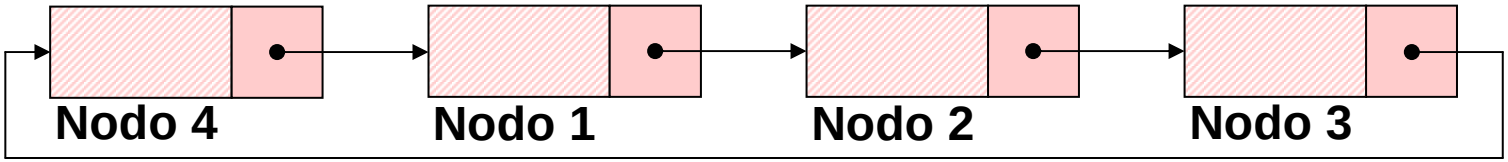
Listas Circulares

Listas lineares encadeadas possuem algumas deficiências. Como, por exemplo, dado um ponteiro p para um nodo de uma lista linear encadeada, não podemos atingir nenhum nodo que antecede o apontado por p.

Contudo, se fizermos uma pequena alteração na estrutura de lista que temos trabalhado, fazendo com que o campo next do último nodo ao invés de conter NULL armazene o endereço do primeiro nodo da lista.

Listas Circulares

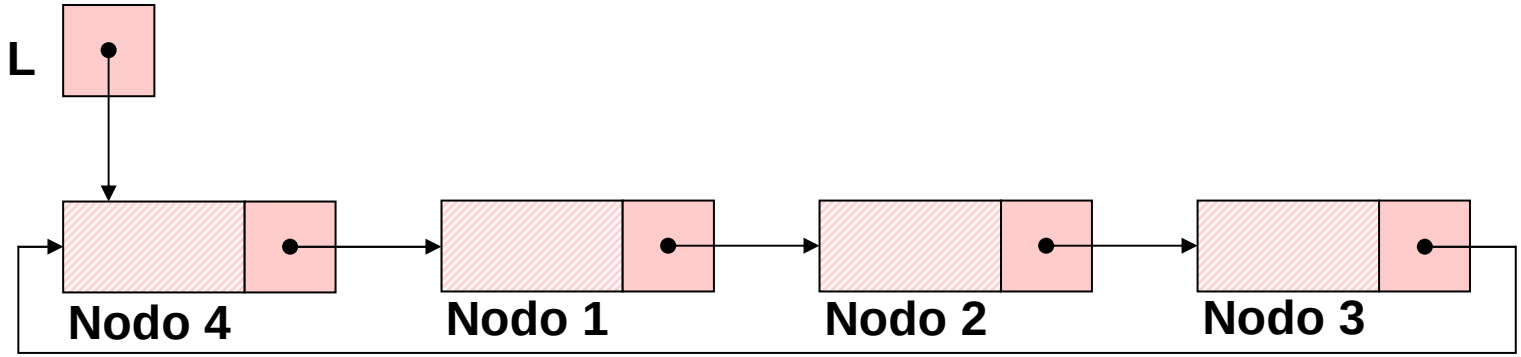
Este tipo de lista é denominado lista circular, possuindo a forma abaixo:



Uma pergunta surge: para qual elemento apontar para que se tenha uma referência a lista circular?

Listas Circulares

Uma convenção útil é fazer com que o ponteiro esterno para a lista circular aponte para o último elemento.



Pois desta forma se tem acesso direto ao último e primeiro elemento.

Listas Circulares

Como podemos observar a definição do TAD LISTA_CIRCULAR é praticamente a mesma do TAD LISTA_ENC, apenas algumas pequenas alterações são necessárias nas operações.

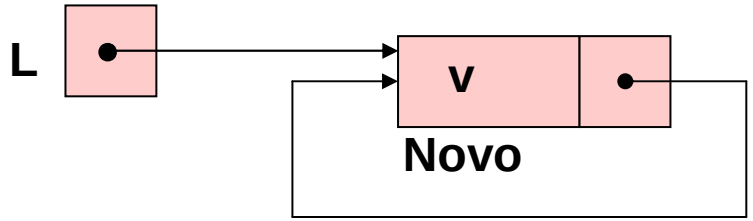
```
typedef struct nodo
{
   int inf;
   struct nodo * next;
}NODO;
typedef NODO * LISTA_CIRCULAR;
void cria_lista (LISTA_CIRCULAR *);
int eh_vazia (LISTA_CIRCULAR);
int tam (LISTA_CIRCULAR);
void ins (LISTA_CIRCULAR *, int, int);
int recup (LISTA_CIRCULAR, int);
void ret (LISTA_CIRCULAR *, int);
```

