```
/***************************************************************************
*  $MCI Módulo de implementação: MAT  Matriz
*
*  Arquivo gerado:            MATRIZ.c
*  Letras identificadoras:    MAT
*
*  Nome da base de software:  Arcabouço para a automação de testes de
programas redigidos em C
*
*  Projeto: Trabalho 2 - Programação Modular
*  Autores: GB - Gustavo Bach
*           JG - João Lucas Gardenberg
*           MV - Michelle Valente
*
*  $HA Histórico de evolução:
*     Versão  Autor     Data        Observações
*     2.00  GB,JG,MV   11/abr/2013   reformulação de todas as funções
*     1.00  GB,JG,MV   28/mar/2013   início desenvolvimento
*
***************************************************************************/

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

#include "LISTA.H"

#define MATRIZ_OWN
#include "MATRIZ.H"
#undef MATRIZ_OWN

/***********************************************************************
*
*  $TC Tipo de dados: MAT Matriz
*
*
***********************************************************************/

      typedef struct tgMatriz
      {

      int Altura ;
            /* Número de linhas da matriz */

      int Largura ;
            /* Número de colunas da matriz */
```

```
      LIS_tppLista Matriz ;
            /* Ponteiro para a lista de listas */

      } tpMatriz ;

/***** Protótipos das funções encapsuladas no módulo *****/

      static void InicializarMatriz( ptMatriz pMatriz ) ;

      static void DestruirElemento ( void * pValor ) ;

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

/***************************************************************************
*
*  Função: MAT  Criar matriz
*  ****/

      MAT_tpCondRet MAT_CriarMatriz( ptMatriz * pMatriz, int Altura, int
Largura )

      {

      int IteradorAltura, IteradorLargura ;
      LIS_tpCondRet CondRetLis ;

      *pMatriz = ( tpMatriz * ) malloc( sizeof( tpMatriz ) ) ;

      InicializarMatriz( *pMatriz ) ;

      CondRetLis = LIS_CriarLista ( &(*pMatriz)->Matriz, DestruirElemento ) ;
      if( CondRetLis == LIS_CondRetFaltouMemoria )
      {
            return MAT_CondRetFaltouMemoria ;
      } /* if */

      (*pMatriz)->Altura = Altura ;
      (*pMatriz)->Largura = Largura ;

      for( IteradorAltura = 0 ; IteradorAltura < Altura ; IteradorAltura++ )
      {
            LIS_tppLista pLista = NULL ;

            CondRetLis = LIS_CriarLista ( &pLista, DestruirElemento ) ;
            if( CondRetLis == LIS_CondRetFaltouMemoria )
            {
                  return MAT_CondRetFaltouMemoria ;
```

```
            } /* if */

            for( IteradorLargura = 0 ; IteradorLargura < Largura ;
IteradorLargura++ )
            {
                  CondRetLis = LIS_InserirElementoApos( pLista, NULL ) ;
                  if( CondRetLis == LIS_CondRetFaltouMemoria )
                  {
                  return MAT_CondRetFaltouMemoria ;
                  } /* if */
            } /* for */

            CondRetLis = LIS_InserirElementoApos( (*pMatriz)->Matriz, pLista )
            if( CondRetLis == LIS_CondRetFaltouMemoria )
            {
                  return MAT_CondRetFaltouMemoria ;
            } /* if */

            LIS_IrInicioLista( pLista ) ;
      } /* for */

      LIS_IrInicioLista( (*pMatriz)->Matriz ) ;

      return MAT_CondRetOK ;

      } /* Fim função: MAT  Criar matriz */

/***************************************************************************
*
*  Função: MAT  Destruir matriz
*  ****/

      MAT_tpCondRet MAT_DestruirMatriz( ptMatriz pMatriz )
      {

      int IteradorAltura ;
      LIS_tpCondRet CondRetLis ;

