

Listas Duplamente Encadeadas

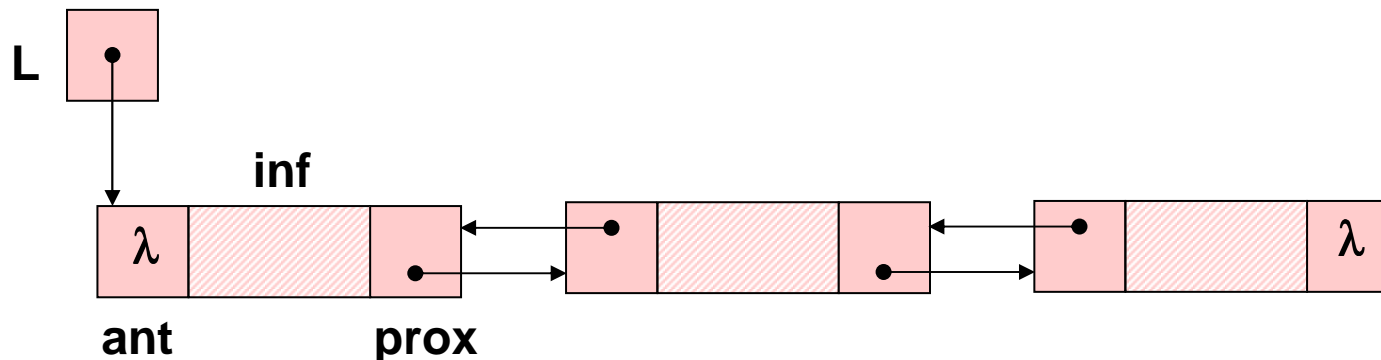
Como vimos, uma lista circular possui vantagens sobre uma lista linear, contudo esta ainda possui limitações.

Por exemplo, não podemos percorrê-la no sentido contrário ou ainda para inserirmos ou retirarmos um k -ésimo elemento temos que ter um ponteiro para seu antecessor.

Com o objetivo de sanar estas limitações surgiram as *listas duplamente encadeadas*.

Listas Duplamente Encadeadas

Em uma *lista duplamente encadeada* os elementos possuem três campos: o campo *inf* o qual contém a informação, o campo *ant* que possui um ponteiro para o elemento antecessor e o campo *prox* que é uma referência para o elemento que sucede.



Listas Duplamente Encadeadas

Definiremos agora o TAD LISTA_DUP_ENC:

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

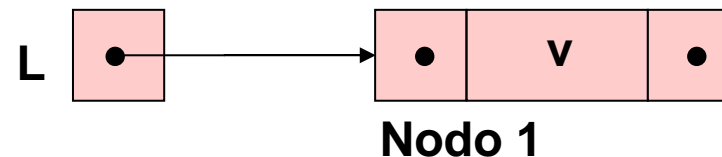
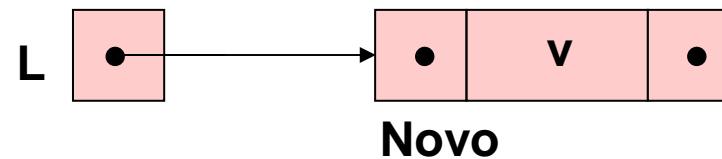
```

void cria_lista (LISTA_DUP_ENC *pl)
{
    *pl=NULL;
}
int eh_vazia (LISTA_DUP_ENC l)
{
    return (l == NULL);
}
int tam (LISTA_DUP_ENC l)
{
    int cont;
    for (cont=0; l!= NULL; cont++)
        l = l->prox;
return (cont); }