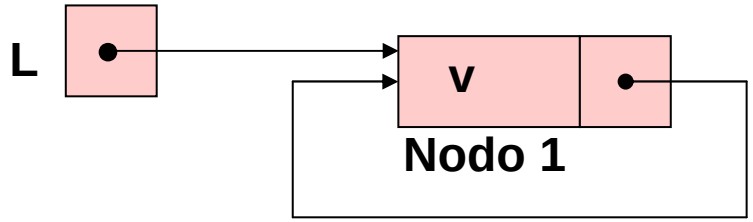
```
void cria_lista (LISTA_CIRCULAR *pl)
{
   *pl=NULL;
}

int eh_vazia (LISTA_CIRCULAR l)
{
   return (l == NULL);
}
```

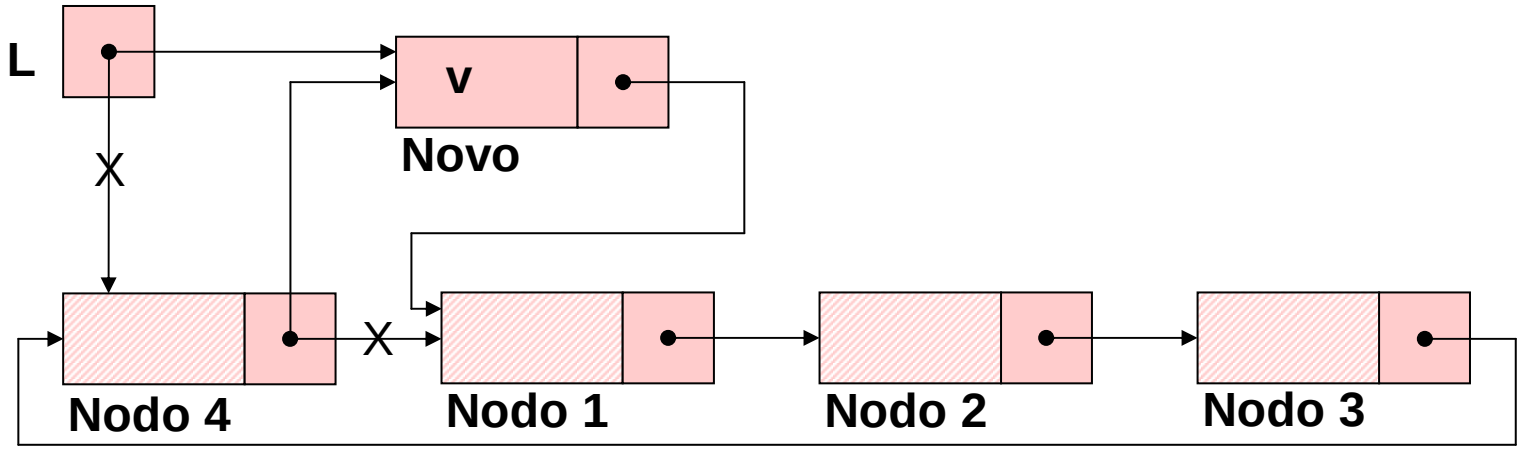
```
int tam (LISTA_CIRCULAR l)
{
   if (l==NULL)
      return (0);
   else
   {
      LISTA_CIRCULAR aux;
      int cont;
      for (cont=1, aux=l->next; aux!=l ;
     cont++)
         aux = aux->next;
      return (cont);    } }
```

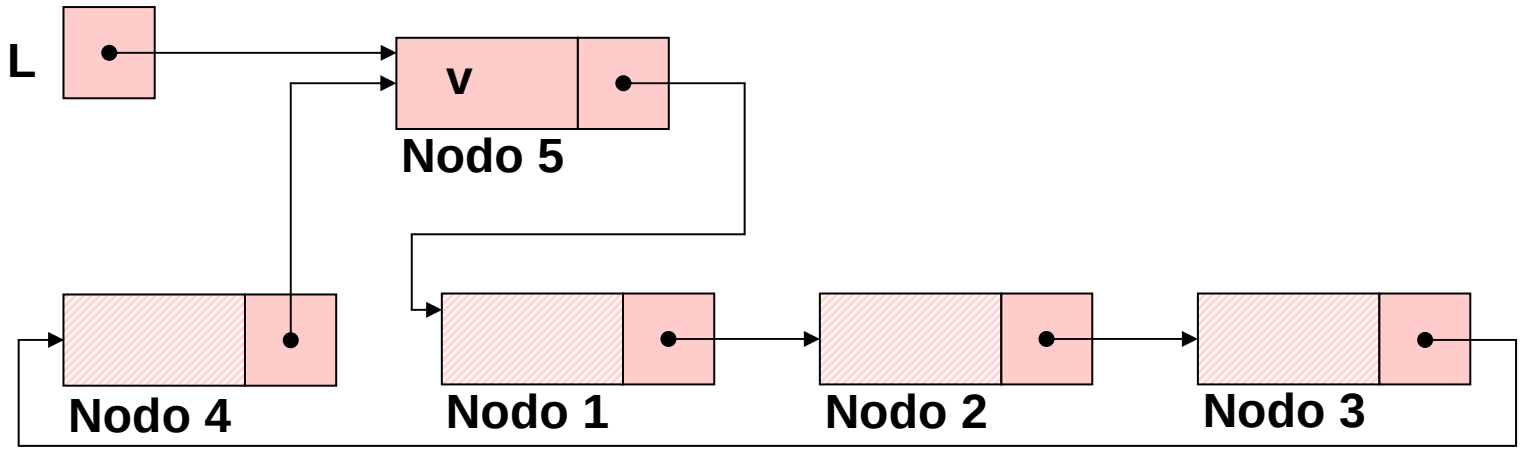
Esquema do processo da inserção de um nó da lista circular. (situação um)



