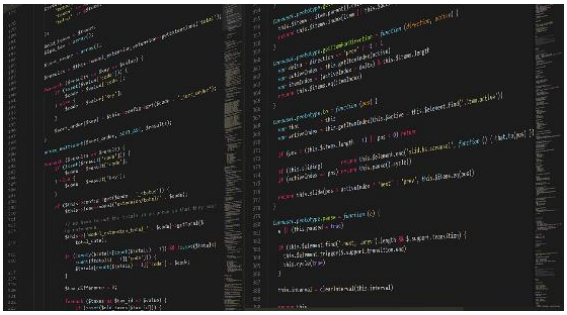
TÉCNICO LISBOA

Pokeristars:

Relatório de projeto



Curso: Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

UC: Programação

Autores: (Grupo 28)

- Afonso Alemão 96135 afonso.alemao@tecnico.ulisboa.pt

- Rui Daniel 96317 ruipcdaniel@tecnico.ulisboa.pt

Data: 10/05/2020

TÉCNICO LISBOA

Introdução

O nosso projeto *Pokeristars* encontra-se dividido em vários ficheiros:

- (1) ProjRuiAfonso.c //main
- (2) RAfuncauxiliares.c //funções auxiliares
- (3) RAmodoc.c //modo c: linha de comando
- (4) RAmododianddx.c //modo ficheiro de baralho e ficheiro de baralho extra
- (5) RAshuffle.c //Modo ficheiro de *shuffling*
- (6) ProjRuiAfonso.h

e também contém uma makefile.

O ficheiro ProjRuiAfonso.c contém o main do nosso programa, e os outros ficheiros contém o que o seu nome indica. Por exemplo RAfuncauxiliares.c, contém as iniciais dos nossos nomes R (Rui), A (Afonso), seguindo da indicação de que neste ficheiro se encontram funções auxiliares aos modos desenvolvidos neste projeto.

Nas próximas secções deste relatório iremos descrever de forma clara e concisa as nossas soluções para os problemas resolvidos. Iremos clarificar as nossas opções de implementação para as funções mais importantes do nosso programa e a forma como se estruturaram os dados para facilitar a produção de soluções e recolha de dados estatísticos.

Main

A função main verifica quais os modos escolhidos pelo utilizador, nomeadamente o modo c (linha de comando), -s1 (shuffle), modo -di e -dx (ficheiro de baralhos) e -o (identifica o modo de escrita de resultados num ficheiro), e procede ao encaminhamento do fluxo do programa para as funções correspondentes a cada modo. Caso algum destes modos seja incorretamente invocado a função deverá imprimir “-1\n” para stdout.

Caso o modo escolhido pelo utilizador seja o c, imprime no fim o resultado retornado pela função modoc. Caso seja escolhido um dos modos -s1 ou -d com o modo -o, será aberto o ficheiro em argv [4], para onde serão enviados os outputs do programa, caso contrário serão imprimidos em stdout. Isto é regulado pela variável ponteiroresultados.

Optámos por esta implementação de diferentes modos em funções distintas para tornar o nosso programa mais eficiente, e tornar mais fácil a leitura e compreensão do fluxo do código do nosso programa.

Modo c

No modo c, entram as informações relativas ao número de cartas a ser analisadas, e é no início deste que as cartas em argv são colocadas no vetor carta. Ocorrem as verificações relativas à validade de cada uma das cartas e se há cartas iguais nos modos de 5,7, 9 e em cada mão do modo de 10 cartas. Seguidamente, caso haja 5 cartas estas são enviadas para a função tipodemao, se houver 7 cartas para a função chamarcartas, se houver 9 para a funcaonovecartas e se houver 10 para a funcaodezcartas. O que estas funções retornarem é guardado em resultado, que será imprimido.