```

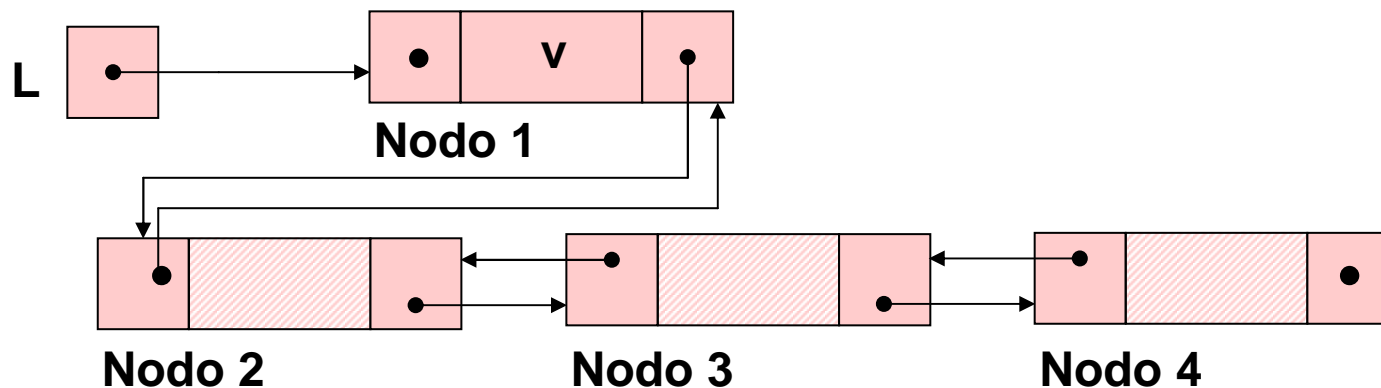
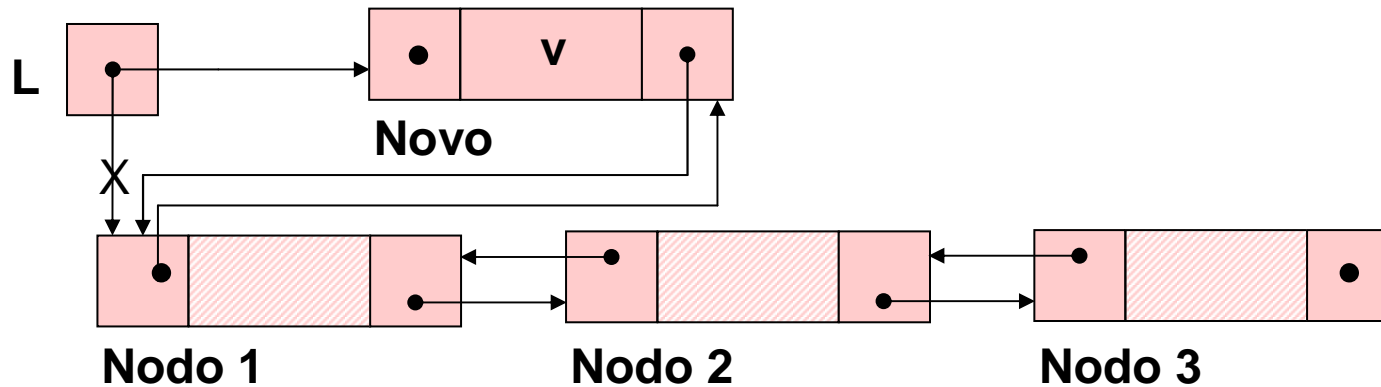
Listas Duplamente Encadeadas

Esquema do processo da inserção de um nó da lista duplamente encadeada. (situação um)



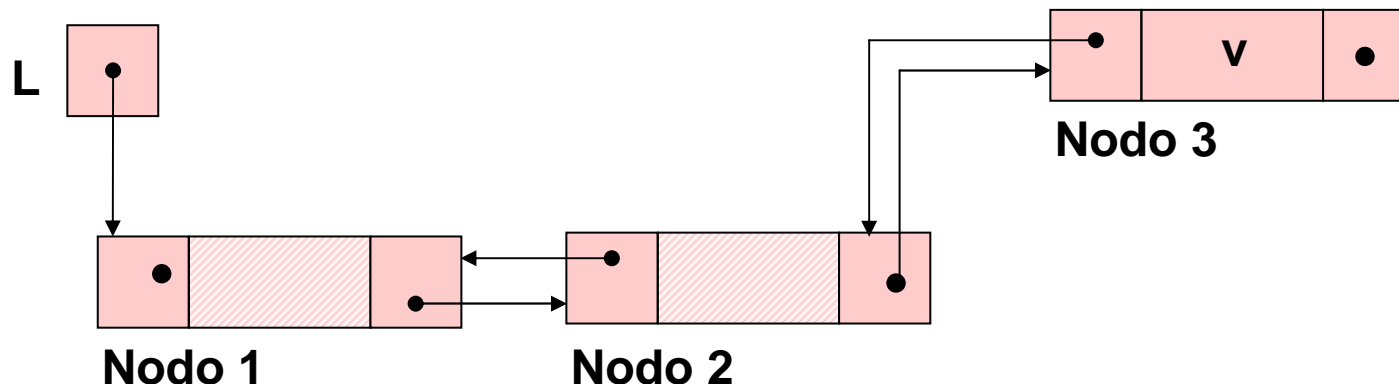
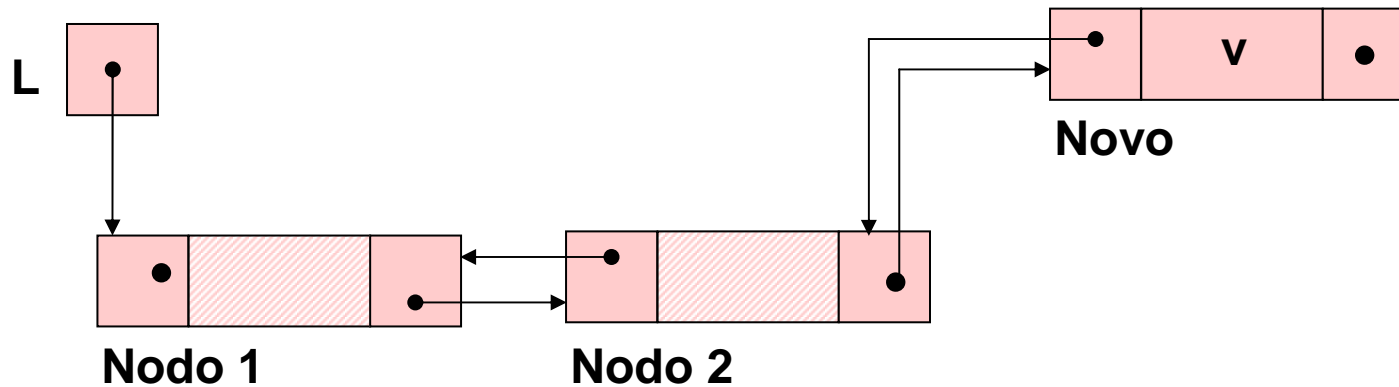
Listas Duplamente Encadeadas

