Projeto Final Lp1

DOMINO

Daniel Mendes 26385
Gonçalo Silva 26329
Versão 2.0
Terça, 27 de Dezembro de 2016

Índice

Table of contents

Índice das estruturas de dados

Estruturas de dados

Lista das estruturas de dados com uma breve descrição:

Índice dos ficheiros

Lista de ficheiros

Lista de todos os ficheiros com uma breve descrição:

Documentação da classe

Referência à estrutura iniciarpeca

```
#include <projetolp1primparte.h>
```

Campos de Dados

- int **nbar**
- int **nsob**
- int **npecasint**
- int **nseq**
- **PECA * pfirst**
- **SOBROU * psobrou**
- **PECAINT * pfirstint**
- **SEQUENCIA * pfirstseq**

Documentação dos campos e atributos

int nbar

int npecasint

int nseq

int nsob

PECA* pfirst

PECAINT* pfirstint

SEQUENCIA* pfirstseq

SOBROU* psobrou

A documentação para esta estrutura foi gerada a partir do seguinte ficheiro:

- **projetolp1primparte.h**

Referência à estrutura peca

`#include <projetolp1primparte.h>`

Campos de Dados

- char * **pecastr**
- struct **peca** * **pnext**

Documentação dos campos e atributos

char* pecastr

struct peca* pnext

A documentação para esta estrutura foi gerada a partir do seguinte ficheiro:

- **projetolp1primparte.h**

Referência à estrutura pecaint

`#include <projetolp1primparte.h>`

Campos de Dados

- int **dir**
- int **esq**
- struct **pecaint** * **pnextint**

Documentação dos campos e atributos

int dir

int esq

struct pecaint* pnextint

A documentação para esta estrutura foi gerada a partir do seguinte ficheiro:

- **projetolp1primparte.h**

Referência à estrutura sequencia

`#include <projetolp1primparte.h>`

Campos de Dados

- char * **seqstring**

Documentação dos campos e atributos

char* seqstring

A documentação para esta estrutura foi gerada a partir do seguinte ficheiro:

- **projetolp1primparte.h**

Referência à estrutura sobrou

```
#include <projetolp1primparte.h>
```

Campos de Dados

- char * **sobroustr**
- struct **sobrou** * **proximo**

Documentação dos campos e atributos

struct sobrou* proximo

char* sobroustr

A documentação para esta estrutura foi gerada a partir do seguinte ficheiro:

- **projetolp1primparte.h**

Documentação do ficheiro

Referência ao ficheiro main.c

```
#include "projetolp1primparte.h"
```

Funções

- int **main** (int argc, char *argv[])
  Função main.

Documentação das funções

int main (int *argc*, char * *argv*[])

Função main.

Função main

Parâmetros:

int	argc - numero de argumento do argv
char	* argv[] - array strings

```
42                                  {
49      main_projetolp1primparte(argc, argv);
50      return 0;
51      }
```

Referência ao ficheiro projetolp1primparte.c

```
#include "projetolp1primparte.h"
```

Funções

- int **main_projetolp1primparte** (int argc, char *argv[])
  Função main_projectolp1primparte.
- void **criarpecas** (char pecas[][**COLSTRING**])
  Criar peças string.
- void **imprimirpecas** (char pecas[][**COLSTRING**])
  Imprime as pecas em string.
- void **imprimirpecasint** (char pecas[][**COLSTRING**], int pecasint[][**COL**])
  Imprimir pecas em inteiros.
- void **baralhos** (**INICIARPECA** *b, char pecas[][**COLSTRING**], char baralho[][**COLSTRING**], char **sobrou**[][**COLSTRING**], int num)
  Criar baralhos/mãos aleatórios.
- void **convert_mao_string_to_int** (**INICIARPECA** *b, int pecastotal)
  Converter peças string para int.
- void **convert_mao_int_to_string** (**INICIARPECA** *b, int pecastotal)
  Converter peças int para string.
- void **rempeca** (**INICIARPECA** *b, int pecastotal, int npecasremover)
  Remover peças de uma ou mais mãos.
- void **inserir_peca_baralho** (**INICIARPECA** *b, char pecas[**COLSTRING**])
  Inserir pecas no baralho (lista ligada)
- void **inserir_sobrou_baralho** (**INICIARPECA** *b, char pecas[**COLSTRING**])
  Inserir pecas que sobraram do baralho (lista ligada)
- void **inserir_pecaint_baralho** (**INICIARPECA** *b, int dir, int esq)
  Inserir pecas do tipo int no baralho (lista ligada)
- char * **create_dyn_string** (char str[])
  Criar um array dinamico para strings.
- void **addpeca** (**INICIARPECA** *b, int pecastotal, int npecas)
  Adicionar peças numa mão/mãos.
- **PECA** * **find_peca_baralho** (**INICIARPECA** *b, char novapeca[])
  Procurar peça no baralho.
- void **create_array_pecas** (**INICIARPECA** *b, unsigned int n)
  Criar array dinamico para pecas.
- void **inserir_seq_iniciarpeca** (**INICIARPECA** *b, char seq[])
  Inserir sequencias no array dinamico.
- int **ordenarseq** (**INICIARPECA** *b, int num)
  Sequencias e ordenar sequencia por ordem decrescente.
- void **save_txt_jogo** (**INICIARPECA** b, char fname[])
  Guardar num ficheiro de texto.
- void **save_bin** (**INICIARPECA** b, char fnome[])
  Guardar num ficheiro binario.
- void **load_txt_jogo** (**INICIARPECA** *b, char fname[])
  Ler de um ficheiro de texto.

