Listas Duplamente Encadeadas

Com base no que foi visto implemente a operação recup() 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);
```

```c
int recup (LISTA_DUP_ENC l, int k)
{
    if (k < 1 || k > tam(l))
    {
        printf ("\nERRO! Consulta invalida.\n");
        exit (3);
    }
    for (;k>1;l=l->prox,k--) ;
    return (l->inf);
}
```

Listas Duplamente Encadeadas

Com base no que foi visto implemente a operação ret() que compõem o T A D 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);
```

Listas Duplamente Encadeadas

Dicas:

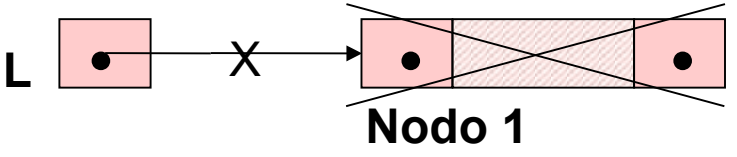
A posição é válida?

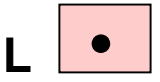
Todas as situações de remoção são tratadas da mesma forma?

UNIVASF

Listas Duplamente Encadeadas

Esquema do processo da retirada de um nó da lista duplamente encadeada. (situação um, remoção do único elemento)

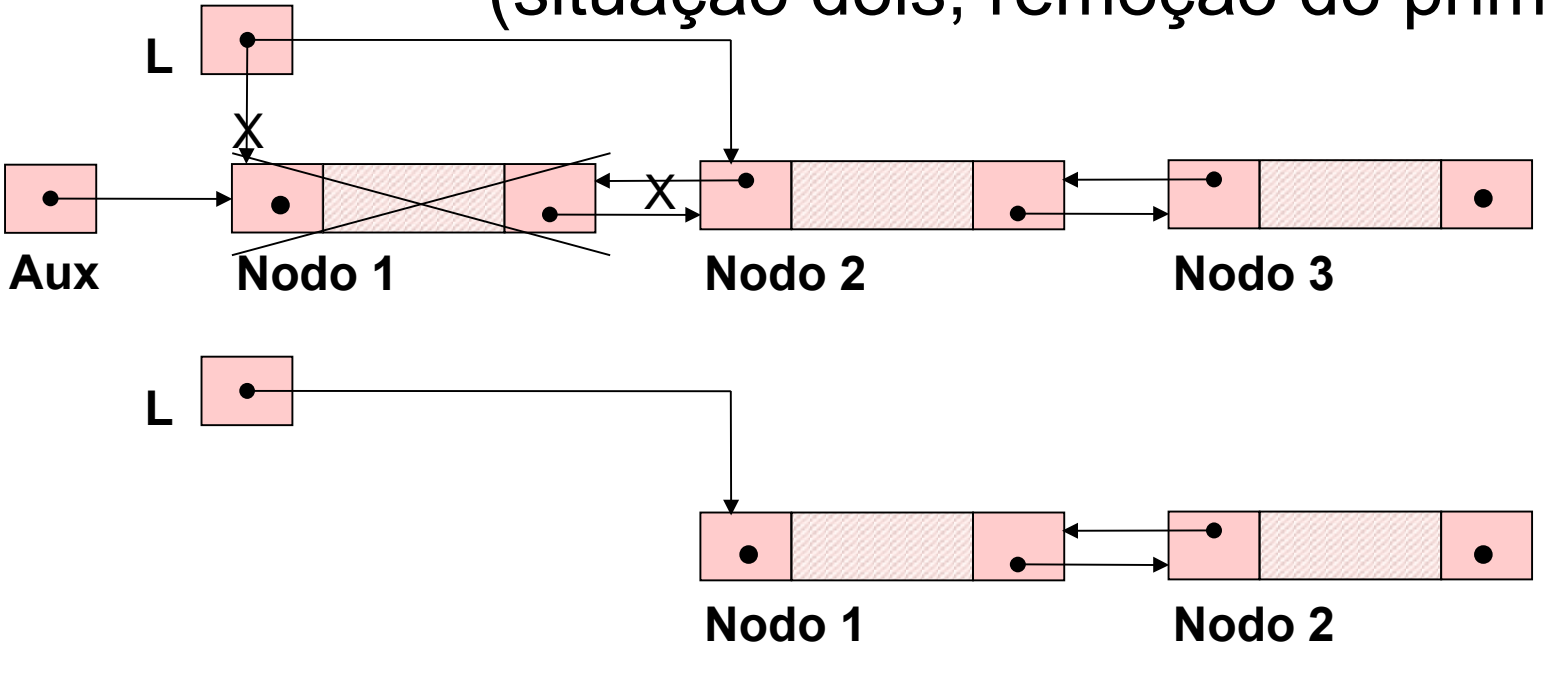




Listas Duplamente Encadeadas

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

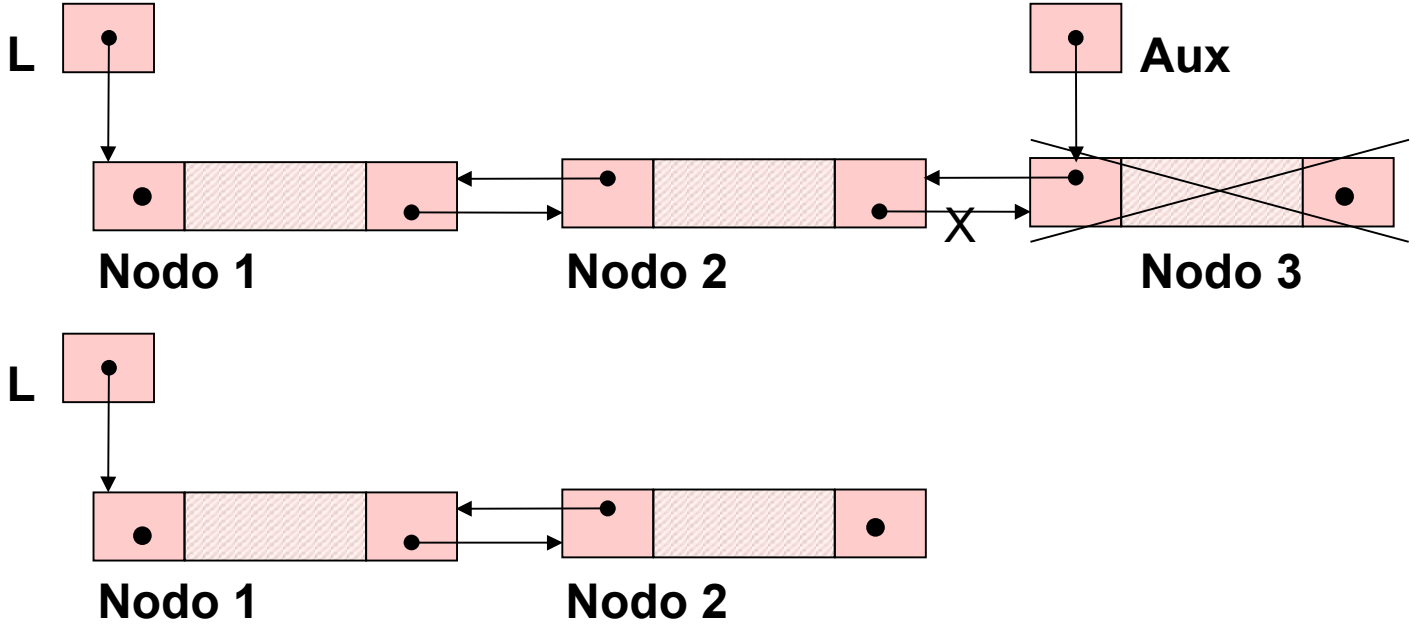
(situação dois, remoção do primeiro)