      for( IteradorAltura = 0 ; IteradorAltura < pMatriz->Altura ;
IteradorAltura++ )
      {
            void * pLista;

            CondRetLis = LIS_IrFinalLista( pMatriz->Matriz ) ;
            if( CondRetLis == LIS_CondRetListaNaoExiste )
            {
                  return MAT_CondRetMatrizNaoExiste ;
            } /* if */
```

```
            CondRetLis = LIS_ObterValor( pMatriz->Matriz, &pLista ) ;
            if( CondRetLis == LIS_CondRetListaNaoExiste )
            {
                  return MAT_CondRetMatrizNaoExiste ;
            } /* if */

            CondRetLis = LIS_DestruirLista( ( LIS_tppLista ) pLista ) ;
            if( CondRetLis == LIS_CondRetListaNaoExiste )
            {
                  return MAT_CondRetMatrizNaoExiste ;
            } /* if */

            LIS_ExcluirElemento( pMatriz->Matriz ) ;
      } /* for */

      CondRetLis = LIS_DestruirLista( pMatriz->Matriz ) ;
      if( CondRetLis == LIS_CondRetListaNaoExiste )
      {
            return MAT_CondRetMatrizNaoExiste ;
      } /* if */

      return MAT_CondRetOK ;

      } /* Fim função: MAT  Destruir matriz */

/***************************************************************************
*
*  Função: MAT  Inserir valor no elemento
*  ****/

      MAT_tpCondRet MAT_InserirValor( ptMatriz pMatriz, void * pElemento, int
Linha , int Coluna )
      {

      void * pValor ;
      LIS_tppLista pLista ;
      LIS_tpCondRet CondRetLis ;

      CondRetLis = LIS_IrInicioLista( pMatriz->Matriz ) ;
      if( CondRetLis == LIS_CondRetListaNaoExiste )
      {
            return MAT_CondRetMatrizNaoExiste ;
      } else if ( CondRetLis == LIS_CondRetListaVazia ) {
            return MAT_CondRetMatrizVazia ;
      } /* if */

      CondRetLis = LIS_AvancarElementoCorrente( pMatriz->Matriz, Linha-1 ) ;
```

```
if ( CondRetLis == LIS_CondRetListaVazia )
{
      return MAT_CondRetMatrizVazia ;
} else if ( CondRetLis == LIS_CondRetFimLista ) {
      return MAT_CondRetFimLinhas ;
} /* if */

CondRetLis = LIS_ObterValor( pMatriz->Matriz, &pValor ) ;
if( CondRetLis == LIS_CondRetListaNaoExiste )
{
      return MAT_CondRetMatrizNaoExiste ;
} else if ( CondRetLis == LIS_CondRetListaVazia ) {
      return MAT_CondRetMatrizVazia ;
} /* if */

pLista = (LIS_tppLista) pValor ;

CondRetLis = LIS_IrInicioLista( pLista ) ;
if( CondRetLis == LIS_CondRetListaNaoExiste )
{
      return MAT_CondRetMatrizNaoExiste ;
} else if ( CondRetLis == LIS_CondRetListaVazia ) {
      return MAT_CondRetMatrizVazia ;
} /* if */

CondRetLis = LIS_AvancarElementoCorrente( pLista, Coluna-1 ) ;
if ( CondRetLis == LIS_CondRetListaVazia )
{
      return MAT_CondRetMatrizVazia ;
} else if ( CondRetLis == LIS_CondRetFimLista ) {
      return MAT_CondRetFimColunas ;
} /* if */

CondRetLis = LIS_AlterarValor( pLista, pElemento ) ;
if( CondRetLis == LIS_CondRetListaNaoExiste )
{
      return MAT_CondRetMatrizNaoExiste ;
} else if ( CondRetLis == LIS_CondRetListaVazia ) {
      return MAT_CondRetMatrizVazia ;
} /* if */

CondRetLis = LIS_IrInicioLista( pLista ) ;
if( CondRetLis == LIS_CondRetListaNaoExiste )
{
      return MAT_CondRetMatrizNaoExiste ;
} else if ( CondRetLis == LIS_CondRetListaVazia ) {
      return MAT_CondRetMatrizVazia ;
} /* if */
```

```
      CondRetLis = LIS_IrInicioLista( pMatriz->Matriz ) ;
      if( CondRetLis == LIS_CondRetListaNaoExiste )
      {
            return MAT_CondRetMatrizNaoExiste ;
      } else if ( CondRetLis == LIS_CondRetListaVazia ) {
            return MAT_CondRetMatrizVazia ;
      } /* if */

      return MAT_CondRetOK ;

      } /* Fim função: MAT  Inserir valor no elemento */

/***************************************************************************
