

# Listas Duplamente Encadeadas

Com base no que foi visto implemente a operação destruir() que compõem o TAD LISTA\_DUP\_ENC.

```
1  typedef struct nodo
2  {
3      int inf;
4      struct nodo * ant;
5      struct nodo * prox;
6  }NODO;
7  typedef NODO * LISTA_DUP_ENC;
8  void cria_lista (LISTA_DUP_ENC *);
9  int eh_vazia (LISTA_DUP_ENC);
10 int tam (LISTA_DUP_ENC);
11 void ins (LISTA_DUP_ENC *, int, int);
12 int recup (LISTA_DUP_ENC, int);
13 void ret (LISTA_DUP_ENC *, int);
14 void destruir (LISTA_DUP_ENC);
```

```
1 void destruire (LISTA_DUP_ENC l)
2 {
3     LISTA_DUP_ENC aux;
4     while (l)
5     {
6         aux = l;
7         l = l->prox;
8         free (aux);
9     }
10 }
```

## Listas Duplamente Encadeadas

Implemente, no TAD LISTA\_DUP\_ENC, a seguinte operação:

```
void inverter_lista (LISTA_DUP_ENC *pl);
```


a qual recebe uma referência para uma lista duplamente encadeada e inverte a ordem de seus elementos.

```
1 void inverter_lista (LISTA_DUP_ENC *p1)
2 {
3     if (*p1) {
4         LISTA_DUP_ENC aux; /* NODO *aux; */
5         do {
6             aux=(*p1)->prox;
7             (*p1)->prox=(*p1)->ant;
8             (*p1)->ant=aux;
9             if (aux)
10                *p1=aux;
11        }while (aux);
12    }
13 }
```

# Listas Duplamente Encadeadas

Com base no que foi visto sobre o TAD LISTA\_DUP\_ENC, implemente, utilizando recursividade, as operações que julgar viáveis.

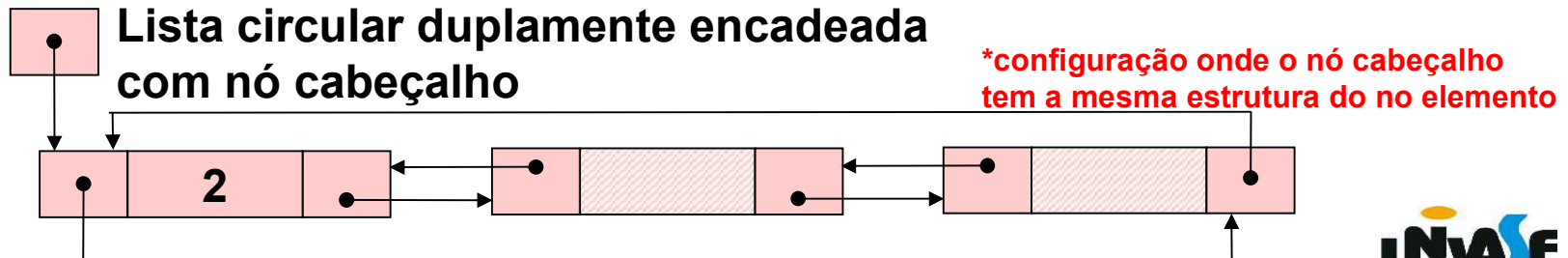
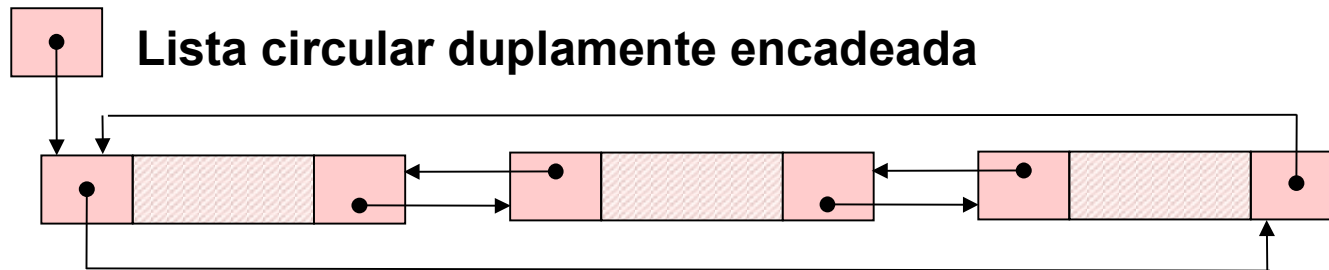
```
1  typedef struct nodo
2  {
3      int inf;
4      struct nodo * ant;
5      struct nodo * prox;
6  }NODO;
7  typedef NODO * LISTA_DUP_ENC;
8  void cria_lista (LISTA_DUP_ENC *);
9  int eh_vazia (LISTA_DUP_ENC);
10 int tam (LISTA_DUP_ENC);
11 void ins (LISTA_DUP_ENC *, int, int);
12 int recup (LISTA_DUP_ENC, int);
13 void ret (LISTA_DUP_ENC *, int);
14 void destruir (LISTA_DUP_ENC);
```



