# 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.

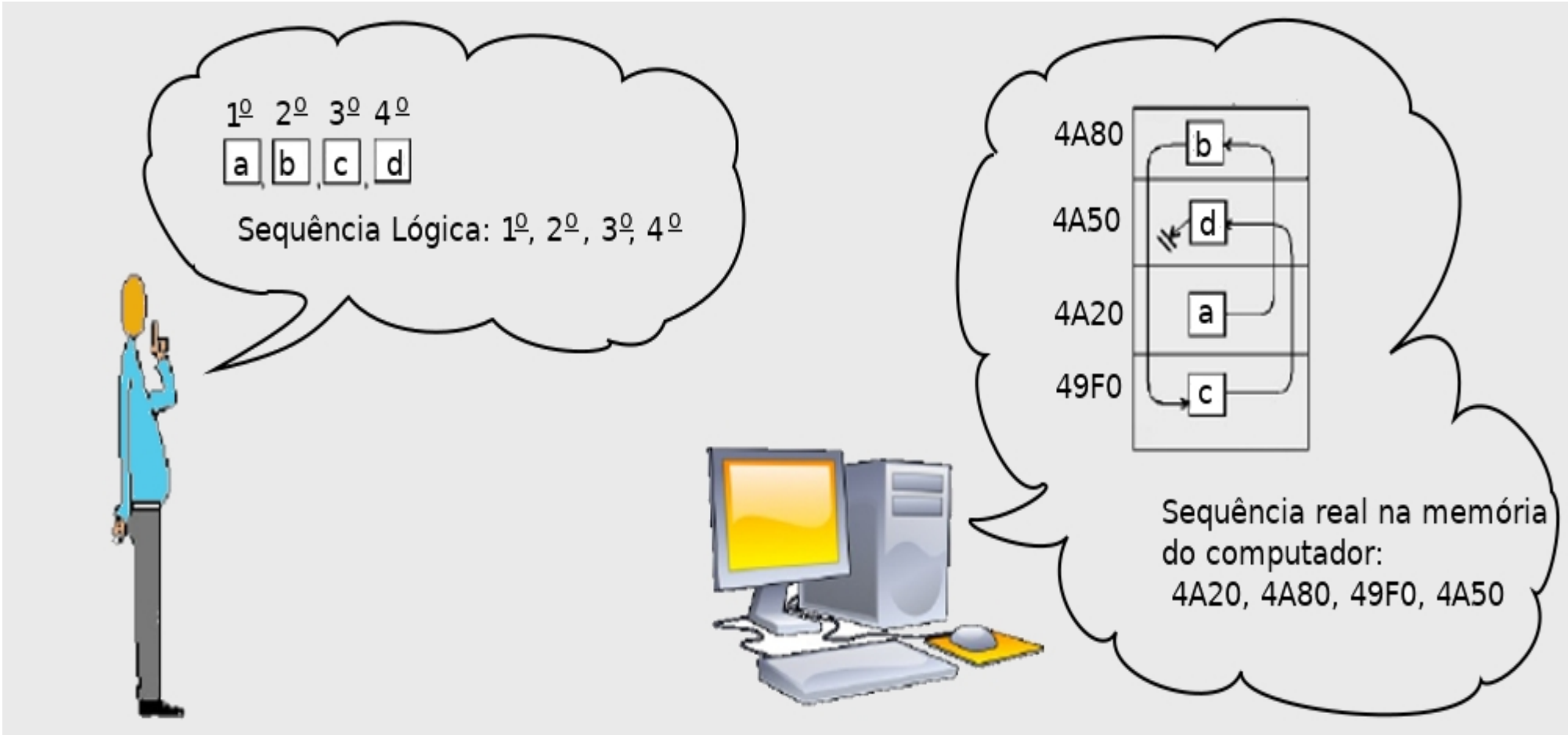
- Listas podem ser manipuladas em qualquer ponto da sua sequência de itens.

- 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 programador de aplicação abstrai/entende a lista por essa sequência lógica;

• É possível efetuar operações de busca, remoção ou inserção em qualquer posição dessa sequência lógica.

