

INICIO void LIS\_EsvaziarLista( LIS\_tppLista pLista ){

AE--->

```
AE->
    tpElemLista * pElem ;
    tpElemLista * pProx ;

    #ifdef _DEBUG
    assert( pLista != NULL ) ;
    #endif

    pElem = pLista->pOrigemLista ;
AS->
```

AI 1->

```
AE->
    while ( pElem != NULL ){
        pProx = pElem->pProx ;
        LiberarElemento( pLista , pElem ) ;
        pElem = pProx ;
    } /* while */
AS->
```

AI 2->

```
AE->
    LimparCabeca( pLista );
AS->
```

AS----->

FIM

}

## Sequência 1

AE -> Existe uma lista a ser esvaziada

AI 1 -> pElem aponta para o nó corrente da lista

AI 2 -> pElem = NULL, ou seja, chegou no fim da lista

As-> A lista é esvaziada

## Repetição 1

AE -> AI1

AS-> ASgeral

AINV -> existe 2 conjuntos: o dos nós já excluídos e os do conjunto a excluir. pElem aponta para o elemento do conjunto a excluir

(1) AE → AINV

- a. Pela AE, pElem aponta para o nó corrente da lista e todos estão a excluir. O conjunto já excluído está vazio, vale AINV.

(2) AE && ( c == F ) → AS

- a. Não entra: Pela AE, pElem != NULL. Como (c==F), pElem == NULL (não existe lista para ser esvaziada). Neste caso vale a AS pois o elemento pesquisado não foi encontrado.

(3) AE && ( c == T ) (+) B → AINV

- a. Pela AE, , pElem aponta para o nó corrente da lista. Como ( c == T ), pElem != NULL.. Este então passa do conjunto a excluir para o já excluído e pElem é reposicionado para outro nó de a excluir, vale AINV.

(4) AINV && ( c == T ) (+) B → AINV

- a. Para que a AINV continue valendo, B deve garantir que um elemento passe de a excluir para o já excluído e pElem seja reposicionado.

(5) AINV && ( c == F ) → AS

- a. Para ( c == F ), pElem == NULL e todos os nós estão em já excluído, vale AS.

(6) Término

- a. Como a cada ciclo, B garante que um elemento de a excluir passe para já excluído e o conjunto a excluir possui um número finito de elementos, a repetição termina em um número finito de passos.

## Sequência 2

AE -> AINV

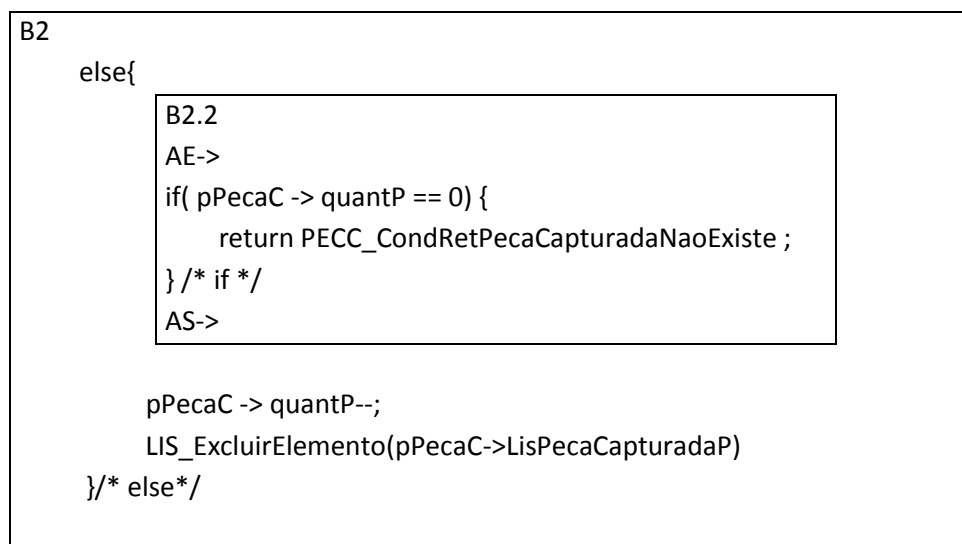
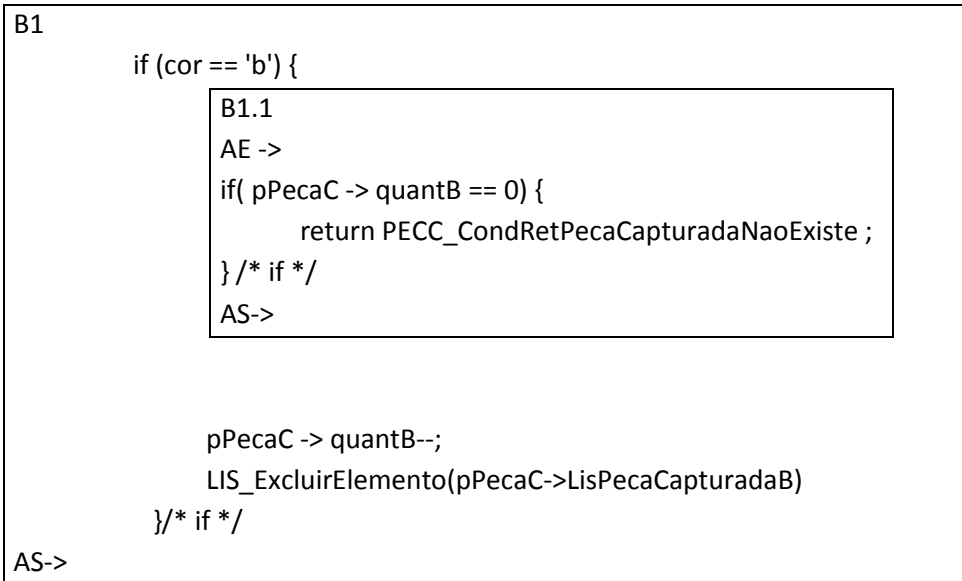
AS-> AINV