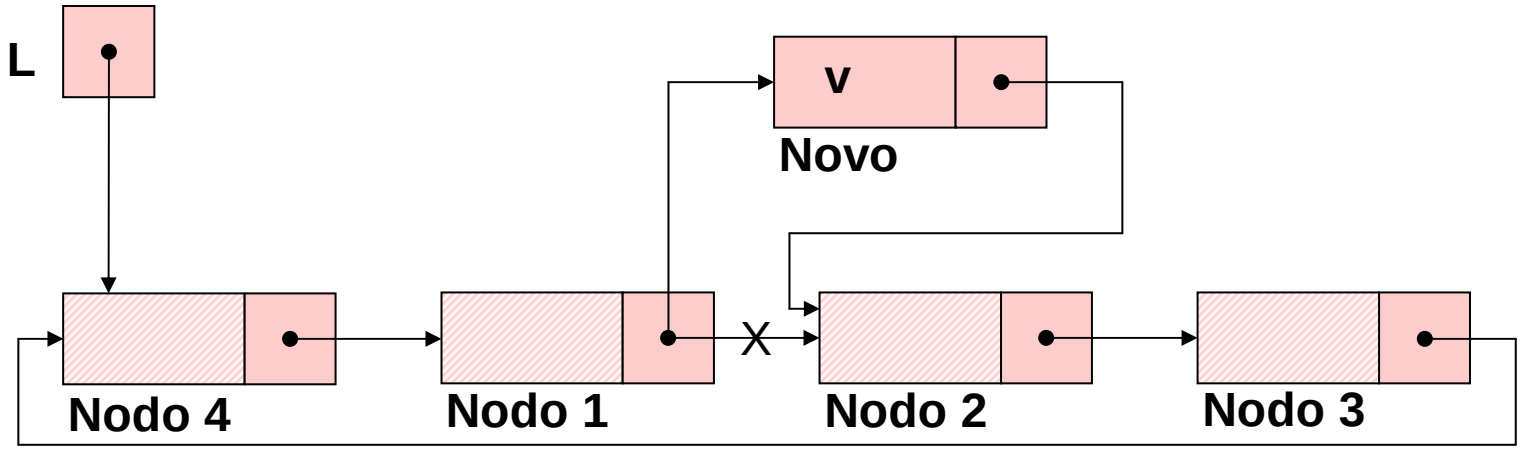


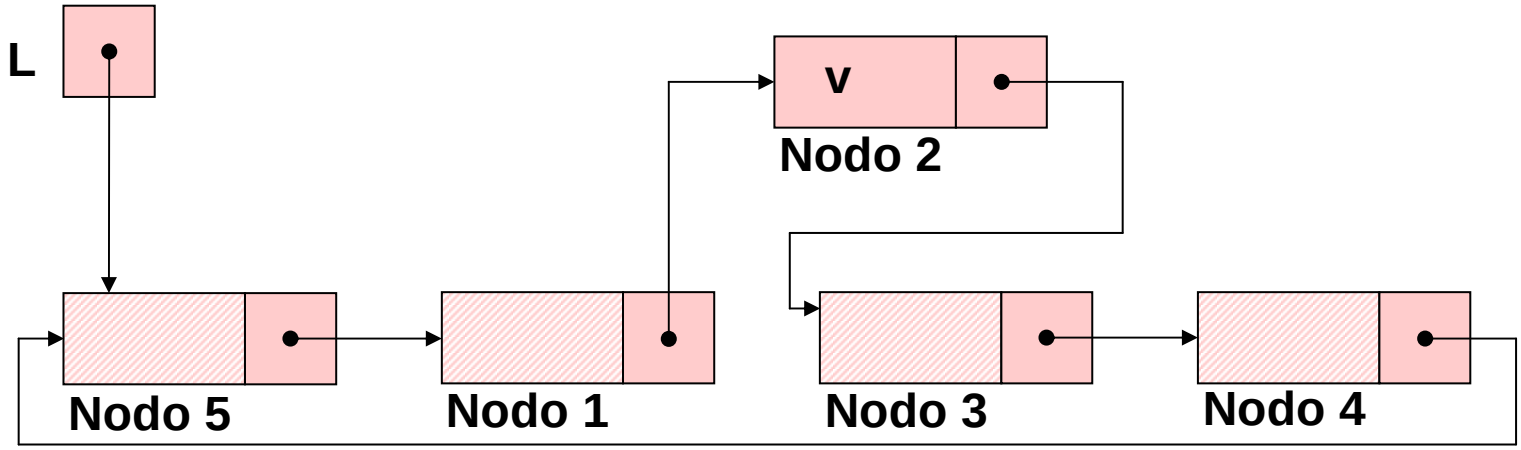
Esquema do processo da inserção de um nó da lista circular. (situação dois)





