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

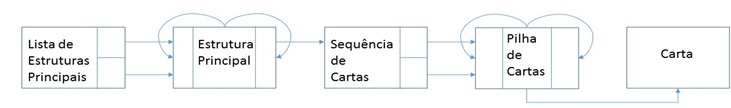
Baralho:  
.52 cartas  
.4 Naipes  
.13 cartas por Naipe

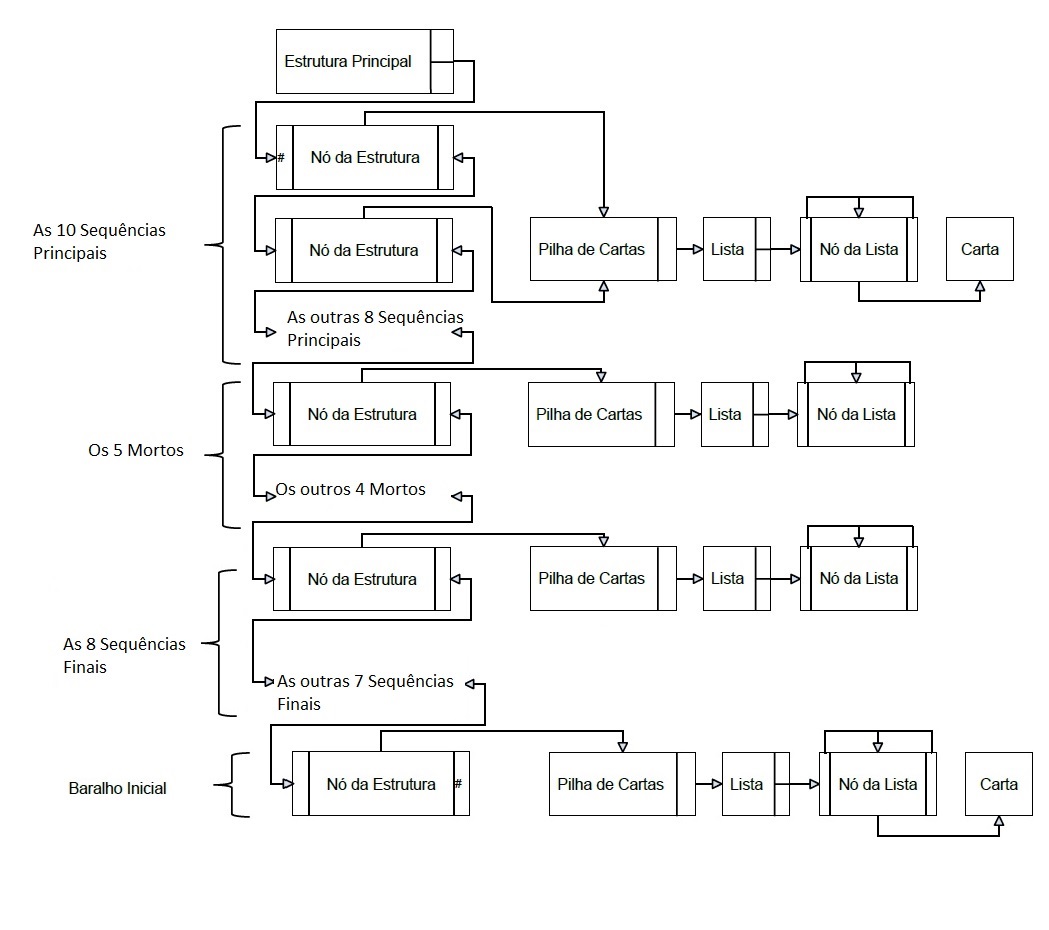
Sequência de 5 cartas (Monte de 5):  
.Máximo de 5 cartas de mesmo ou diferentes naipes  
.Pode ser sequência vazia  
.Primeira carta é virada para cima e as outras para baixo  
.São 6 sequência de 5 cartas

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

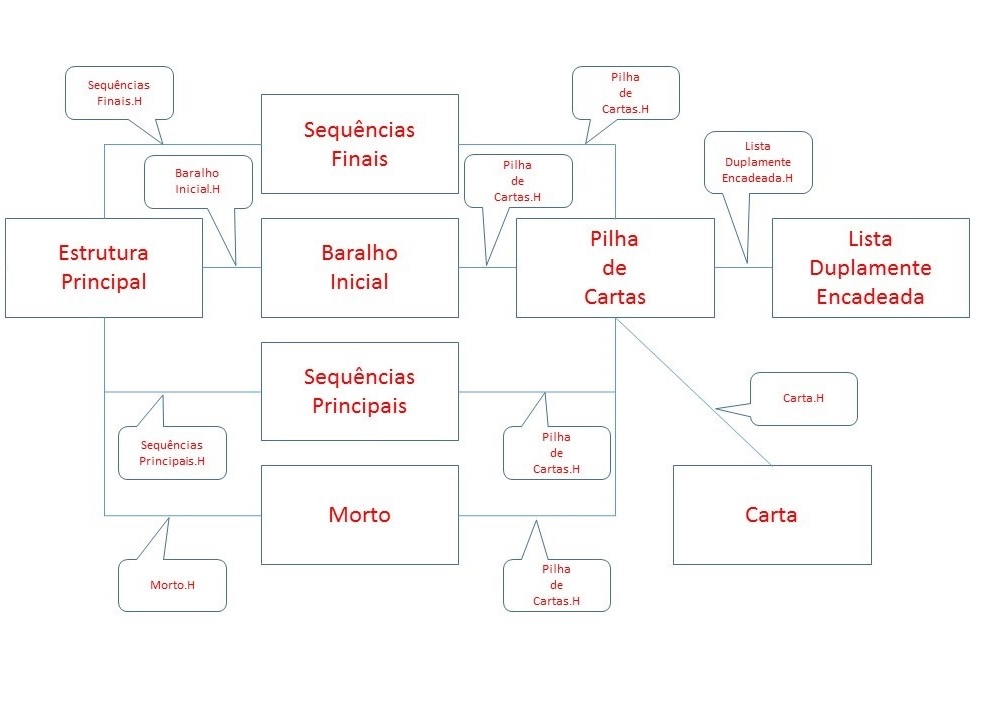
Sequência de 13 cartas (Finalizada):  
.São 13 cartas de mesmo naipe  
.Não pode ser sequência vazia  
.É uma sequência ordenada (A,2,3,4,5,6,7,8,9,10,J,Q,K)  
  
Baralho Inicial:  
.São 104 cartas de mesmo ou diferentes naipes  
.Não pode ser uma sequência vazia  
.Não precisa ser ordenada  
.Precisa 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);