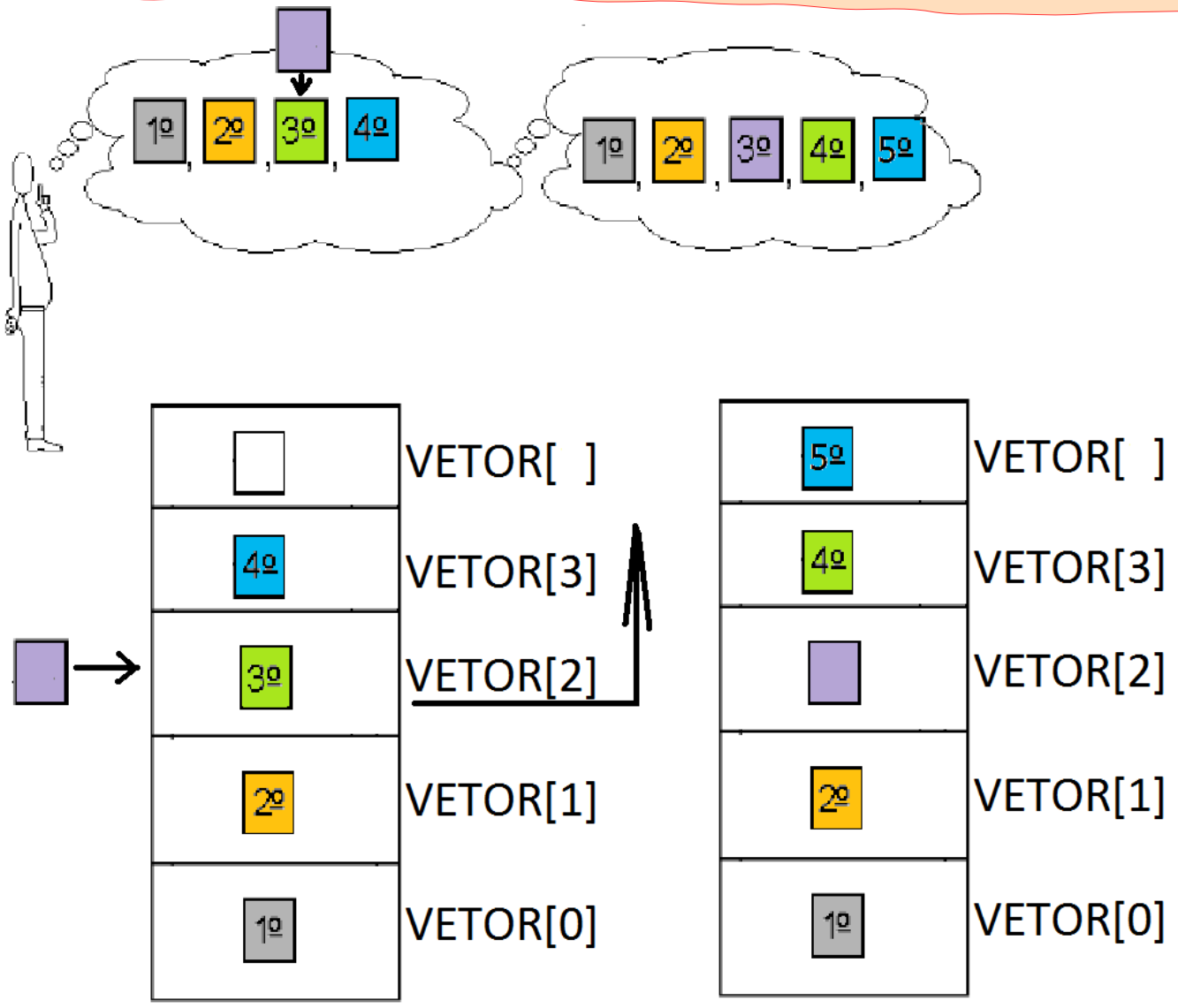
A implementação não encadeada simplesmente utiliza um vetor sem que os nós da lista possuam elos (links) explícitos de ligação para encadeamento com seus vizinhos. Nesse caso, os nós vizinhos na sequência lógica também são vizinhos na sequência física dos endereços de memória.

Exemplo na figura: o 2º elemento tem seu antecessor no endereço imediatamente anterior e seu sucessor no endereço imediatamente posterior ao seu.



Lista encadeada ou lista não encadeada? Qual a melhor implementação ?

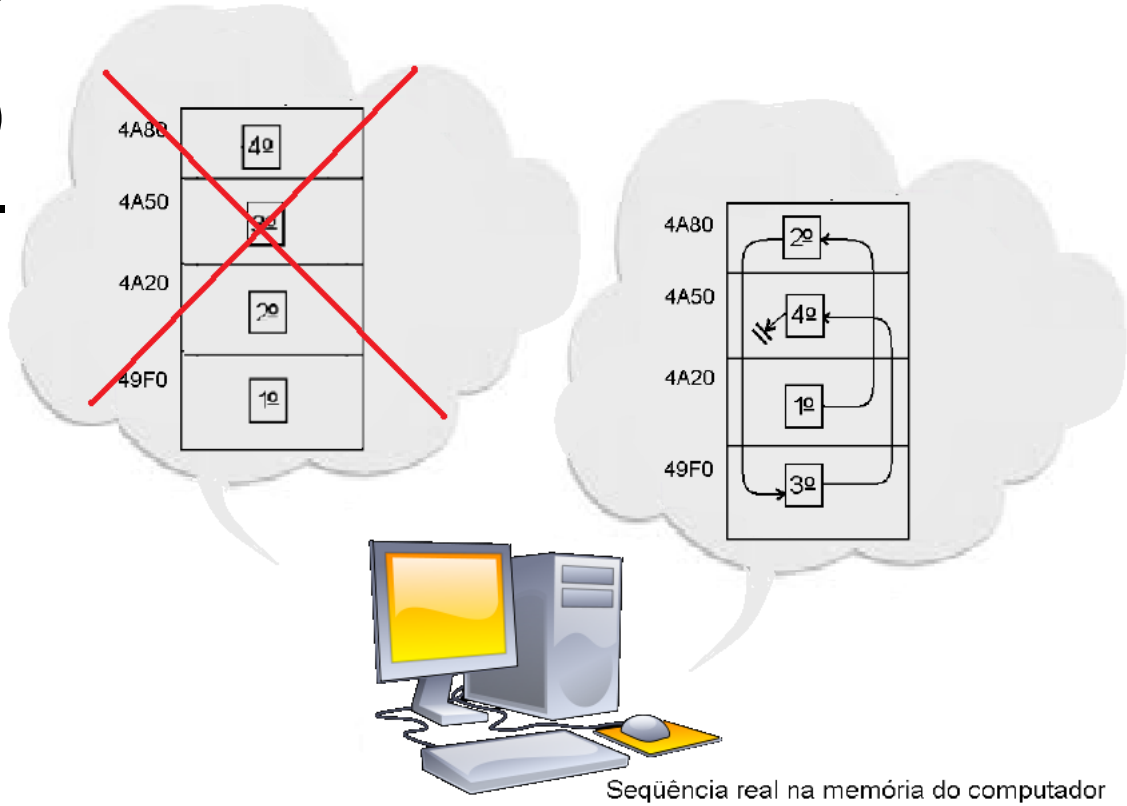
A manipulação da lista não encadeada sobre vetor pode ser muito custosa, pois o gerenciamento de espaços pode implicar em mover dados.