Função tipodemao:

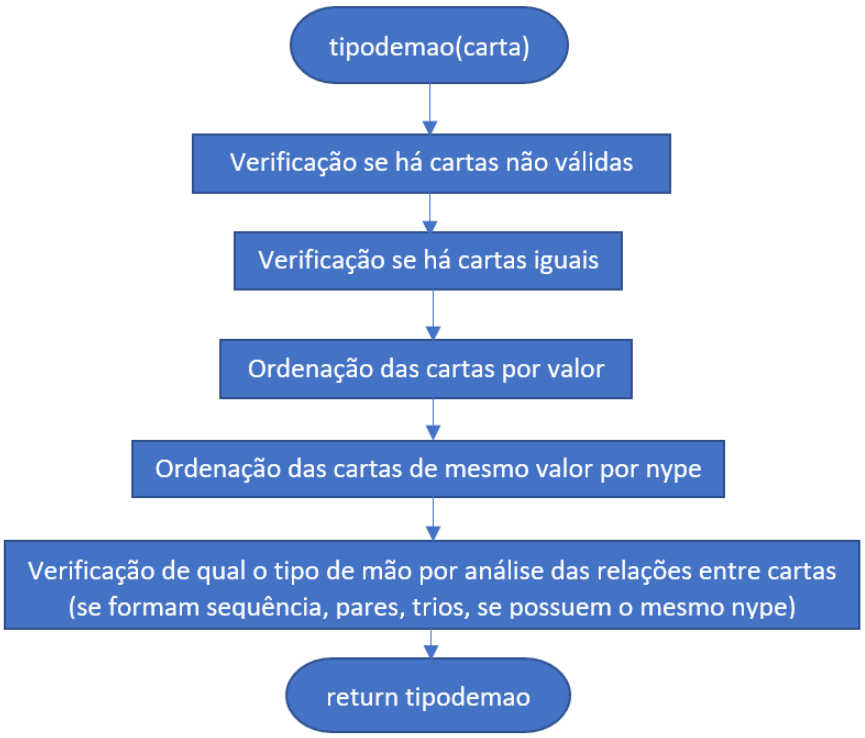


Figura 1: Fluxograma da função tipodemao

Esta é uma das funções mais importantes de todo o projeto, verifica se a mão se cartas é válida, ordena-as e procede à identificação de qual o tipo de mão. Nesta parte que aparece extremamente resumida no fluxograma iremos agora explicar o que foi feito:

Primeiramente identificamos vários indicadores para o tipo de mão: se é uma sequência, se é constituído pelas 5 cartas mais altas e se todas as cartas possuem o mesmo nype. (tipo de mão=t).

Caso tenhamos uma sequência, se as cartas não possuírem o mesmo nype temos um t= ‘5’, caso possuam o mesmo nype, se forem as 5 cartas mais altas o t= ‘10’, caso contrário temos um t= ‘9’. Também caso não haja sequência e as cartas forem todas do mesmo nype temos um t= ‘6’. Seguidamente e caso nenhum dos tipos anteriores se confirme, vamos fazer mais verificações:

Se houver 4 cartas iguais temos um t= ‘8’. Caso haja 3 cartas iguais e as outras 2 também diferentes e diferentes das restantes temos um t= ‘4’, caso estas outras 2 sejam iguais entre si e diferentes das restantes o t=’7’. Caso haja um só par temos t=’2’, e no caso de haver 2 pares diferentes entre si temos t=’2’. Em todos os outros casos t=’1’.

Ao longo de todo este processo, assim que é descoberto o tipo de mão, este é logo retornado. Optámos por esta implementação lógica, pois pareceu muito mais eficiente inicialmente identificar os casos de sequência e nype igual, e caso correspondesse a um desses casos sair logo da função, e só por fim identificar os casos do parágrafo anterior que necessitam de mais alguma lógica adicional, pois por exemplo um pair pode estar em diferentes locais do conjunto de 5 cartas, enquanto que a verificação de que se trata duma sequência é muito mais simples.

Função funcaodezcartas:

Visa receber 2 mãos e retornar qual destas a melhor, ou se houve empate. Em primeiro lugar, as cartas são todas validadas, e após isso as cartas do jogador 1 colocadas em x1 e as do jogador 2 colocadas em x2. Também é atribuído um valor do nype e um valor do tipo de carta a cada uma das cartas de cada jogador, após a determinação do tipo de mão de cada jogador, que automaticamente ordena as cartas por valor e por nype, nesta ordem de relevância.