# **Listas Circulares Duplamente Encadeadas com Nó Cabeçalho**

# Listas Dup. Encad. Circ. com NC

Também podemos construir *listas circulares duplamente encadeadas* ou *listas circulares duplamente encadeadas com nó cabeçalho*.



```
1  typedef struct nodo
2  {
3      int inf;
4      struct nodo * ant;
5      struct nodo * prox;
6  }NODO;
7  typedef NODO * LISTA_CIR_DUP_ENC_NC;
8  void cria_lista (LISTA_CIR_DUP_ENC_NC *);
9  int eh_vazia (LISTA_CIR_DUP_ENC_NC);
10 int tam (LISTA_CIR_DUP_ENC_NC);
11 void ins (LISTA_CIR_DUP_ENC_NC, int, int);
12 int recup (LISTA_CIR_DUP_ENC_NC, int);
13 void ret (LISTA_CIR_DUP_ENC_NC, int);
14 void destruir (LISTA_CIR_DUP_ENC_NC);
```

## Listas Dup. Encad. Circ. com NC

Com base no que foi visto implemente a operação `criar_lista()` que compõem o TAD `LISTA_CIR_DUP_ENC_NC`.

```
typedef struct nodo
{
    int inf;
    struct nodo * ant;
    struct nodo * prox;
}NODO;
typedef NODO * LISTA_CIR_DUP_ENC_NC;
void cria_lista (LISTA_CIR_DUP_ENC_NC *);
int eh_vazia (LISTA_CIR_DUP_ENC_NC);
int tam (LISTA_CIR_DUP_ENC_NC);
void ins (LISTA_CIR_DUP_ENC_NC, int, int);
int recup (LISTA_CIR_DUP_ENC_NC, int);
void ret (LISTA_CIR_DUP_ENC_NC, int);
void destruir (LISTA_CIR_DUP_ENC_NC);
```

```
1 void cria_lista (LISTA_CIR_DUP_ENC_NC *p1)
2 {
3     *p1 = (LISTA_CIR_DUP_ENC_NC) malloc (sizeof(NODO));
4     if (!(*p1))
5     {
6         printf ("\nERRO! Memoria insuficiente!\n");
7         exit (2);
8     }
9     (*p1)->inf=0;
10    (*p1)->ant=(*p1)->prox=*p1;
11 }
```

## Listas Dup. Encad. Circ. com NC

Com base no que foi visto implemente a operação `eh_vazia()` que compõem o TAD `LISTA_CIR_DUP_ENC_NC`.

```
typedef struct nodo
{
    int inf;
    struct nodo * ant;
    struct nodo * prox;
}NODO;
typedef NODO * LISTA_CIR_DUP_ENC_NC;
void cria_lista (LISTA_CIR_DUP_ENC_NC *);
int eh_vazia (LISTA_CIR_DUP_ENC_NC);
int tam (LISTA_CIR_DUP_ENC_NC);
void ins (LISTA_CIR_DUP_ENC_NC, int, int);
int recup (LISTA_CIR_DUP_ENC_NC, int);
void ret (LISTA_CIR_DUP_ENC_NC, int);
void destruir (LISTA_CIR_DUP_ENC_NC);
```

```
1 int eh_vazia (LISTA_CIR_DUP_ENC_NC l)
2 {
3     return (l->inf == 0);
4 }
```

