INF1301 – Programação Modular

Trabalho 2: Paciência Spider

Nome: Daniel Siles – 1311291
Fernando Homem – 1211971
Mateus Castro – 1213068

Informações para o modelo:

Exigências do ENUNCIADO:

1) As cartas de um baralho começam a ser divididas em dez sequências em baralhadas com os valores e naipes visíveis de apenas uma única carta e mais cinco montes.

2) O trabalho deverá ter um módulo "pilha de cartas" a ser utilizada pelos os módulos baralho inicial, em cada uma das 10 sequências principais (mostram apenas a última carta) administrados por outro módulo, nos cincos montes (outro módulo) e nas sequências finais (outro módulo).

3) Deverá ter módulo carta que representa uma carta.

4)4 níveis de dificuldade:

1 naipe (fácil),

2 naipes (médio),

3 naipes (difícil),

4 naipes (muito difícil).

5) O módulo "pilha de cartas" deverá utilizar a estrutura lista duplamente encadeada genérica com cabeça devidamente encapsulada em um módulo que será acoplado a outro.

6) Uma estrutura para cada jogo. Esta será uma lista de listas, cada nó desta lista de listas apontará para o cabeça de uma lista necessária, ou seja, cada nó apontará para lista da sequência1, outro para a lista da sequência2, etc..., outro apontará para lista de monte.

Carta:
.Visível ou Escondida
.Naipe
.Numeração

Baralho:
.52 cartas
.4 Naipes
.13 cartas por Naipe

Sequência de 5 cartas (Monte de 5):
.Máximo de 5 cartas de mesmo ou diferentes naipes
.Pode ser sequência vazia
.Primeira carta é virada para cima e as outras para baixo
.São 6 sequência de 5 cartas

Sequência de 6 cartas (Monte de 6):
.Máximo de 6 cartas de mesmo ou diferentes naipes
.Pode ser sequência vazia
.Primeira carta é virada para cima e as outras para baixo
.São 4 sequências de 6 cartas

Sequência de 10 cartas (Morto):
.São 10 cartas de mesmo ou diferentes naipes
.Não pode ser sequência vazia
.Todas viradas para baixo
.São 5 sequência de 10 cartas

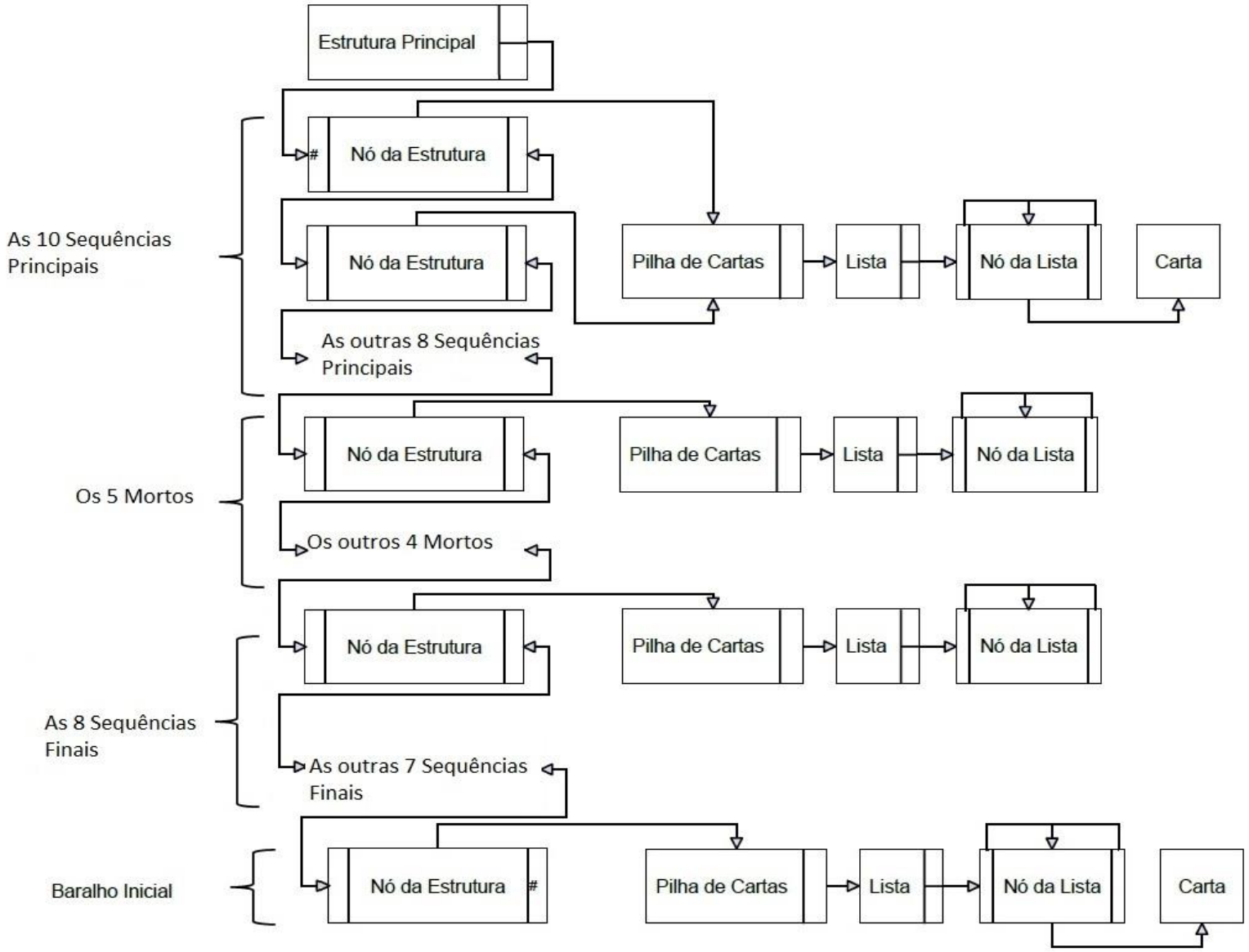
Sequência de 13 cartas (Finalizada):
.São 13 cartas de mesmo naipe
.Não pode ser sequência vazia
.É uma sequência ordenada (A,2,3,4,5,6,7,8,9,10,J,Q,K)