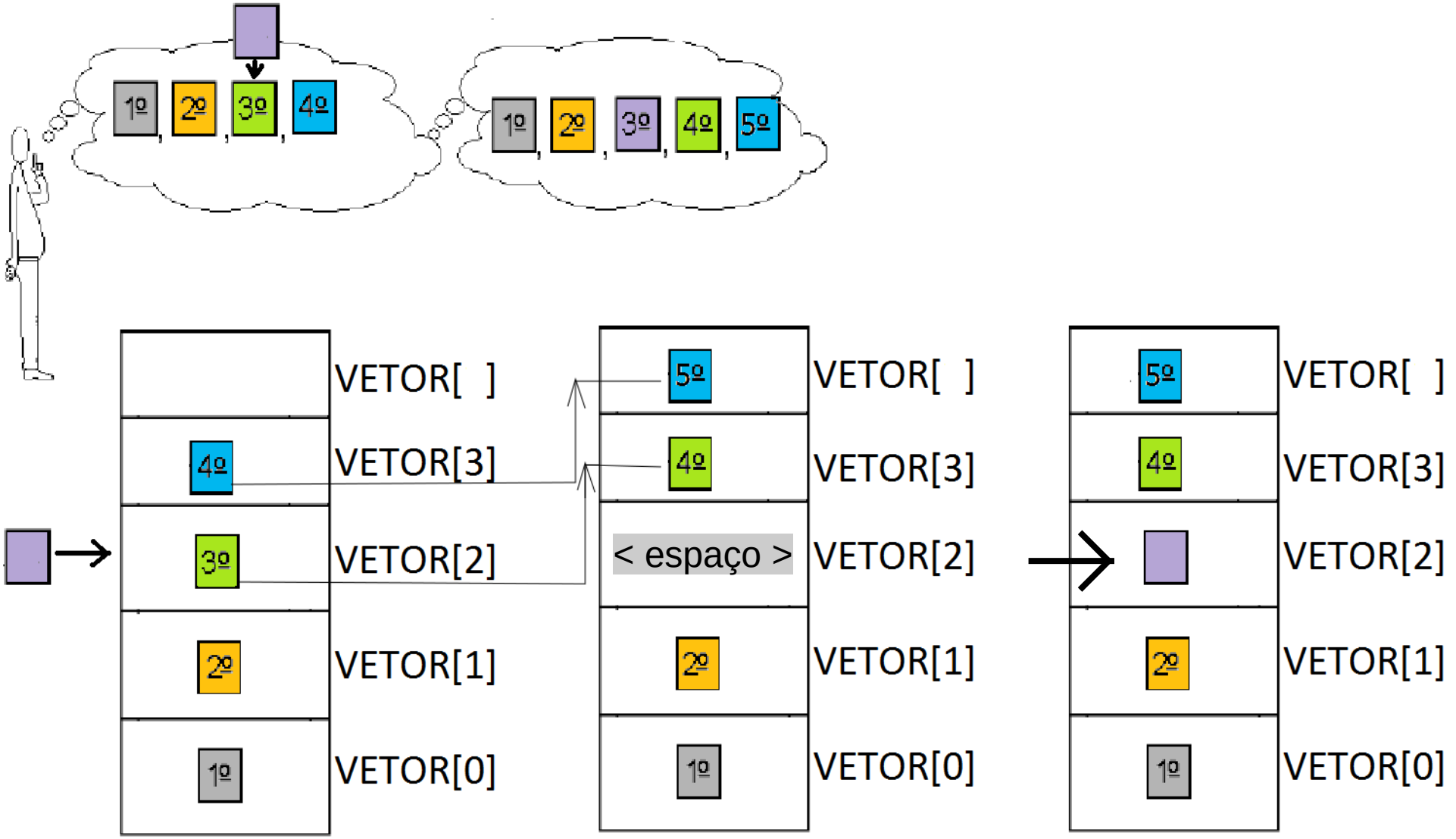


Lista encadeada ou lista não encadeada?

Qual é a melhor opção para a implementação da lista?

