Pontifícia Universidade Católica
do Rio de Janeiro

INF1301 - Programação Modular - 2016.1

Prof. Flavio Bevilacqua

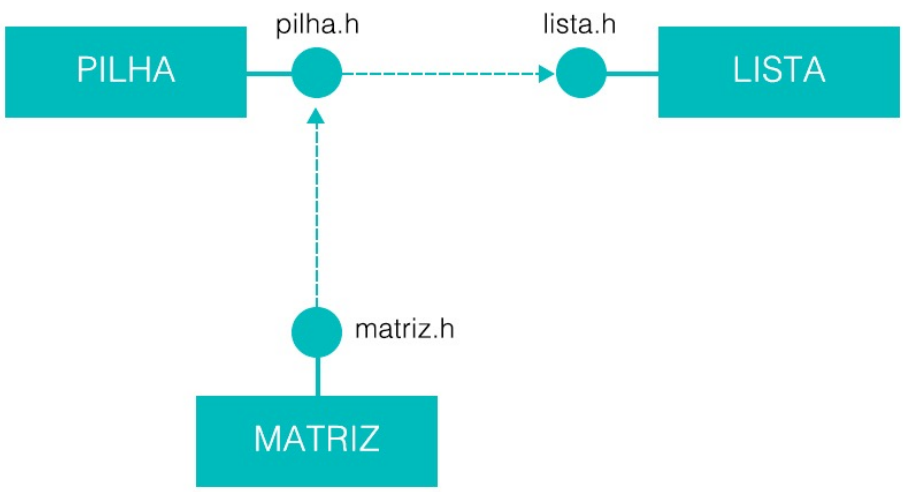
TRABALHO 3 - Arquitetura da Matriz de Pilhas

Gabriel Da Silva Gomes - 1412845

Gustavo Severo Barros - 1421713

João Pedro Masset Lacombe Dias Garcia - 1211768

1. Modelo de Arquitetura



Funções disponibilizadas em cada interface:

I. Lista - LIS

LIS_tppLista LIS_CriarLista (void (* ExcluirValor) (void * pDado));

Assertivas de Entrada:

- Deve existir um ponteiro para a função que processa a exclusão do valor referenciado pelo elemento a ser excluído;

Assertivas de Saída:

- Se existe memória disponível, a lista é criada e seu ponteiro aponta para NULL;

void LIS_DestruirLista(LIS_tppLista pLista);

Assertivas de Entrada:

- Deve existir um ponteiro para a lista a ser destruida;

Assertivas de Saída:

- Se a lista existe, ela é esvaziada, destruida e seu ponteiro aponta para NULL;

void LIS_EsvaziarLista(LIS_tppLista pLista);

Assertivas de Entrada:

- Deve existir um ponteiro para a lista a ser esvaziada;

Assertivas de Saída:

- Se a lista existe, ela é esvaziada;

LIS_tpCondRet LIS_InserirElementoAntes(LIS_tppLista pLista, void * pValor);

Assertivas de Entrada:

- Deve existir um ponteiro para a lista a ser utilizada;
- Deve existir um valor a ser inserido na lista;

Assertivas de Saída:

- Se a lista existe e existe memória disponível, o valor é inserido antes do valor corrente;

LIS_tpCondRet LIS_InserirElementoApos(LIS_tppLista pLista, void * pValor);

Assertivas de Entrada:

- Deve existir um ponteiro para a lista a ser utilizada;
- Deve existir um valor a ser inserido na lista;

Assertivas de Saída:

- Se a lista existe e existe memória disponível, o valor é inserido após o valor corrente;

LIS_tpCondRet LIS_ExcluirElemento(LIS_tppLista pLista);

Assertivas de Entrada:

- Deve existir um ponteiro para a lista a ser utilizada;

Assertivas de Saída:

- Se a lista existe e ela não é vazia, o elemento corrente da lista é excluido;

char LIS_ObterValor(LIS_tppLista pLista);

Assertivas de Entrada:

- Deve existir um ponteiro para a lista a ser utilizada;

Assertivas de Saída:

- Se a lista existe, se não for vazia, a referência para o elemento corrente da lista é retornado;

void IrInicioLista(LIS_tppLista pLista);

