```
/***************************************************************************
*  $MCI Módulo de implementação: TLIS Teste lista de símbolos
*
*  Arquivo gerado:              TestLIS.c
*  Letras identificadoras:      TLIS
*
*  Nome da base de software:    Arcabouço para a automação de testes de
programas redigidos em C
*  Arquivo da base de software: D:\AUTOTEST\PROJETOS\LISTA.BSW
*
*  Projeto: INF 1301 / 1628 Automatização dos testes de módulos C
*  Gestor:  LES/DI/PUC-Rio
*  Autores: avs
*
*  $HA Histórico de evolução:
*     Versão  Autor    Data     Observações
*     4       avs   01/fev/2006 criar linguagem script simbólica
*     3       avs   08/dez/2004 uniformização dos exemplos
*     2       avs   07/jul/2003 unificação de todos os módulos em um só
projeto
*     1       avs   16/abr/2003 início desenvolvimento
*
***************************************************************************/

#include    <string.h>
#include    <stdio.h>
#include    <malloc.h>

#include    "TST_Espc.h"

#include    "Generico.h"
#include    "LerParm.h"

#include    "Lista.h"


static const char RESET_LISTA_CMD         [ ] = "=resetteste"     ;
static const char CRIAR_LISTA_CMD         [ ] = "=criarlista"     ;
static const char DESTRUIR_LISTA_CMD      [ ] = "=destruirlista"  ;
static const char ESVAZIAR_LISTA_CMD      [ ] = "=esvaziarlista"  ;
static const char INS_ELEM_ANTES_CMD      [ ] = "=inselemantes"   ;
static const char INS_ELEM_APOS_CMD       [ ] = "=inselemapos"    ;
static const char OBTER_VALOR_CMD         [ ] = "=obtervalorelem" ;
static const char EXC_ELEM_CMD            [ ] = "=excluirelem"    ;
static const char IR_INICIO_CMD           [ ] = "=irinicio"       ;
static const char IR_FIM_CMD              [ ] = "=irfinal"        ;
static const char AVANCAR_ELEM_CMD        [ ] = "=avancarelem"    ;
```

```
#define TRUE  1
#define FALSE 0

#define VAZIO     0
#define NAO_VAZIO 1

#define DIM_VT_LISTA   10
#define DIM_VALOR     100

LIS_tppLista   vtListas[ DIM_VT_LISTA ] ;

/***** Protótipos das funções encapuladas no módulo *****/

   static void DestruirValor( void * pValor ) ;

   static int ValidarInxLista( int inxLista , int Modo ) ;

/*****  Código das funções exportadas pelo módulo  *****/


/***************************************************************************
*
*  $FC Função: TLIS &Testar lista
*
*  $ED Descrição da função
*     Podem ser criadas até 10 listas, identificadas pelos índices 0 a 10
*
*     Comandos disponíveis:
*
*     =resetteste
*           - anula o vetor de listas. Provoca vazamento de memória
*     =criarlista                   inxLista
*     =destruirlista                inxLista
*     =esvaziarlista                inxLista
*     =inselemantes                 inxLista  string  CondRetEsp
*     =inselemapos                  inxLista  string  CondRetEsp
*     =obtervalorelem               inxLista  string  CondretPonteiro
*     =excluirelem                  inxLista  CondRetEsp
*     =irinicio                     inxLista
*     =irfinal                      inxLista
*     =avancarelem                  inxLista  numElem CondRetEsp
*
***************************************************************************/

   TST_tpCondRet TST_EfetuarComando( char * ComandoTeste )
   {
```

```
int inxLista  = -1 ,
    numLidos   = -1 ,
    CondRetEsp = -1  ;

  LIS_tpCondRet CondRetObtido ;

char   StringDado[  DIM_VALOR ] ;
char * pDado ;

  void * pDadoAux ;

int ValEsp = -1 ;

int i ;

int numElem = -1 ;

StringDado[ 0 ] = 0 ;

/* Efetuar reset de teste de lista */

   if ( strcmp( ComandoTeste , RESET_LISTA_CMD ) == 0 )
   {

      for( i = 0 ; i < DIM_VT_LISTA ; i++ )
      {