- void **load_bin** (**INICIARPECA** *b, char fname[])
  Ler de um ficheiro binario.
- void **procurar_padrao** (char arrayfinalcompleto[][150], int y)
  Funçao procurar e substituir padrão.
- void **seq_inicial** (char baralho[][**COLSTRING**])
  Faz sequencias a partir de uma sequencia inicial.
- void **retirar_mao_jogadores** (char baralho[][**COLSTRING**], int num)
  Cria as sequencias possiveis apartir de uma sequencia inicial com 2 ou mais mãos.

Documentação das funções

void addpeca (INICIARPECA * *b*, int *pecastotal*, int *npecas*)

Adicionar peças numa mão/mãos.

Adicionar peças na mão, sendo que as peças inseridas sao adicionadas á mão inicial e completada com peças aleatorias para preencher as mãos. Ex: o jogador tem uma mão de 7 peças, quer adicionar mais duas, ou seja adiciona essas duas, e sao adicionadas automaticamente 5 peças aleatorias, para completar duas mãos (14 peças)

Parâmetros:

INICIARPECA	* b , estrutura do tipo INICIARPECA
pecastotal	Valor do tipo int, utilizado para definir as peças totais usadas pelo jogador
npecas	Valor do tipo int, utilizado para definir o numero de peças a ser adicionado pelo jogador

int aleatorio=0 - Variavél onde é guardada o numero de peças total mais as peças adicionadas pelo jogador. Posteriormente, este valor é subtraido conforme as mãos usadas

int j=0 - Variavél usada nas funções seguintes

int k=0 - Variavél usada nas funções seguintes

char novapeca[4] - Usado para guardar a peça a ser inserida pelo jogador, para depois adicionar

char invertnewpeca[4] - Usado para guardar a peça invertida da novapeca inserida

```
  666                                                         {
  674     int aleatorio=0;
  676     int j=0;
  678     int k=0;
  680     char novapeca[4];
  682     char invertnewpeca[4]={};
  683
  684     SOBROU * pauxsob = NULL;
  685     PECA * pauxbar = NULL;
  686
  687     aleatorio=pecastotal+npecas;
  688     // Verificar se a soma da numero de pecas total com as peças que quer adicionar
está no primeiro baralho, entre 0 e 6. Se sim, subtrai o valor aleatorio por 6 e guarda em
aleatório o numero de peças que vao ser lançadas aleatóriamente. E assim sucessivamente para
os outros.
  689     if(aleatorio>0 && aleatorio<6){
  690         aleatorio=6-aleatorio;
  691     }else if(aleatorio>6 && aleatorio<13){
  692             aleatorio=13-aleatorio;
  693     }else if(aleatorio>13 && aleatorio<20){
  694         aleatorio=20-aleatorio;
```

```
  695     }else if(aleatorio>20 && aleatorio<27){
  696         aleatorio=27-aleatorio;
  697     }
  698     // Por exemplo, caso o utilizador adiciona 5 pecas, entao 5+7 (7 do baralho original)
=12 logo 2 pecas sao aleatorias e as outras 5 manual
  699
  700     pauxsob = b->psobrou;
  701     //inserir as pecas do sobrou para o baralho
  702     for(j=pecastotal;j<=(pecastotal+aleatorio);j++){
  703         inserir_peca_baralho(b,pauxsob->sobroustr);
  704         pauxsob=pauxsob->proximo;
  705     }
  706     // Imprime a mão/baralho mais as peças aleatorias
  707     pauxbar = b->pfirst;
  708     while(pauxbar!=NULL)
  709     {
  710         printf("%s\n",pauxbar->pecastr);
  711         pauxbar=pauxbar->pnext;
  712     }
  713
  714
  715     // Inserir peca (novapeca)
  716     int h=0;
  717
  718     for(k=0;k<npecas;k++){
  719             printf("Insira a peca");
  720             scanf("%s",novapeca);
  721
  722             // Inverter novapeca
  723             int d=strlen(novapeca)-1;
  724             for(h=0;h<(strlen(novapeca));h++){
  725                 invertnewpeca[d]=novapeca[h];
  726                 d--;
  727             }
  728             // Verifica se a peca da mão(baralho[]) é igual a novapeca, ou á invertida
  729             PECA * pauxfind = NULL;
  730             PECA * pauxfindinv = NULL;
  731
  732             pauxfind = find_peca_baralho(b,novapeca);
  733
  734             pauxfindinv = find_peca_baralho(b,invertnewpeca);
  735
  736             if(pauxfind != NULL || pauxfindinv != NULL){
  737                         printf("A peca ja existe!\n");
  738                         k--;
  739
  740             }else{
  741                         pauxbar = b->pfirst;
  742
  743                         //Caso contrario ele insere a nova peça na ultima posição da
mão (baralho[]) e imprime
  744
  745                         inserir_peca_baralho(b,novapeca);
  746
  747                         //imprime baralho todo
  748                         while(pauxbar!=NULL)
  749                         {
  750                             printf("%s\n",pauxbar->pecastr);
  751                             pauxbar=pauxbar->pnext;
  752                         }
  753             }
  754     }
  755 }
```