## Listas Dup. Encad. Circ. com NC

Com base no que foi visto implemente a operação tam() que compõem o TAD LISTA\_CIR\_DUP\_ENC\_NC.

```
typedef struct nodo
{
    int inf;
    struct nodo * ant;
    struct nodo * prox;
}NODO;
typedef NODO * LISTA_CIR_DUP_ENC_NC;
void cria_lista (LISTA_CIR_DUP_ENC_NC *);
int eh_vazia (LISTA_CIR_DUP_ENC_NC);
int tam (LISTA_CIR_DUP_ENC_NC);
void ins (LISTA_CIR_DUP_ENC_NC, int, int);
int recup (LISTA_CIR_DUP_ENC_NC, int);
void ret (LISTA_CIR_DUP_ENC_NC, int);
void destruir (LISTA_CIR_DUP_ENC_NC);
```

```
1 int tam (LISTA_CIR_DUP_ENC_NC l)
2 {
3     return (l->inf);
4 }
```

## Listas Dup. Encad. Circ. com NC

Com base no que foi visto implemente a operação `ins()` que compõem o TAD `LISTA_CIR_DUP_ENC_NC`.

```
typedef struct nodo
{
    int inf;
    struct nodo * ant;
    struct nodo * prox;
}NODO;
typedef NODO * LISTA_CIR_DUP_ENC_NC;
void cria_lista (LISTA_CIR_DUP_ENC_NC *);
int eh_vazia (LISTA_CIR_DUP_ENC_NC);
int tam (LISTA_CIR_DUP_ENC_NC);
void ins (LISTA_CIR_DUP_ENC_NC, int, int);
int recup (LISTA_CIR_DUP_ENC_NC, int);
void ret (LISTA_CIR_DUP_ENC_NC, int);
void destruir (LISTA_CIR_DUP_ENC_NC);
```

# Listas Dup. Encad. Circ. com NC

## Dicas:

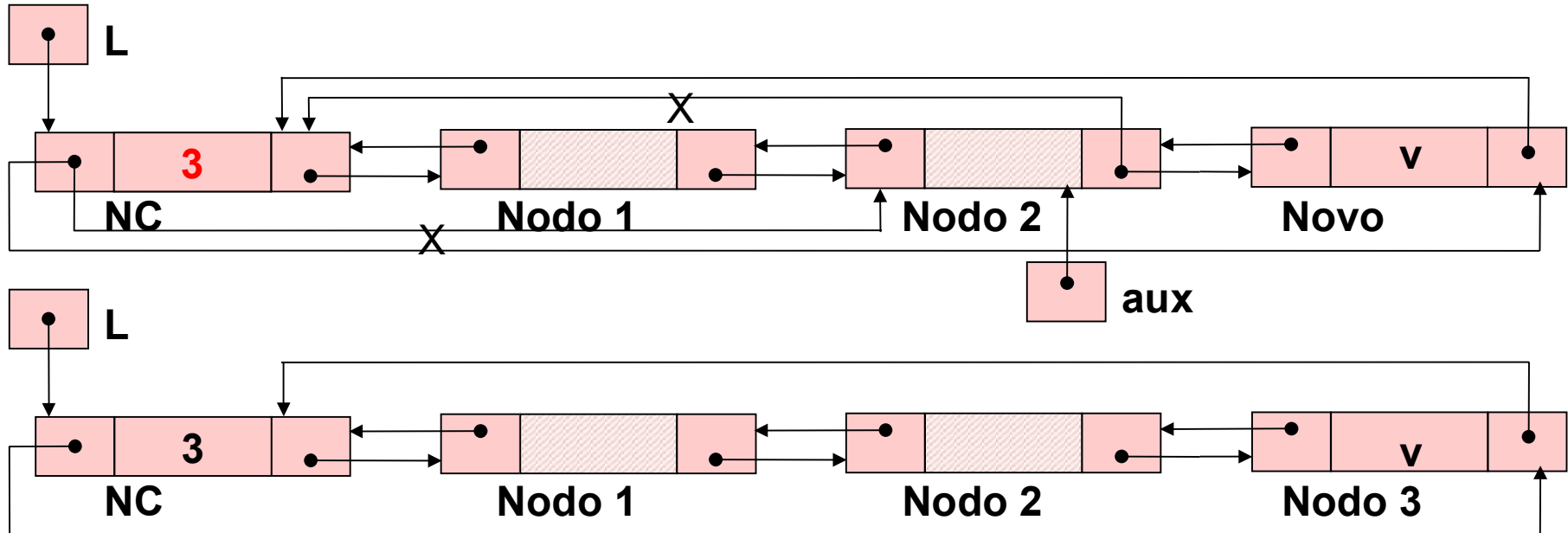
A posição é válida?