Lista encadeada ou lista não encadeada? Qual a melhor implementação ?

Por questões de eficiência das operações de manipulação estudaremos as listas estáticas e dinâmicas implementadas por encadeamento.

Nesse caso o TDA-Lista implementa a sequência lógica como um encadeamento por meio de um campo de ligação a cada nó. Vizinhos na sequência lógica não necessariamente serão vizinhos na sequência (física) dos endereços de memória.

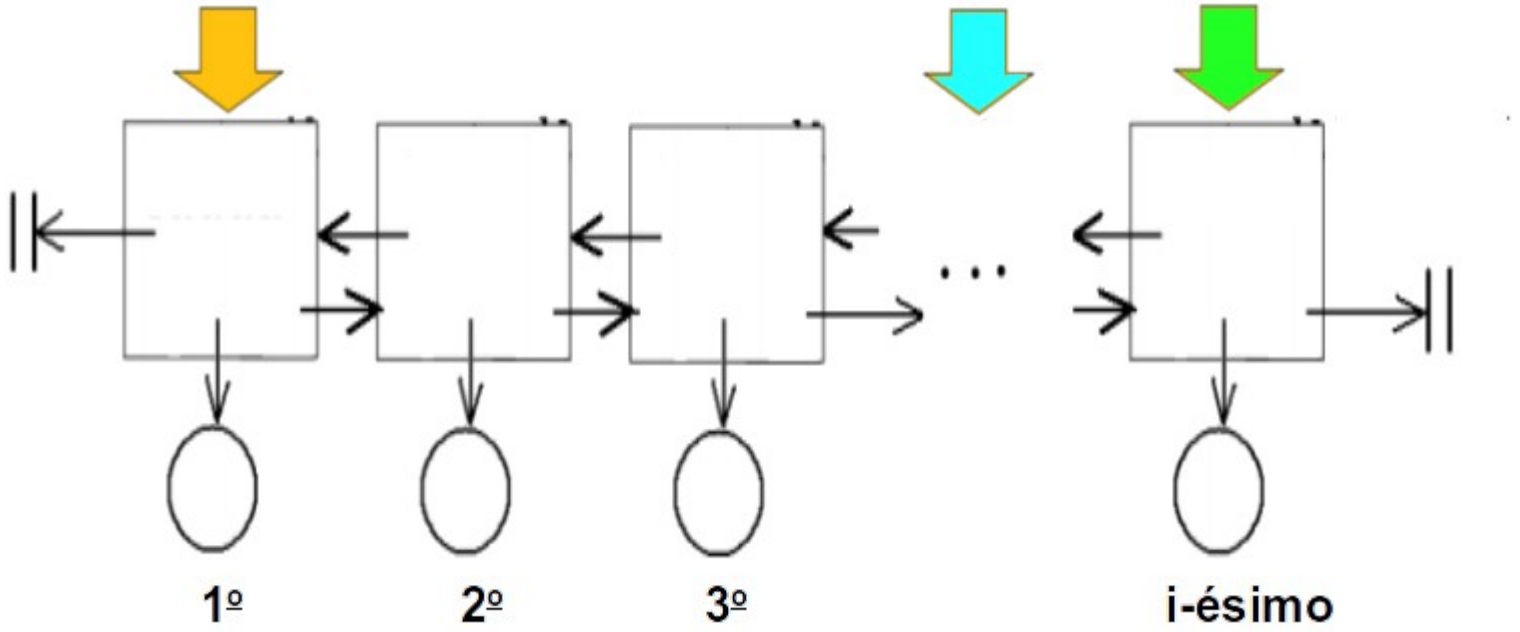


Seqüência real na memória do computador

Interface do TDA – Lista:

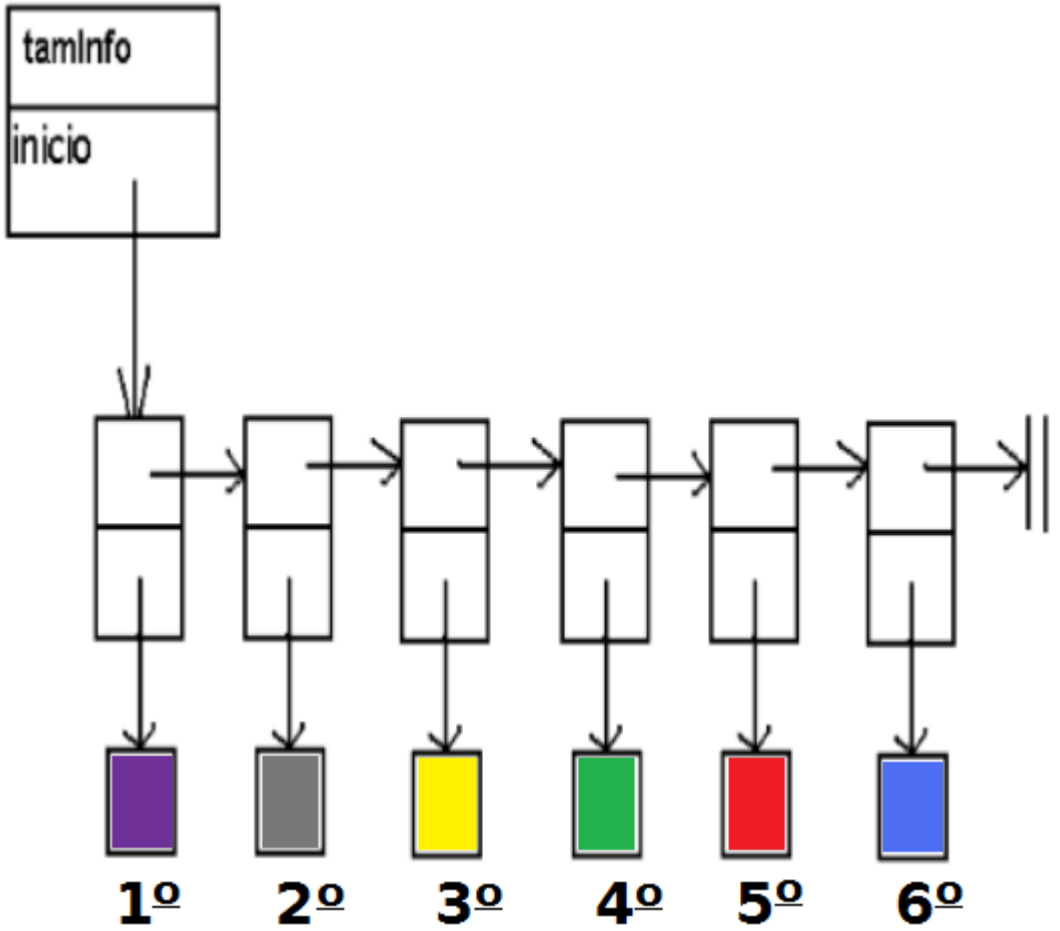
Além das operações usuais de criação, destruição, reinicialização, a lista permite buscas, inserções e remoções em **qualquer ponto** da sua sequência lógica:

buscas, inserções e remoções: inicio, final ou ponto intermediário



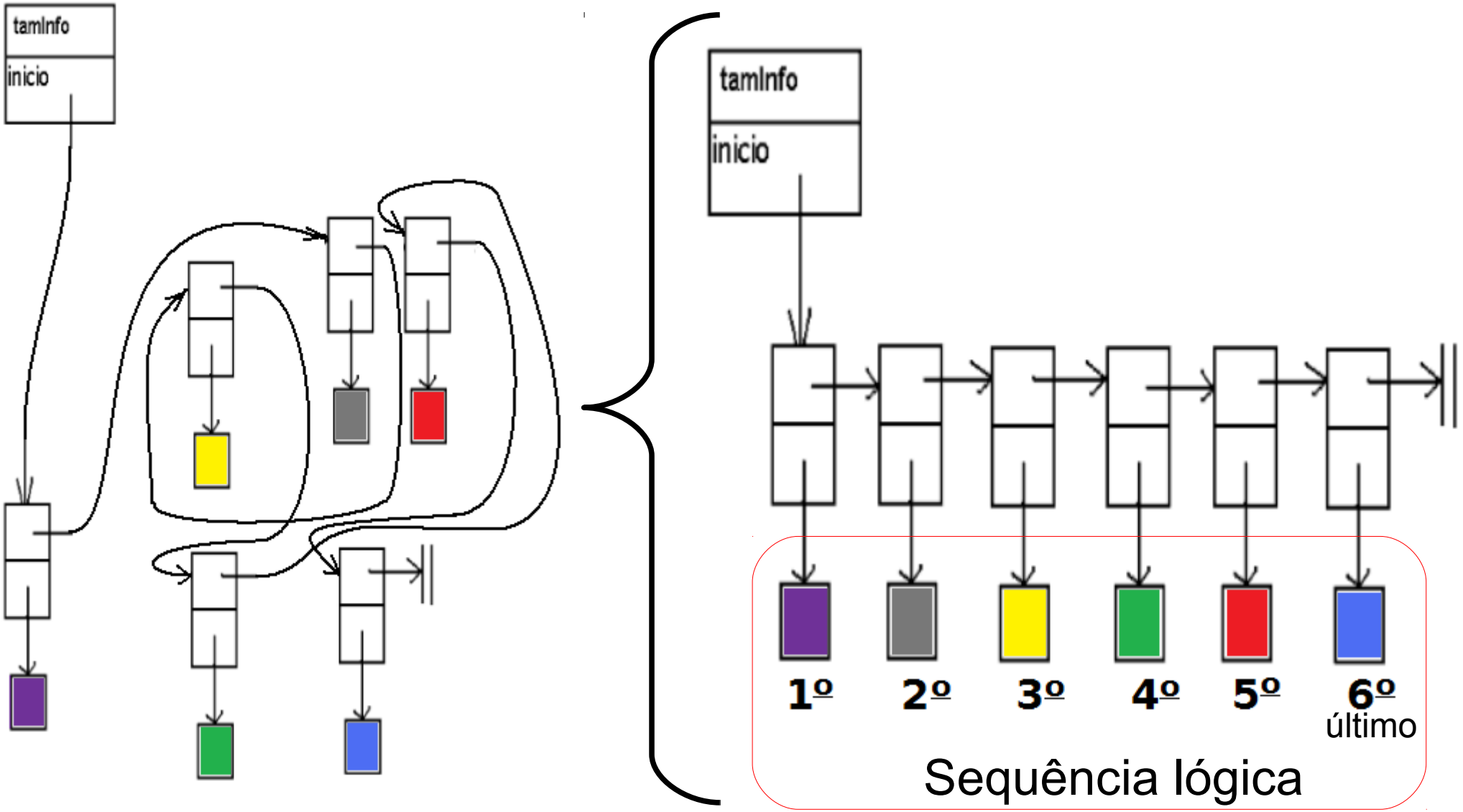
Inicialmente iremos estudar a LDSE:

Lista com encadeamento simples



LDSE Vazia:

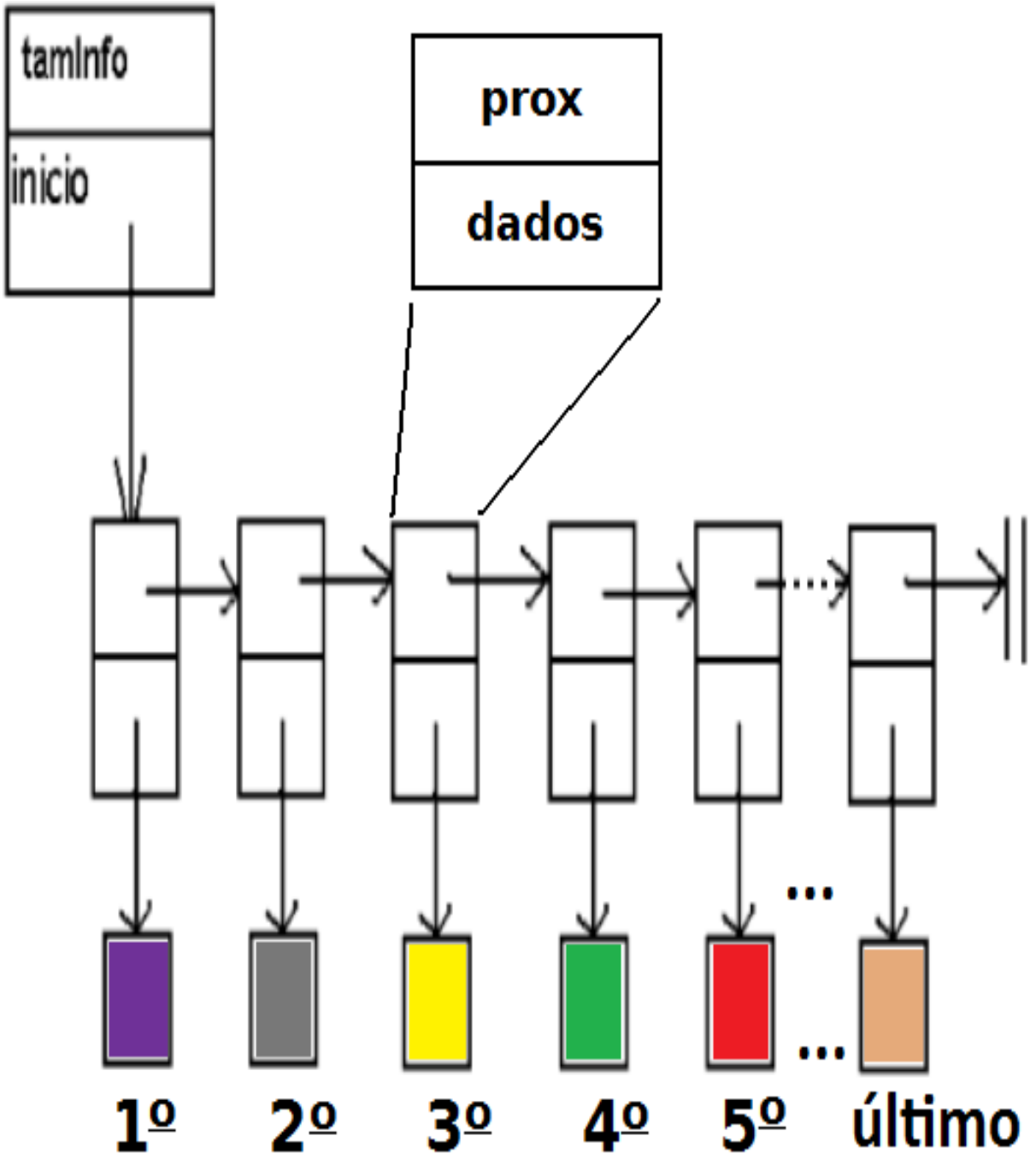
LDSE Não Vazia:



TDA – LDSE

```
#include “stdio.h”
#include “stdlib.h”
...
/* nó de dados */
typedef struct noLDSE{
        void *dados
        struct noLDSE *prox;
} NoLDSE;

/* Descritor */
typedef struct LDSE{
    int tamInfo;
    NoLDSE *inicio;
} LDSE;
```



Interface da LDSE:

a) int cria(ppLDSE pp, int tamInfo);
b) void reinicia(pLDSE p);
c) void destroi(ppLDSE pp);
d) int testaVazia(pLDSE p);
e) int tamanho(pLDSE p);

→ Operações ordinárias presentes em todos os TDAs vistos até aqui

f) int buscaOprimeiro(pLDSE p, void *reg);
g) int insereNovoPrimeiro(pLDSE p, void *novo);
h) int removeOprimeiro(pLDSE p, void *reg);

i) int buscaOultimo(pLDSE p, void *reg);
j) int insereNovoUltimo(pLDSE p, void *novo);
k)int removeOultimo(pLDSE p, void *reg);

→ Operações nas extremidades da lista

l) int buscaNaPosLog(pLDSE p, void *reg, unsigned int posLog);
m) int insereNaPosLog (pLDSE p, void *novo,unsigned int posLog);
n) int removeDaPosLog(pLDSE p, void *reg, unsigned int posLog);

→ **Operações em alguma posição lógica especifica da**

Interface da LDSE:

a) int cria(ppLDSE pp, int tamInfo);
b) void reinicia(pLDSE p);
c) void destroi(ppLDSE pp);
d) int testaVazia(pLDSE p);
e) int tamanho(pLDSE p);

Logicamente idênticas às correspondentes na PDSE.

