Alocação Encadeada – Nó de cabeçalho

Com base no que foi visto implemente a operação ret() que compõem o TAD LISTA_ENC_NC.

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

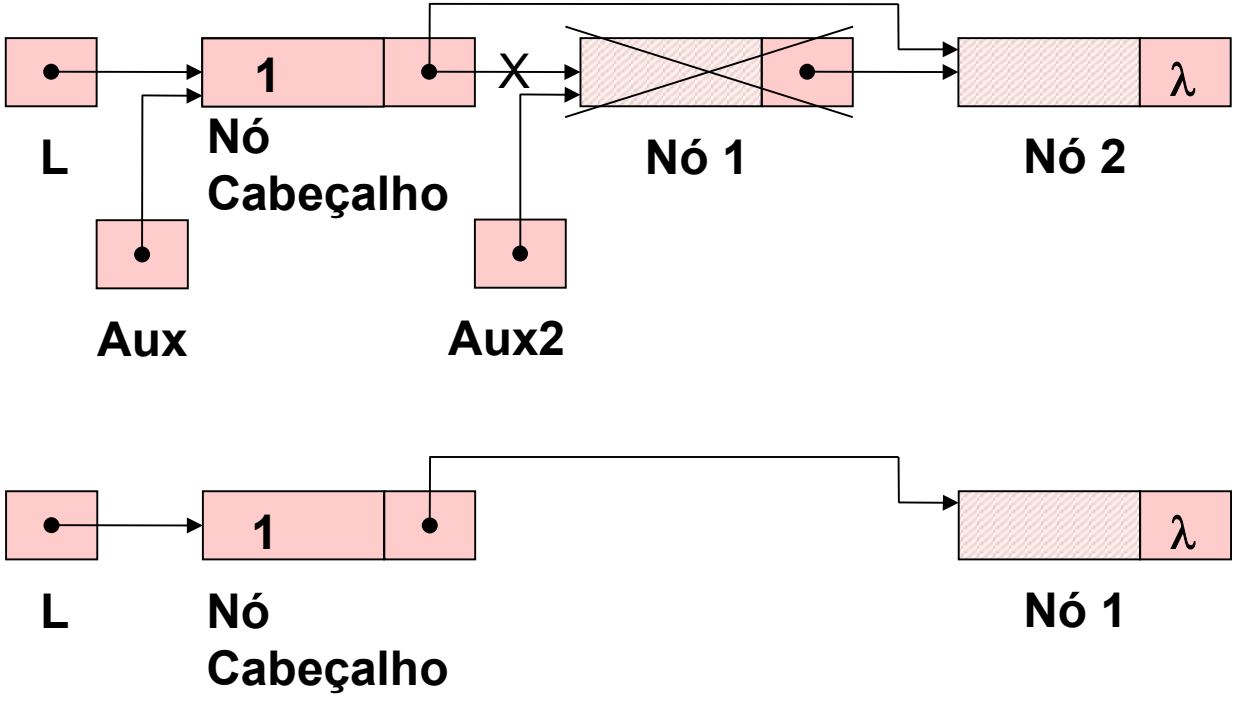
Alocação Encadeada – Nó de cabeçalho

Como ocorreu na operação de inserção, este caso particular, da existência do nó cabeçalho com a mesma estrutura de um elemento da lista, também elimina a ocorrência de duas situações na operação de remoção.

UNIVASF

Alocação Encadeada – Nó de cabeçalho

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



```c
void ret (LISTA_ENC_NC l, int k)
{
    NODO *aux, *aux2;
    if (k < 1 || k > l->inf)
     {
        printf ("\nERRO! Posição invalida para retirada.\n");
        exit (4);
      }
    for (aux=l; k>1; k--, aux=aux->next);
    aux2 = aux->next;
    aux->next = aux2->next;
    free (aux2);
    l->inf--;
}
```

Alocação Encadeada – Nó de cabeçalho

Com base no que foi visto implemente a operação destruir() que compõem o TAD LISTA_ENC_NC.

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

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

```
1  void destruir (LISTA_ENC_NC l)
2  {
3      while (l)
4      {
5          free(l);
6          l = l->next;
7      }
8  }
```

Listas Circulares – Alocação Encadeada

UNIVASF

Listas Circulares

Listas lineares encadeadas possuem alguma deficiência?