Esquema do processo da inserção de um nó da lista duplamente encadeada. (situação dois)



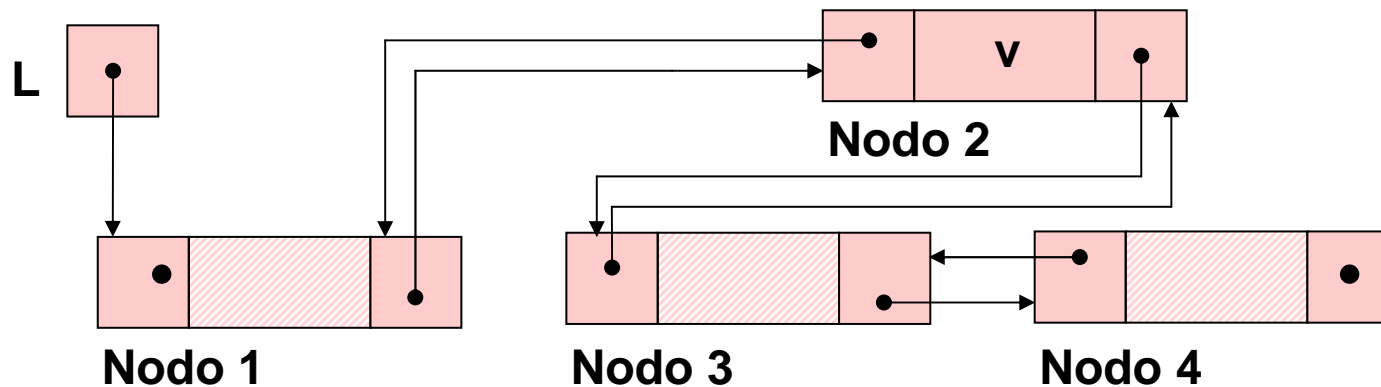
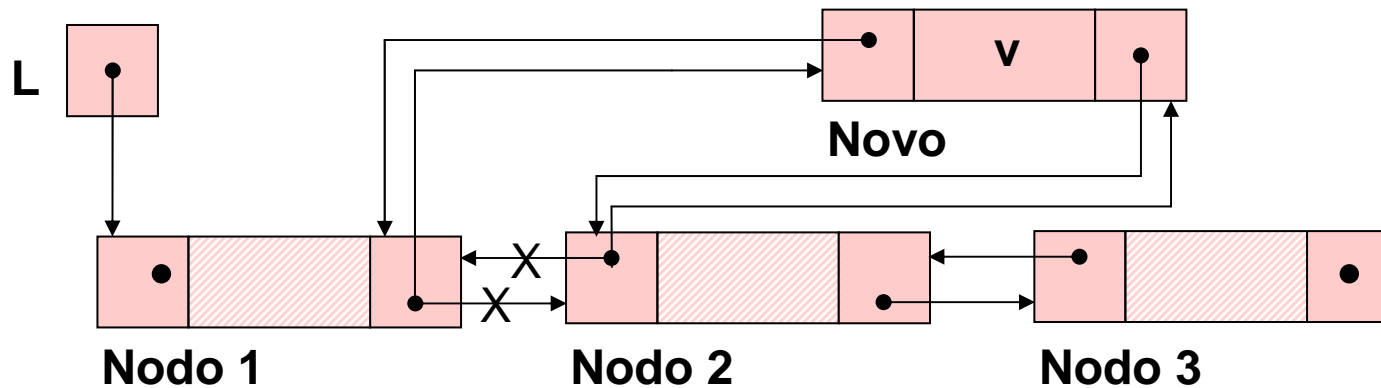
Listas Duplamente Encadeadas