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



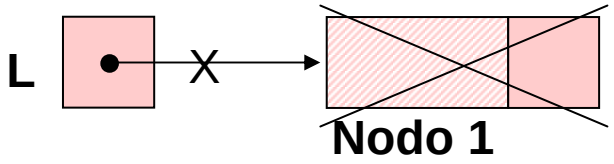


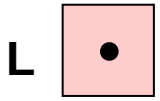
```
void ins (LISTA_CIRCULAR *pl, int v, int k)
{
  NODO *novo;
  if (k < 1 || k > tam(*pl)+1)
  {
    printf ("\nERRO! Posição invalida para insercao.\n");
    exit (1);
  }
  novo = (NODO *) malloc (sizeof(NODO));
  if (!novo)
  {
    printf ("\nERRO! Memoria insuficiente!\n");
    exit (2);
  }
```

```
novo->inf = v;
if (*pl==NULL)    {
   novo->next=novo;
   *pl = novo;    }
else
{
   LISTA_CIRCULAR aux=*pl;
   if (k==tam(*pl)+1)
      *pl=novo;
   for (; k>1; aux=aux->next, k--);
   novo->next = aux->next;
   aux->next = novo;      } }
```

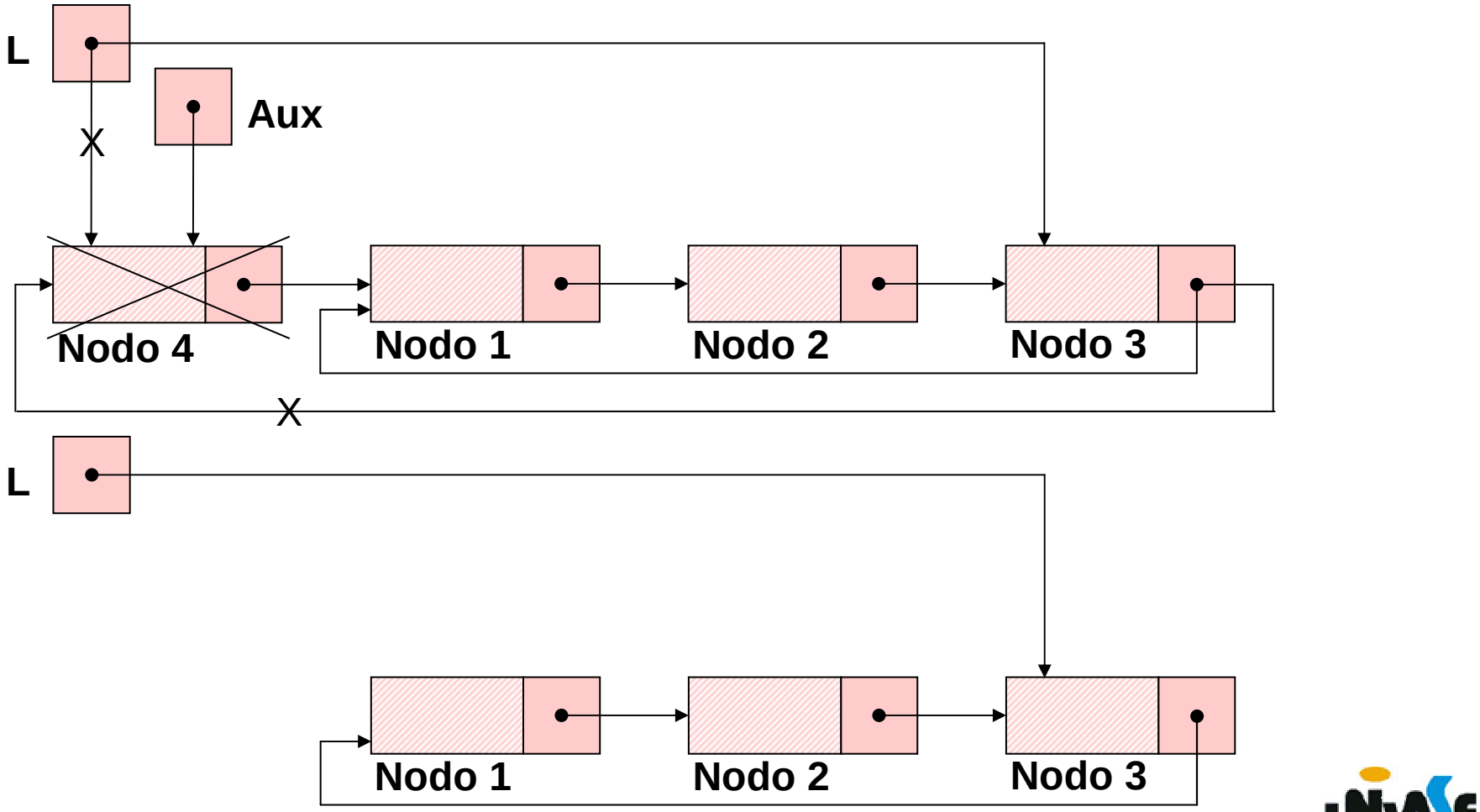
```
int recup (LISTA_CIRCULAR l, int k)
{
   if (k < 1 || k > tam(l))
   {
      printf ("\nERRO! Consulta invalida.\n");
      exit (3);
   }
   for (;k>0;k--)
      l=l->next;
   return (l->inf);
}
```

