Argumentação de Corretude

Função: TAB_CriarTabuleiro, encontra-se no módulo TABULEIRO.c

```
/* Assertivas de Entrada da Função */
TAB_tpCondRet TAB_CriarTabuleiro(int numColunas, int numLinhas,
            TAB_tppTabuleiro* novoTabuleiro){

      /* Bloco 1 */
      int lin, col;
      (*novoTabuleiro) =  (TAB_tppTabuleiro) malloc(sizeof(TAB_tpTabuleiro));
      /* Fim Bloco 1*/

      /* Assertiva Intermediária 1*/

      /*Bloco 2*/
      if ((*novoTabuleiro) == NULL ){
            /*Bloco 2.1*/
            return TAB_CondRetErro ;
            /*Fim Bloco 2.1*/
      } /* if */
      /*Fim Bloco 2*/

      /*Assertiva Intermediária 2*/

      /*Bloco 3*/
      (*novoTabuleiro)->posicoes = (PEC_tppPeca**)
malloc(sizeof(PEC_tppPeca*)*numLinhas);
      /*Fim Bloco 3*/

      /*Assertiva Intermediária 3*/

      /*Bloco 4*/
      for(lin =0; lin < numLinhas; lin++){
            /*Bloco 4.1*/
            (*novoTabuleiro)->posicoes[lin] = (PEC_tppPeca*)
      malloc(sizeof(PEC_tppPeca)*numColunas);
            /*Fim Bloco 4.1*/

            /*Assertiva Intermediária 4*/

            /*Bloco 4.2*/
            for(col = 0; col < numColunas; col++){
                  /*Bloco 4.2.1*/
                  (*novoTabuleiro)->posicoes[lin][col] =NULL;
                  /*Fim Bloco 4.2.1*/
```

```
            }
            /*Fim Bloco 4.2*/
        }

        /*Assertiva intermediária 5*/

        /*Bloco 5*/
        (*novoTabuleiro)->linhas = numLinhas;
        (*novoTabuleiro)->colunas = numColunas;

        return TAB_CondRetOK;
        /*Fim Bloco 5*/
}/* Fim função: TAB  Criar Tabuleiro */
/*Assertivas de Saída da Função*/
```

- **Argumentação de Sequência**

Assertivas de Entrada da Função:

- Um inteiro de 0 a 7 para o valor da coluna
- Um inteiro de 0 a 7 representando o valor da largura
- Um endereço de memória para armazenar o novo tabuleiro a ser criado

Assertivas de Saída da Função:

- Estados correto da criação de novo tabuleiro
  OU
- Condição de erro na criação de novo tabuleiro

Assertiva Intermediária 1:

- **Assertivas de Entrada (1)**

-- Um endereço de memória para armazenar o novo tabuleiro foi criado

- **Assertivas de Saída (1)**

-- Variável "novoTabuleiro" não nula OU retorna condição de erro TAB_CondRetErro.

Assertiva Intermediária 2:

- **Assertivas de Entrada (2) == Assertivas de Saída (1)**

-- Variável "novoTabuleiro" não nula OU retorna condição de erro TAB_CondRetErro.

- **Assertivas de Saída (2)**

-- Alocação de espaço na memória para a variável posições, presente na struct Tabuleiro. O resultado é o alocamento de espaço do tamanho tpPeça (tipo peça) vezes a quantidade de linhas (numLinhas) que o tabuleiro terá.

Assertiva Intermediária 3:

- **Assertivas de Entrada (3) == Assertivas de Saída (2)**

-- Alocação de espaço na memória para a variável posições, presente na struct Tabuleiro. O resultado é o alocamento de espaço do tamanho tpPeça (tipo peça) vezes a quantidade de linhas (numLinhas) que o tabuleiro terá.

- **Assertivas de Saída (3)**

--"Lin" e "col" passaram por todas as posições do tabuleiro.

Assertiva Intermediária 5:

- **Assertivas de Entrada (5) == Assertivas de Saída (3)**

--"Lin" e "col" passaram por todas as posições do tabuleiro.

- **Assertivas de Saída (3)**

  -- Criação do tabuleiro bem sucedida, retorna ok.