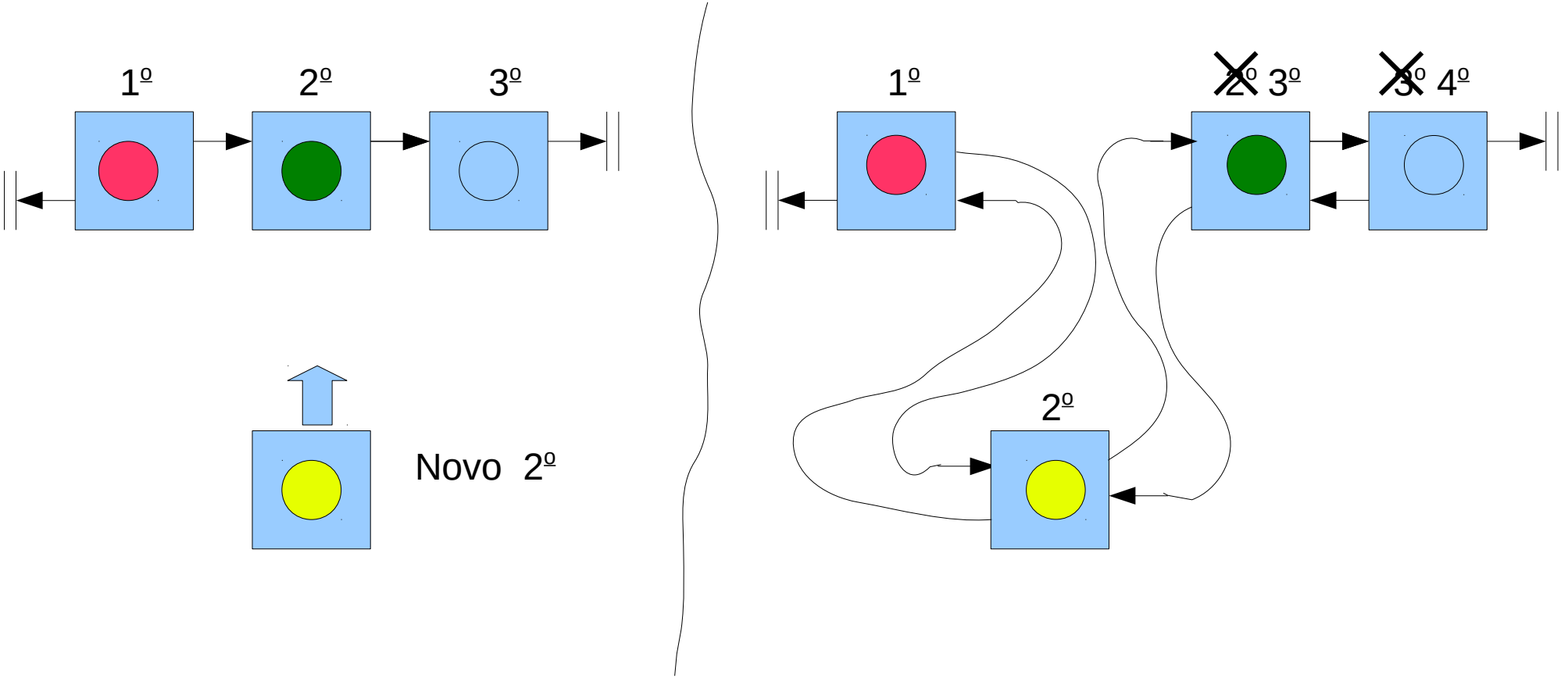
# Lista encadeada ou lista não encadeada? Qual é a melhor opção ?

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



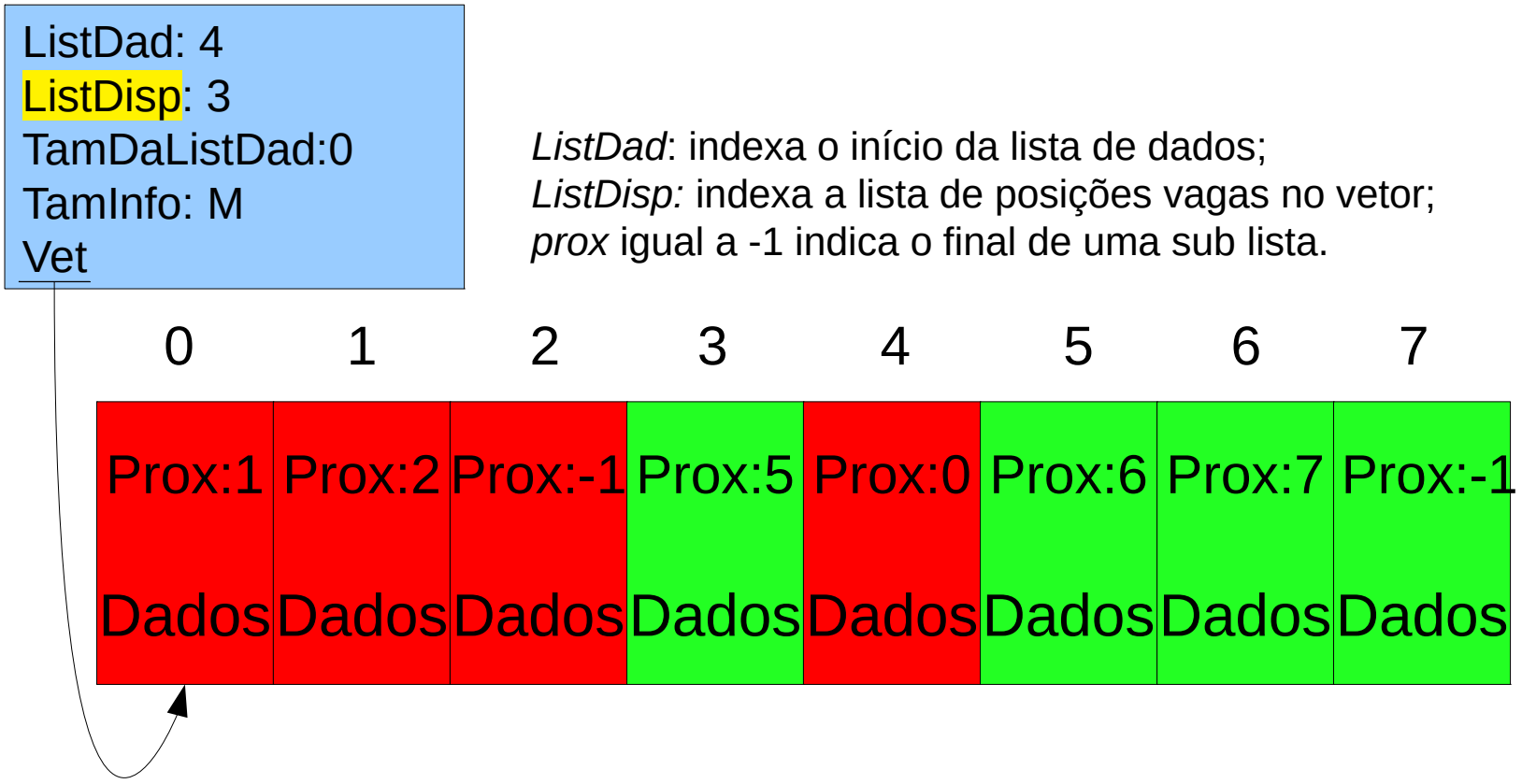
# Lista encadeada ou lista não encadeada? Qual é a melhor opção ?