Tem espaço na memória para armazenar mais um elemento?

Todas as situações de inserção são tratadas da mesma forma?

# Listas Dup. Encad. Circ. com NC

Esquema do processo da inserção de um nó da lista circular duplamente enca-deada com nó cabeçalho.



```
1 void ins (LISTA_CIR_DUP_ENC_NC l, int v, int k)
2 {
3     LISTA_CIR_DUP_ENC_NC aux, novo;
4     if (k < 1 || k > tam(l)+1) {
5         printf ("\nERRO! Posição invalida para insercao.\n");
6         exit (1);
7     }
8     novo = (LISTA_CIR_DUP_ENC_NC) malloc (sizeof(NODO));
9     if (!novo) {
10        printf ("\nERRO! Memoria insuficiente!\n");
11        exit (2);
12    }
```

```
13     novo->inf = v;
14     for (aux=l; k>1; aux=aux->prox, k--);
15     novo->prox = aux->prox;
16     novo->ant = aux;
17     aux->prox = novo;
18     novo->prox->ant=novo;
19     l->inf++;
20 }
```

## Listas Dup. Encad. Circ. com NC

Com base no que foi visto implemente a operação `recup()` que compõem o TAD `LISTA_CIR_DUP_ENC_NC`.

```
typedef struct nodo
{
    int inf;
    struct nodo * ant;
    struct nodo * prox;
}NODO;
typedef NODO * LISTA_CIR_DUP_ENC_NC;
void cria_lista (LISTA_CIR_DUP_ENC_NC *);
int eh_vazia (LISTA_CIR_DUP_ENC_NC);
int tam (LISTA_CIR_DUP_ENC_NC);
void ins (LISTA_CIR_DUP_ENC_NC, int, int);
int recup (LISTA_CIR_DUP_ENC_NC, int);
void ret (LISTA_CIR_DUP_ENC_NC, int);
void destruir (LISTA_CIR_DUP_ENC_NC);
```

```
1 int recup (LISTA_CIR_DUP_ENC_NC l, int k)
2 {
3     if (k < 1 || k > tam(l))
4     {
5         printf ("\nERRO! Consulta invalida.\n");
6         exit (3);
7     }
8     for (;k>0;k--)
9         l=l->prox;
10    return (l->inf);
11 }
```

## Listas Dup. Encad. Circ. com NC

Com base no que foi visto implemente a operação ret() que compõem o TAD LISTA\_CIR\_DUP\_ENC\_NC.

```
typedef struct nodo
{
    int inf;
    struct nodo * ant;
    struct nodo * prox;
}NODO;
typedef NODO * LISTA_CIR_DUP_ENC_NC;
void cria_lista (LISTA_CIR_DUP_ENC_NC *);
int eh_vazia (LISTA_CIR_DUP_ENC_NC);
int tam (LISTA_CIR_DUP_ENC_NC);
void ins (LISTA_CIR_DUP_ENC_NC, int, int);
int recup (LISTA_CIR_DUP_ENC_NC, int);
void ret (LISTA_CIR_DUP_ENC_NC, int);
void destruir (LISTA_CIR_DUP_ENC_NC);
```