f) int buscaOprimeiro(pLDSE p, void *reg);
g) int insereNovoPrimeiro(pLDSE p, void *novo);
h) int removeOprimeiro(pLDSE p, void *reg);

Logicamente idênticas às correspondentes na PDSE.

i) int buscaOultimo(pLDSE p, void *reg);
j) int insereNovoUltimo(pLDSE p, void *novo);

Logicamente idênticas às correspondentes de busca na cauda e inserção na FDSE da sétima questão na lista de exercícios sobre filas

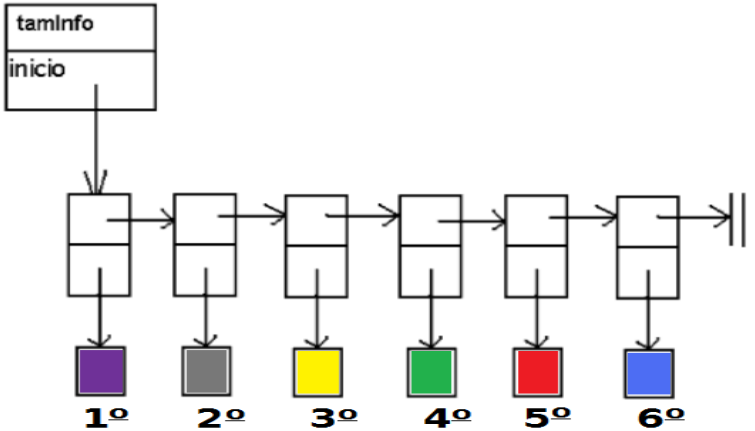
k)int removeOultimo(pLDSE p, void *reg);

l) int buscaNaPosLog(pLDSE p, void *reg, unsigned int posLog);
m) int insereNaPosLog (pLDSE p, void *novo,unsigned int posLog);
n) int removeDaPosLog(pLDSE p, void *reg, unsigned int posLog);

Serão abordadas adiante

ponteiro para o descritor

```
e) tamanhoDaLista(pLista)
    cont=0
    SE(vazia(pLista) == não)
        aux1 = primeiro(pLista)
        ENQUANTO( aux1 ≠ nulo) FAÇA:
            cont=cont+1
            aux1 = aux1->proximo
    RETORNA cont
```

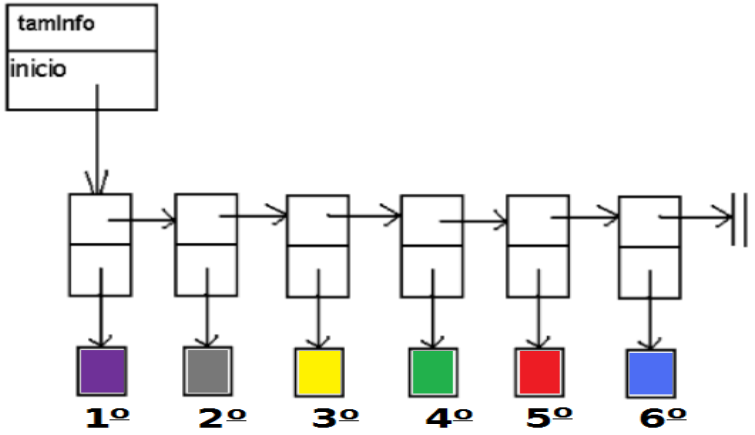



```
i) buscaOultimo(pLista)
    SE(vazia(pLista) == não)
        aux1 = primeiro(pLista)
        ENQUANTO(proximo(aux1) ≠ nulo) FAÇA:
                aux1 = aux1->proximo
        acessa(dados(aux1))
        RETORNA sucesso
     SENAO
       RETORNA fracasso;
```

ponteiro para
a informação
a ser inserida

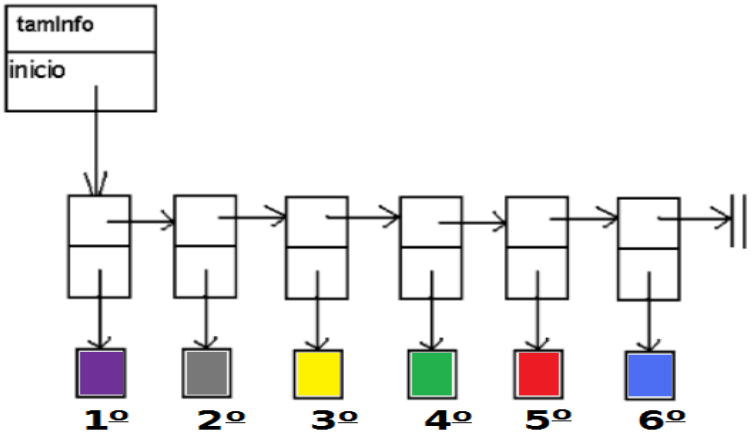
j) insereNovoUltimo(pLista, pNovo)

```
    temp=alocaMemoria(tamInfo(pLista))
      ...
      ... // acrescente verificação da alocação e tratamento de erro
      ...
    copia(dados(temp), pNovo, tamInfo(pLista) );
    anula(proximo(temp))  // aterra o ponteiro temp→proximo
    aux1 = primeiro(pLista)
    SE(aux1 == nulo)
        pLista->inicio=temp;
     SENAO
        ENQUANTO(proximo(aux1) ≠ nulo) FAÇA:
                aux1 = proximo(aux1)
        aux1->proximo=temp
    RETORNA sucesso
```



ponteiro para o descritor

```
k) removeOultimo(pLista)
    SE(vazia(pLista) == não)
        aux1 = primeiro(pLista)
        aux2 = proximo(aux1)
        SE( aux2 == nulo)
            Libera(primeiro(pLista))
            anula(pLista->inicio)
            RETORNA sucesso
        SENAO
            ENQUANTO( aux2-> proximo ≠ nulo) FAÇA:
                aux1 = aux2
                aux2 = aux2-> proximo
            libera(aux2)
            aux1->proximo = nulo
            RETORNA sucesso
    RETORNA fracasso
```



ATENÇÃO

A respeito das Operações nas <u>extremidades</u> da lista:

insereNovoPrimeiro(...) → também é capaz de inserir em lista vazia;

insereNovoUltimo(...) → *é a* única a acrescentar (*append*) um item ao fim da lista.