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



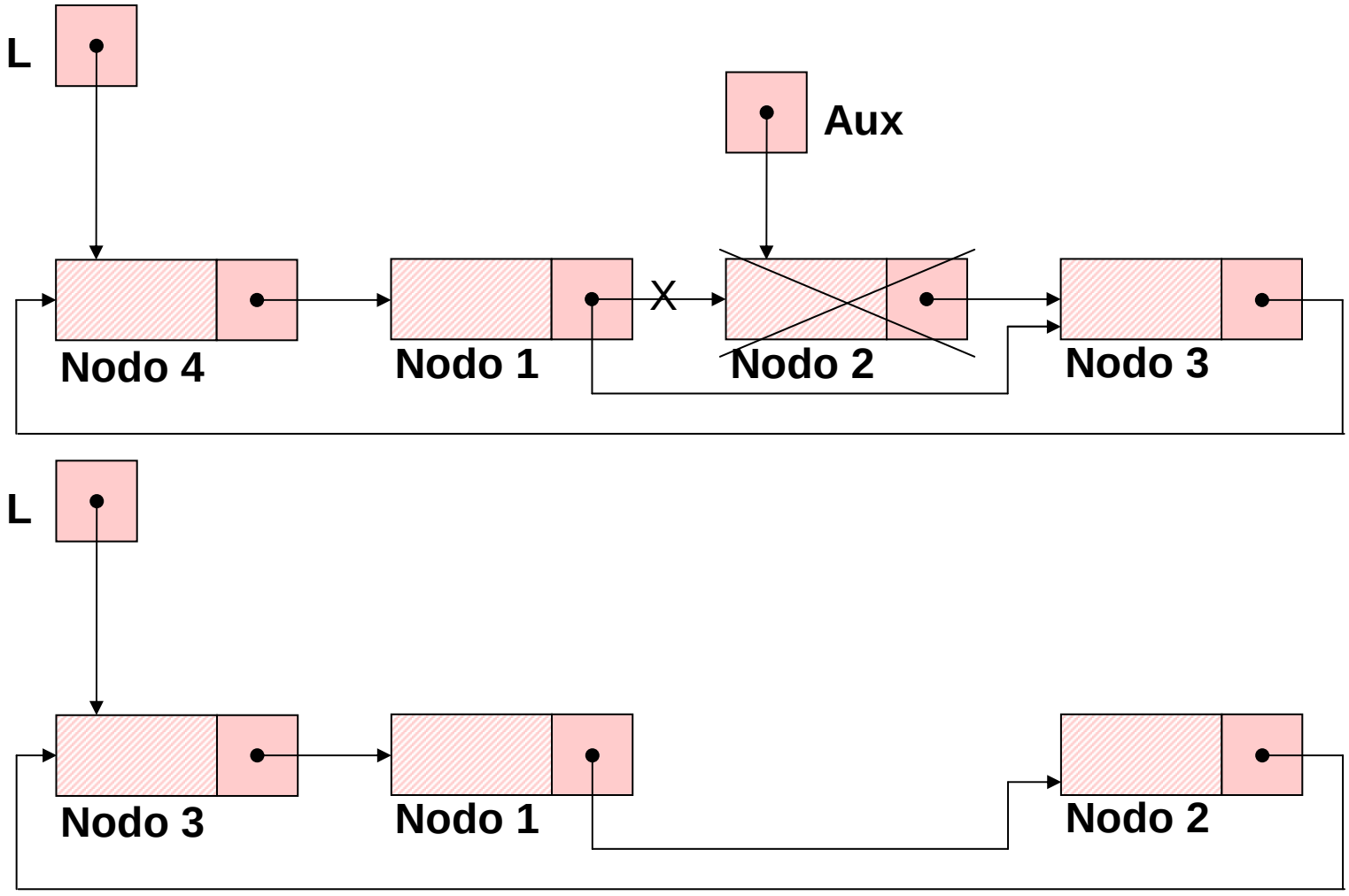


Esquema do processo da retirada de um nó da lista circular. (situação dois)



Listas Circulares

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



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

```
    else
    {
       NODO *aux, *aux2;
       int i;
       for (aux=*pl, i=k; i>1; i--, aux=aux->next);
       aux2 = aux->next;
       aux->next = aux2->next;
       if (k==tam(*pl))
          *pl=aux;
       free (aux2);
    }
}
```

Listas Circulares – Nó de Cabeçalho

O conceito de ***nó de cabeçalho*** também pode ser empregado nas listas circulares. Por exemplo, podemos utilizar o nó de cabeçalho como um ponto de verificação para testar se alista inteira foi atravessada.