void baralhos (INICIARPECA * *b*, char *pecas*[][COLSTRING], char *baralho*[][COLSTRING], char *sobrou*[][COLSTRING], int *num*)

Criar baralhos/mãos aleatórios.

Cria um array de peças baralhadas (baralho), sendo esta a mão do jogador. E guarda noutro array (sobrou) as peças não utilizadas na mão.

Parâmetros:

INICIARPECA	* b , estrutura do tipo INICIARPECA
pecas[][COLSTRING]	Array do tipo char onde recebe as peças totais do jogo.
baralho[][COLSTRING]	Array do tipo char, onde irão ser guardadas as peças baralhadas da mão do jogador
sobrou[][COLSTRING]	Array do tipo char, onde irão ser guardadas as peças que sobraram, ou seja, que não foram usadas na mão do jogador
num	Valor do tipo int, utilizado para definir o numero de mãos a ser usado pelo jogador

srand((unsigned)time(NULL)) - Função srand(),responsável por alimentar o rand() e gerar números aleatórios

int pecastotal=0 - Número de peças total que o jogador tem na mão

int i=0 - Variavél usada nos ciclos

int aleatorio - Variavél que guarda número gerado aleatório

int array[28] - Array com as posicoes de 0 a 27 das pecas das mãos

int temp - Variavél temporária para guardas conteudo de array[i]

```
  251
{
  261     srand((unsigned)time(NULL));
  263     int pecastotal=0;
  265     int i=0;
  267     int aleatorio;
  269     int array[28];
  271     int temp;
  272
  273     PECA * pauxbar = NULL;
  274     SOBROU * pauxsob = NULL;
  275
  276     // Número de peças total que o jogador tem na mão, numero de mãos (1,2,3 ou 4)
multiplicado pelas pecas possiveis de cada mão (7)
  277     pecastotal = num * 7;
  278     //Caso o numero de mãos escollhidas seja igual ou inferior a 4, entra neste if
  279     if(num<=4){
  280             // Cria array posicoes das pecas totais
  281             for(i=0;i<28;i++){
  282                 array[i] = i;
  283             }
  284             // Baralha essas 28 pecas e guarda em array, de forma aleatória
  285             for(i=0;i<28;i++){
  286                 temp=array[i];
  287                 aleatorio = rand() % 28;
  288                 array[i]=array[aleatorio];
  289                 array[aleatorio]=temp;
  290             }
  291             // Guarda num array baralho[] as peças aleatórias até ao numero de pecas
total definida pelo jogador e imprime
  292             printf("MAO:\n");
  293             for(i=0;i<pecastotal;i++){
  294                 //strcpy(baralho[i],pecas[array[i]]);
  295                 //printf("%s\n",baralho[i]);
  296                 inserir_peca_baralho(b,pecas[array[i]]);
  297             }
  298
  299                 //imprime mao do baralho
  300                 pauxbar=b->pfirst;
  301
```

```
  302                 while(pauxbar!=NULL)
  303                 {
  304                     printf("%s\n",pauxbar->pecastr);
  305                      pauxbar=pauxbar->pnext;
  306                 }
  307             // Guarda num array sobrou[] as peças aleatórias desde o numero de pecas
total definida pelo jogador até as 27 possiveis e imprime
  308             printf("SOBROU:\n");
  309             for(i=pecastotal;i<28;i++){
  310                 //strcpy(sobrou[i],pecas[array[i]]);
  311                 //printf("%s\n",sobrou[i]);
  312
  313                 inserir_sobrou_baralho(b,pecas[array[i]]);
  314
  315             }
  316
  317             //imprime sobras do baralho
  318             pauxsob=b->psobrou;
  319
  320                 while(pauxsob!=NULL)
  321                 {
  322                     printf("%s\n",pauxsob->sobroustr);
  323                      pauxsob=pauxsob->proximo;
  324                 }
  325
  326     }else if(num>4){
  327         printf("Nao pode escolher mais que 4 baralhos!\n");
  328     }
  329 }
```

