T2 de Programação Modular: Documentação

Grupo: André Mazal Krauss, Danillo Catalão e Albert Uler

Estão contidos nesse documento a especificação de requisitos, arquitetura completa e modelo estrutural do programa.

Especificação de Requisitos:

**O Jogo:**

• O programa deve possibilitar o jogo entre 2,4 ou 6 jogadores.

• Os jogadores são dividos igualmente em 2 equipes.

• A primeira equipe a marcar 12 ou mais pontos ganha o jogo.

• Ambas as equipes começam com 0 pontos.

• Equipes marcam pontos vencendo mãos.

• Somente uma equipe pode ganhar a mão.

• O jogo é composto por um número variável de mãos.

• Uma mão é iniciada sempre que o jogo começar ou se após uma mão terminar, nenhuma equipe tiver ganho o jogo.

• Cada mão pode valer 1,3,6,9 ou 12 pontos, mas inicialmente começam valendo 1 ponto.

• Uma mão pode ter 1,2 ou 3 rodadas.

• Somente uma equipe pode ganhar a rodada.

• A primeira equipe a ganhar 2 rodadas, ganha a mão.

• Ao ganhar uma mão, a equipe marca a pontuação vigente da mão e encerra-se aquela mão.

• Jogadores realizam jogadas na sua vez apenas.

• O sentido que os jogadores realizam suas jogadas é fixo durante o jogo.

• As jogadas são intercaladas por jogadores de cada equipe.

• Na primeira mão um jogador é escolhido aleatoriamente para realizar a primeira jogada da primeira rodada.

• Nas mão subsequentes, o jogador a realizar a primeira jogada é o jogador seguinte ao que realizou primeira jogada da mão anterior.

**O Baralho:**

• O jogo utiliza um baralho convencional retirando-se as cartas '8','9','10' e "Coringa".

• O valor das cartas é determinado pelo seu número, independente do naipe, inicialmente na seguinte ordem crescente: 4 < 5 < 6 < 7 < Q < J < K < Ás < 2 < 3

• O baralho é embaralhado ao final de cada mão.

• O baralho inicialmente está embaralhado.

**As Mãos e as Rodadas:**

• No ínicio de cada mão, cada jogador recebe 3 cartas do baralho, estas visíveis apenas ao próprio jogador.

• Em seguida, uma carta deverá ser retirada do baralho e ser visível a todos os jogadores, esta carta recebe o nome de 'vira'.

• A carta imediatamente superior(de acordo com a ordem crescente de valor inicial) a 'vira' determinará 4 cartas 'manilha' daquela mão, sendo elas a carta em questão nos quatro naipes.

• No caso em que a vira é a carta '3', ao invés da carta imediatamente superior a vira, a carta '4' nos quatro naipes determinará as manilhas daquela mão

• As manilhas da mão vigoram somente nas mão em que elas forem definidas.

• A ordem crescente de valor é atualizada, o valor da manilha é retirada, e colocada como a maior da ordem.

• Após os procedimentos de distribuição das cartas e da determinação das manilhas, inicia-se uma rodada.

• Uma nova rodada é iniciada sempre que após o término de uma rodado nenhuma equipe tiver ganho a mão.

• Na vez de cada jogador, o mesmo deverá revelar aos outros jogadores a carta que deseja jogar,ou joga-la sem revelar aos outros jogadores caso seja a 2ª ou 3ª rodada, esta carta não poderá mais ser utilizada nesta mão.

• Ganha a rodada, a equipe do jogador que tiver a carta com o maior valor revelada.

• O jogador que realizará a primeira jogada nas rodadas seguintes a primeira, será o que teve a carta com maior valor revelada na última rodada.

• Caso jogadores da mesma equipe tenham revelado e jogado a carta com maior valor na rodada anterior, quem realizará a primeira jogada da rodada seguinte será escolhido aleatoriamente entre eles.