- A estratégia encadeada dispensa a movimentação de dados:



# Lista encadeada ou lista não encadeada? Qual é a melhor opção ?

- A estratégia encadeada dispensa a movimentação de dados
  Inclusive é possível implementar a lista em um vetor incorporando o conceito de encadeamento e suas vantagens:




*ListDad*: indexa o início da lista de dados;
*ListDisp:* indexa a lista de posições vagas no vetor;
*prox* igual a -1 indica o final de uma sub lista.

# Lista encadeada ou lista não encadeada? Qual é a melhor opção ?

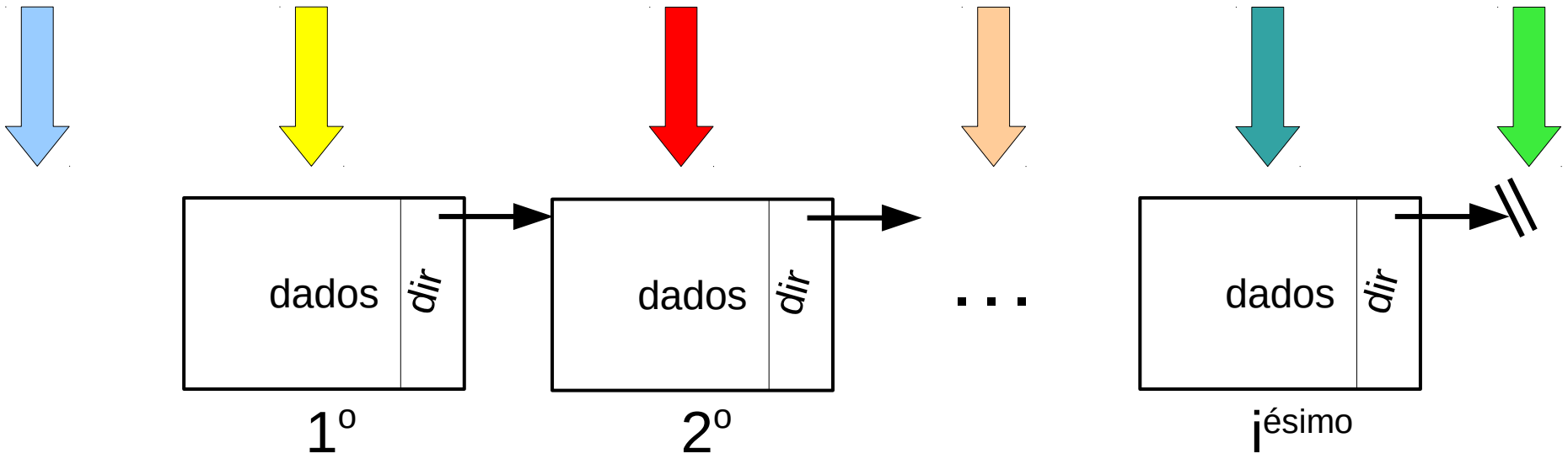
- A estratégia encadeada dispensa a movimentação de dados;

- Portanto, por questões de eficiência nas operações, estudaremos as listas estáticas e dinâmicas implementadas por encadeamento;