void convert_mao_int_to_string (INICIARPECA * *b*, int *pecastotal*)

Converter peças int para string.

Converte peças inteiro para strings

Parâmetros:

INICIARPECA	* b , estrutura do tipo INICIARPECA
pecastotal	Valor do tipo int, utilizado para definir as peças totais usadas pelo jogador

```
  370 {
  377     PECA * paux = NULL;
  378     PECA * pauxbar = NULL;
  379     PECAINT * pauxint = NULL;
  380     char str[4];
  381     int aux1 = 0 ,aux2 = 0,i = 0;
  382     int tamanhobar=b->nbar;
  383
  384     paux = b->pfirst;
  385     //percorro o baralho inicial e meto tudo vazio
  386     while(paux!=NULL)
  387     {
  388             paux->pecastr=NULL;
  389             b->nbar--;
  390             paux = paux->pnext;
  391     }
  392
  393
  394     //percorro a lista das pecas int e transformo e coloco de volta no baralho original
(b->pfirst)
  395     pauxint = b->pfirstint;
  396     for(i=0;i<tamanhobar;i++)
  397     {
  398         aux1=pauxint->dir;
  399         aux2=pauxint->esq;
  400         str[0]=aux1+'0';
  401         str[1]='|';
```

```
402         str[2]=aux2+'0';
403
404         inserir_peca_baralho(b,str);
405
406         pauxint = pauxint->pnextint;
407     }
408
409     pauxbar = b->pfirst;
410
411     //imprime baralho todo
412     while(pauxbar!=NULL)
413     {
414         printf("%s\n",pauxbar->pecastr);
415         pauxbar=pauxbar->pnext;
416     }
417 }
```

void convert_mao_string_to_int (INICIARPECA * *b*, int *pecastotal*)

Converter peças string para int.

Converte peças strings para inteiro

Parâmetros:

INICIARPECA	* b , estrutura do tipo INICIARPECA
pecastotal	Valor do tipo int, utilizado para definir as peças totais usadas pelo jogador

```
337 {
338     PECA * paux = NULL;
339     PECAINT * pauxbar = NULL;
340     char aux1[4];
341     char aux2[4];
342
343     paux = b->pfirst;
344
345     while(paux!=NULL)
346     {
347         strcpy(aux1,paux->pecastr);
348         strcpy(aux2,paux->pecastr);
349         aux1[1]='\0';
350         aux2[0]=aux2[2];
351         aux2[1]='\0';
352
353         inserir_pecaint_baralho(b,atoi(aux1),atoi(aux2));
354
355         paux = paux->pnext;
356     }
357
358     pauxbar = b->pfirstint;
359
360     //imprime baralho todo
361     while(pauxbar!=NULL)
362     {
363         printf("%d|%d\n",pauxbar->dir,pauxbar->esq);
364         pauxbar=pauxbar->pnextint;
365     }
366 }
```

void create_array_pecas (INICIARPECA * *b*, unsigned int *n*)

Criar array dinamico para pecas.

Remove peças de uma ou mais mãos, sendo que as peças removidas são substituidas por X|X, e é pedido ao utilizador para inserir tantas peças quantas removeu. Sendo tudo verificado para não remover peças que não existam e que não sejam adicionas peças repetidas.