Após isto verificamos qual jogador tem a melhor mão, se as mãos forem diferentes temos um vencedor. Caso tenham o mesmo tipo de mão, temos de ir verificar, tendo em conta qual é esse tipo de mão, qual a melhor das mãos tendo em conta os critérios desse tipo de mão.

Para não tornar esta leitura massuda, vou dar 2 exemplos desta análise:

Por exemplo para o tipo de mão ‘1’ basta irmos comparando a x1[0] com a x2[0], x1[1] com a x2[1], x1[2] com a x2[2], x1[3] com a x2[3] e a x1[4] com a x2[4], e no primeiro caso em que uma seja maior que outra, temos um vencedor. Isto ocorre porque a relevância neste tipo de mão é do género 0,1,2,3,4, sendo a carta [0] a mais relevante. Caso todas estas cartas resultem em empate procedemos à mesma análise por nype, através da funcaoanalisenype.

Um outro exemplo mais complexo é o da análise de empate com tipo de mão ‘2’, em que em primeiro lugar temos de analisar se um dos pares de uma mão é maior que o par da outra mão tendo para isso de determinar onde está o par, que como temos a mão ordenada pode estar em XXabc(tipo 1), aXXbc(tipo 2), abXXc(tipo 3) ou em abcXX(tipo 4). Se houver empate temos de analisar as cartas com a ordem de relevância partindo sempre do par de cada mão e só depois para as outras cartas, para isso temos de considerar os 16 tipos de confrontos que podemos ter entre mãos, por exemplo, tipo1 vs tipo1, tipo1 vs tipo 2…

No tipo de mão 5/9 que envolve sequências ainda tivemos de ter em conta uma análise em separada para o caso especial do género 5C 4C 3C 2C AC, em que a relevância é diferente do valor das cartas.

Toda esta implementação que descrevi foi realizada com nested if statements pois foi a forma que nos pareceu mais simples e rápida para a implementação de todas estas condições.

No final são colocadas as cartas do jogador vencedor no vetor de entrada carta e é retornado o índice do jogador vencedor ou 0 em caso de empate.

Função chamarcartas:

Recebe 7 cartas, imprime as 5 cartas da melhor mão possível e retorna o seu tipo de mão.