Assertivas de Entrada:

- Deve existir um ponteiro para a lista a ser utilizada;

Assertivas de Saída:

- Se a lista existe, se não for vazia, torna corrente o primeiro elemento da lista;

void IrFinalLista(LIS_tppLista pLista);

Assertivas de Entrada:

- Deve existir um ponteiro para a lista a ser utilizada;

Assertivas de Saída:

- Se a lista existe, se não for vazia, torna corrente o último elemento da lista;

LIS_tpCondRet LIS_AvancarElementoCorrente(LIS_tppLista pLista, int numElem);

Assertivas de Entrada:

- Deve existir um ponteiro para a lista a ser utilizada;
- Deve existir um valor representando o número de elementos que o elemento corrente deve avançar;

Assertivas de Saída:

- Se a lista existe, se não for vazia e o valor for encontrado, a referência do valor é retornada;

LIS_tpCondRet LIS_ProcurarValor(LIS_tppLista pLista, void * pValor);

Assertivas de Entrada:

- Deve existir um ponteiro para a lista a ser utilizada;
- Deve existir um ponteiro para o valor a ser procurado;

Assertivas de Saída:

- Se a lista existe, se não for vazia, a referência do valor procurado é o elemento corrente;

II. Matriz - MAT

MAT_tpCondRet MAT_CriarMatriz(MAT_tpMatriz **pMatriz, int NumeroLinhas, int NumeroColunas);

Assertivas de Entrada:

- Deve existir um valor representando o número de linhas da matriz a ser criada;
- Deve existir um valor representando o número de colunas da matriz a ser criada;
- Deve existir um ponteiro por onde será passado, por referência, a matriz a ser criada;

Assertivas de Saída:

- Se o número de linhas e/ou colunas não for igual ou menor que 0 e exista memória disponível, pMatriz é atualizada com um ponteiro para matriz com o valor de linhas e colunas;

MAT_tpCondRet MAT_DestruirMatriz(MAT_tpMatriz ** pMatriz);

Assertivas de Entrada:

- Deve existir um ponteiro para a matriz a ser destruida;

Assertivas de Saída:

- Se a matriz existe, ela é destruida e seu ponteiro aponta para NULL;

MAT_tpCondRet MAT_InserirValor(MAT_tpMatriz *pMatriz, PIL_tppPilha pPilha);

Assertivas de Entrada:

- Deve existir um ponteiro para a matriz a ser utilizada;
- Deve existir um ponteiro para a pilha a ser inserida;

Assertivas de Saída:

- Se a matriz existe e seu elemento corrente existe, ponteiro para pilha é inserido na cabeça da pilha do elemento corrente da matriz;

MAT_tpCondRet MAT_AdicionarLinha(MAT_tpMatriz * pMatriz);

Assertivas de Entrada:

- Deve existir um ponteiro para a matriz a ser utilizada;

Assertivas de Saída:

- Se a matriz existe e existe memória disponível, um nova linha é adicionada a matriz e o número de linhas é incrementada;

MAT_tpCondRet MAT_AdicionarColuna(MAT_tpMatriz * pMatriz);

Assertivas de Entrada:

- Deve existir um ponteiro para a matriz a ser utilizada;

Assertivas de Saída:

- Se a matriz não é nula e existe memória disponível, um nova coluna é adicionada a matriz e o número de colunas é incrementada;

MAT_tpCondRet MAT_RemoverLinha(MAT_tpMatriz * pMatriz);

Assertivas de Entrada:

- Deve existir um ponteiro para a matriz a ser utilizada;

Assertivas de Saída:

- Se a matriz não é nula e seu elemento corrente existe, a última linha da matriz é removida e o número de linhas é decrementada;

MAT_tpCondRet MAT_RemoverColuna(MAT_tpMatriz * pMatriz);

Assertivas de Entrada:

- Deve existir um ponteiro para a matriz a ser utilizada;

Assertivas de Saída:

- Se a matriz não é nula e seu elemento corrente existe, a última coluna da matriz é removida e o número de linhas é decrementada;