- **Argumentação de Seleção**

Bloco 2

1. **AE2 && (Condição == verdade) + Bloco 2.1 => AS2:**

   -- Pela AE2, o ponteiro para o novoTabuleiro pode ser NULL, caso o espaço não seja alocado corretamente. Como (Condição == verdade), a função retorna condição de erro, satisfazendo assim a AS2.

2. **AE2 && (Condição == falsa) => AS2:**

   -- Pela AE2, o ponteiro para o novo tabuleiro pode não ser NULL, caso o arquivo exista. Como (Condição == falsa), nada acontece e o código continua. A variável "novoTabuleiro", portanto, não está apontando para NULL, e pode ser utilizada, satisfazendo assim a AS2.

- **Argumentação de Repetição**

Bloco 4

1. **AE4 => AINV:**

-- Pela AE4, lin aponta para o primeiro elemento horizontal no tabuleiro. Todos os elementos estão no conjunto "a pesquisar" e o conjunto "já pesquisou" está vazio, valendo assim a AINV.

2. **AE4 && (Condição == falsa) => AS4:**

-- Para que (Condição == falsa), nessa repetição, é necessário que lin > numLinhas (no caso, a largura do tabuleiro), ou seja, o tabuleiro será lido até onde necessário, valendo assim a AS4.

3. **AE4 && (Condição == verdade) + Bloco 4.1 + Bloco 4.2 => AINV:**

-- Pela AE4, lin aponta para o primeiro elemento. Como (Condição == verdade), a posição terá alocado espaço para as peças pela quantidade de colunas no tabuleiro e a posição linha x coluna será preenchida com NULL. A realização deste passo garante que lin será reposicionado para um novo elemento, e AINV é válida.

4. **AINV && (Condição == verdade) + Bloco 4.1 + Bloco 4.2 => AINV:**

-- Para que AINV continue valendo, o passo 3 deve garantir que um elemento passe do conjunto "a pesquisar" para "já pesquisou" e lin seja reposicionado.

5. **AINV && (Condição == falsa) => AS4.1:**

-- Para que (Condição == falsa), o elemento apontado por lin é maior do que numLinhas (no caso, a largura do tabuleiro). Neste último caso, o conjunto "a pesquisar" está vazio, e vale a AS4.1.

6. **Término:**

-- Como o conjunto "a pesquisar" é finito, e o passo 3 a cada ciclo retira um elemento deste conjunto, a repetição termina em um número finito de ciclos, que é igual à largura do tabuleiro.

- **Argumentação de Sequência**

Assertiva Intermediária 4:

- Existe um espaço alocado na memória para o elemento posições de acordo com a quantidade de colunas do novo tabuleiro. "lin" aponta para uma posição que pertence a extensão de numLinhas.

- **Argumentação de Repetição**

Bloco 4.2

1. **AE4.2 => AINV:**

-- Pela AE4.2, col aponta para o primeiro elemento vertical no tabuleiro. Todos os elementos estão no conjunto "a pesquisar" e o conjunto "já pesquisou" está vazio, valendo assim a AINV.

2. **AE4.2 && (Condição == falsa) => AS4:**

-- Para que (Condição == falsa), nessa repetição, é necessário que col > numColunas (no caso, a altura do tabuleiro), ou seja, o tabuleiro será lido até onde necessário, valendo assim a AS4.2.

3. **AE4.2 && (Condição == verdade) + Bloco 4.2.1 => AINV:**

-- Pela AE4, col aponta para o primeiro elemento. Como (Condição == verdade), a posição linha x coluna será preenchida com NULL. A realização deste passo garante que lin será reposicionado para um novo elemento, e AINV é válida.

4. **AINV && (Condição == verdade) + Bloco 4.2.1 => AINV:**

-- Para que AINV continue valendo, o passo 3 deve garantir que um elemento passe do conjunto "a pesquisar" para "já pesquisou" e col seja reposicionado.

5. **AINV && (Condição == falsa) => AS4.1:**

-- Para que (Condição == falsa), o elemento apontado por col é maior do que numColunas (no caso, a altura do tabuleiro). Neste último caso, o conjunto "a pesquisar" está vazio, e vale a AS4.2.

6. **Término:**

-- Como o conjunto "a pesquisar" é finito, e o passo 3 a cada ciclo retira um elemento deste conjunto, a repetição termina em um número finito de ciclos, que é igual à altura do tabuleiro.