Interface da 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

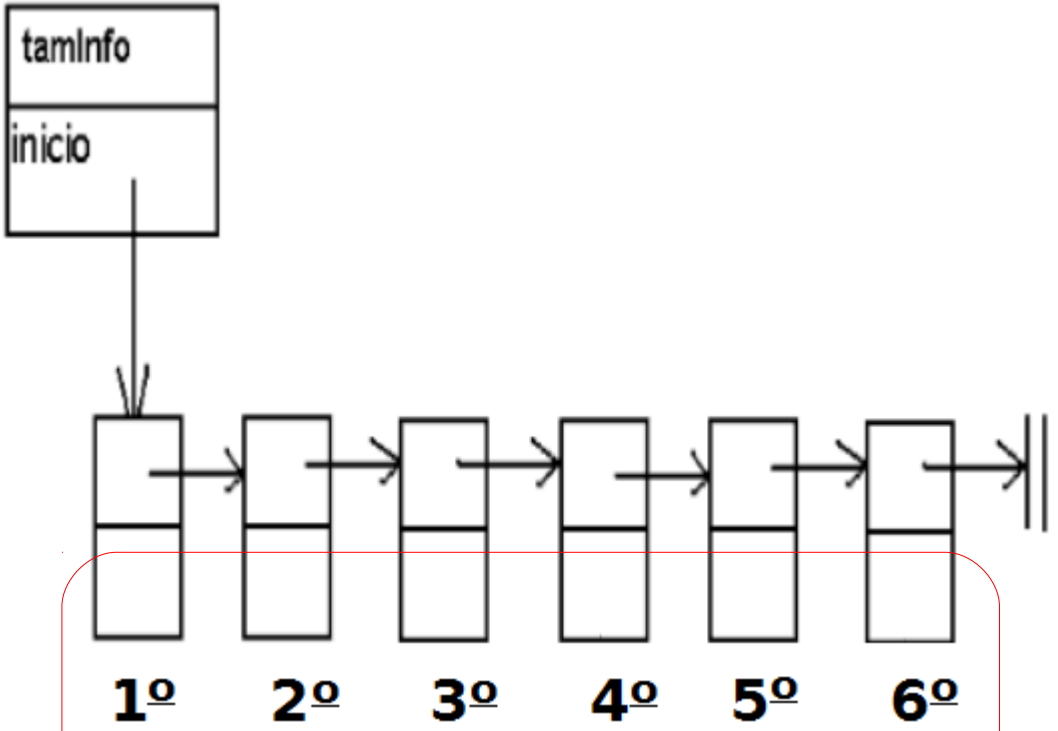


Inicialmente iremos estudar a LSE:

## Lista com encadeamento simples

VAZIA


NÃO VAZIA

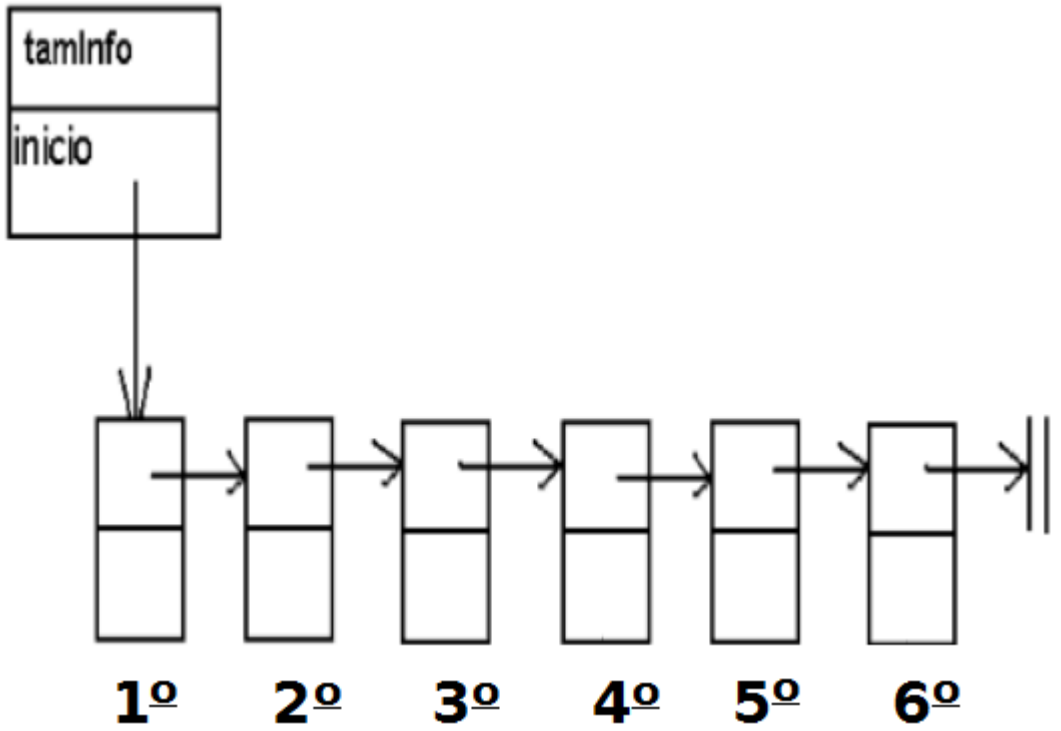



Sequência lógica

# Est. de Dados – LSE

```
/* nó de dados */
typedef struct noLSE{
    info dados;
    struct nLSE *prox;
} NoLSE;

/* descritor */
typedef struct{
    int taminfo;
    NoLSE *inicio;
}LSE;
```