AI 3-> pElem é o elemento a excluir , pProx é o próximo nó do conjunto a excluir. pElem é incluído no conjunto já excluído e pElem= pProx.

---

```
INICIO PECC_tpCondRet PECC_RemoverPecaCapturada ( char cor ) {
```

```
AE---->
```



```
    return PECC_CondRetOK;
```

```
AS----->
```

```
}
```

## Seleção 1

AE -> Tem que existir uma string que guarda a cor do elemento

AS-> Um elemento (no) deve ser excluído da lista

(1)  $AE \ \&\& \ (c == T) \ (+) \ B1 \rightarrow AS$

Pela AE, cor é uma cor valida, como  $(c == T)$  cor = 'b', então executa B1, valendo AS.

(2)  $AE \ \&\& \ (c == F) \ (+) \ B2 \rightarrow AS$

Pela AE, cor é uma cor valida, como  $(c == F)$  cor = 'p', então executa B2, valendo AS.

## Seleção 2 (B1.1)

AE -> Quantidade de peças na lista é 0.

AS-> Se  $(c == T)$  fim da função e nenhum elemento é retirado da lista. Se não vale ASgeral.

(1)  $AE \ \&\& \ (c == T) \ (+) \ B \rightarrow AS$

Pela AE, não existe elemento a ser excluído, já que  $(c == T)$ , então executa B, valendo AS.

(2)  $AE \ \&\& \ (c == F) \rightarrow AS_{\text{geral}}$

Pela AE, existe elemento a ser excluído, como  $(c == F)$ , vale ASgeral.

## Sequência 1 (dentro de B1)

AE -> AE do bloco B1.1

AS -> ASgeral

## Seleção 3 (B2.2)

AE -> Quantidade de peças na lista é 0.

AS-> Se  $(c == T)$  fim da função e nenhum elemento é retirado da lista. Se não vale ASgeral.

(1)  $AE \ \&\& \ (c == T) \ (+) \ B \rightarrow AS$

Pela AE, não existe elemento a ser excluído, já que  $(c == T)$ , então executa B, valendo AS.

(2)  $AE \ \&\& \ (c == F) \rightarrow AS_{\text{geral}}$

Pela AE, existe elemento a ser excluído, como  $(c == F)$ , vale  $AS_{\text{geral}}$ .

**Sequência 2 1** (dentro de B2)

AE  $\rightarrow$  AE do bloco B2.2

AS  $\rightarrow AS_{\text{geral}}$