Baralho Inicial:
.São 104 cartas de mesmo ou diferentes naipes
.Não pode ser uma sequência vazia
.Não precisa ser ordenada
.Precisa ser embaralhada

Estrutura Principal
.Novo Jogo-> Começar um jogo novo.
.Continuar/Carregar Jogo -> continuar um jogo já salvo.
.Salvar-> Salvar jogo corrente.
.Dificuldade->Selecionar o nível de dificuldade do Jogo.
(Fácil – 1 Naipe, Médio – 2 Naipes, Difícil – 3 Naipes, Muito Difícil – 4 Naipes)
.Sair-> fechar a aplicação.

Modelo Estrutural:

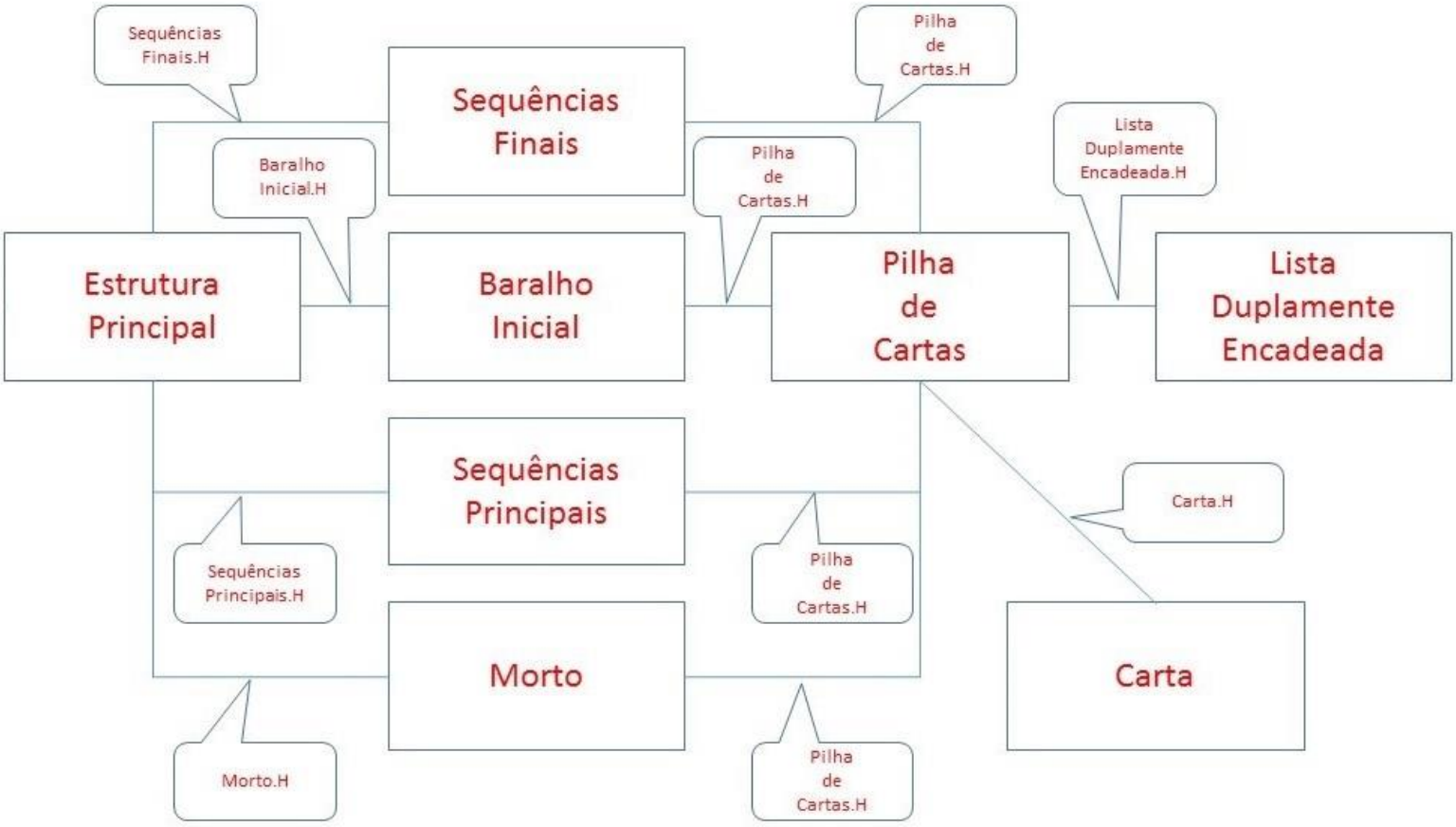
Exemplo de Modelo Estrutural



As Assertivas Estruturais do Exemplo de Modelo:

. Se List->primeiro == Nó da Lista, então Nó da List ->ant = NULL
. Se Nó da Lista for o último nó, então Nó da Lista->prox = NULL
. Pilha de Cartas aponta para uma Lista ou NULL
. Todos os nós da Estrutura que representam um Morto apontam para uma Pilha de Cartas, que aponta para uma Lista, que contém 0 ou 10 nós.
.Todos os Nós da Estrutura que representam um Sequência Final apontam para uma Pilha de Cartas, que aponta para uma Lista, que contém ou 13 nós.
.Todos os Nós da Estrutura que respresentam um Baralho Inicial apontam para uma Pilha de Cartas, que ponta para uma Lista, que contém 104 ou 0 nós.

Diagrama de Arquitetura



Módulos:

1)Carta.H/Carta.C