• As cartas jogadas,e a vira retornam ao baralho, assim que uma equipe ganha a mão.

• A ordem crescente de valor retorna ao estado inicial, assim que uma equipe ganha a mão.

**O Truco:**

• Antes de realizar sua jogada, o jogador tem a opção de 'pedir truco', caso sua equipe não tenha pedido truco anteriormente nessa mão e caso o valor vigente da partida não seja 12 pontos.

• O jogador seguinte ao que pediu truco pode aceitar,recusar ou retrucar o pedido de truco.

• Ao aceitar um pedido de truco, a pontuação da mão aumenta para o valor imediatamente superior disponível. O jogador que pediu truco prossegue sua jogada.

• Ao recusar um pedido de truco, equipe cujo jogador pediu truco ganha aquela mão.

• Ao retrucar um pedido de truco, a pontuação da mão aumenta para o valor imediatamente superior disponível e o jogador pede truco, mas quem irá aceitar,recusar ou retrucar será o jogador que pediu truco anteriormente.

**Critérios de Desempate:**

• Nos casos em que forem reveladas e jogadas mais de uma manilha em uma rodada, utiliza-se de uma segunda ordem de valor crescente: manilha de outro < manilha de espadas < manilha de copas < manilha de paus. Ganha aquela rodada a equipe do jogador que tiver jogado a manilha com maior valor.

• Caso jogadores de equipes distintas tenham jogado a carta com o maior valor da 1ª rodada, ganha a mão a equipe que vencer a 2ª ou 3ª rodada.

• Caso jogadores de equipes distintas tenham jogado a carta com o maior valor da 2ª ou da 3ª rodada, ganha a mão a equipe que tiver ganho a 1ª rodada.

• Caso jogadores de equipes distintas tenham jogado a carta com o maior valor nas 3 rodada, nenhuma equipe ganha aquela mão. Encerra-se aquela mão.

**Mãos Especiais:**

• Se uma e apenas uma equipe tiver 11 pontos marcados após o término de uma mão, inicia-se uma mão que começa valendo 3 pontos chamada de 'mão de onze'.

• Nas mãos de onze, os jogadores da equipe que tiver 11 pontos podem olhar as cartas dos outros jogadores daquela equipe antes de iniciar-se a primeira rodada.

• Nas mãos de onze, a equipe que tiver 11 pontos pode optar por dar prosseguimento a mão de onze ou recusar a mão de onze antes da primeira rodada começar.

• Ao dar prosseguimento a mão de onze, inicia-se a primeira rodada.

• Ao recusar uma mão de onze, a equipe que não tiver 11 pontos marcados marca 1 ponto. Encerra-se a mão de onze.

• Se ambas as equipe tiverem 11 pontos marcados após o término de uma mão, inicia-se uma mão onde nenhum jogador pode ver as próprias cartas nem as de qualquer jogador, chamada de 'mão de ferro'.

• Não é permitido a nenhum jogador pedir truco nas mãos de onze nem nas mão de ferro.

Arquitetura:



Modelo Estrutural:



Descrição das funções:

**Módulo Lista:**

* **LIS\_tppLista LIS\_CriarLista( void ( \* ExcluirValor ) ( void \* pDado ) ) :**
* Cria uma lista genérica duplamente encadeada. Os possíveis tipos são desconhecidos a priori. A tipagem é implicita. Não existe identificador de tipo associado à lista.
* Parâmetros:
* ExcluirValor - ponteiro para a função que processa a exclusão do valor referenciado pelo elemento a ser excluído.
* Valor retornado - Se executou corretamente retorna o ponteiro para a lista. Este ponteiro será utilizado pelas funções que manipulem esta lista. Se ocorreu algum erro, por exemplo, falta de memória ou dados errados, a função retornará NULL. Não será dada mais informação quanto ao problema ocorrido.
* **void LIS\_DestruirLista( LIS\_tppLista pLista ) :**
* Destrói a lista fornecida. O parâmetro ponteiro para a lista não é modificado. Se ocorrer algum erro durante a destruição, a lista resultará estruturalmente incorreta. OBS. não existe previsão para possíveis falhas de execução.
* **void LIS\_EsvaziarLista( LIS\_tppLista pLista ) :**
* Elimina todos os elementos, sem contudo eliminar a lista.
* Parâmetros:
* pLista - ponteiro para a lista a ser esvaziada.
* **LIS\_tpCondRet LIS\_InserirElementoAntes ( LIS\_tppLista pLista , void \* pValor ) :**
* Insere novo elemento antes do elemento corrente. Caso a lista esteja vazia, insere o primeiro elemento da lista. O novo elemento criado se torna o elemento corrente.
* Parâmetros:
* pLista - ponteiro para a lista onde deve ser inserido o elemento
* pValor - ponteiro para o valor do novo elemento (pode ser NULL)
* Valor retornado: LIS\_CondRetOK ou LIS\_CondRetFaltouMemoria
* **LIS\_tpCondRet LIS\_InserirElementoApos( LIS\_tppLista pLista , void \* pValor ) :**
* Insere novo elemento após o elemento corrente. Caso a lista esteja vazia, insere o primeiro elemento da lista. O novo elemento criado se torna o elemento corrente.
* Parâmetros:
* pLista - ponteiro para a lista onde deve ser inserido o elemento
* pValor - ponteiro para o valor do novo elemento(pode ser NULL)
* Valor retornado: LIS\_CondRetOK ou LIS\_CondRetFaltouMemoria
* **LIS\_tpCondRet LIS\_ExcluirElemento( LIS\_tppLista pLista ) :**
* Exclui o elemento corrente da lista dada. Se existir o elemento à esquerda do corrente será o novo corrente. Se não existir e existir o elemento à direita, este se tornará corrente. Se este também não existir a lista tornou-se vazia.
* Parâmetros:
* pLista - ponteiro para a lista na qual deve excluir.
* Valor retornado: LIS\_CondRetOK ou LIS\_CondRetListaVazia
* **void \* LIS\_ObterValor( LIS\_tppLista pLista ) :**
* Obtem a referência para o valor contido no elemento corrente da lista.
* Parâmetros:
* pLista - ponteiro para a lista de onde se quer o valor
* Valor retornado: não NULL(se o elemento corrente existe) ou NULL(se a lista estiver vazia). Pode retornar NULL se o valor inserido no elemento for NULL.
* **void IrInicioLista( LIS\_tppLista pLista ) :**
* Torna corrente o primeiro elemento da lista. Faz nada se a lista está vazia.
* Parâmetros:
* pLista - ponteiro para a lista a manipular
* **void IrFinalLista( LIS\_tppLista pLista ) :**
* Torna corrente o elemento final da lista. Faz nada se a lista está vazia.
* Parâmetros:
* pLista - ponteiro para a lista a manipular
* **LIS\_tpCondRet LIS\_AvancarElementoCorrente( LIS\_tppLista pLista , int numElem ) :**
* Avança o elemento corrente numElem elementos na lista. Se numElem for positivo avança em direção ao final. Se numElem for negativo avança em direção ao início. numElem pode ser maior do que o número de elementos existentes na direção desejada. Se numElem for zero somente verifica se a lista está vazia.
* Parâmetros:
* pLista - ponteiro para a lista a ser manipulada
* numElem - o número de elementos a andar
* Valor retornado
* CondRetOK - se numElem elementos tiverem sido andados
* CondRetFimLista - se encontrou o fim da lista antes de andar numElem elementos
* CondRetListaVazia - se a lista está vazia

Retornos possíveis:

* LIS\_CondRetOK - Concluiu corretamente
* LIS\_CondRetListaVazia - A lista não contém elemento
* LIS\_CondRetFimLista - Foi atingido o fim de lista
* LIS\_CondRetNaoAchou - Não encontrou o valor procurado
* LIS\_CondRetFaltouMemoria - Faltou memória ao tentar criar um elemento de lista

**Módulo Geraalt:**

* **void ALT\_Inicializar( unsigned int Base ) :**
* Inicializa o gerador de números aleatórios. Esta função deve ser chamada antes de qualquer outra do módulo GERAALT.
* Parâmetros:
* Base
* se == 0 utiliza a semente normal (PI 3141592653 ): isto gerará a mesma seqüência de números aleatórios a seguir da inicialização. Esta semente é tida como uma boa semente, a distribuição é virtualmente uniforme e a cardinalidade é a maior possível.
* se == 1 utiliza a PI | time( ): isto gera uma seqüência de números aleatórios diferente a cada inicialização
* outros - utiliza o valor fornecido como semente deve ser tomado cuidado pois uma semente mal escolhida pode levar à geração de uma seqüência não aleatória, ou de cardinalidade limitada.
* **int ALT\_GerarDistUniforme( int Inf , int Sup ) :**
* Gerar um número inteiro uniformemente distribuído em um intervalo dado
* Parâmetros
* Inf - limite inferior, inclusive
* Sup - limite superior, inclusive
* Valor retornado: Inf <= x <= Sup
* Assertivas de entradas esperadas: Inf <= Sup. Dados errados provocarão um comportamento errático no programa.
* **int ALT\_GerarFrequencia( int numElem , int \* vetElem , int Limite ) :**
* Gerar distribuição segundo tabela de frequências
* Parâmetros:
* numElem - número de elementos (intervalos) da tabela o número de intervalos é na realidade igual a numElem + 1
* vetElem - vetor de elementos da tabela. Os valores dos elementos são números entre 0 e Limite – 1. Os elementos devem estar em ordem crescente. Os elementos correspondem a uma função cumulativa.
* Limite - é o limite superior da distribuição de frequências.
* **void ALT\_GerarPermutacao( int tamVetor , int \* Vetor ) :**
* Gera uma permutação de n elementos uniformemente distribuída. O algoritmo utilizado é o "Knuth shuffle" descrito na Wikipedia.
* Parâmetros:
* tamVetor - número de elementos a serem embaralhados
* Vetor - elementos a serem embaralhados. O vetor deve conter pelo menos tamVetor elementos. Caso não contenha, os resultados serão impreditíveis, possivelmente desastrosos

**Módulo Baralho:**

* **LIS\_tppLista BAR\_CriarBaralho( void ( \* ExcluirValor ) ( void \* pDado ) ) :**
* Cria uma lista duplamente encadeada vazia.
* Parâmetros:
* Ponteiro para a função que processa a exclusão do valor referenciado pelo elemento a ser excluído.
* Retorna um ponteiro para a lista que representa o baralho.

* **BAR\_tpCarta BAR\_PedirCarta( LIS\_tppLista Baralho ) :**
* Retorna um elemento aleatório na lista recebida como parâmetro e o remove da lista.
* Parâmetros:
* Baralho – ponteiro para a lista que representa o baralho do qual a carta será removida
* Valor de retorno: um ponteiro para uma carta ou NULL.
* **LIS\_tpCondRet BAR\_DevolverCarta( LIS\_tppLista Baralho , BAR\_tpCarta Carta ) :**
* Devolve para a lista que representa o baralho a carta passada como parâmetro.
* Parâmetros:
* Baralho – Ponteiro para a cabeça da lista baralho
* Carta – Variável do tipo Carta
* Valor de retorno: LIS\_CondRetOK ou LIS\_CondRetFaltouMemoria.
* **LIS\_tpCondRet BAR\_SetarBaralho( LIS\_tppLista Baralho ) :**
* Preenche uma lista do tipo LIS\_tppLista com as 54 cartas convencionais de um baralho, incluindo dois coringas.
* Parâmetros:
* Baralho – Ponteiro para a cabeça da lista baralho
* Valor de retorno: LIS\_CondRetOK ou LIS\_CondRetFaltouMemoria.
* **LIS\_tpCondRet BAR\_SetarBaralhoTruco( LIS\_tppLista Baralho ):**
* Preenche uma lista do tipo LIS\_tppLista com as 40 cartas do jogo de truco.
* Parâmetros:
* Baralho – Ponteiro para a cabeça da lista baralho
* Valor de retorno: LIS\_CondRetOK ou LIS\_CondRetFaltouMemoria.
* **void BAR\_DestruirBaralho( LIS\_tppLista Baralho ) :**
* Desaloca a lista que representa o baralho.
* Parâmetros:
* Baralho – Ponteiro para a cabeça da lista baralho.