         vtListas[ i ] = NULL ;
      } /* for */

      return TST_CondRetOK ;

   } /* fim ativa: Efetuar reset de teste de lista */

/* Testar CriarLista */

   else if ( strcmp( ComandoTeste , CRIAR_LISTA_CMD ) == 0 )
   {

      numLidos = LER_LerParametros( "i" ,
                 &inxLista ) ;

      if ( ( numLidos != 1 )
        || ( ! ValidarInxLista( inxLista , VAZIO )))
      {
         return TST_CondRetParm ;
      } /* if */
```

```
      LIS_CriarLista( &vtListas[ inxLista ], DestruirValor ) ;

      return TST_CompararPonteiroNulo( 1 , vtListas[ inxLista ] ,
         "Erro em ponteiro de nova lista."  ) ;

   } /* fim ativa: Testar CriarLista */

/* Testar Esvaziar lista */

   else if ( strcmp( ComandoTeste , ESVAZIAR_LISTA_CMD ) == 0 )
   {

      numLidos = LER_LerParametros( "i" ,
                        &inxLista ) ;

      if ( ( numLidos != 1 )
        || ( ! ValidarInxLista( inxLista , NAO_VAZIO )))
      {
         return TST_CondRetParm ;
      } /* if */

      LIS_EsvaziarLista( vtListas[ inxLista ] ) ;

            return TST_CondRetOK ;

   } /* fim ativa: Testar Esvaziar lista lista */

/* Testar Destruir lista */

   else if ( strcmp( ComandoTeste , DESTRUIR_LISTA_CMD ) == 0 )
   {

      numLidos = LER_LerParametros( "i" ,
                        &inxLista ) ;

      if ( ( numLidos != 1 )
        || ( ! ValidarInxLista( inxLista , NAO_VAZIO )))
      {
         return TST_CondRetParm ;
      } /* if */

      LIS_DestruirLista( vtListas[ inxLista ] ) ;
      vtListas[ inxLista ] = NULL ;

      return TST_CondRetOK ;

   } /* fim ativa: Testar Destruir lista */
```

```
      /* Testar inserir elemento antes */

         else if ( strcmp( ComandoTeste , INS_ELEM_ANTES_CMD ) == 0 )
         {

            numLidos = LER_LerParametros( "isi" ,
                       &inxLista , StringDado , &CondRetEsp ) ;

            if ( ( numLidos != 3 )
              || ( ! ValidarInxLista( inxLista , NAO_VAZIO )) )
            {
               return TST_CondRetParm ;
            } /* if */

            pDado = ( char * ) malloc( strlen( StringDado ) + 1 ) ;
            if ( pDado == NULL )
            {
               return TST_CondRetMemoria ;
            } /* if */

            strcpy( pDado , StringDado ) ;


            CondRetObtido = LIS_InserirElementoAntes( vtListas[ inxLista ] ,
pDado ) ;

            if ( CondRetObtido != LIS_CondRetOK )
            {
               free( pDado ) ;
            } /* if */

            return TST_CompararInt( CondRetEsp , CondRetObtido ,
                     "Condicao de retorno errada ao inserir antes."
) ;

         } /* fim ativa: Testar inserir elemento antes */

      /* Testar inserir elemento apos */

         else if ( strcmp( ComandoTeste , INS_ELEM_APOS_CMD ) == 0 )
         {

            numLidos = LER_LerParametros( "isi" ,
                       &inxLista , StringDado , &CondRetEsp ) ;

            if ( ( numLidos != 3 )
              || ( ! ValidarInxLista( inxLista , NAO_VAZIO )) )
```

```
            {
               return TST_CondRetParm ;
            } /* if */

            pDado = ( char * ) malloc( strlen( StringDado ) + 1 ) ;
            if ( pDado == NULL )
            {
               return TST_CondRetMemoria ;
            } /* if */

            strcpy( pDado , StringDado ) ;


            CondRetObtido = LIS_InserirElementoApos( vtListas[ inxLista ] ,
pDado ) ;

            if ( CondRetObtido != LIS_CondRetOK )
            {
               free( pDado ) ;
            } /* if */

            return TST_CompararInt( CondRetEsp , CondRetObtido ,
                     "Condicao de retorno errada ao inserir apos."