A implementação de um TAD LISTA_CIRCULAR_COM_NC é sugerida como uma exercício de fixação.

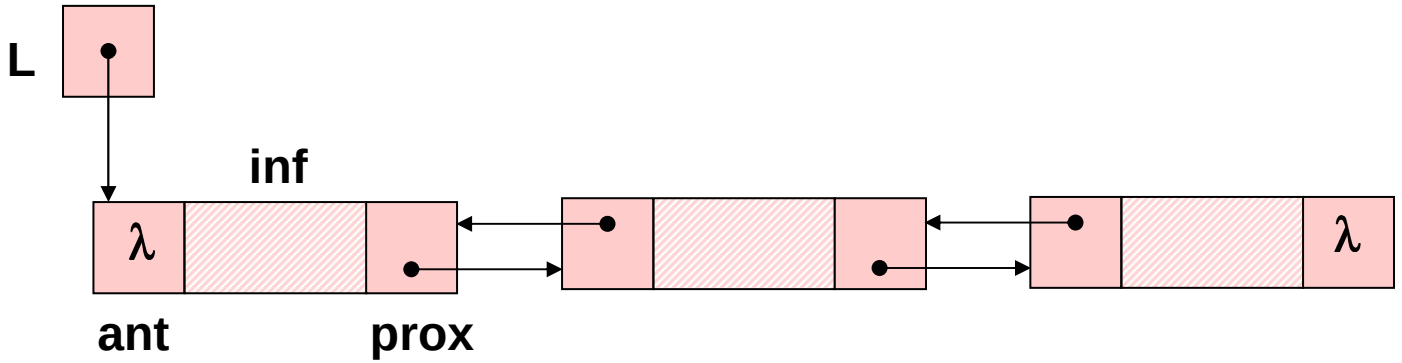
Listas Duplamente Encadeadas

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

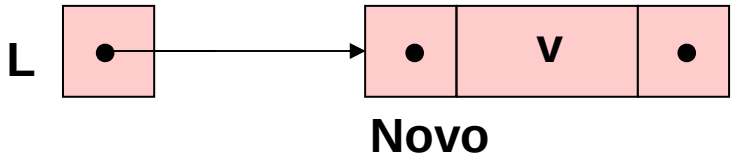
Definiremos agora o TAD LISTA_DUP_ENC:

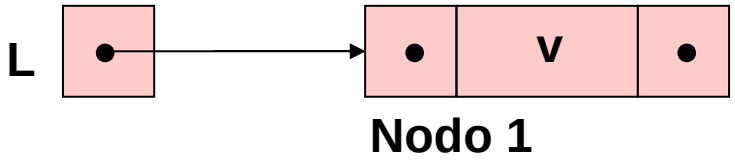
```
typedef struct nodo
{
    int inf;
    struct nodo * ant;
    struct nodo * prox;
}NODO;
typedef NODO * LISTA_DUP_ENC;
void cria_lista (LISTA_DUP_ENC *);
int eh_vazia (LISTA_DUP_ENC);
int tam (LISTA_DUP_ENC);
void ins (LISTA_DUP_ENC *, int, int);
int recup (LISTA_DUP_ENC, int);
void ret (LISTA_DUP_ENC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

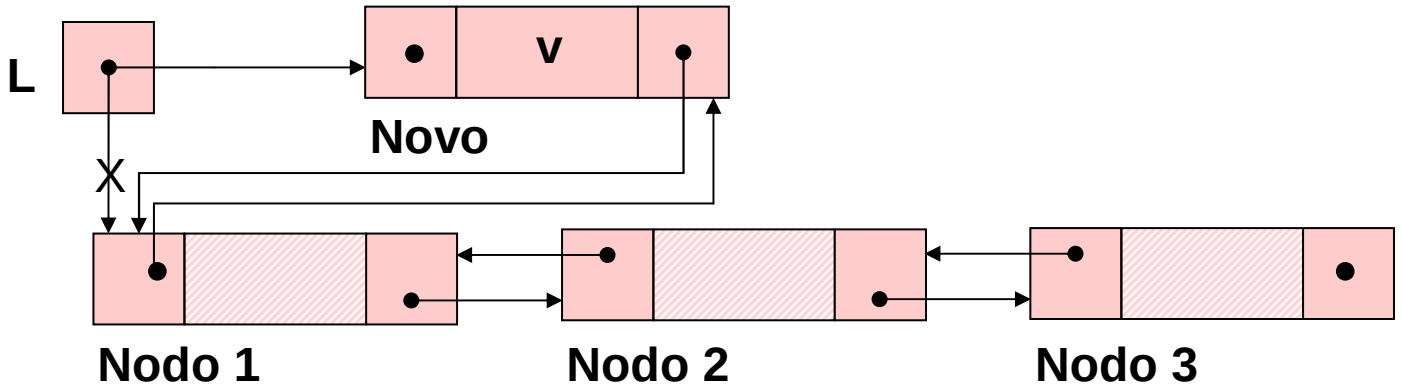
Esquema do processo da inserção de um nó da lista duplamente encadeada. (situação um)