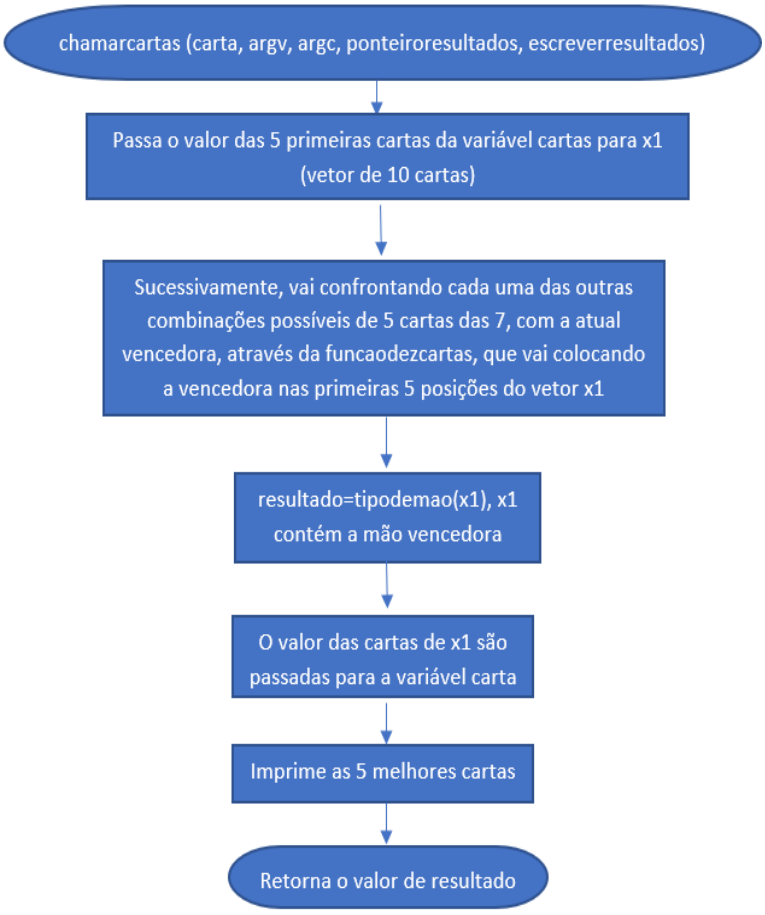


Figura 2: Fluxograma da função chamarcartas

Utilizámos um loop que gera todas as 21 combinações possíveis de 5 cartas entre as 7 para as ir sucessivamente confrontando através da funcaodezcartas.

Optámos por um sistema em que a mão que ganha é colocada no início do vetor x1, e a próxima mão entra no fim deste, de modo a não ser necessário a realização de todos os confrontos, mas sim apenas dos confrontos entre a melhor mão até ao momento com a próxima mão até termos uma mão vencedora. Desta forma melhoramos a eficiência do nosso programa.

Função funcaonovecartas:

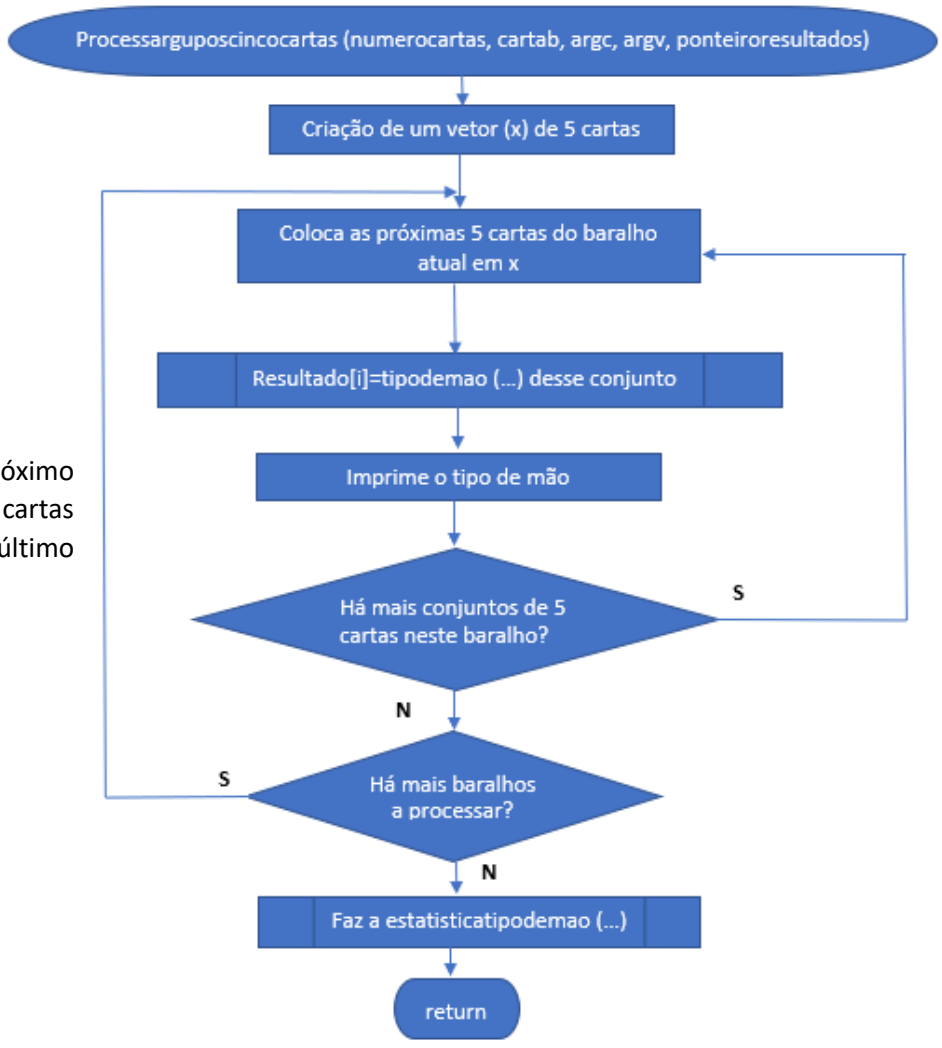
A função chamarcartas é chamada duas vezes, para selecionar as melhores cartas do jogador 1 (primeiras 5 cartas do vetor carta) e do jogador 2 (últimas 5 cartas do vetor carta). Por fim, é feito um confronto entre estas duas mãos a partir da funcaodezcartas e o valor de lá retornado é novamente retornado para main.