Sim.

Qual?

Dado um ponteiro ***p*** para um nodo de uma lista linear encadeada, não podemos atingir nenhum nodo que antecede o apontado por ***p***.

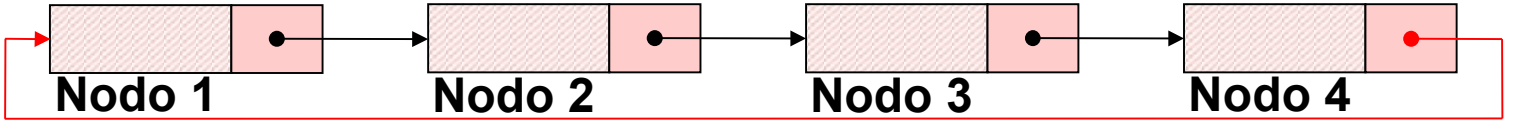
Contudo, se fizermos uma pequena alteração na estrutura da lista que temos trabalhado eliminaremos esta restrição.

UNIVASF

Listas Circulares

Se fizermos com que o campo ***next*** do último nodo, ao invés de conter ***NULL,*** armazenar o endereço do primeiro nodo da lista, resolveremos este problema.

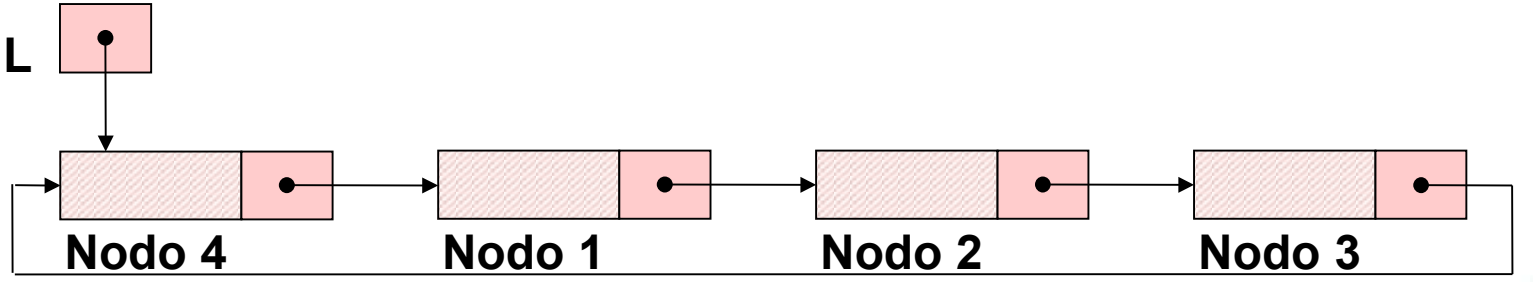
Este tipo de lista é denominado lista circular, possuindo a forma abaixo:



Listas Circulares

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

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



Listas Circulares

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

```c
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 destruir (LISTA_CIRCULAR);
```

Listas Circulares

Com base no que foi visto implemente a operação cria_lista() que compõem o TAD LISTA_CIRCULAR.

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

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

Listas Circulares

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

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

```c
int eh_vazia (LISTA_CIRCULAR l)
{
    return (!l);
}
```

Listas Circulares

Com base no que foi visto implemente a operação tam() que compõem o TAD LISTA_CIRCULAR.

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

UNIVASF

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

Listas Circulares

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

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

Listas Circulares

Dicas:

A posição é válida?

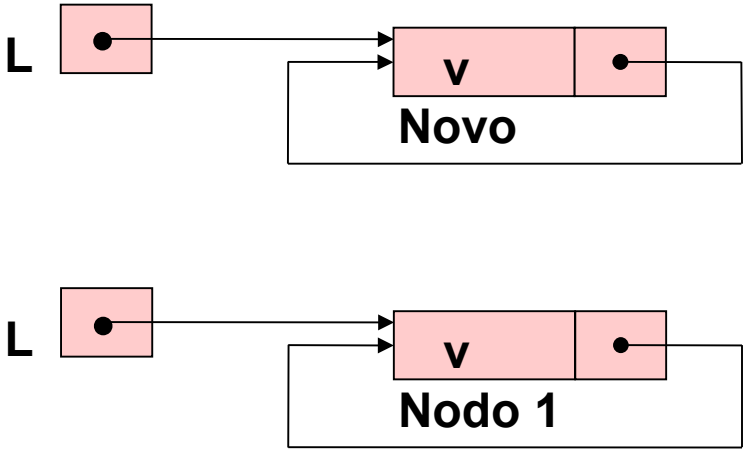
Tem espaço na memória para armazenar mais um elemento?