Listas Duplamente Encadeadas

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

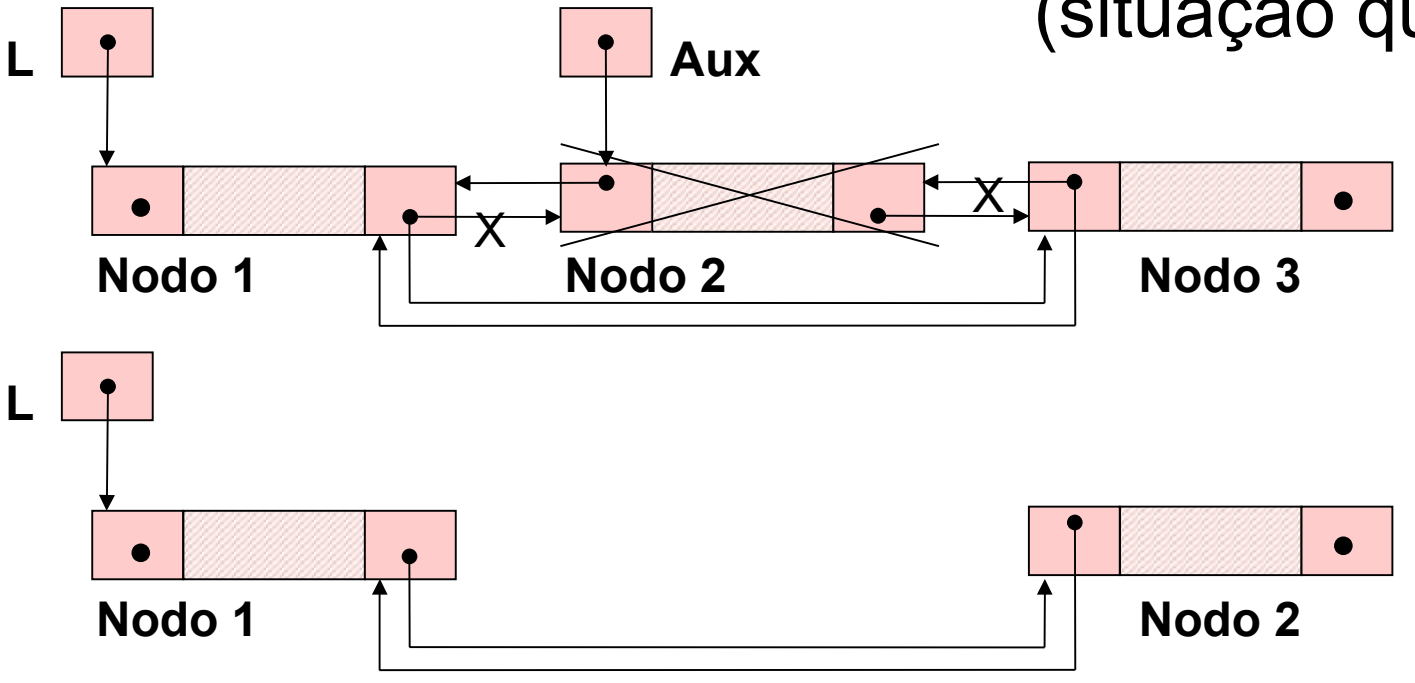
(situação três, remoção do último)



Listas Duplamente Encadeadas

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

(situação quatro)



```c
void ret (LISTA_DUP_ENC *pl, int k) {
   LISTA_DUP_ENC aux;
   if (k < 1 || k > tam(*pl)) {
      printf ("\nERRO! Posição invalida para retirada.\n");
      exit (4);
   }
   if (k==1) { /*situações um e dois*/
      aux = *pl;
      *pl = aux->prox;
      if (*pl) /*situação dois*/
         (*pl)->ant=NULL;
      free (aux);
   } else { /*situações três e quatro*/
      for (aux=(*pl)->prox; k>2; k--, aux=aux->prox);
      aux->ant->prox = aux->prox;
      if (aux->prox) /*situação quatro*/
         aux->prox->ant = aux->ant;
      free (aux);
   }
}
```

