```
/***************************************************************************
*
*  Função: LIS  &Esvaziar lista
*  ****/
AE
   LIS_tpCondRet LIS_EsvaziarLista( LIS_tppLista pLista )
   {

      tpElemLista * pElem ;
      tpElemLista * pProx ;

         if ( pLista == NULL ) /* Lista não existe */
         {
                return LIS_CondRetListaNaoExiste;
         }/* if */
AI1
      pElem = pLista->pOrigemLista ;
AI2
      while ( pElem != NULL )
      {
         pProx = pElem->pProx ;
AI5
         LiberarElemento( pLista , pElem ) ;
AI6
         pElem = pProx ;
      } /* while */
AI3
      LimparCabeca( pLista ) ;
AI4
         return LIS_CondRetOK;

   } /* Fim função: LIS  &Esvaziar lista */

AS
```

## Sequência:

**AE:** - pLista é um ponteiro válido para uma lista

- Valem as assertivas estruturais da lista duplamente encadeada

**AS:** - se a lista existia agora a lista é vazia

- se a lista existia o conteúdo dos elementos da lista são desalocados segundo a função ExcluirValor fornecida

- valem as assertivas estruturais da lista duplamente encadeada

AI1: - A lista existe

AI2: - pElem aponta para o primeiro elemento da lista

AI3: - todos os elementos da lista foram liberados

AI4: - a cabeça da lista foi liberada

## Seleção

AE = AE

AS = AI1 ou AS

1) AE && (C==T) + B → AS

Pela AE, pLista pode ser NULL. Como (C==T), a lista não existe. Neste caso B retorna a condição "ListaNaoExiste", pLista continua sendo um ponteiro válido para uma lista, respeitando as assertivas estruturais, valendo AS.

2) AE && (C==F) → AI1

Pela AE, a lista pode apontar para qualquer lista válida. Como (C==F), a lista existe, valendo AI1.

## Repetição

AE = AI2

AS = AI3

AINV: - existem dois conjuntos: já liberados e a liberar.

- pElem aponta para um elemento a liberar.

1 ) AE → AINV

Pela AE, pElem aponta para o primeiro elemento da lista. Existem dois conjuntos: a liberar e liberados. O primeiro contém todos os elementos e o segundo está vazio. Como pElem aponta para um elemento da lista, necessariamente este é do conjunto a liberar, valendo AINV.

2) AE && (C == F) → AS

Pela AE, pElem aponta para o primeiro elemento da lista, mas para que (C==F) o elemento precisa ser NULL, significando que a lista está vazia. Vale então a AS, pois não existem elementos alocados.

3) AE && (C==T) + B → AINV

Pela AE, pElem aponta para o primeiro elemento da lista. Como (C==T), o Bloco B passa um elemento de a liberar para o conjunto já liberado. Continuam existindo os dois conjuntos e pElem passa a apontar para outro elemento de a liberar, valendo AINV

4) AINV && (C==T) + B → AINV

Para que AINV continue valendo a cada ciclo, B deve garantir que um elemento passa do conjunto a ordenar para já ordenado e pElem seja reposicionado.

5) AINV && (C==F) → AS

Pela AINV, há dois conjuntos, sendo que todos os elementos estão em já liberados . A C==F indica que pElem não aponta mais para elementos da lista (a liberar está vazio). Todos os elementos foram liberados, indicando AS válida.

6) Término

Como cada ciclo retira um elemento do conjunto a liberar, e a quantidade de elementos é finita, então a repetição termina num número finito de passos.

## Sequência

AE = AINV

AS = AINV

AI5 : - pProx aponta para o elemento seguinte a pElem

AI6: - elemento apontado por pElem foi liberado