.CAR_tpCondRet CAR_criarCarta(CAR_tpCarta *pCarta);
.CAR_tpCondRet CAR_destruirCarta(CAR_tpCarta pCarta);
.CAR_tpCondRet CAR_editarCarta(CAR_tpCarta pCarta, char NovaFace, char NovoNaipe, char NovaPosicao);
.CAR_tpCondRet CAR_retornaPosicao(CAR_tpCarta pCarta, char* pPosicao);
.CAR_tpCondRet CAR_retornaNaipe(CAR_tpCarta pCarta, char* pNaipe);
.CAR_tpCondRet CAR_retornaFace(CAR_tpCarta pCarta, char* pFace);
.CAR_tpCondRet CAR_imprimeCarta(CAR_tpCarta pCarta);

2)ListaDuplamenteEncadeda.H/ListaDuplamenteEncadeda.c

. LIS_tpCondRet LIS_CriarLista(LIS_tppLista * pLista, void(*ExcluirValor) (void * pDado));
. void LIS_DestruirLista(LIS_tppLista pLista);
. void LIS_EsvaziarLista(LIS_tppLista pLista);
. LIS_tpCondRet LIS_InserirElementoAntes(LIS_tppLista pLista, void * pValor);
. LIS_tpCondRet LIS_InserirElementoApos(LIS_tppLista pLista, void * pValor);
. LIS_tpCondRet LIS_ExcluirElemento(LIS_tppLista pLista);
. void * LIS_ObterValor(LIS_tppLista pLista);
. void LIS_IrInicioLista(LIS_tppLista pLista);
. void LIS_IrFinalLista(LIS_tppLista pLista);
. LIS_tpCondRet LIS_AvancarElementoCorrente(LIS_tppLista pLista, int numElem);
. LIS_tpCondRet LIS_ProcurarValor(LIS_tppLista pLista, void * pValor);
. LIS_tpCondRet LIS_retornaNumElementos(LIS_tppLista pLista, int *Num);