Modo -di

Iremos agora abordar as decisões tomadas na implementação do modo -di.

Optamos também por ler todas as cartas de um ficheiro ao mesmo tempo através de um loop, onde íamos sucessivamente lendo carta a carta para um vetor auxiliar, incrementando o número de cartas, e alocando espaço de memória para a colocação da carta no vetor de cartas, pois assim seria mais eficiente, visto que não seria necessário no futuro regressar a estas leituras, e de uma só vez todas as cartas de todos os baralhos ficariam colocadas num vetor que após ser processado é desalocado.

Para exemplificar apresentamos um fluxograma para o modo d1, ou seja a para a função processargruposcincocartas (figura x) , sendo que para os modos d2,d3,d4, o processo é bastante semelhantes mudando apenas o número de cartas processadas por baralho, que são respetivamente, 50 (10 conjuntos de 5 cartas), 49 (7 conjuntos de 7 cartas), 45 (5 conjuntos de 9 cartas), 50 (5 conjuntos de 10 cartas).



Passagem para o próximo baralho, ignorando as cartas não processadas do último baralho.

Figura 3: Fluxograma da função processargruposcincocartas

Para o tratamento estatístico dos dados de di, criámos uma função chamada estatisticatipodemao que compara os sucessivos valores dos tipos de mão processados nos conjuntos de 5,7,9, ou 10 cartas. Colocámos estes valores do tipo de mão num vetor chamado resultados, e para cada valor de resultados, incrementamos a posição correspondente no vetor das frequências absolutas de cada mão. A obtenção destes resultados para o modo d1, d2 resulta simplesmente da análise do valor retornado por tipodemao () e por chamarcartas (), já para d3 e d4 tivemos de dividir as 9/10 cartas nas mãos de cada um dos 2 jogadores do confronto, e individualmente verificar o tipo de mão de cada jogador. Após isto, obtivemos a frequência relativa pela divisão pelo número de mãos total e imprimimos estes valores.

Já para o modo d3 e d4, foi necessário entre "faz estatisticatipodemao(…)" e "return", também ir à função produzdadosintese, que diretamente através dos resultados anteriores nas funções processar grupos de 9 e de 10 cartas, incrementa em freqabsoluta[0] em caso de empate ou de existência de mãos inválidas, em freqabsoluta[i] se ganhou o jogador i.