Interface da LSE:

a) LSE * cria(int tamInfo);
b) void reinicia(LSE *p);
c) LSE * destroi(LSE *p);
d) int testaVazia(LSE *p);
e) int tamanho(LSE *p);

→ Operações ordinárias presentes em todas as estruturas de dados vistas até aqui

f) int buscaOprimeiro(LSE *p, info *reg);
g) int insereNovoPrimeiro(LSE *p, info *novo);
h) int removeOprimeiro(LSE *p, info *reg);

i) int buscaOultimo(LSE *p, info *reg);
j) int insereNovoUltimo(LSE *p, info *novo);
k)int removeOultimo(LSE *p, info *reg);

→ Operações nas extremidades da lista, são similares às correspondentes utilizadas nas pilhas e/ou filas

l) int buscaNaPosLog(LSE *p, info *reg, int posLog);
m) int insereNaPosLog (LSE *p, info *novo, int posLog);
n) int removeDaPosLog(LSE *p, info *reg, int posLog);

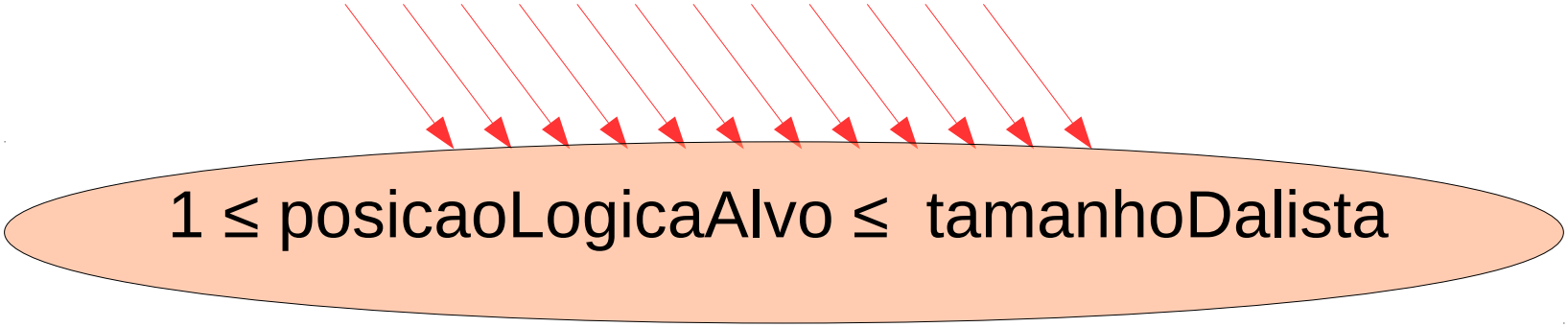
→ **Operações em alguma posição lógica especificada**

**ATENÇÃO**

**A Respeito das Operações em Posições Lógicas <u>Existentes</u>**

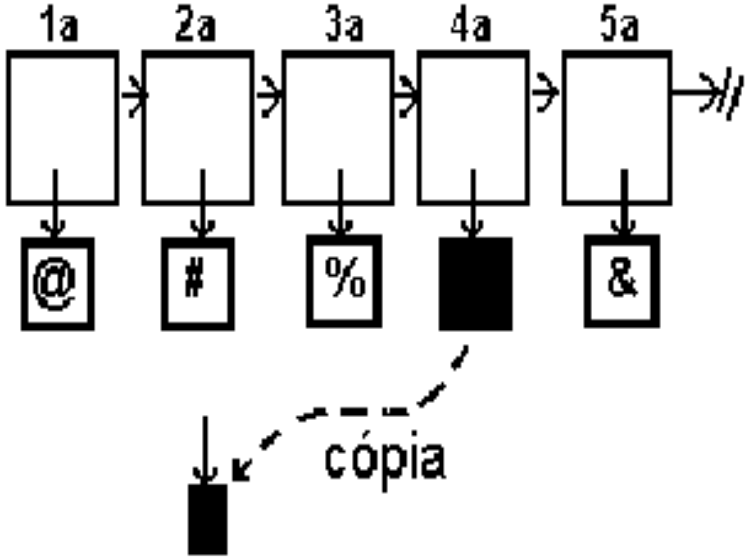
***buscaNaposicaoLogica(...),***
***removeDaposicaoLogica(...) e***
***insereNaPosicaoLogica(...)***