**Módulo Interface:**

* **void INT\_ExibirMesa( tpElemLista\* Descarte ) :**
* Exibe as cartas que foram jogadas na mesa e por quais jogadores até então.
* Parâmetros:
* Ponteiro para a lista que contém as cartas jogadas.
* **void INT\_InicializaJogo( tpElemLista\* jogadores, Equipe\* equipeA, Equipe\* equipeB ) :**
* Pergunta quantos jogadores farão parte do jogo e seus nomes.
* Armazena na lista jogadores os nomes dos jogadores.
* Preenche o campo equipe de cada jogador com A e B intercaladamente.
* Inicializa a pontuação de cada equipe com 0 pontos.
* Parâmetros:
* Um ponteiro para cada equipe, que armazena a pontuação e os membros da equipe. E o ponteiro para a lista de jogadores.
* **void INT\_ExibeMão( LIS\_tppLista jogadores ) :**
* Exibe as cartas da mão do jogador.
* Parâmetros:
* Ponteiro para a Cabeça da Lista de Jogadores, tendo o elemento corrente apontando para o nó do jogador em questão.
* **Carta\* INT\_Fazjogada( LIS\_tppLista jogadores ) :**
* Dá ao jogador a opção de escolher a carta que irá jogar.
* Recebe a carta escolhida pelo jogador.
* Valor de retorno: Ponteiro para a carta escolhida.
* Parâmetros:
* Ponteiro para a Cabeça da Lista de Jogadores, tendo o elemento corrente apontando para o nó do jogador em questão.
* **Int INT\_PedirTruco() :**
* Exibe a opção de Pedir Truco ou Prosseguir com a jogada.
* Recebe a escolha do jogador
* Valor de retorno: 1 para caso tenho pedido truco, 0 para caso tenha prosseguido com a jogada.
* **int INT\_ResponderTruco( ) :**
* Exibe as seguintes opções: Aceitar, Recusar ou Retrucar.
* Recebe a escolha do jogador.
* Valor de retorno: 1 caso tenha aceitado, retorna 2 caso tenha recusado, retorna 3 caso tenha retrucado
* **void INT\_TerminarRodada(LIS\_tppLista jogadores ) :**
* Exibe o nome do jogador que ganhou aquela rodada, e a sua equipe. Em casos de empate exibe que aquela rodada não teve vencedor.
* Parâmetros:
* Ponteiro para a Cabeça da Lista de Jogadores, tendo o elemento corrente apontando para o nó do jogador que venceu a rodada.

* **void INT\_TerminarMão( Equipe\* equipe, int\* pontuação ) :**
* Exibe a equipe que venceu aquela mão, e quantos pontos ela marcou.
* Exibe a pontuação total da equipe.
* Parâmetros:
* Ponteiro para a equipe que venceu a mão.
* Inteiro que indica quantos pontos a equipe marcou.
* **void INT\_TerminarJogo( Equipe\* equipeVenceu, Equipe\* equipePerdeu ) :**
* Exibe a pontuação de cada equipe.
* Exibe a equipe que venceu o jogo.
* Parâmetros:
* Ponteiro para as duas equipes.