Esquema do processo da inserção de um nó da lista duplamente encadeada. (situação três)



Listas Duplamente Encadeadas

Esquema do processo da inserção de um nó da lista duplamente encadeada. (situação quatro)




```

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

```

```
novo->inf = v;  
if (k==1)  
{  
    novo->ant = NULL;  
    novo->prox = *pl;  
    *pl = novo;  
    if ((*pl)->prox)  
        (*pl)->prox->ant=novo;  
}  
else  
{  
    LISTA_DUP_ENC aux;  
    for (aux=*pl; k>2; aux=aux->prox, k--);
```

```
novo->prox = aux->prox;
```

```
aux->prox = novo;
```

```
novo->ant=aux;
```

```
if (novo->prox)
```

```
    novo->prox->ant=novo;
```

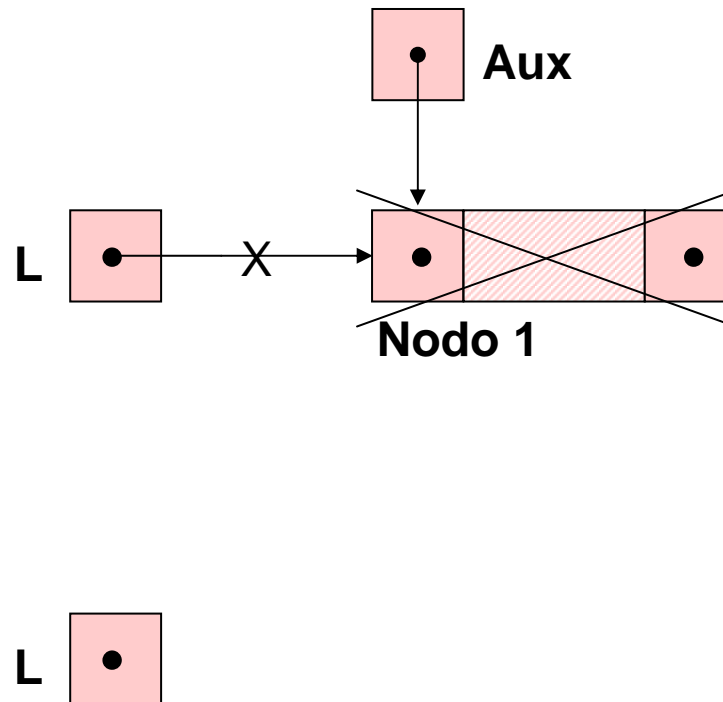
```
}
```

```
}
```

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

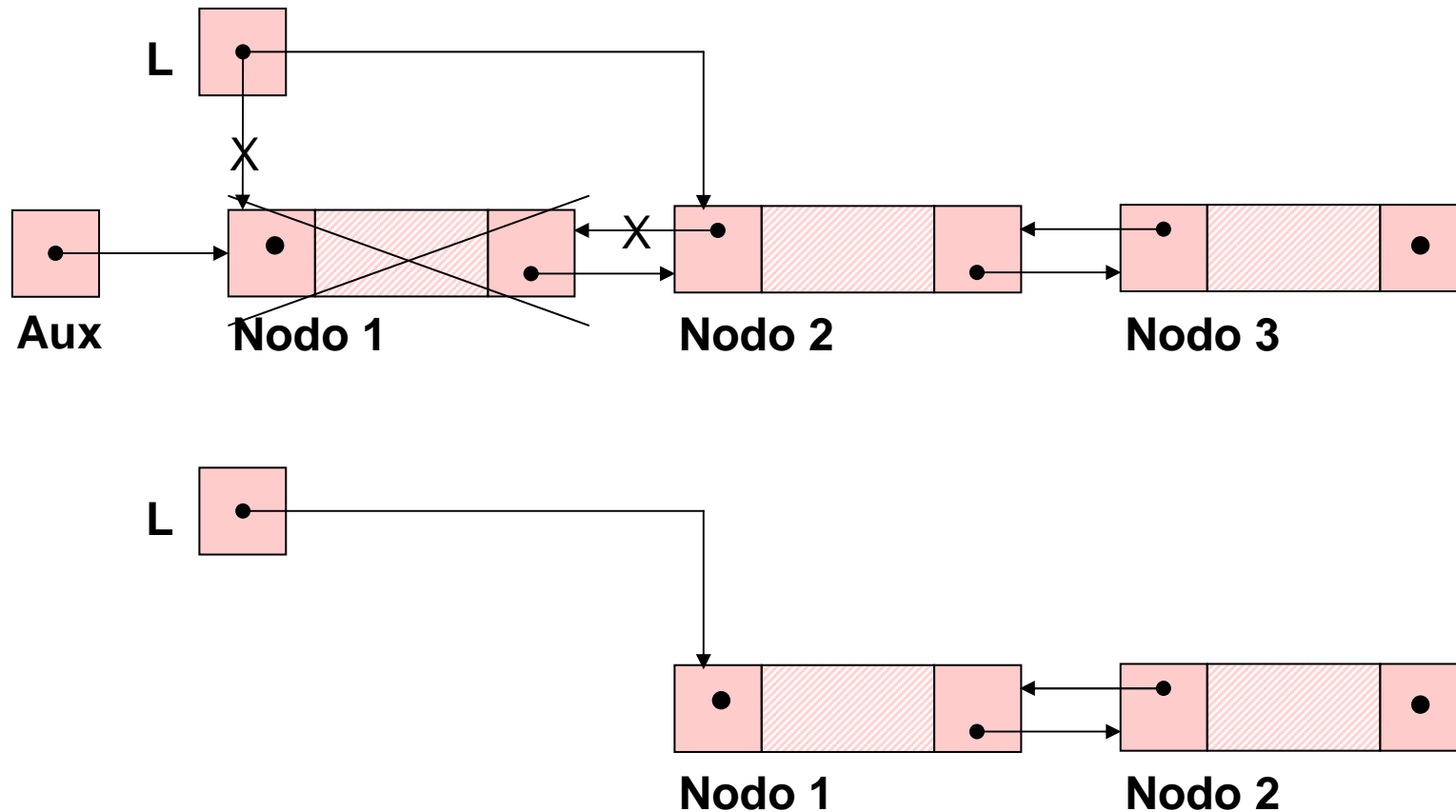
Listas Duplamente Encadeadas

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



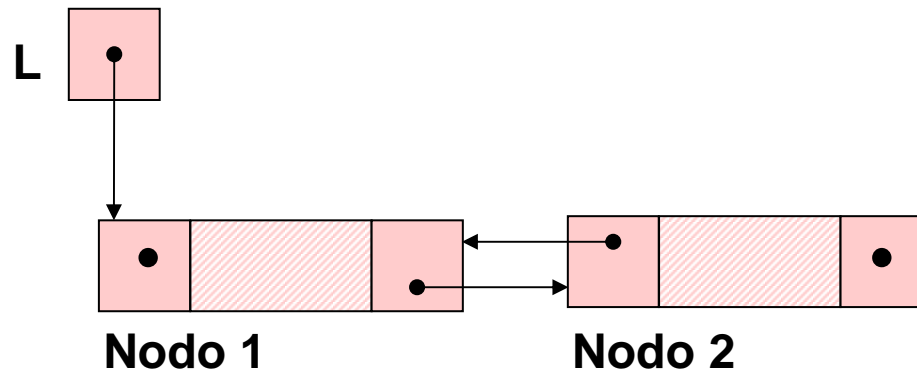
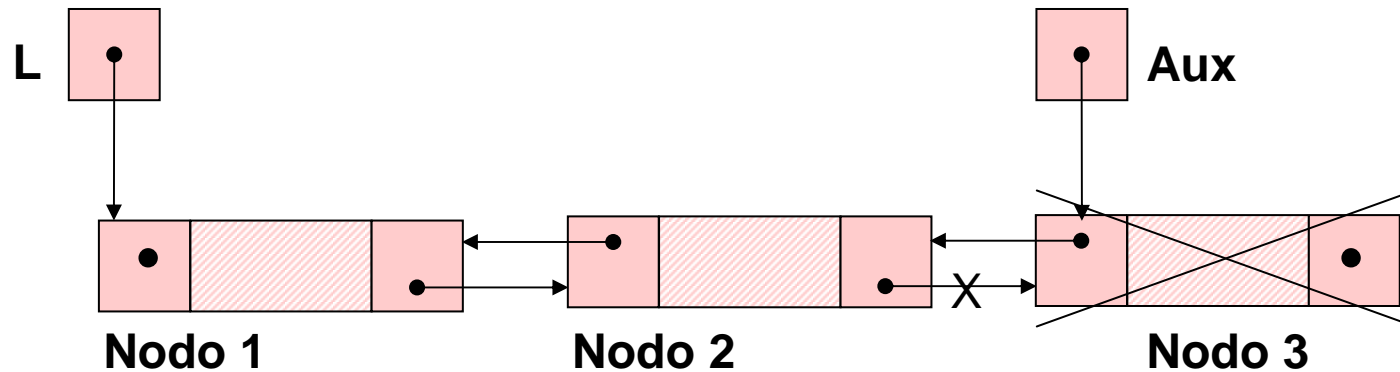
Listas Duplamente Encadeadas

