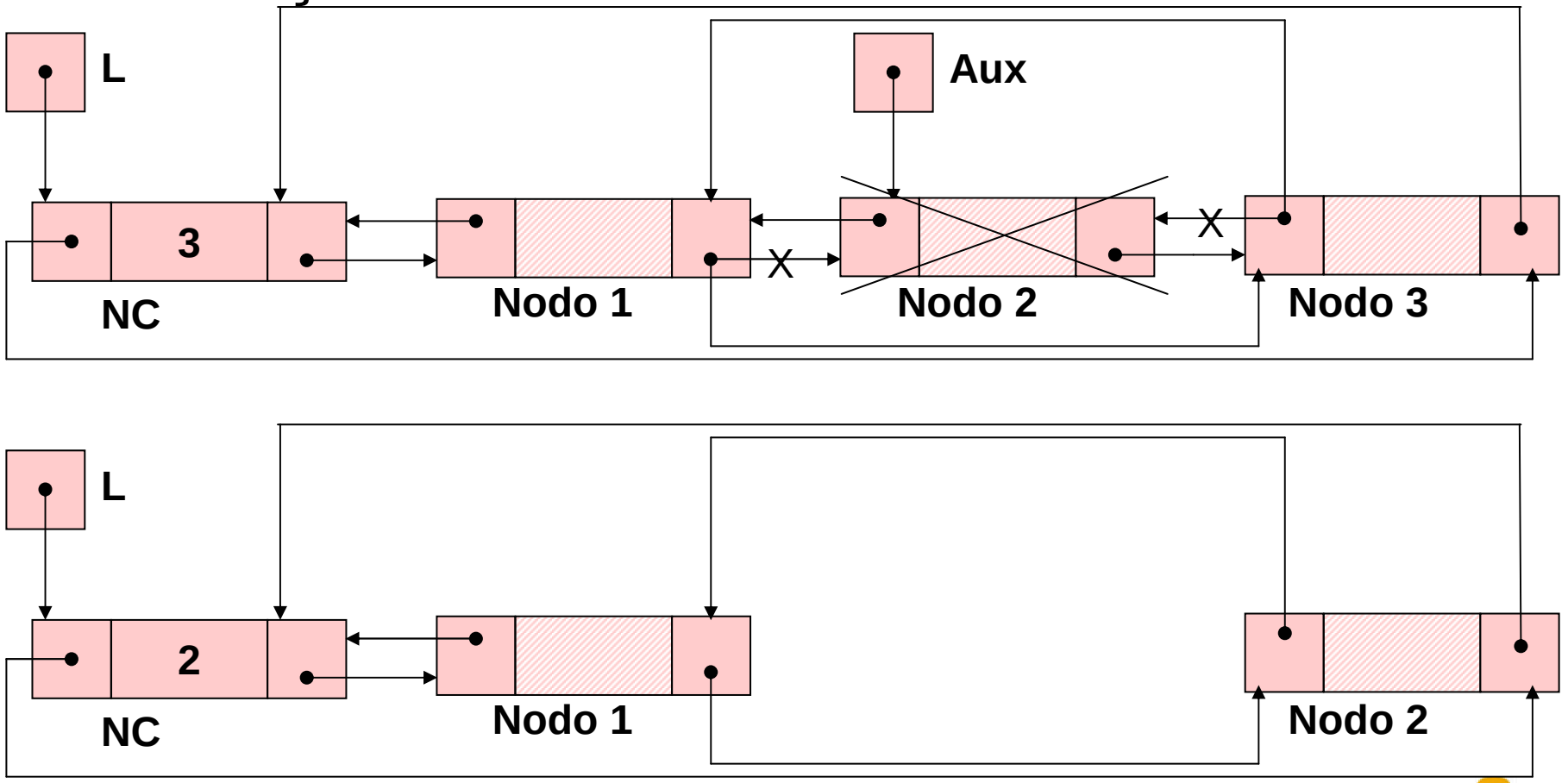
```
int recup (LISTA_CIR_DUP_ENC_NC l, int k)
{
   if (k < 1 || k > tam(l))
   {
      printf ("\nERRO! Consulta invalida.\n");
      exit (3);
   }
   for (;k>0;k--)
      l=l->prox;
   return (l->inf);
}
```

Listas Duplamente Encadeadas

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



```
void ret (LISTA_CIR_DUP_ENC_NC l, int k)
{
   if (k < 1 || k > tam(l))
   {
      printf ("\nERRO! Posição invalida para
     retirada.\n");
      exit (4);
   }
   l->inf--;
   for (; k>0; k--, l=l->prox);
   l->ant->prox = l->prox;
   l->prox->ant = l->ant;
   free (l);
}
```

Listas Duplamente Encadeadas – Exercício

Implemente, no TAD LISTA_CIR_DUP_ENC_NC, a seguinte operação:

void inverter_lista (LISTA_CIR_DUP_ENC_NC l);

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

Listas não inteiras e não homogêneas

Evidentemente, o campo ***inf*** de um nó numa lista não precisa necessariamente armazenar, apenas, um valor inteiro. Pode-se, por exemplo, representar uma lista de strings, por uma lista encadeada, tornando assim, necessários nós contendo as strings de caracteres em seus campos ***inf***. Tais nós poderiam ser declarados como:

```
typedef struct node
{
    char inf [100];
    struct node *prox;
} NODE;
```

Listas não inteiras e não homogêneas

É possível que determinada aplicação exija nós com mais de um item de informação, por exemplo, podemos necessitar de uma lista de registros de estudantes. Onde, cada registro, contém as seguintes informações: nome do estudante, número de identificação da faculdade, endereço, coeficiente de rendimento e área de especialização. Os nós para tal implementação podem ser declarados assim:

Listas não inteiras e não homogêneas

```
typedef struct
{
    char nome [30];
    char id [9];
    char end [100];
    float cr;
    char earea [20];
} INF;
typedef struct node
{
    INF inf;
    struct node *prox;
} NODE;
```

Listas não inteiras e não homogêneas

Para representar listas não-homogêneas (as que contém nós de diversos tipos), podemos utilizar uma união. Exemplo:

```
 #define INTGR     1
 #define FLT       2
 #define STRING    3
 typedef struct node
{
        int etype;
        union inf
        {
                int ival;
                float fval;
                char sval[20];
        } element;
        struct node *prox;
 } NODE;
```

Listas não inteiras e não homogêneas

define um nó cujos itens podem ser inteiros, números de ponto flutuante ou strings, dependendo do valor de etype correspondente. Como uma união é suficientemente grande para armazenar seu maior componente, as funções ***sizeof*** e ***malloc*** podem ser usadas para alocar armazenamento para o nó. Evidentemente, fica sob a responsabilidade do programador usar os componentes de um nó, conforme for apropriado.

Como exercício de fixação, implemente a operação de inserção de um nó um uma lista circular duplamente encadeada não homogênea. Onde, os nós podem ter o campo com informação do tipo inteiro, ponto flutuante ou string. A lista não deve possuir valores replicados.

Disciplinas de acesso

Muitas vezes é útil impor, para manipulação de uma certa estrutura de dados, restrições quanto à visibilidade de seus componentes ou quanto à ordem que deve ser respeitada para se efetuarem operações, como inserção ou retiradas, etc. Isto ajuda na modelagem de certos processos que ocorrem no mundo real.

Com o tempo e a prática, foram identificadas algumas disciplinas de acesso aplicadas a estruturas de dados, úteis em diversas aplicações. Dois casos dos mais importantes são casos particulares de listas com disciplinas de acesso, denominados: filas e pilhas.