Todas as situações de inserção são tratadas da mesma forma?

UNIVASF

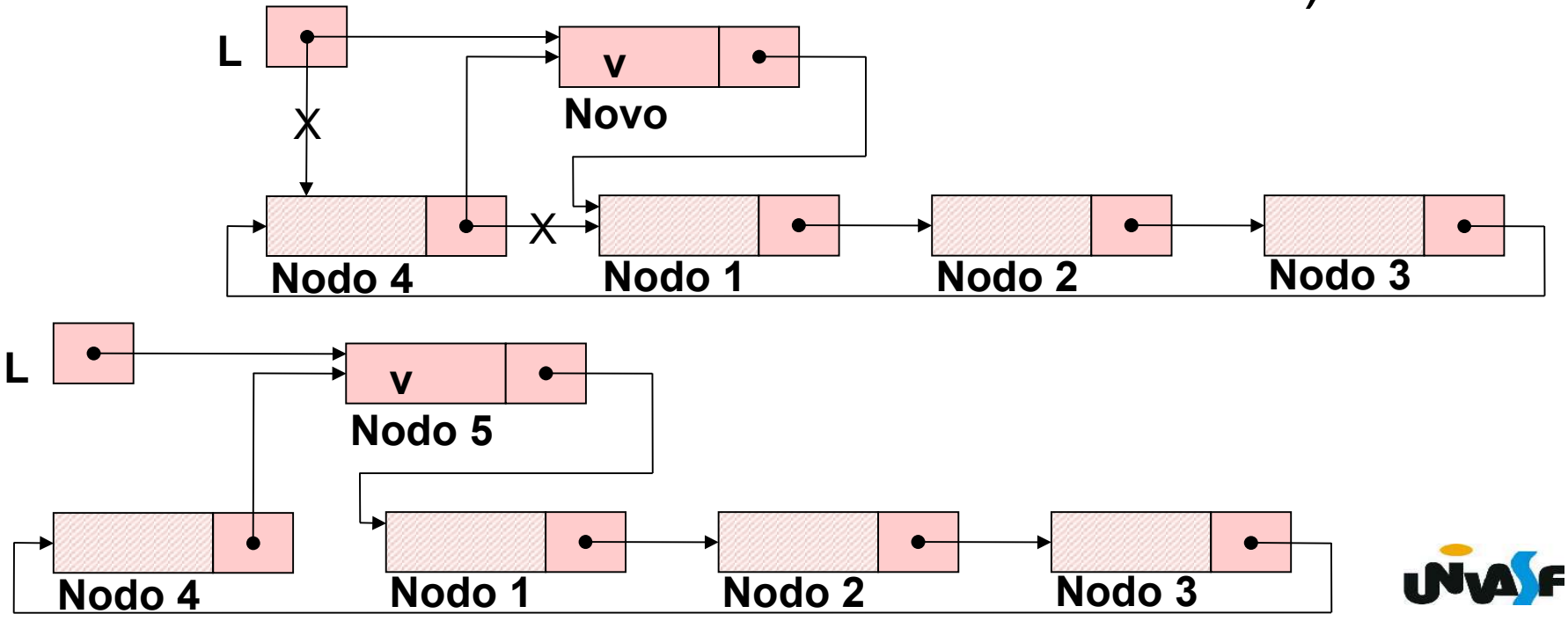
Listas Circulares

Esquema do processo da inserção de um nó da lista circular. (situação um, inserir um elemento em uma lista vazia)