*
*  Função: MAT  Obter valor do elemento
*  ****/

      MAT_tpCondRet MAT_ObterValor( ptMatriz pMatriz, int Linha, int Coluna,
void ** pValor )
      {

      void * pValorObtido ;
      LIS_tppLista pLista ;
      LIS_tpCondRet CondRetLis ;

      CondRetLis = LIS_IrInicioLista( pMatriz->Matriz ) ;
      if( CondRetLis == LIS_CondRetListaNaoExiste )
      {
            return MAT_CondRetMatrizNaoExiste ;
      } else if ( CondRetLis == LIS_CondRetListaVazia ) {
            return MAT_CondRetMatrizVazia ;
      } /* if */

      CondRetLis = LIS_AvancarElementoCorrente( pMatriz->Matriz, Linha-1 ) ;
      if ( CondRetLis == LIS_CondRetListaVazia )
      {
            return MAT_CondRetMatrizVazia ;
      } else if ( CondRetLis == LIS_CondRetFimLista ) {
            return MAT_CondRetFimLinhas ;
      } /* if */

      CondRetLis = LIS_ObterValor( pMatriz->Matriz, &pValorObtido ) ;
      if( CondRetLis == LIS_CondRetListaNaoExiste )
      {
            return MAT_CondRetMatrizNaoExiste ;
      } else if ( CondRetLis == LIS_CondRetListaVazia ) {
            return MAT_CondRetMatrizVazia ;
      } /* if */
```

```
pLista = ( LIS_tppLista ) pValorObtido ;

CondRetLis = LIS_IrInicioLista( pLista ) ;
if( CondRetLis == LIS_CondRetListaNaoExiste )
{
      return MAT_CondRetMatrizNaoExiste ;
} else if ( CondRetLis == LIS_CondRetListaVazia ) {
      return MAT_CondRetMatrizVazia ;
} /* if */

CondRetLis = LIS_AvancarElementoCorrente( pLista, Coluna-1 ) ;
if ( CondRetLis == LIS_CondRetListaVazia )
{
      return MAT_CondRetMatrizVazia ;
} else if ( CondRetLis == LIS_CondRetFimLista ) {
      return MAT_CondRetFimColunas ;
} /* if */

CondRetLis = LIS_ObterValor( pLista, pValor ) ;
if( CondRetLis == LIS_CondRetListaNaoExiste )
{
      return MAT_CondRetMatrizNaoExiste ;
} else if ( CondRetLis == LIS_CondRetListaVazia ) {
      return MAT_CondRetMatrizVazia ;
} /* if */

CondRetLis = LIS_IrInicioLista( pLista ) ;
if( CondRetLis == LIS_CondRetListaNaoExiste )
{
      return MAT_CondRetMatrizNaoExiste ;
} else if ( CondRetLis == LIS_CondRetListaVazia ) {
      return MAT_CondRetMatrizVazia ;
} /* if */

CondRetLis = LIS_IrInicioLista( pMatriz->Matriz ) ;
if( CondRetLis == LIS_CondRetListaNaoExiste )
{
      return MAT_CondRetMatrizNaoExiste ;
} else if ( CondRetLis == LIS_CondRetListaVazia ) {
      return MAT_CondRetMatrizVazia ;
} /* if */

return MAT_CondRetOK ;

} /* Fim função: MAT  Obter valor do elemento  */
```

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

/***************************************************************************
*
*  $FC Função: MAT  Inicializar Matriz
*
*  $ED Descrição da função
*     Inicializa a estrutura da matriz.
*
***************************************************************************/

      void InicializarMatriz( ptMatriz pMatriz )
      {

      pMatriz->Altura = 0 ;

      pMatriz->Altura = 0 ;

      pMatriz->Matriz = NULL ;

      } /* Fim função: MAT  Inicializar matriz */

/***************************************************************************
*
*  $FC Função: MAT  Destruir elemento
*
*  $ED Descrição da função
*     Função de destruir elemento apontada pelo ponteiro para função
*     utilizado na função LIS_CriarLista do módulo Lista.
*
***************************************************************************/

      void DestruirElemento ( void * pValor )
      {

      if( pValor != NULL )
      {
            pValor = NULL ;
            free(pValor) ;
      } /* if */

      } /* Fim função: MAT  Destruir elemento */

/********** Fim do módulo de implementação: MAT  Matriz **********/
```