MAT_tpCondRet MAT_IrParaCoordenada(MAT_tpMatriz * pMatriz, int linha, int coluna);

Assertivas de Entrada:

- Deve existir um ponteiro para a matriz a ser utilizada;
- Deve existir um valor representando o número da linha e da coluna da matriz para onde o nó corrente irá avançar;

Assertivas de Saída:

- Se a matriz não é nula, se seu elemento corrente existe e o número da linha e coluna não for igual ou menor a 0 e esteja dentro do limite do número de linhas e colunas da matriz, o nó corrente da matriz avança para o número da linha e da coluna;

MAT_tpCondRet MAT_IrPara(MAT_tpMatriz * pMatriz, MAT_tpCoord Coordenada);

Assertivas de Entrada:

- Deve existir um ponteiro para a matriz a ser utilizada;
- Deve existir o valor representando a coordenada para onde o nó corrente irá avançar;

Assertivas de Saída:

- Se a matriz não é nula, se seu elemento corrente existe e se sua coordena existe, o nó corrente da matriz avança para a coordenada;

MAT_tpCondRet MAT_ObterValorCorr(MAT_tpMatriz *pMatriz, PIL_tppPilha pPilha);

Assertivas de Entrada:

- Deve existir um ponteiro para a matriz a ser utilizada;

- Deve existir um ponteiro para a pulha a ser utilizada;

Assertivas de Saída:

- Se a matriz não é nula, se seu elemento corrente existe e se sua coordena existe, o ponteiro para pilha contido na cabeça da pilha do elemento corrente da matriz atualiza pPilha;

III. Pilha - PIL

PIL_tppPilha PIL_CriarPilha( );

Assertivas de Saída:

- Se existe memória disponível, retorna um ponteiro para cabeça da pilha criada;

PIL_tpCondRet PIL_DestruirPilha(PIL_tpPilha *pPilha);

Assertivas de Entrada:

- Deve existir um ponteiro para a pilha a ser destruida;

Assertivas de Saída:

- Se a pilha não é nula, ela é destruida e seu ponteiro aponta para NULL;

PIL_tpCondRet PIL_EmpilhaValor(PIL_tppPilha *pPilha, char ValorParm);

Assertivas de Entrada:

- Deve existir um ponteiro para a pilha a ser utilizada;
- Deve existir um valor a ser inserido na pilha;

Assertivas de Saída:

- Se existe memória disponível e a pilha não é nula, o valor a ser inserido é inserido no topo da pilha e o número de elementos da pilha é incrementado;

PIL_tpCondRet PIL_DesempilhaValor(PIL_tppPilha *pPilha);

Assertivas de Entrada:

- Deve existir um ponteiro para a pilha a ser utilizada;

Assertivas de Saída:

- Se a pilha não é nula e não é vazia, o valor do topo da pilha é removido e o número de elementos da pilha é decrementado;

PIL_tpCondRet PIL_ObterValorTopo(PIL_tppPilha *pPilha, char * ValorParm);

Assertivas de Entrada:

- Deve existir um ponteiro para a pilha a ser utilizada;

- Deve existir um ponteiro por onde será passado, por referência, o valor do topo da pilha;

Assertivas de Saída:

- Se a pilha não é nula e não é vazia, ValorParm é atualizado;

PIL_tpCondRet PIL_ObterTamanho(PIL_tppPilha *pPilha, int * pTamanhoPilha);