# Listas Dup. Encad. Circ. com NC

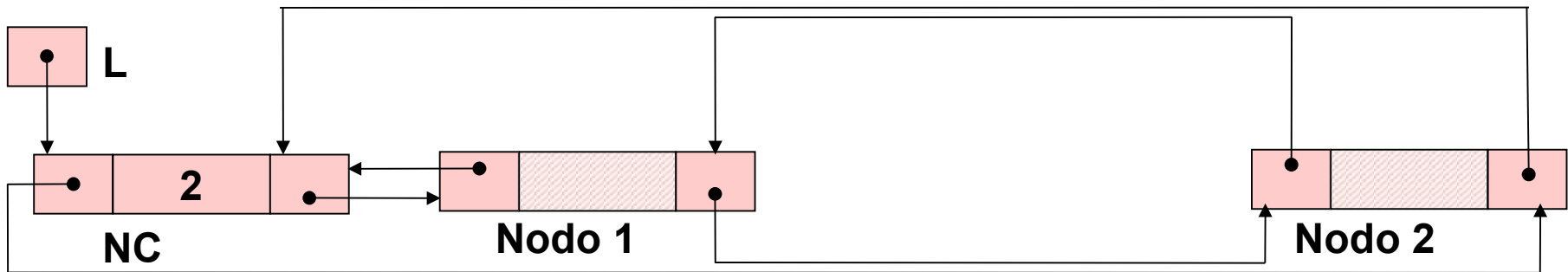
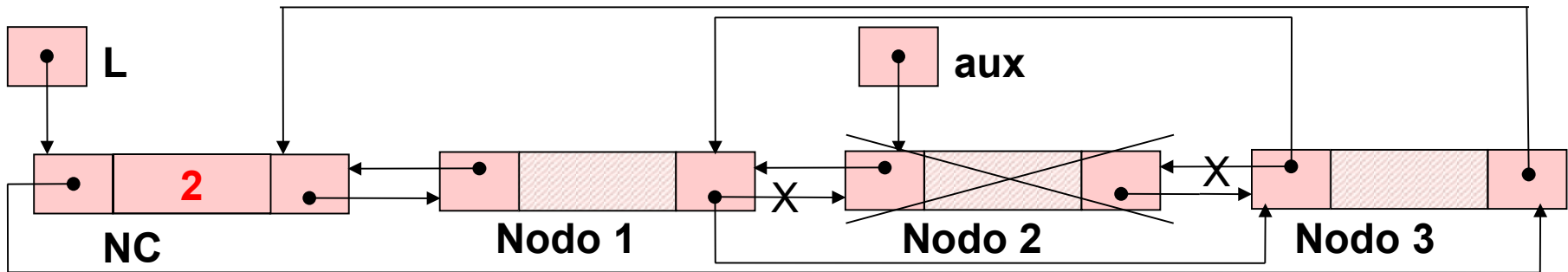
## Dicas:

A posição é válida?

Todas as situações de remoção são tratadas da mesma forma?

# Listas Dup. Encad. Circ. com NC

Esquema do processo da retirada de um nó da lista circular duplamente encadeada com nó cabeçalho.



```
1 void ret (LISTA_CIR_DUP_ENC_NC l, int k)
2 {
3     if (k < 1 || k > tam(l))
4     {
5         printf ("\nERRO! Posição invalida para retirada.\n");
6         exit (4);
7     }
8     l->inf--;
9     for (; k>0; k--, l=l->prox);
10    l->ant->prox = l->prox;
11    l->prox->ant = l->ant;
12    free (l);
13 }
```

## Listas Dup. Encad. Circ. com NC

Com base no que foi visto implemente a operação destruir() que compõem o TAD LISTA\_CIR\_DUP\_ENC\_NC.

```
typedef struct nodo
{
    int inf;
    struct nodo * ant;
    struct nodo * prox;
}NODO;
typedef NODO * LISTA_CIR_DUP_ENC_NC;
void cria_lista (LISTA_CIR_DUP_ENC_NC *);
int eh_vazia (LISTA_CIR_DUP_ENC_NC);
int tam (LISTA_CIR_DUP_ENC_NC);
void ins (LISTA_CIR_DUP_ENC_NC, int, int);
int recup (LISTA_CIR_DUP_ENC_NC, int);
void ret (LISTA_CIR_DUP_ENC_NC, int);
void destruir (LISTA_CIR_DUP_ENC_NC);
```

## Listas Dup. Encad. Circ. com NC

Implemente, no TAD LISTA\_CIR\_DUP\_ENC\_NC, a seguinte operação:

```
void inverter_lista (LISTA_CIR_DUP_ENC_NC I);
```

a qual recebe uma referência para uma lista circular duplamente encadeada com nó cabeçalho e inverte a ordem de seus elementos.