```
/***************************************************************************
*  $MCI Módulo de implementação: CEL  Célula
*
*  Arquivo gerado:            CELULA.c
*  Letras identificadoras:    CEL
*
*  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
*     1.00  GB,JG,MV   12/abr/2014   Correções das condições de retorno
*                                    e parâmetros por referência.
*     1.00  GB,JG,MV   03/abr/2014   Início do desenvolvimento.
*
***************************************************************************/

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

#define CELULA_OWN
#include "CELULA.h"
#undef CELULA_OWN

/***************************************************************************
*
*  $TC Tipo de dados: CEL Célula
*
*
***************************************************************************/

   typedef struct tgCelula {

      int EstadoAtual ;
            /* Estado atual da célula
            *
            *$EED Assertivas estruturais
            *   Variável booleana.
            *   Se for 1, a célula atualmente está preenchida.
            *   Se for 0, a célula atualmente está vazia.
            *   É inicializado com 0. */

      int EstadoCorreto ;
            /* Estado correto da célula
```

```
            *
            *$EED Assertivas estruturais
            *   Variável booleana.
            *   Se for 1, o estado correto da célula é preenchida.
            *   Se for 0, o estado correto da célula é vazia.
            *   É inicializado com 0. */

   } tpCelula ;

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

/***************************************************************************
*
*  Função: CEL Criar célula
*  ****/

   CEL_tpCondRet CEL_CriarCelula( ptCelula * pCelula )
   {

      if( *pCelula != NULL )
      {
            CEL_DestruirCelula( *pCelula ) ;
      } /* if */

      *pCelula = ( tpCelula * ) malloc( sizeof( tpCelula ) ) ;
      if( *pCelula == NULL )
      {
            return CEL_CondRetFaltouMemoria ;
      } /* if */

      /* Inicializar os estados com 0 (vazia) */

      (*pCelula)->EstadoAtual = 0 ;
      (*pCelula)->EstadoCorreto = 0 ;

      return CEL_CondRetOK ;

   } /* Fim função: CEL Criar célula */

/***************************************************************************
*
*  Função: CEL Destruir célula
*  ****/

   CEL_tpCondRet CEL_DestruirCelula( ptCelula pCelula )
   {

      if( pCelula != NULL )
```

```
      {
      free( pCelula ) ;
      pCelula = NULL ;
      } /* if */

      return CEL_CondRetOK ;

   } /* Fim função: CEL Destruir célula */

/***************************************************************************
*
*  Função: CEL Obter estado atual da célula
*  ****/

   CEL_tpCondRet CEL_ObterEstadoAtual( ptCelula pCelula, int * Estado )
   {

      if( pCelula == NULL ) {
      return CEL_CondRetCelulaNaoExiste ;
      } /* if */

      if( pCelula->EstadoAtual == 1 )
      {
      /* Passar por referência o estado atual (preenchida) */

            *Estado = 1 ;
            return CEL_CondRetOK ;
      } else if ( pCelula->EstadoAtual == 0 ) {
      /* Passar por referência o estado atual (vazia) */

            *Estado = 0 ;
            return CEL_CondRetOK ;
      } else {
      /* Se o estado estiver diferente de 1 ou 0, é um estado inválido */

            *Estado = -1 ;
            return CEL_CondRetEstadoInvalido ;
      } /* if */

   } /* Fim função: CEL Obter estado atual da célula */

/***************************************************************************
*
*  Função: CEL Obter estado correto da célula
*  ****/

   CEL_tpCondRet CEL_ObterEstadoCorreto( ptCelula pCelula, int * Estado )
   {
```

```
      if( pCelula == NULL ) {
      return CEL_CondRetCelulaNaoExiste ;
      } /* if */

      if( pCelula->EstadoCorreto == 1 )
      {
      /* Passar por referência o estado correto (preenchida) */

            *Estado = 1 ;
            return CEL_CondRetOK ;
      } else if ( pCelula->EstadoCorreto == 0 ) {
      /* Passar por referência o estado correto (vazia) */

            *Estado = 0 ;
            return CEL_CondRetOK ;
      } else {
      /* Se o estado estiver diferente de 1 ou 0, é um estado inválido */

            *Estado = -1 ;
            return CEL_CondRetEstadoInvalido ;
      } /* if */

   } /* Fim função: CEL Obter estado correto da célula */

/***************************************************************************
*
*  Função: CEL Alterar estado atual da célula
*  ****/

   CEL_tpCondRet CEL_AlterarEstadoAtual( ptCelula pCelula )
   {

      if( pCelula == NULL ) {
      return CEL_CondRetCelulaNaoExiste ;
      } /* if */

      if( pCelula->EstadoAtual == 1 )
      {
      /* Se o estado atual é 1 (preenchida), vira 0 (vazia) */

            pCelula->EstadoAtual = 0 ;
            return CEL_CondRetOK ;
      } else if ( pCelula->EstadoAtual == 0 ) {
      /* Se o estado atual é 0 (vazia), vira 1 (preenchida) */

            pCelula->EstadoAtual = 1 ;
            return CEL_CondRetOK ;
```

```
      } else {
      /* Se o estado estiver diferente de 1 ou 0, é um estado inválido */

            return CEL_CondRetEstadoInvalido ;
      } /* if */

   } /* Fim função: CEL Alterar estado atual da célula */

/***************************************************************************
*
*  Função: CEL Alterar estado correto da célula
*  ****/

   CEL_tpCondRet CEL_AlterarEstadoCorreto( ptCelula pCelula )
   {

      if( pCelula == NULL ) {
      return CEL_CondRetCelulaNaoExiste ;
      } /* if */

      if( pCelula->EstadoCorreto == 1 )
      {
      /* Se o estado correto é 1 (preenchida), vira 0 (vazia) */

            pCelula->EstadoCorreto = 0 ;
            return CEL_CondRetOK ;
      } else if ( pCelula->EstadoCorreto == 0 ) {
      /* Se o estado correto é 0 (vazia), vira 1 (preenchida) */

      pCelula->EstadoCorreto = 1 ;
      return CEL_CondRetOK ;
      } else {
      /* Se o estado estiver diferente de 1 ou 0, é um estado inválido */

            return CEL_CondRetEstadoInvalido ;
      } /* if */

   } /* Fim função: CEL Alterar estado correto da célula */

/***************************************************************************
*
*  Função: CEL Comparar estados da célula
*  ****/

   CEL_tpCondRet CEL_CompararEstados( ptCelula pCelula, int * Comparacao )
   {

      if( pCelula == NULL ) {
```

```
      return CEL_CondRetCelulaNaoExiste ;
      } /* if */

      if( ( pCelula->EstadoAtual == 0 && pCelula->EstadoCorreto == 0 ) || (
pCelula->EstadoAtual == 1 && pCelula->EstadoCorreto == 1 ) )
      {
      /* Passa por referência a comparação dos estados (iguais) */

            *Comparacao = 0 ;
            return CEL_CondRetOK ;
      } else if ( ( pCelula->EstadoAtual == 1 && pCelula->EstadoCorreto == 0 )
      /* Passa por referência a comparação dos estados (diferentes) */

            *Comparacao = 1 ;
            return CEL_CondRetOK ;
      } else {
      /* Se algum estado estiver diferente de 1 ou 0, é um estado inválido */

            *Comparacao = -1 ;
            return CEL_CondRetEstadoInvalido ;
      } /* if */

   } /* Fim função: CEL Comparar estados atual e correto da célula */

/********** Fim do módulo de implementação: CEL  Célula **********/
```