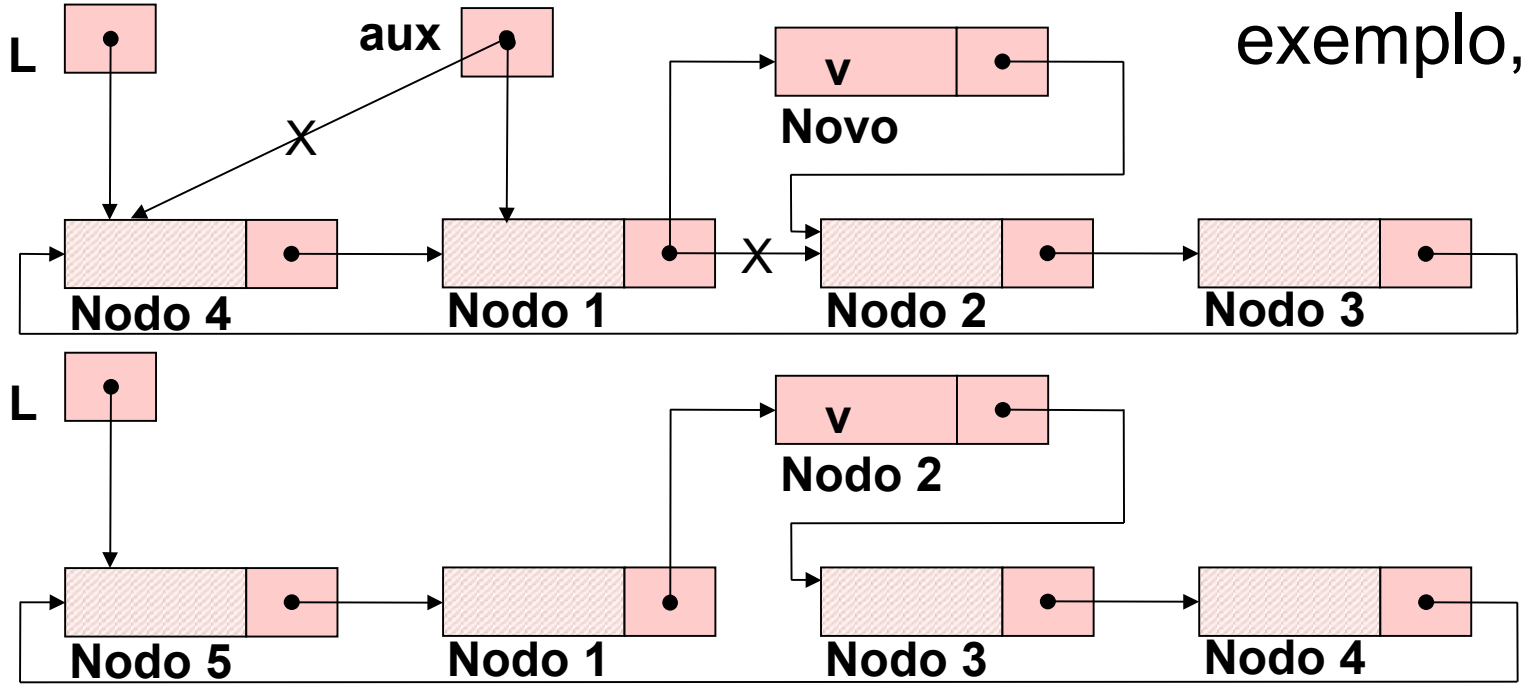
Listas Circulares

Esquema do processo da inserção de um nó da lista circular. (situação dois, novo último elemento em uma lista não vazia)



Listas Circulares

Esquema do processo da inserção de um nó da lista circular. (situação três, elemento nas demais posições em lista não vazia, por exemplo, o 2º)



```c
void ins (LISTA_CIRCULAR *pl, int v, int k) {
    NODO *novo;
    int tamanho = tam(*pl);
    if (k < 1 || k > tamanho+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;
```

```
14      if (*pl==NULL)    {
15          novo->next=novo;
16          *pl = novo;
17      } else {
18          LISTA_CIRCULAR aux=*pl;
19          if (k==tamanho+1)
20              *pl=novo;
21          else
22              for (; k>1; aux=aux->next, k--);
23          novo->next = aux->next;
24          aux->next = novo;
25      }
26  }
```

Listas Circulares

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

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

UNIVASF

```c
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);
}
```

Listas Circulares

Com base no que foi visto implemente a operação ret() que compõem o TAD LISTA_CIRCULAR.