Esquema do processo da retirada de um nó da lista duplamente encadeada. (situação dois)



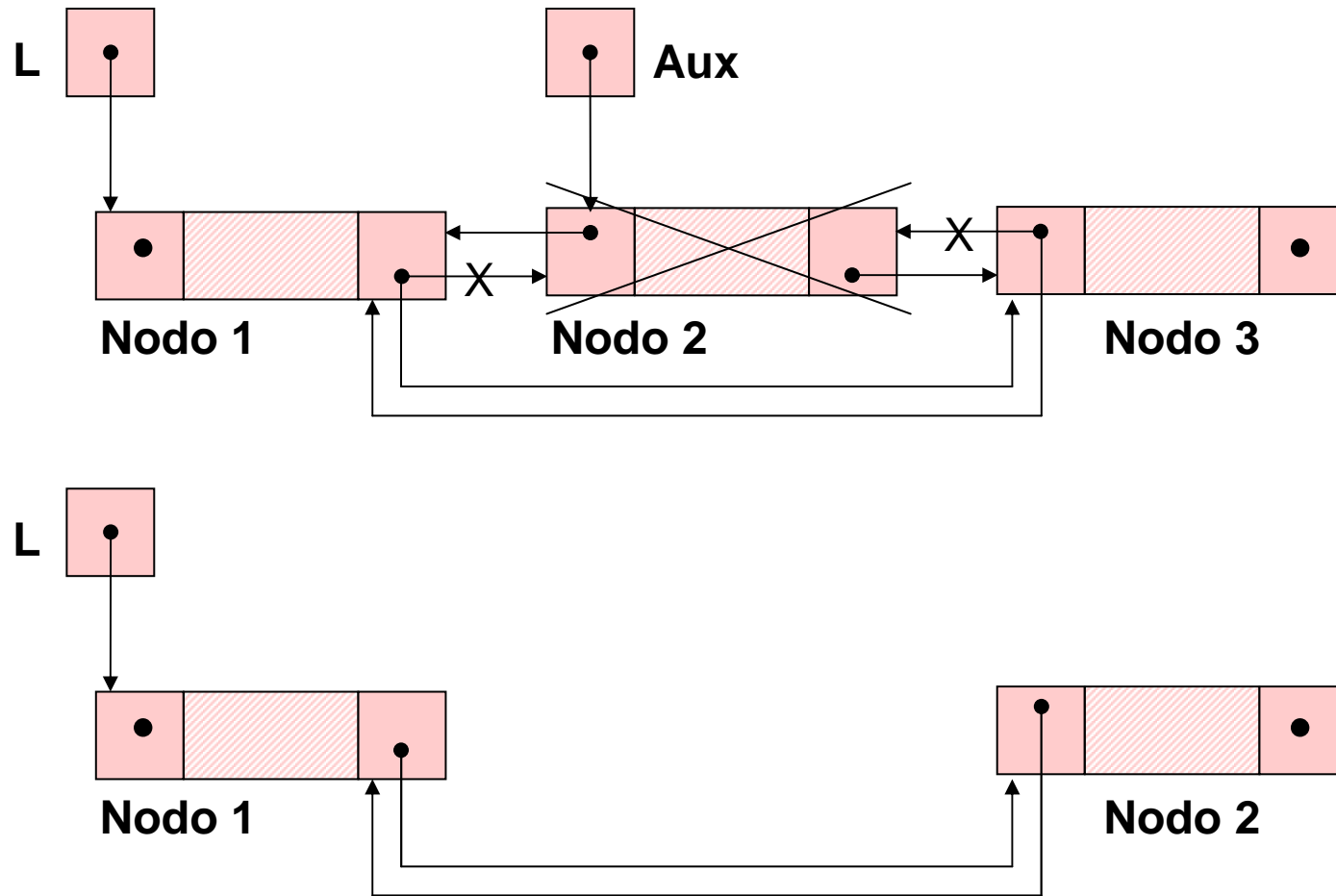
Listas Duplamente Encadeadas

Esquema do processo da retirada de um nó da lista duplamente encadeada. (situação três)