Modo -dx

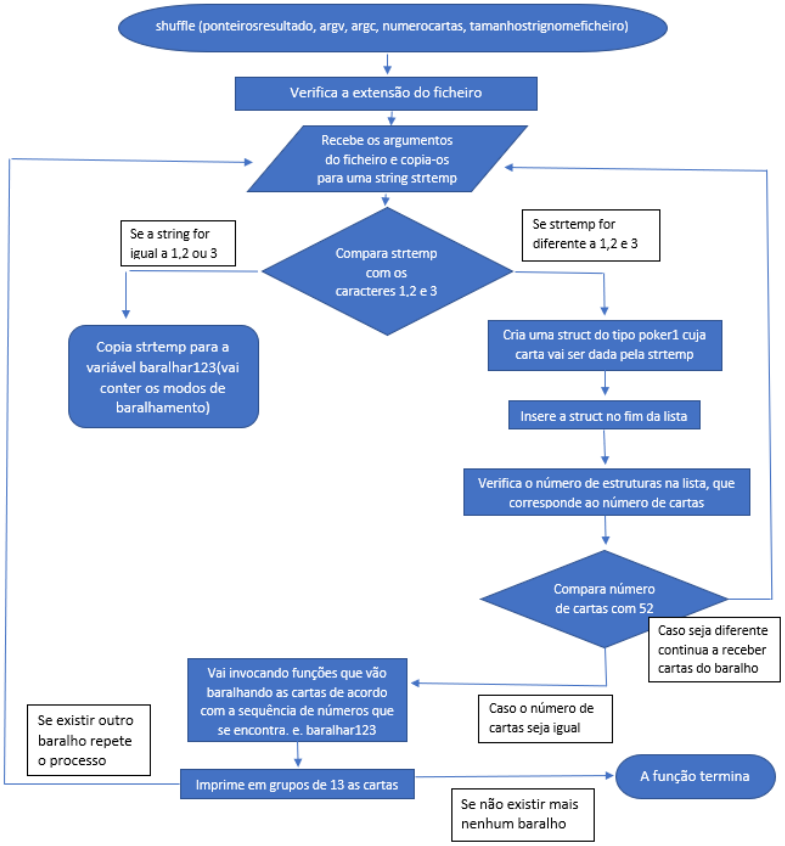
Para a realização do modo dx, as cartas lidas do ficheiro foram encaminhadas para a função torneiodepoker. Esta função é das mais complexas do nosso programa e visa a simulação de uma partida de póquer. Para cada uma das rondas o nosso programa faz os seguintes passos: Verificação dos acumuladores de fold e de idas a jogo de cada jogador para verificar quem vai a jogo e recebe cartas; distribuição de 2 cartas para cada um dos jogadores que vai a jogo, e das 5 cartas seguintes para todos eles; para cada jogador verificação se a sua mão é maior ou igual ao seu índice, se sim esse jogador vai a jogo e é imprimido o conjunto das suas melhores cartas, juntamente com o seu índice;

De seguida todos os jogadores que vão a jogo são confrontados em playerstofinalconfront, que verifica quais os que possuem com a mão com o valor máximo e os coloca num vetor chamado vencedores, assim poupamos tempo de execução pois futuramente não será necessário efetuar tantos confrontos; após isto todos os jogadores com a mão de maior valor serão confrontados na função confronto e o jogador que perder vai sendo removido do vetor dos vencedores, por uma função chamada retiranumerovetor; caso haja empate passamos para o próximo confronto; quando o vetor vencedores possuir apenas os índices vencedores, são imprimidos as melhores cartas do vencedor de menor índice e o índice de todos os vencedores; após isto é atribuída uma pontuação a cada um dos vencedores dividindo 1 ponto por todos os vencedores;

Após este procedimento em todas as rondas, partimos então para a produção da estatística das mãos vencedores, e para o print da tabela classificativa ordenada dos jogadores por ordem decrescente de pontuação pelas funções estatisticamaoganhadora (que imprime para cada mão a sua frequência relativa), e quadroclassificativo (que ordena os jogadores com um algoritmo do género do bubble sort), respetivamente.

Shuffle

No modo shuffle implementaram-se listas duplamente ligadas, constituídas por um conjunto de estruturas do tipo poker1,ou seja, que contêm uma carta do baralho, e dois apontadores, um que aponta para a estrutura que se encontra na próxima posição da lista (poker1->next) e outro que aponta para a estrutura que se encontra na posição anterior da lista (poker1->prev).



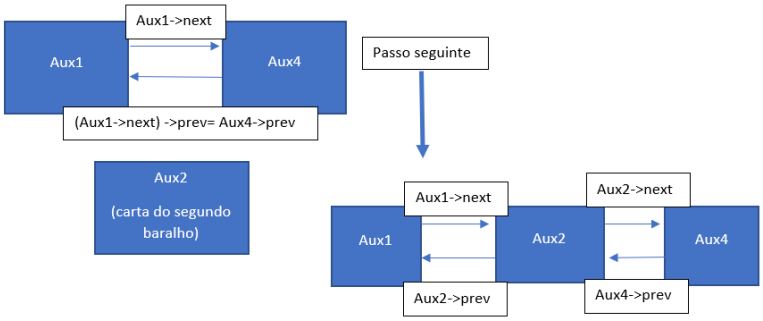
Para cada baralho lido do ficheiro, são aplicados os modos de baralhar, guardados no vetor baralhar123. Existem 3 modos distintos de baralhar: no modo 1 aplica-se uma intercalação das cartas; no modo 2 uma inversão e intercalação; e no modo 3 um corte.