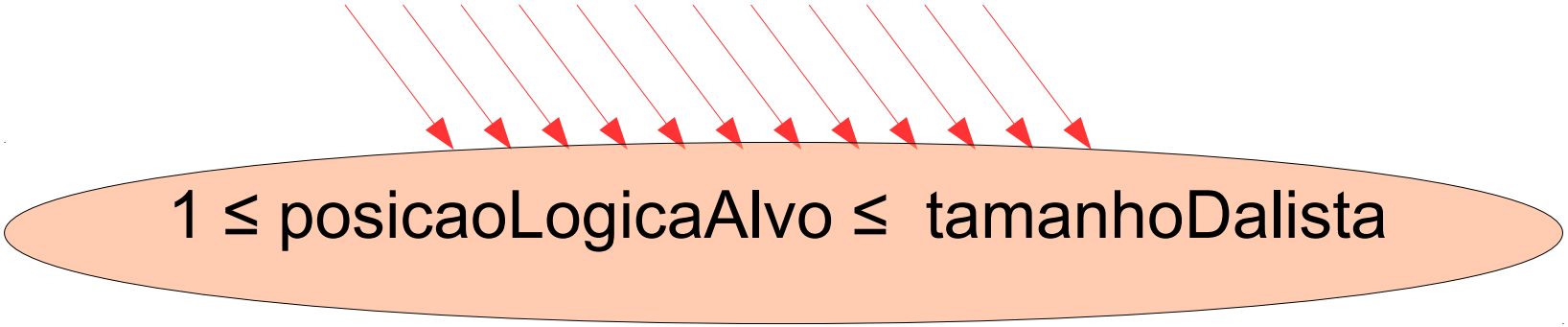
Isso ocorre em decorrência de aspectos conceituais do modelo TDA

ATENÇÃO

A Respeito das Operações em Posições Lógicas Existentes

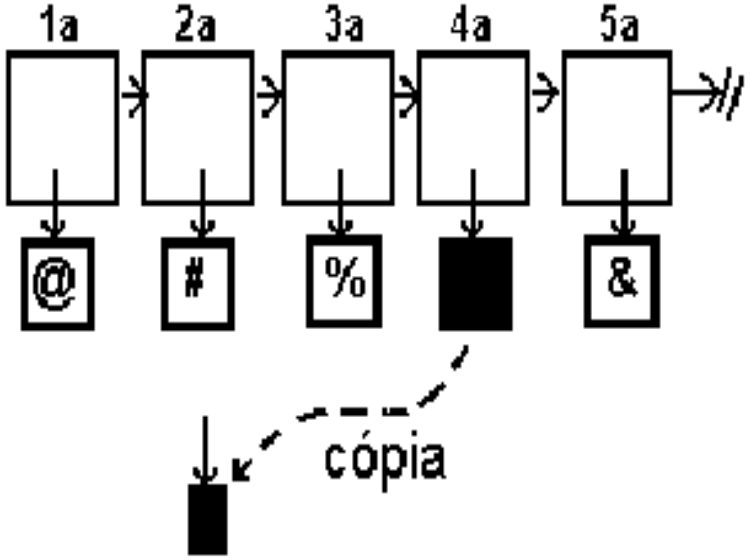
buscaNaposicaoLogica(...), removeDaposicaoLogica(...) e insereNaPosicaoLogica(...)

Essas operações exigem que a posição lógica alvo já exista na lista, ou seja:

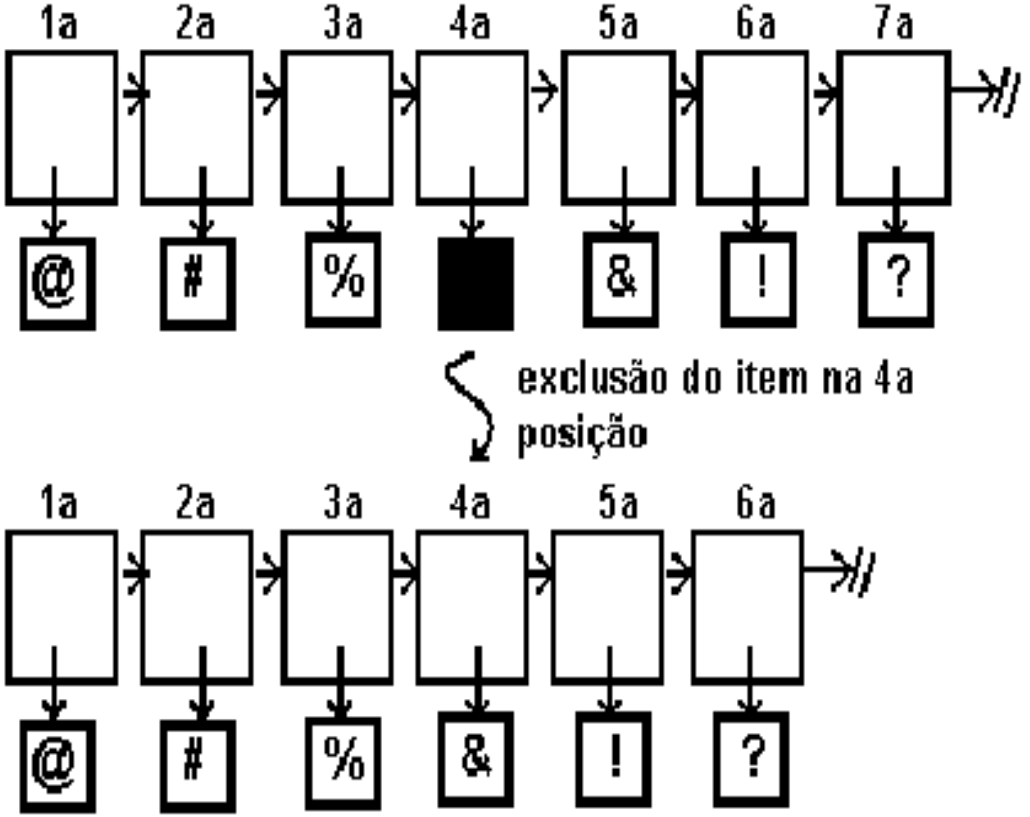


Exemplos de operações: Busca e remoção de uma posição lógica em uma LDSE

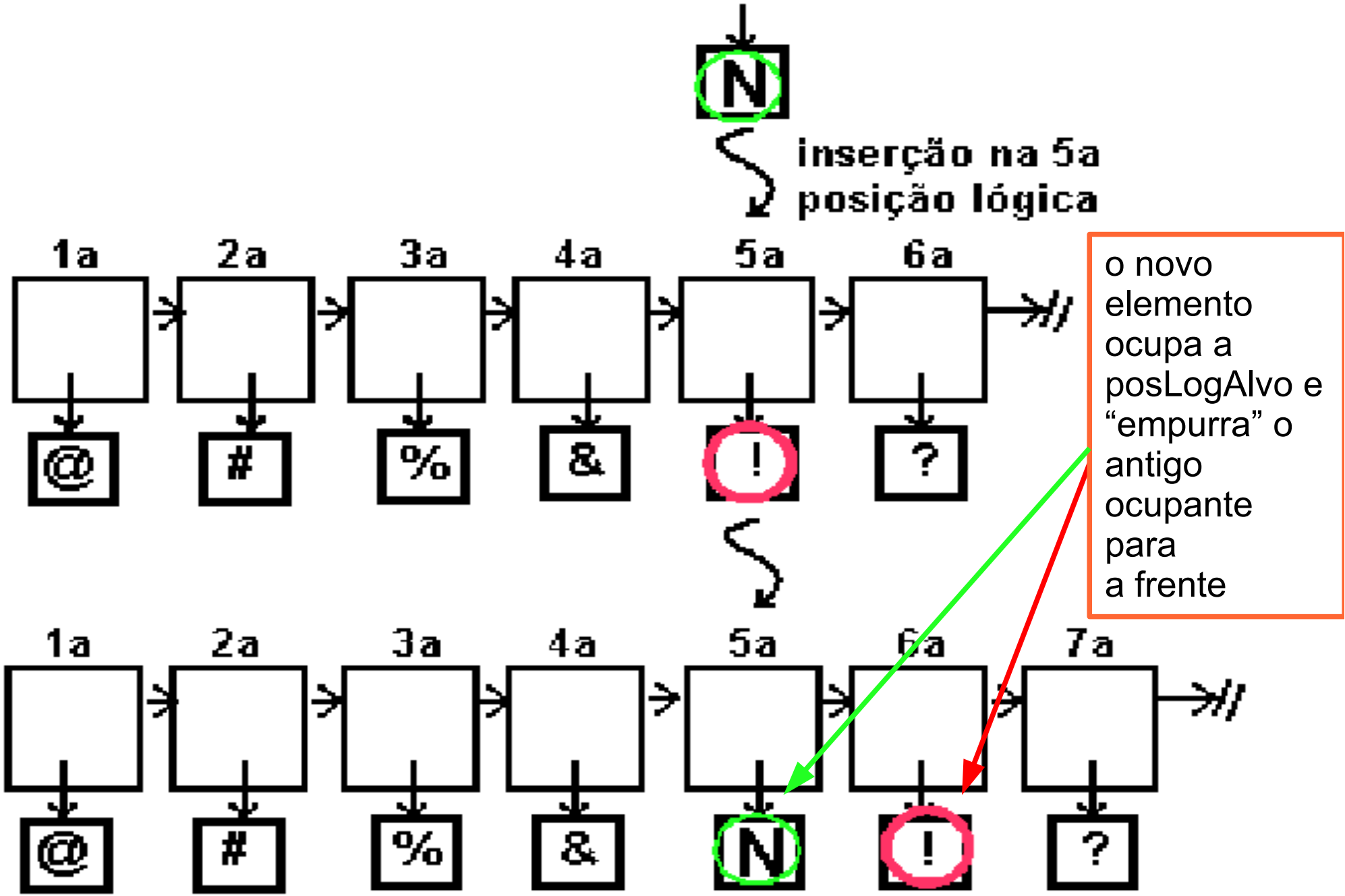
(A) Busca por um item na 4a posição lógica.



(B) Exclusão do item na 4a posição lógica.



C) Exemplo de operação de inserção em uma posição lógica



Operações em Posições Lógicas Existentes na Lista

1 ≤ posicaoLogicaAlvo ≤ tamanhoDalista

Mas, por que a *insereNaPosicaoLogica(...)* não insere em lista vazia?

- RESPOSTA:

  A lista vazia não possui posições ocupadas, então não seria correto inserir em uma posição que (ainda) não existe na lista!

  - ✔ Suponha que, apesar da lista estar VAZIA, o programa de aplicação tenha solicitado: *insereNaPosicaoLogica(pLista, novo, 30);*
  - ✔ Também suponha que o TDA tenha realizado tal inserção
    - ✘ Nesse caso o programa de aplicação receberia uma informação de inserção bem sucedida na 30ª posição da lista, o que <u>não existe</u>!
    - ✘ Isso seria inconsistente com o estado da lista, pois o novo item será de fato o primeiro (único) e não o 30º
    - ✘ O TDA (contendo 1 item) estaria inconsistente com a aplicação (a qual inferiria, erradamente, a existência do TDA contendo pelo menos 31 itens)!

Operações em Posições Lógicas Existentes na Lista

1 ≤ posicaoLogicaAlvo ≤ tamanhoDalista

Por que a *insereNaPosicaoLogica(...)* não insere em posições maiores que o tamanho atual da lista?