Listas Duplamente Encadeadas

Esquema do processo da retirada de um nó da lista duplamente encadeada. (situação quatro)




```
void ret (LISTA_DUP_ENC *pl, int k)  
{  
    NODO *aux;  
    if (k < 1 || k > tam(*pl))  
    {  
        printf ("\nERRO! Posição invalida para  
        retirada.\n");  
        exit (4);  
    }  
    if (k==1)  
    {  
        aux = *pl;  
        *pl = aux->prox;
```

```
    if (*pl)
        (*pl)->ant=NULL;
    free (aux);
}
else
{
    for (aux=(*pl)->prox; k>2; k--, aux=aux->prox);
    aux->ant->prox = aux->prox;
    if (aux->prox)
        aux->prox->ant = aux->ant;
    free (aux);
}
}
```

Listas Duplamente Encadeadas – Exercício

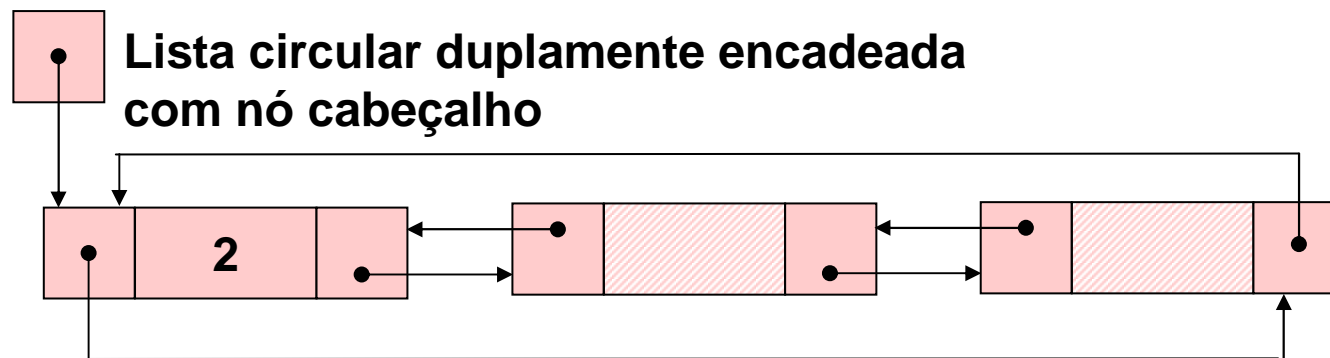
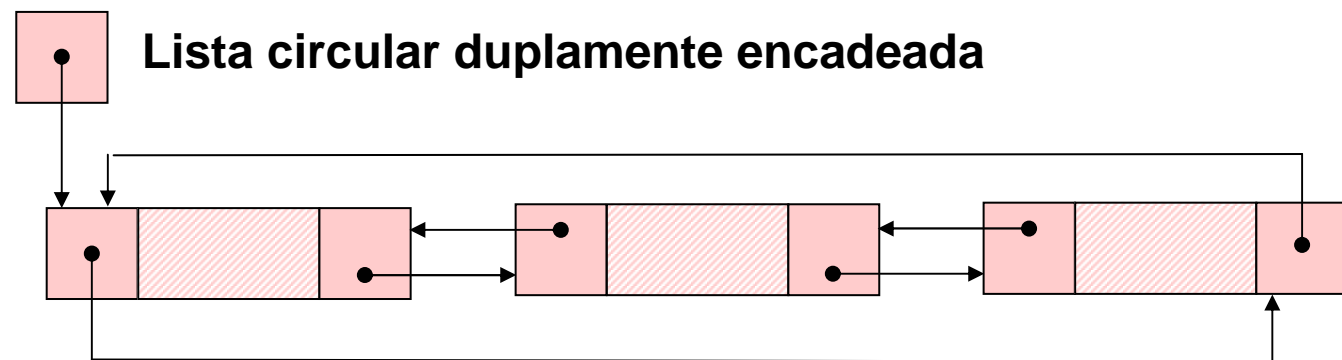
Implemente, no TAD LISTA_DUP_ENC, a seguinte operação:

void inverter_lista (LISTA_DUP_ENC *pl);