Figura 4: Fluxograma da função shuffle

Função modointercalar:

Durante a execução desta mesma função inicializou-se Aux2 como apontador para a struct que contém a carta 27 e Aux1 foi inicializado com o valor do endereço de Listhead. Deste modo, de forma a intercalar as cartas da primeira metade com as da segunda usou-se um esquema igual aquele que se encontra em baixo representado (repetiu-se esse esquema 26 vezes), e que requer a salvaguarda do valor de Aux2->next e Aux1->next para que Aux1 e Aux2 tenham esse valor no próximo ciclo. No fim da função, Aux2 apontará para a estrutura com a última carta da lista, pelo Aux2->next deverá acabar apontando para NULL.

Lógica de inserção de uma carta da segunda na metade entre duas da primeira:



Sendo: Aux4=Aux1->next;

Colocamos a carta Aux2 entre Aux1 e Aux4:

Aux2->next=Aux4;
Aux2->prev=Aux1;
Aux4->prev=Aux2;
Aux1->next=Aux2;

Figura 5: Fluxograma da função modointercalar

Função modoinverter:

A partir desta função pretendemos inverter a segunda metade do baralho. Sendo assim, inicializou-se Aux1 com o valor do endereço da estrutura que contem a carta 27 e inicializou-se Aux2 com o valor do endereço da estrutura que contem a carta 52. Durante a função é usado um ciclo for no qual se troca o valor das cartas da estrutura apontada por Aux1 pelos valor das cartas da estrutura apontada por Aux2. O loop é executado 13 vezes e no fim de cada loop Aux1 é incrementado passando apontar para a estrutura que se encontra á sua frente na lista (Aux1->next). O que acontece com Aux2 no fim do loop é o oposto daquilo que acontece com Aux1, isto é, Aux2 passa a ter o valor de Aux2->prev (fica a apontar para a estrutura que fica atrás na lista).

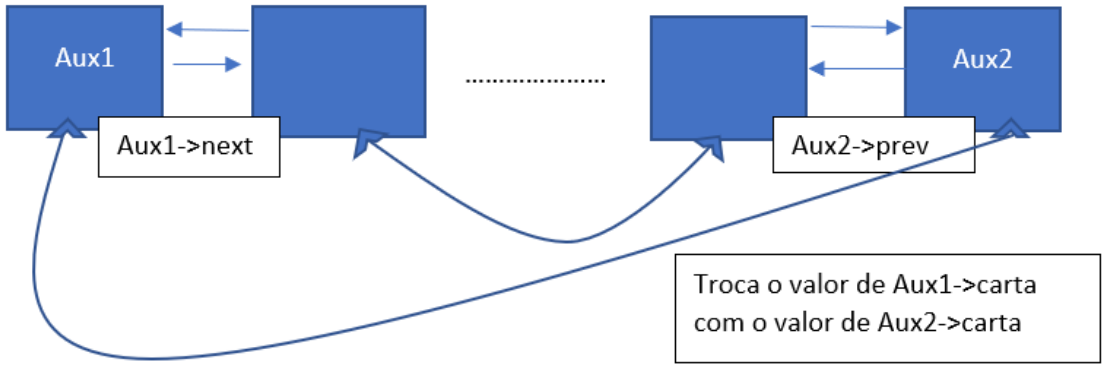


Figura 6: Fluxograma da função modoinverter

Função modocorte:

Neste caso inicializou-se Aux1 com o valor do endereço da estrutura que contém a carta 1(Listhead) e inicializou-se Aux2 com o valor do endereço da estrutura que contém a carta 27. Durante a função é usado um ciclo for no qual se troca o valor das cartas da estrutura apontada por Aux1 pelo valor das cartas da estrutura apontada por Aux2. O loop é executado 26 vezes e no fim de cada loop Aux1 Aux2 são incrementados passando apontar para a estrutura que se encontra á sua frente na lista (Aux1->next no caso de Aux1 e Aux2->next no caso de Aux2).