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

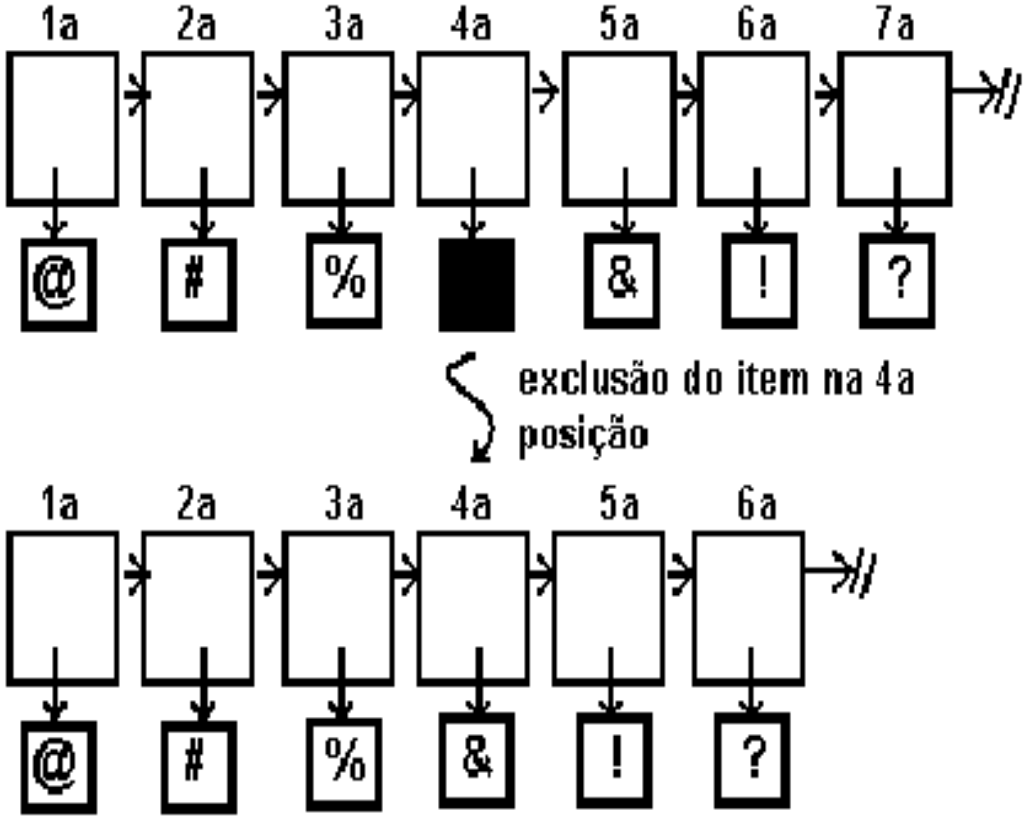


# Exemplos de operações: Busca e remoção de uma posição lógica em uma LDSE

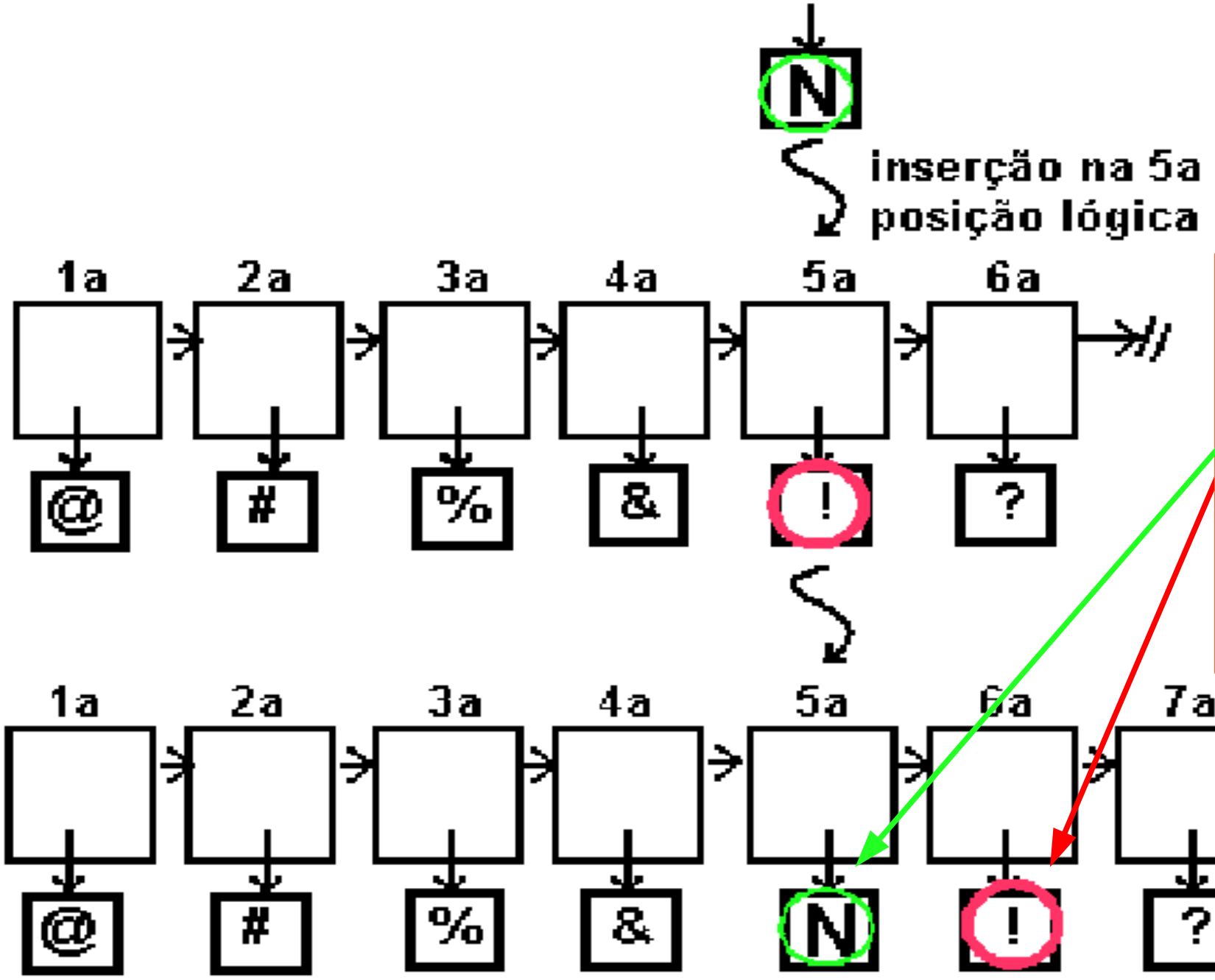
(A) Busca por um item na 4a posição lógica.