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);
```

```c
void destruir (LISTA_DUP_ENC l)
{
    LISTA_DUP_ENC aux;
    while (l)
    {
        aux = l;
        l = l->prox;
        free (aux);
    }
}
```

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.

UNIVASF

```c
void inverter_lista (LISTA_DUP_ENC *pl)
{
    if (*pl) {
        LISTA_DUP_ENC aux; /* NODO *aux; */
        do {
            aux=(*pl)->prox;
            (*pl)->prox=(*pl)->ant;
            (*pl)->ant=aux;
            if (aux)
                *pl=aux;
        }while (aux);
    }
}
```

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);
```

UNIVASF

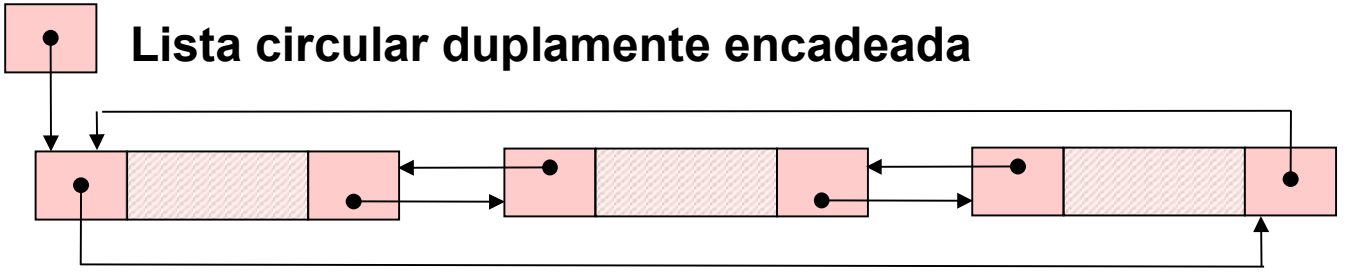
Listas Circulares Duplamente Encadeadas com Nó Cabeçalho

UNIVASF

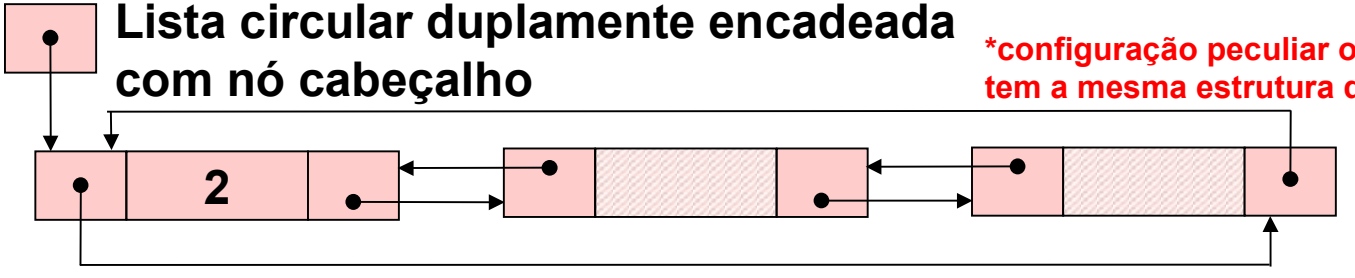
Listas Dup. Encad. Circ. com NC

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

Lista circular duplamente encadeada



Lista circular duplamente encadeada com nó cabeçalho



***configuração peculiar onde o nó cabeçalho tem a mesma estrutura do no elemento**

```c
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

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);
```

```c
void cria_lista (LISTA_CIR_DUP_ENC_NC *pl)
{
    *pl = (LISTA_CIR_DUP_ENC_NC) malloc (sizeof(NODO));
    if (!(*pl))
    {
        printf ("\nERRO! Memoria insuficiente!\n");
        exit (2);
    }
    (*pl)->inf=0;
    (*pl)->ant=(*pl)->prox=*pl;
}
```

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);
```

```c
int eh_vazia (LISTA_CIR_DUP_ENC_NC l)
{
    return (l->inf == 0);
}
```

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);
```

```c
int tam (LISTA_CIR_DUP_ENC_NC l)
{
    return (l->inf);
}
```

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?