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

UNIVASF

Listas Circulares

Dicas:

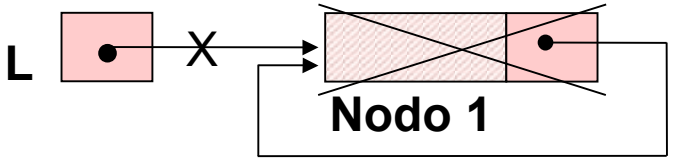
A posição é válida?

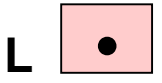
Todas as situações de remoção são tratadas da mesma forma?

UNIVASF

Listas Circulares

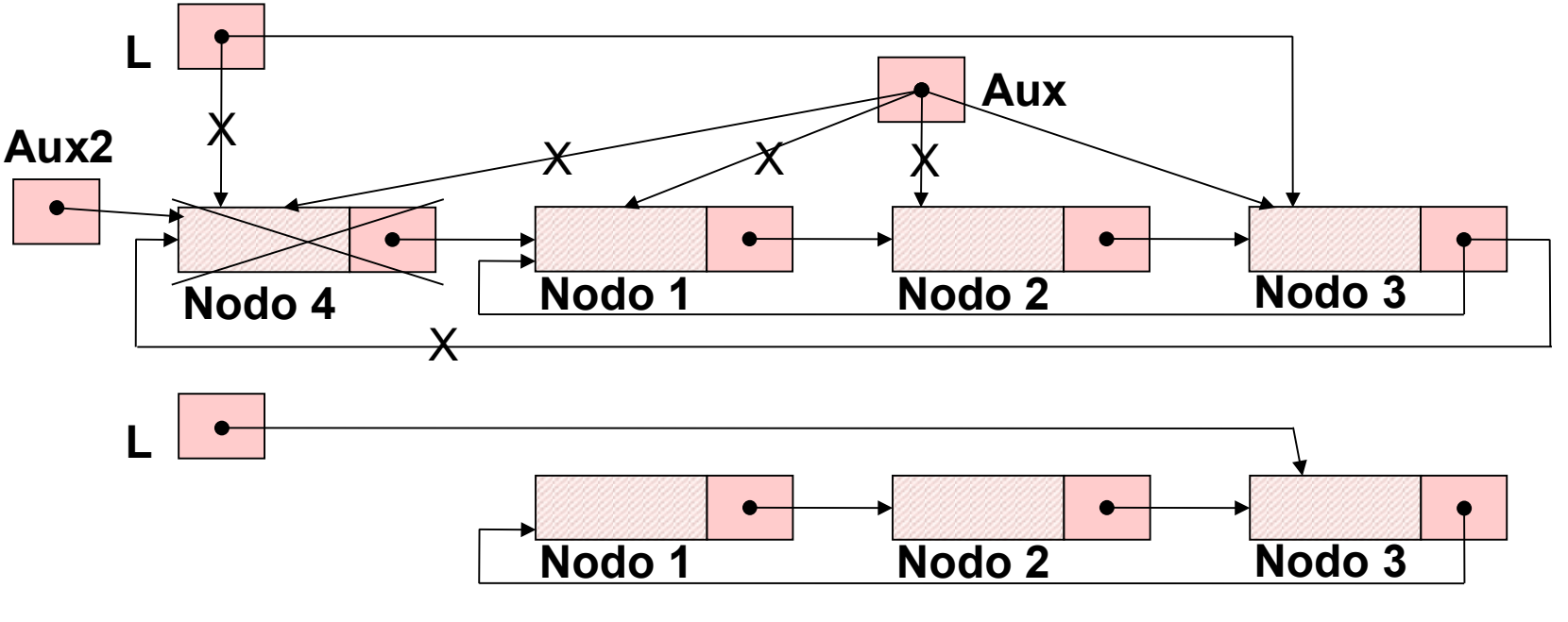
Esquema do processo da retirada de um nó da lista circular. (situação um, removendo o único elemento)