3)PilhadeCartas.H/PilhadeCartas.C

.PILHA_tpCondRet PILHA_criarPilha(PILHA_tpPilha *pPilha);
.PILHA_tpCondRet PILHA_pushPilha(PILHA_tpPilha pPilha, CAR_tpCarta* c);
.PILHA_tpCondRet PILHA_popPilha(PILHA_tpPilha pPilha, CAR_tpCarta *pCarta);
.PILHA_tpCondRet PILHA_liberaPilha(PILHA_tpPilha pPilha);
.PILHA_tpCondRet PILHA_verificaPilhaVazia(PILHA_tpPilha pPilha);
.PILHA_tpCondRet PILHA_imprimePilha(PILHA_tpPilha pPilha);
.PILHA_tpCondRet PILHA_retornaNumElem(PILHA_tpPilha pPilha, int *num);

4)BaralhoInicial.H/BaralhoInicial.C

.BAR_tpCondRet BAR_criarBaralho(BAR_tpBaralho *pBaralho);
.BAR_tpCondRet BAR_liberarBaralho(BAR_tpBaralho pBaralho);
.BAR_tpCondRet BAR_inicializarBaralho(BAR_tpBaralho pBaralho, int numNaipes);
. BAR_tpCondRet BAR_embaralharBaralho(BAR_tpBaralho pBaralho);
. BAR_tpCondRet BAR_imprimeBaralho(BAR_tpBaralho baralho);
. BAR_tpCondRet BAR_popBaralho(BAR_tpBaralho bBaralho, CAR_tpCarta *cCarta);

5)SequênciasPrincipais.H/
SequenciasPrincipais.C

.SQP_tpCondRet SQP_criarSequencia(SQP_tpSQPrincipal *SQPrincipal);
.SQP_tpCondRet SQP_liberaSequencia(SQP_tpSQPrincipal sqPrincipal);
.SQP_tpCondRet SQP_removeDaSequencia(SQP_tpSQPrincipal sqTira, CAR_tpCarta cCarta, PILHA_tpPilha *pPilhaGuarda);
.SQP_tpCondRet SQP_adicionaNaSequencia(PILHA_tpPilha pPilhaTira, SQP_tpSQPrincipal sqRecebe);
.SQP_tpCondRet SQP_verificaSequenciaCompleta(SQP_tpSQPrincipal SQP);
.SQP_tpCondRet SQP_retornaPilha(SQP_tpSQPrincipal sqp, PILHA_tpPilha * pilha);
.SQP_tpCondRet SQP_pushSQP(SQP_tpSQPrincipal sSQP, CAR_tpCarta cCarta);
.SQP_tpCondRet SQP_popSQP(SQP_tpSQPrincipal sSQP, CAR_tpCarta *Carta);

6)SequênciasFinais.H/
SequênciasFinais.C

.SQF_tpCondRet SQF_criarSQFinal(SQF_tpSQFinal *SQFinal);
.SQF_tpCondRet SQF_liberaSQFinal(SQF_tpSQFinal SQFinal);
.SQF_tpCondRet SQF_inicializaSQFinal(SQF_tpSQFinal SQFinal, PILHA_tpPilha sqFinal);
.SQF_tpCondRet SQF_retornaPilha(SQF_tpSQFinal SQF, PILHA_tpPilha *pPilha);

7)Morto.H/Morto.C

.MOR_tpCondRet MOR_criarMorto(MOR_tpMorto *mMorto, PILHA_tpPilha pPilha);
.MOR_tpCondRet MOR_liberaMorto(MOR_tpMorto mMorto);
.MOR_tpCondRet MOR_popMorto(MOR_tpMorto mMorto, CAR_tpCarta *cartaPop);
.MOR_tpCondRet MOR_retornaMorto(MOR_tpMorto mMorto, PILHA_tpPilha * pilha);

8)EstruturaPrincipal.C

.int ESP_iniciaNovoJogo(LIS_tppLista * ListaPrincipal);
.int distribuiBaralho(BAR_tpBaralho pBaralho, SQP_tpSQPrincipal vSQP[], MOR_tpMorto mMorto[]);
.int escolheDificuldade(void);
.int checarDificuldade(int dificuldade);
.void CriaPilha(PILHA_tpPilha *pPilha, BAR_tpBaralho bBaralho, int nCartas);
.void DestruirBaralho(void * bBaralho);
.void DestruirSeqFinal(void * sSeqFinal);
.void DestruirSeqPrincipal(void * sSeqPrincipal);
.void DestruirMorto(void * mMorto);
.void DestruirLista(void * pLista);
.int distribuiMortoJogo(LIS_tppLista ListaPrincipal);
.int ESP_realizaJogada(LIS_tppLista ListaPrincipal, int sq_de, int sq_para, CAR_tpCarta carta);
.int ESP_salvaJogo(LIS_tppLista ListaPrincipal, char nome[]);
.int ESP_CarregaJogoSalvo(LIS_tppLista *ListaPrincipal, char nome[]);
.void ESP_ImprimeJogo(LIS_tppLista ListaPrincipal);
.int main(void);