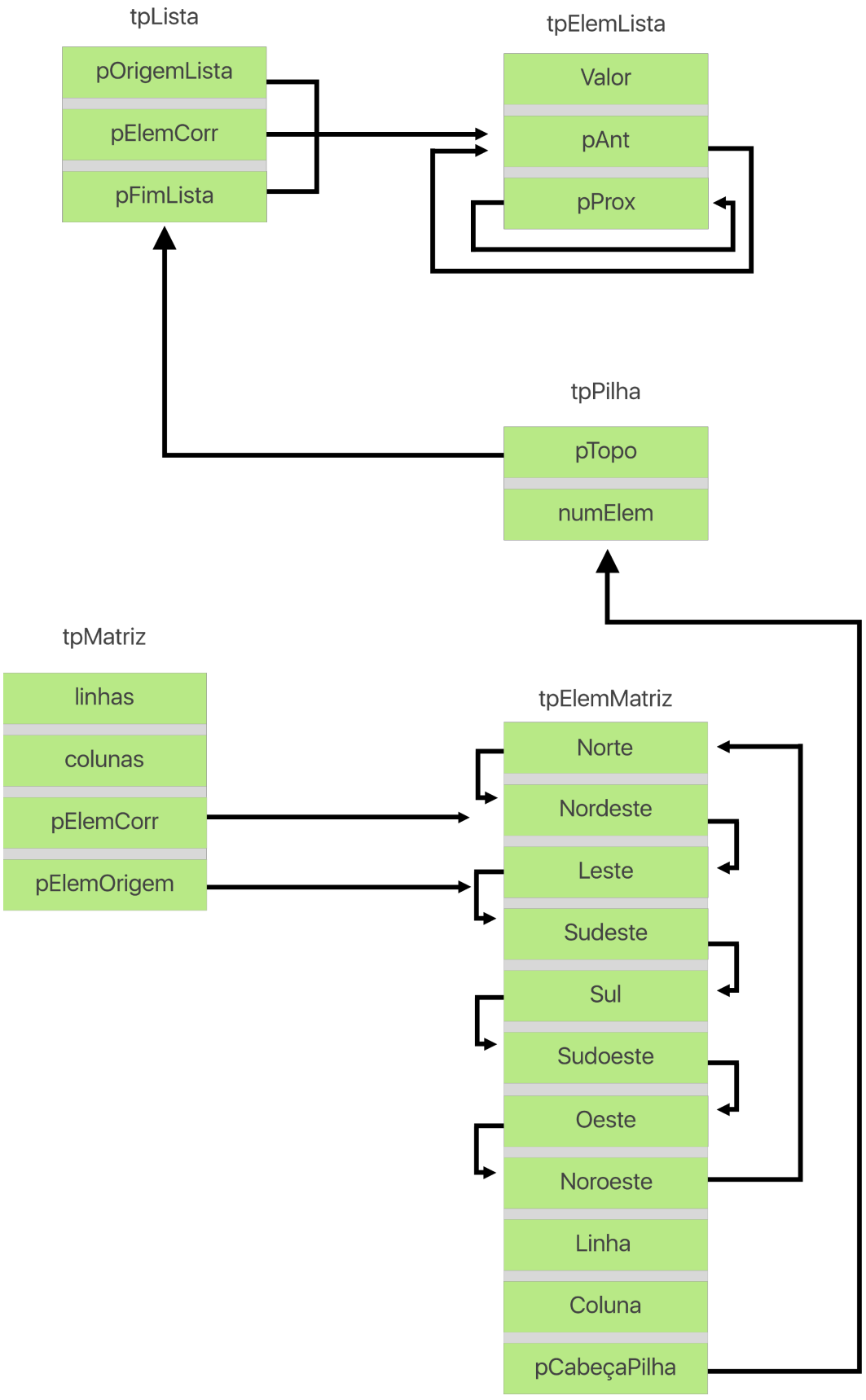
Assertivas de Entrada:

- Deve existir um ponteiro para a pilha a ser utilizada;
- Deve existir um ponteiro por onde será passado, por referência, o tamanho da pilha;

Assertivas de Saída:

- Se a pilha não é nula, pTamanhoPilha é atualizado;

2. Modelo Estrutural



Assertivas Estruturais:

- **Matriz de Pilhas** -> Matriz N/M, com N*M elementos, cada um com um ponteiro para o elemento ao norte, nordeste, leste, sudeste, sul, sudoeste, oeste e noroeste, ponteiro para um tipo pilha e valor linha e coluna. Valem as assertivas estruturais de uma pilha com cabeça estruturada com uma lista duplamente encadeada com cabeça.

- **Pilha** -> Pilha com cabeça com ponteiro para um elemento lista e um valor com número de elementos da pilha. Valem as assertivas estruturais de uma lista duplamente encadeada com cabeça.

- **Lista** -> Valem as assertivas estruturais de uma lista duplamente encadeada com cabeça.

Exemplo