RESPOSTA

- Suponha uma lista com N elementos e que o programa de aplicação tenha solicitado: *insereNaPosicaoLogica(pLista, novo, **N+10**);*

- Também suponha que o TDA tenha realizado a inserção:
  - Nesse caso o programa de aplicação receberá uma informação de inserção bem sucedida na (N+10)ª posição da lista, o que não existe!
  - Isso seria inconsistente com o estado da lista, pois a mesma possuia apenas N elementos, mais um implicaria em N+1 elementos e não N+10;
  - Novamente haveria inconsistência entre o tamanho real do TDA e o número de itens que a aplicação acha que já estão inseridos;

Complementando a interface – TDA_LDSE.C

A seguir descreveremos:

- *buscaNaposicaoLogica(..);*
- *removeDaposicaoLogica(...);*
- *insereNaposicaoLogica(...);*

Essas funções operam em uma *posLog* qualquer e por isso não encontram similar nas pilhas e filas.

posLog: posição alvo da operação

```
buscaNaPosLog(pLista, posLog, destino)
    SE (vazia(pLista) == NAO E posLog > 0)
        SE(posLog == 1)
            copia(destino,dados(primeiro(pLista)),tamInfo(pLista))
            RETORNA sucesso
        SENAO /* posLog > 1, no minimo igual a dois */
            cont=2 /* contador de nós */
            aux2 = proximo(primeiro(pLista))
            ENQUANTO(aux2 ≠ nulo E  cont < posLog) FAÇA:
                aux2 = aux2-> proximo
                cont = cont + 1
            SE(aux2 ≠ nulo E posLog == cont)/*aux2 na posicao alvo? */
                 copia(destino, dados(aux2), tamInfo(pLista))
                RETORNA sucesso
 RETORNA fracasso
```

/* ATENÇÃO: a *posLog* tem que existir na lista antes da chamada a uma operação voltada a esta *PosLog*/

```
insereNaPosicaoLogica(pLista, pNovo, posLog)
  SE (posLog > 0 E vazia(pLista) == NAO)
        SE(posLog == 1)
             RETORNA insereNovoPrimeiro(pLista, pNovo)
        SENAO
             cont = 2
             aux1 = primeiro(pLista)
             aux2 = aux1-> proximo
             ENQUANTO(aux2 ≠ nulo E  cont < posLog) FAÇA:
                  aux1 = aux2
                  aux2 = aux2-> proximo
                  cont = cont + 1
            SE(aux2 ≠ nulo E posLog == cont)
             /*aux1 faz referência à posição posLog-1 e aux2 à posLog */
                  aux1->proximo = alocaMemoria(tamInfo(pLista))
                  aux1->proximo ->proximo = aux2
                  copia(dados( aux1-> proximo), pNovo, tamInfo(pLista) );
                  RETORNA sucesso
  RETORNA fracasso
```

/* ATENÇÃO: a *posLog* tem que existir na lista antes da chamada a uma operação voltada a esta *PosLog*/

```
removeDaPosLog(pLista, posLog)
  SE (vazia(pLista) == NAO E posLog > 0)
        SE(posLog == 1)
            RETORNA( removeDoInicio(pLista));
        SENAO /* posLog > 1, no minimo igual a dois */
            cont=2
            aux1 = primeiro(pLista)
            aux2 = aux1-> proximo
            ENQUANTO(aux2 ≠ nulo E  cont < posLog) FAÇA:
                aux1 = aux2
                aux2 = aux2-> proximo
                cont = cont + 1
            SE(aux2 ≠ nulo E posLog == cont)
                aux1->proximo = aux2-> proximo
                liberaNoLista(aux2)
                RETORNA sucesso
  RETORNA fracasso
/* ATENÇÃO: a posLog tem que existir na lista antes da
chamada a uma operação voltada a esta PosLog*/
```

Implemente a LDSE:

1) int cria(ppLDSE pp, int tamInfo);
2) void reinicia(pLDSE p);
3) void destroi(ppLDSE pp);
4) int tamanho(pLDSE p);
5) int testaVazia(pLDSE p);

6) int insereNovoPrimeiro(pLDSE p, void *novo);
7) int insereNovoUltimo(pLDSE p, void *novo);
8) int insereNaPosLog (pLDSE p, void *novo,unsigned int posLog);

9) int buscaNaPosLog(pLDSE p, void *reg, unsigned int posLog);
10) int buscaOultimo(pLDSE p, void *reg);
11) int buscaOprimeiro(pLDSE p, void *reg);

12) int removeDaPosLog(pLDSE p, void *reg, unsigned int posLog);
13) int removeOultimo(pLDSE p, void *reg);
14) int removeOprimeiro(pLDSE p, void *reg);

Implemente uma LDDE com base na LDSE:

1) int cria(ppLDDE pp, int tamInfo);
2) void reinicia(pLDDE p);
3) void destroi(ppLDDE pp);
4) int tamanho(pLDDE p);
5) int testaVazia(pLDSE p);

6) int insereNovoPrimeiro(pLDDE p, void *novo);
7) int insereNovoUltimo(pLDDE p, void *novo);
8) int insereNaPosLog (pLDDE p, void *novo,unsigned int posLog);

9) int buscaNaPosLog(pLDDE p, void *reg, unsigned int posLog);
10) int buscaUltimo(pLDDE p, void *reg);
11) int buscaPrimeiro(pLDDE p, void *reg);

12) int removeDaPosLog(pLDDE p, void *reg, unsigned int posLog);
13) int removeUltimo(pLDDE p, void *reg);
14) int removePrimeiro(pLDDE p, void *reg);

15)int enderecosFisicos(pLDDE p);

Similar à inserção na fila de prioridade

ATENÇÃO

A Respeito das Operações em Posições Lógicas Existentes

1 ≤ posicaoLogicaAlvo ≤ tamanhoDalista

J u s t i f i c a – s e

Essa exigência é obviamente justificada para as funções *buscaNaposicaoLogica(...)* e *removeDaposicaoLogica(...)* pois não teria sentido buscar ou remover um item inexistente na lista.

... E também é válida para a função *insereNaPosicaoLogica(...*), a qual:

Não insere em lista vazia
(para essa finalidade utiliza-se *insereNovoPrimeiro(...)*);

Não insere em posições maiores que o tamanho atual da lista
(para tal finalidade utiliza-se *insereNovoUltimo(...)*);