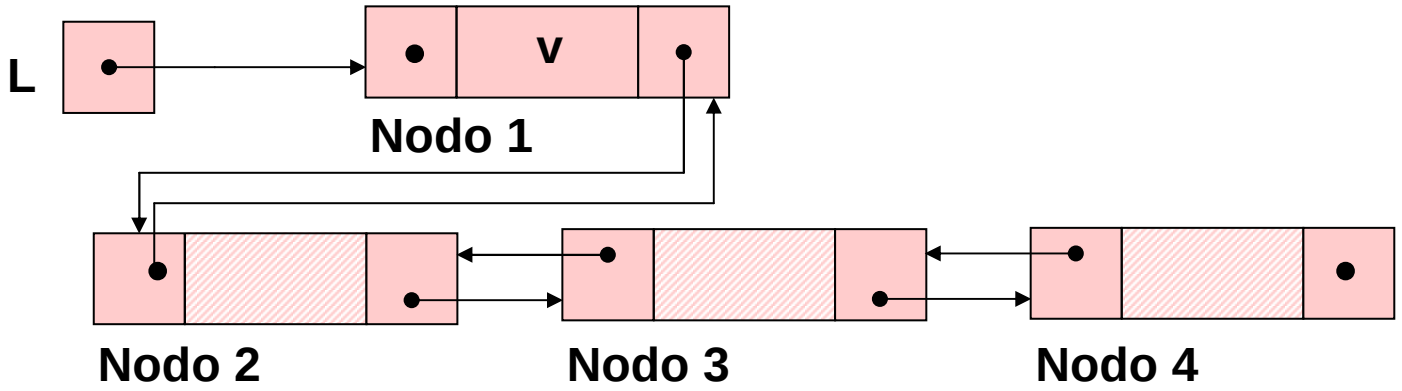




Listas Duplamente Encadeadas

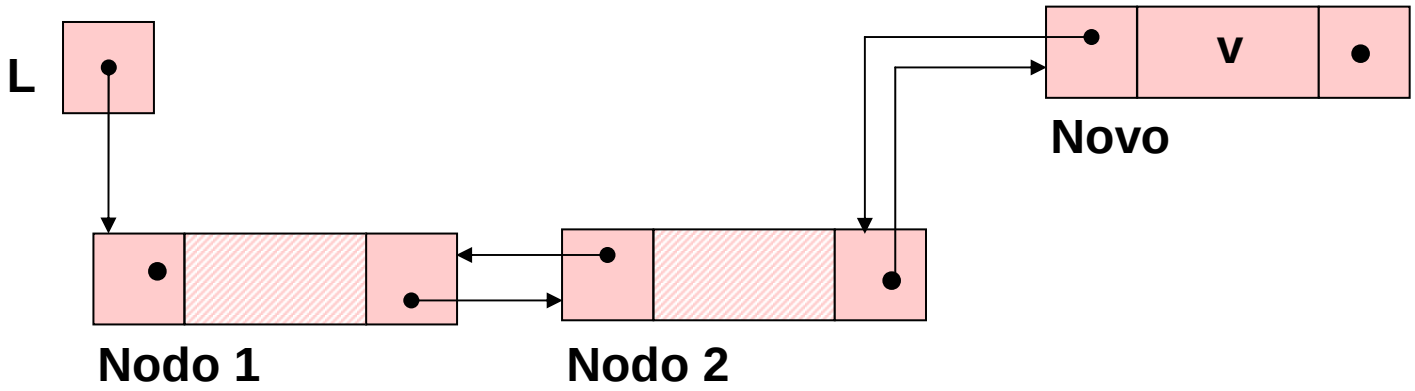
Esquema do processo da inserção de um nó da lista duplamente encadeada. (situação dois)

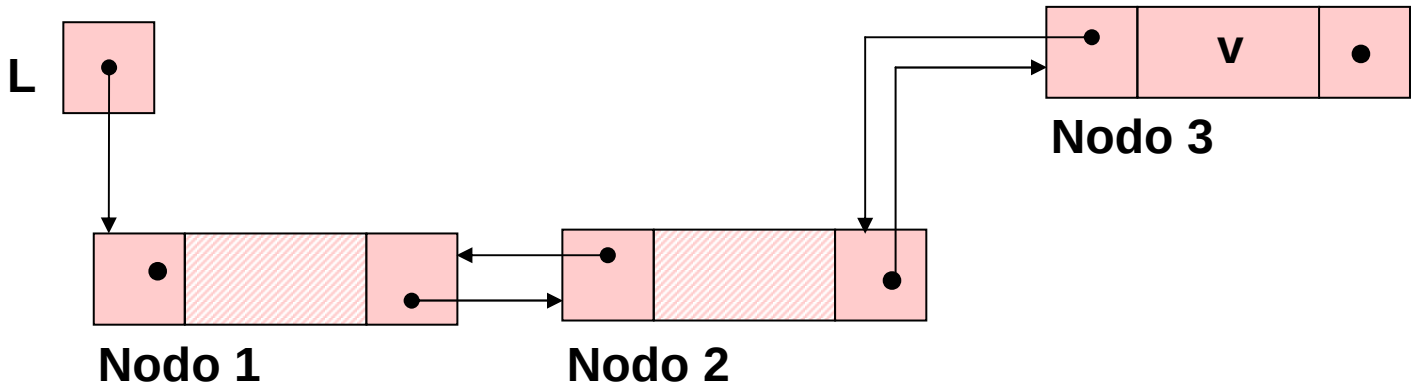




Listas Duplamente Encadeadas

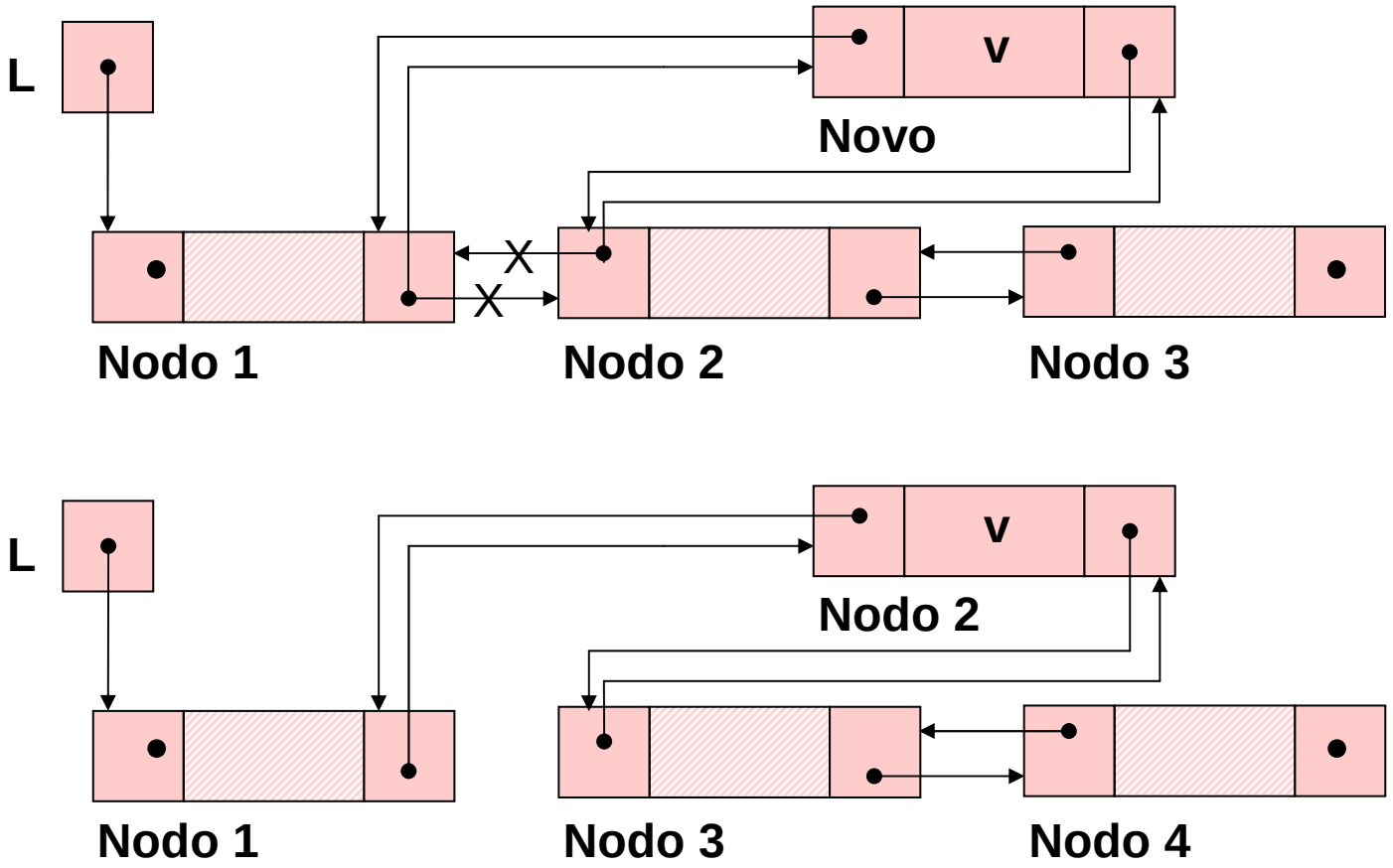
Esquema do processo da inserção de um nó da lista duplamente encadeada. (situação três)





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;
    }
}
```