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

Listas Duplamente Encadeadas

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



Listas Duplamente Encadeadas

Para uma melhor fixação definiremos agora 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);
```

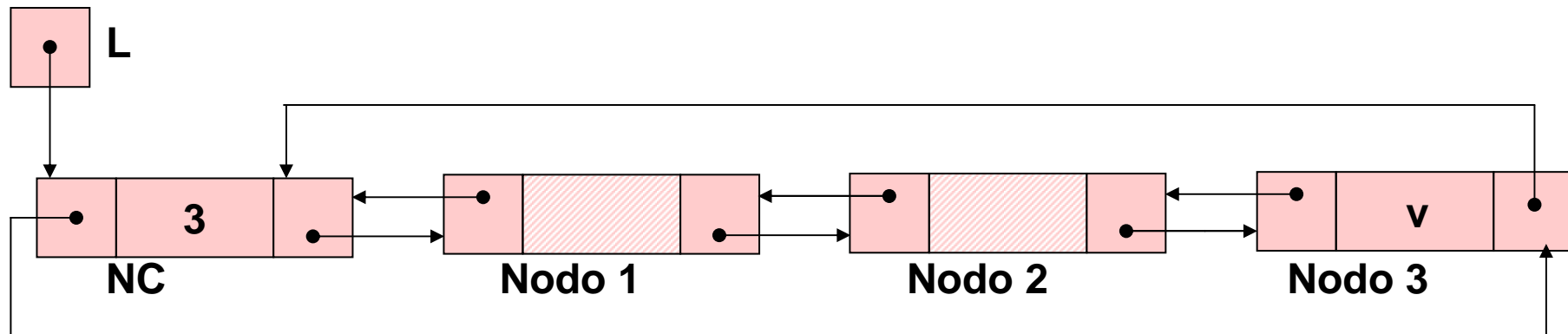
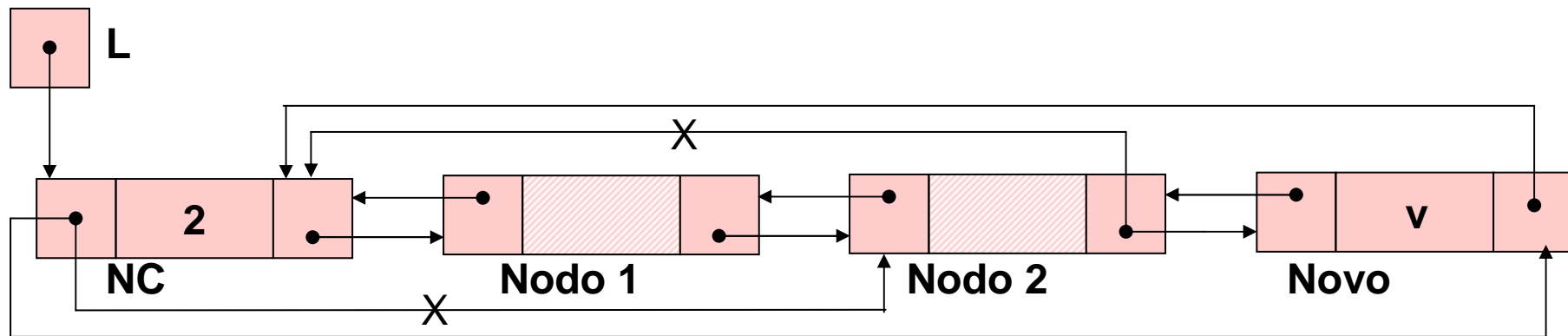
```
void cria_lista (LISTA_CIR_DUP_ENC_NC *pl)  
{  
    NODO *novo;  
    novo = (NODO *) malloc (sizeof(NODO));  
    if (!novo)  
    {  
        printf ("\nERRO! Memoria insuficiente!\n");  
        exit (2);  
    }  
    novo->inf=0;  
    *pl=novo->ant=novo->prox=novo;  
}
```

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

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

Listas Duplamente Encadeadas

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




```

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 ("\n%d %d\n", tam(l)+1, k);
        printf ("\nERRO! Posição invalida para
insercao.\n");
        exit (1);
    }
    novo = (NODO *) malloc (sizeof(NODO));
    if (!novo)
    {
        printf ("\nERRO! Memoria insuficiente!\n");
        exit (2);
    }
}

```

```
novo->inf = v;  
for (aux=l; k>1; aux=aux->prox, k--);  
novo->prox = aux->prox;  
novo->ant = aux;  
aux->prox = novo;  
novo->prox->ant=novo;  
l->inf++;  
}
```