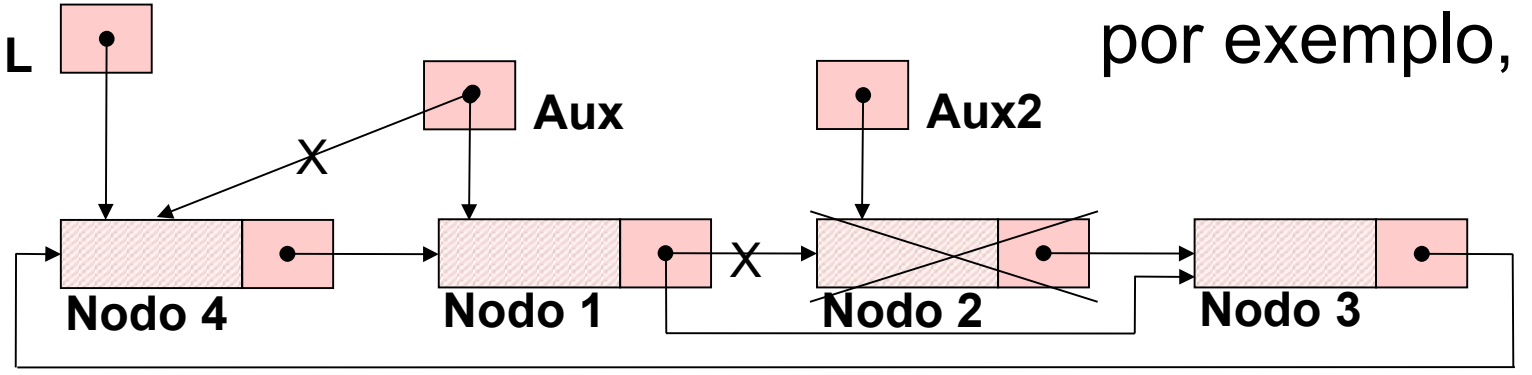
Listas Circulares

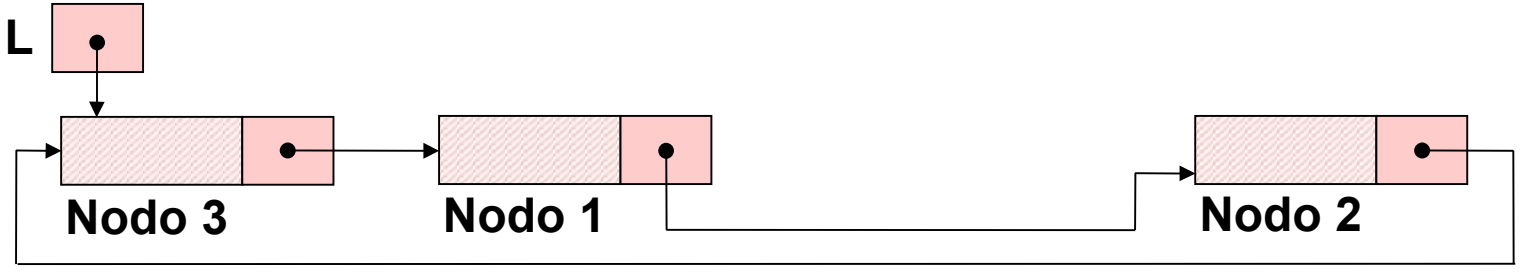
Esquema do processo da retirada de um nó da lista circular. (situação dois, removendo o último elemento)



Listas Circulares

Esquema do processo da retirada de um nó da lista circular. (situação três, removendo um elemento que não seja o único ou o último, por exemplo, o 2º)





```c
void ret (LISTA_CIRCULAR *pl, int k) {
   int tamanho = tam(*pl);
   if (k < 1 || k > tamanho) {
      printf ("\nERRO! Posição invalida para retirada.\n");
      exit (4);
    }
   if (tamanho==1) {
      free (*pl);
      *pl = NULL;
   } else {
      LISTA_CIRCULAR aux, aux2; /* NODO *aux; */
      int i;
      for (aux=*pl, i=k; i>1; i--, aux=aux->next);
      aux2 = aux->next;
      aux->next = aux2->next;
      if (k==tamanho)
         *pl=aux;
      free (aux2);
   }
}
```

Listas Circulares

Com base no que foi visto implemente a operação destruir() que compõem o TAD LISTA_CIRCULAR.

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

UNIVASF

Listas Circulares – Nó de Cabeçalho

O conceito de ***nó de cabeçalho*** também pode ser empregado nas listas circulares.

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