) ;

         } /* fim ativa: Testar inserir elemento apos */

      /* Testar excluir simbolo */

         else if ( strcmp( ComandoTeste , EXC_ELEM_CMD ) == 0 )
         {

            numLidos = LER_LerParametros( "ii" ,
                  &inxLista , &CondRetEsp ) ;

            if ( ( numLidos != 2 )
              || ( ! ValidarInxLista( inxLista , NAO_VAZIO )) )
            {
               return TST_CondRetParm ;
            } /* if */

            return TST_CompararInt( CondRetEsp ,
                      LIS_ExcluirElemento( vtListas[ inxLista ] ) ,
                     "Condição de retorno errada ao excluir."   ) ;

         } /* fim ativa: Testar excluir simbolo */

      /* Testar obter valor do elemento corrente */
```

```
   else if ( strcmp( ComandoTeste , OBTER_VALOR_CMD ) == 0 )
   {

      numLidos = LER_LerParametros( "isi" ,
                 &inxLista , StringDado , &ValEsp ) ;

      if ( ( numLidos != 3 )
        || ( ! ValidarInxLista( inxLista , NAO_VAZIO )) )
      {
         return TST_CondRetParm ;
      } /* if */

            LIS_ObterValor( vtListas[ inxLista ], &pDadoAux ) ;

      pDado = ( char * ) pDadoAux ;

      if ( ValEsp == 0 )
      {
         return TST_CompararPonteiroNulo( 0 , pDado ,
                   "Valor não deveria existir." ) ;
      } /* if */

      if ( pDado == NULL )
      {
         return TST_CompararPonteiroNulo( 1 , pDado ,
                   "Dado tipo um deveria existir." ) ;
      } /* if */

      return TST_CompararString( StringDado , pDado ,
                   "Valor do elemento errado." ) ;

   } /* fim ativa: Testar obter valor do elemento corrente */

/* Testar ir para o elemento inicial */

   else if ( strcmp( ComandoTeste , IR_INICIO_CMD ) == 0 )
   {

      numLidos = LER_LerParametros( "i" , &inxLista ) ;

      if ( ( numLidos != 1 )
        || ( ! ValidarInxLista( inxLista , NAO_VAZIO )) )
      {
         return TST_CondRetParm ;
      } /* if */

      LIS_IrInicioLista( vtListas[ inxLista ] ) ;
```

```
         return TST_CondRetOK ;

      } /* fim ativa: Testar ir para o elemento inicial */

   /* LIS  &Ir para o elemento final */

      else if ( strcmp( ComandoTeste , IR_FIM_CMD ) == 0 )
      {

         numLidos = LER_LerParametros( "i" , &inxLista ) ;

         if ( ( numLidos != 1 )
           || ( ! ValidarInxLista( inxLista , NAO_VAZIO )) )
         {
            return TST_CondRetParm ;
         } /* if */

         LIS_IrFinalLista( vtListas[ inxLista ] ) ;

         return TST_CondRetOK ;

      } /* fim ativa: LIS  &Ir para o elemento final */

   /* LIS  &Avançar elemento */

      else if ( strcmp( ComandoTeste , AVANCAR_ELEM_CMD ) == 0 )
      {

         numLidos = LER_LerParametros( "iii" , &inxLista , &numElem ,
                              &CondRetEsp ) ;

         if ( ( numLidos != 3 )
           || ( ! ValidarInxLista( inxLista , NAO_VAZIO )) )
         {
            return TST_CondRetParm ;
         } /* if */

         return TST_CompararInt( CondRetEsp ,
                   LIS_AvancarElementoCorrente( vtListas[ inxLista ] ,
numElem ) ,
                   "Condicao de retorno errada ao avancar" ) ;

      } /* fim ativa: LIS  &Avançar elemento */

   return TST_CondRetNaoConhec ;
```

```
   } /* Fim função: TLIS &Testar lista */


/*****  Código das funções encapsuladas no módulo  *****/


/***************************************************************************
*
*  $FC Função: TLIS -Destruir valor
*
***************************************************************************/

   void DestruirValor( void * pValor )
   {

      free( pValor ) ;

   } /* Fim função: TLIS -Destruir valor */


/***************************************************************************
*
*  $FC Função: TLIS -Validar indice de lista
*
***************************************************************************/

   int ValidarInxLista( int inxLista , int Modo )
   {

      if ( ( inxLista <  0 )
        || ( inxLista >= DIM_VT_LISTA ))
      {
         return FALSE ;
      } /* if */

      if ( Modo == VAZIO )
      {
         if ( vtListas[ inxLista ] != 0 )
         {
            return FALSE ;
         } /* if */
      } else
      {
         if ( vtListas[ inxLista ] == 0 )
         {
            return FALSE ;
         } /* if */
      } /* if */
```

```
      return TRUE ;

   } /* Fim função: TLIS -Validar indice de lista */

/********** Fim do módulo de implementação: TLIS Teste lista de símbolos
**********/
```