**Módulo Jogo:**

* **void MãoNormal( LIS\_tppLista jogadores, LIS\_tppLista Baralho, int\* pontuação, Equipe\* equipeA, Equipe\* equipeB ) :**
* Utiliza a função BAR\_PedirCarta para preencher a mão de cada jogador com 3 cartas.
* O ponteiro pontuação aponta para o valor 1.
* Define as manilhas da mão.
* Inicializa rodadas até uma equipe vencer a mão.
* Após uma equipe vencer a mão, utiliza a função BAR\_DevolverCarta para retornar as cartas dos jogadores para o baralho.
* Atualiza a pontuação da equipe que vencer a mão, caso haja uma equipe vença a mão.
* Parâmetros:
* Ponteiro para a lista de jogadores, ponteiro para o baralho e ponteiro para a pontuação vigente da mão.
* **void Mãode11( LIS\_tppLista jogadores, LIS\_tppLista Baralho, int\* pontuação, Equipe\* equipeA, Equipe\* equipeB ) :**
* Utiliza a função BAR\_PedirCarta para preencher a mão de cada jogador com 3 cartas.
* Exibe as cartas de cada jogador da equipe com 11 pontos.
* Exibe a opção de aceitar ou recusar a mão para a equipe com 11pontos.
* Aceitar a mão da procedimento a Mãode11, Recusar a mão soma-se 1 ponto para a outra equipe e encerra a Mãode11.
* O ponteiro pontuação aponta para o valor 3.
* Define as manilhas da mão.
* Inicializa rodadas até uma equipe vencer a mão.
* Após uma equipe vencer a mão, utiliza a função BAR\_DevolverCarta para retornar as cartas dos jogadores para o baralho.
* Parâmetros:
* Ponteiro para a lista de jogadores, ponteiro para o baralho e ponteiro para a equipe com 11 pontos.
* **void MãoDeFerro( LIS\_tppLista jogadores, LIS\_tppLista Baralho, Equipe\* equipeA, Equipe\* equipeB ) :**
* Utiliza a função BAR\_PedirCarta para preencher a mão de cada jogador com 3 cartas.
* Define as manilhas da mão.
* Nesta função, as funções do Módulo Interface não exibem para o jogador as cartas da mão.
* Inicializa rodadas até uma equipe vencer a mão.
* Após uma equipe vencer a mão, utiliza a função BAR\_DevolverCarta para retornar as cartas dos jogadores para o baralho.
* Parâmetros:
* Ponteiro para a lista de jogadores e ponteiro para o baralho.
* **void Truco( int retorno, int\* pontuação, Equipe\* equipe ) :**
* Se o parâmetro retorno for 1, chama a função INT\_ResponderTruco.
* Caso essa função retorne 1, a pontuação passa a apontar para o próximo valor disponível.
* Caso essa função retorne 2, soma-se 1 na pontuação da equipe e encerra-se a mão vigente.
* Caso essa função retorne 3, a pontuação passa a apontar para o próximo valor disponível, e é chamada as seguinte funções : INT\_PedirTruco, INT\_ResponderTruco e PedidoDeTruco.
* Se o parâmetro retorno for 0, sai da função.
* Parâmetros :
* retorno : inteiro que representa a escolha do jogador, pedir truco(1) ou prosseguir com a jogada(0)
* Ponteiro para a pontuação vigente da mão.
* Ponteiro para a equipe cujo jogador pediu truco.
* **void EncerraJogo( Equipe\* equipeA, Equipe\* equipeB ) :**
* Compara a pontuação de cada equipe.
* Caso alguma das equipes de 11 ou mais pontos utiliza a função INT\_TerminarJogo e encerra-se o jogo (sai da main).
* Parâmetros:
* Ponteiro para as 2 equipes.