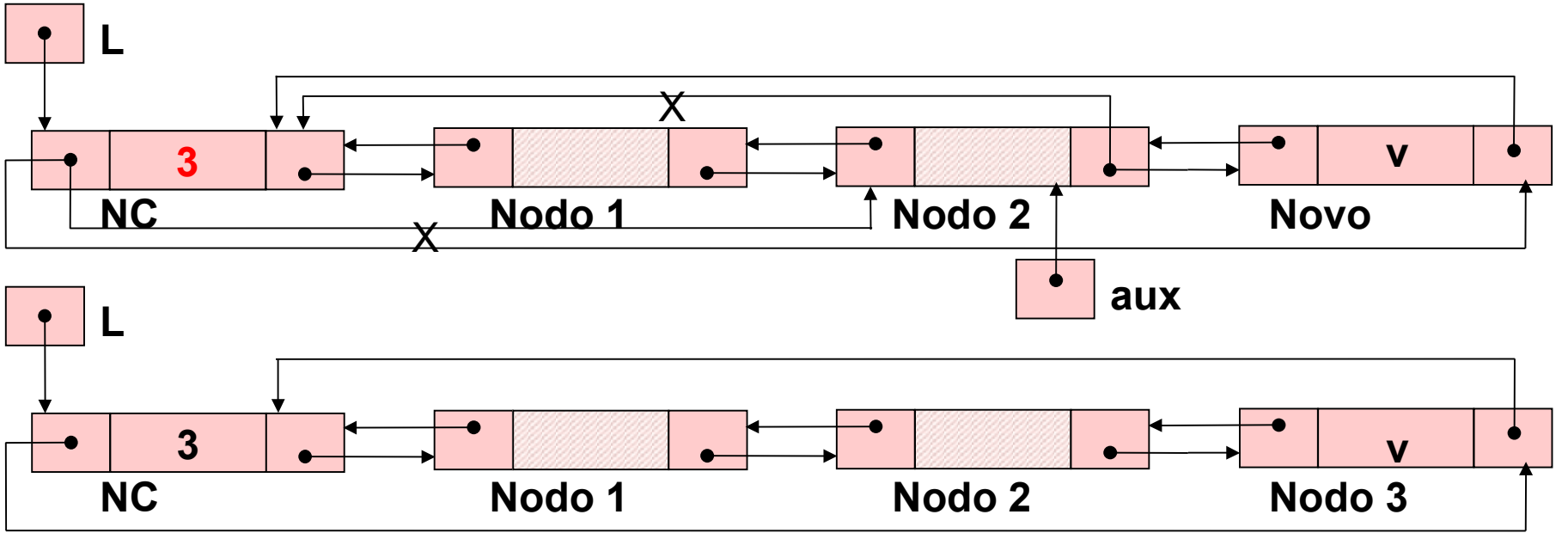
Tem espaço na memória para armazenar mais um elemento?

Todas as situações de inserção são tratadas da mesma forma?

UNIVASF

Listas Dup. Encad. Circ. com NC

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



```c
void ins (LISTA_CIR_DUP_ENC_NC l, int v, int k)
{
    LISTA_CIR_DUP_ENC_NC aux, novo;
    if (k < 1 || k > tam(l)+1) {
        printf ("\nERRO! Posição invalida para insercao.\n");
        exit (1);
    }
    novo = (LISTA_CIR_DUP_ENC_NC) malloc (sizeof(NODO));
    if (!novo) {
        printf ("\nERRO! Memoria insuficiente!\n");
        exit (2);
    }
```

```
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);
```

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);
```

UNIVASF

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);
```