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



# C) Exemplo de operação de inserção em uma posição lógica




o novo elemento ocupa a posLogAlvo e “empurra” o antigo ocupante para a frente

# Complementando a interface – TDA_LDSE.C

A seguir descreveremos os pseudocódigos para:

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

posLog: posição alvo da operação

```
buscaNaPosLog(pLista, posLog, destino)
    SE (vazia(pLista) == NAO E posLog > 0)
            SE(posLog == 1)
                    copia(destino,&(pLista->inicio->dados),pLista->tamInfo)
                    RETORNA sucesso
            SENAO /* posLog > 1, no minimo igual a dois */
                cont=2 /* contador de nós */
                aux1 = pLista->inicio->proximo
                ENQUANTO(aux1 ≠ nulo E  cont < posLog) FAÇA:
                        aux1 = aux1-> proximo
                        cont = cont + 1
                SE(aux1 ≠ nulo E posLog == cont)/*aux1 na posicao alvo? */
                        copia(destino, &(aux1->dados), 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 = pLista->inicio
              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(pLista->tamInfo)
                    aux1->proximo ->proximo = aux2
                    copia( &(aux1-> proximo->dados), pNovo, pLista->tamInfo);
                    RETORNA sucesso
  RETORNA fracasso
                              /* ATENÇÃO: a posLog tem que existir na lista antes da
                              chamada a uma operação voltada a esta PosLog*/
```

```
removeDaPosLog(pLista, pDestino, 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 =pLista->inicio
            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-> próximo
                copia(pDestino,&(aux2->dados),pLista->tamInfo)
                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 LSE com a seguinte interface:

a) LSE * cria(int tamInfo);
b) void reinicia(LSE *p);
c) LSE * destroi(LSE *p);
d) int testaVazia(LSE *p);
e) int tamanho(LSE *p);

f) int buscaOprimeiro(LSE *p, info *reg);
g) int insereNovoPrimeiro(LSE *p, info *novo);
h) int removeOprimeiro(LSE *p, info *reg);

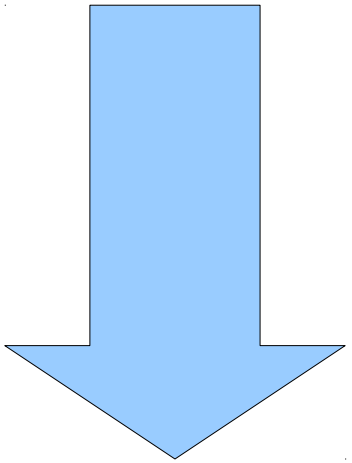
i) int buscaOultimo(LSE *p, info *reg);
j) int insereNovoUltimo(LSE *p, info *novo);
k)int removeOultimo(LSE *p, info *reg);

l) int buscaNaPosLog(LSE *p, info *reg, int posLog);
m) int insereNaPosLog (LSE *p, info *novo, int posLog);
n) int removeDaPosLog(LSE *p, info *reg, int posLog);

o) int enderecosFisicos(pLSE p);

Implemente a LDE (duplamente encadeada) com a mesma funcionalidade da LSE solicitada acima.

# Algumas observações...



# 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(...)*);

## 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 aqui adotados**

# 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 em uma 30ª posição que ainda não existe na lista!
    - ✗ 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 <u>não existe</u>!

  - 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;