Parâmetros:

INICIARPECA	* b , estrutura do tipo INICIARPECA
int	n, quantidade que vamos alocar

```
781 {
787     SEQUENCIA * paux = NULL;
788     SEQUENCIA * pnew = NULL;
789 
790     int i=0;
791     int j=0;
792 
793     if(b->pfirstseq==NULL && b->nseq==0)
794     {
795 
796         pnew=(SEQUENCIA*)malloc(sizeof(SEQUENCIA)*n);
797         b->nseq = n;
798 
799         for(i=0;i<n;i++)
800         {
801             (pnew+i)->seqstring = NULL;
802 
803         }
804         b->pfirstseq=pnew;
805     }else{
806 
807             b->nseq = n;
808             pnew=(SEQUENCIA*)malloc(sizeof(SEQUENCIA)*n);
809             paux=b->pfirstseq;
810 
811 
812         //copiar para o novo espaço
813         for(i=0;i<(b->nseq - 1);i++)
814         {
815             (pnew+i)->seqstring = (paux+i)->seqstring;
816         }
817         for(j=i;j<n;j++)
818         {
819             (pnew+j)->seqstring = NULL;
820         }
821 
822         b->pfirstseq=pnew;
823 
824     }
825 }
```

char* create_dyn_string (char *str*[])

Criar um array dinamico para strings.

Criar um array dinamico para strings

Parâmetros:

char	str[], recebe uma string

```
650 {
655     char *paux=NULL;
656     int slen = strlen(str)+1;
657 
658     paux=(char*)malloc(sizeof(char)*slen);
659 
660     strcpy(paux,str);
661 
662     return paux;
663 }
```

void criarpecas (char *pecas*[][COLSTRING])

Criar peças string.

São criadas todas as peças possiveis que um jogo tem e são copiadas através do strcpy para dentro do array pecas

Parâmetros:

pecas[][COLSTRING]	Array do tipo char guarda as peças totais do jogo.

O que está comentado - Peças teste para usar no ponto R7, visto que pecas aleatorias de 2 ou mais mãos, ele não tem memória para gerar as sequencias

```
169                                             {
174     strcpy(pecas[0],"0|0");
175     strcpy(pecas[1],"0|1");
176     strcpy(pecas[2],"0|2");
177     strcpy(pecas[3],"0|3");
178     strcpy(pecas[4],"0|4");
179     strcpy(pecas[5],"0|5");
180     strcpy(pecas[6],"0|6");
181     strcpy(pecas[7],"1|1");
182     strcpy(pecas[8],"1|2");
183     strcpy(pecas[9],"1|3");
184     strcpy(pecas[10],"1|4");
185     strcpy(pecas[11],"1|5");
186     strcpy(pecas[12],"1|6");
187     strcpy(pecas[13],"2|2");
188     strcpy(pecas[14],"2|3");
189     strcpy(pecas[15],"2|4");
190     strcpy(pecas[16],"2|5");
191     strcpy(pecas[17],"2|6");
192     strcpy(pecas[18],"3|3");
193     strcpy(pecas[19],"3|4");
194     strcpy(pecas[20],"3|5");
195     strcpy(pecas[21],"3|6");
196     strcpy(pecas[22],"4|4");
197     strcpy(pecas[23],"4|5");
198     strcpy(pecas[24],"4|6");
199     strcpy(pecas[25],"5|5");
200     strcpy(pecas[26],"5|6");
201     strcpy(pecas[27],"6|6");
202
204 /*
205     strcpy(pecas[0],"2|5");
206     strcpy(pecas[1],"3|3");
207     strcpy(pecas[2],"2|2");
208     strcpy(pecas[3],"1|1");
209     strcpy(pecas[4],"4|4");
210     strcpy(pecas[5],"5|5");
211     strcpy(pecas[6],"6|6");
212     strcpy(pecas[7],"1|2");
213     strcpy(pecas[8],"6|3");
214     strcpy(pecas[9],"1|4");
215     strcpy(pecas[10],"1|5");
216     strcpy(pecas[11],"0|6");
217     strcpy(pecas[12],"2|1");
218     strcpy(pecas[13],"0|3");
219 */
220
221 }
```

PECA* find_peca_baralho (INICIARPECA * *b*, char *novapeca*[])

Procurar peça no baralho.

Verificar se a peça existe no baralho, caso haja retorna a peça, senão retorna NULL

Parâmetros:

INICIARPECA	* b , estrutura do tipo INICIARPECA
char	novapeca[], recebe a nova peça para verificar se existe no baralho

```
759 {
765     PECA *paux = NULL;
766     paux = b->pfirst;
767
768     while(paux !=NULL){
769         if( strcmp(paux->pecastr,novapeca)==0 ){
770             return paux;
771         }else{
772             paux=paux->pnext;
773         }
774     }
775
776     return NULL;
777 }
```

void imprimirpecas (char *pecas*[][COLSTRING])

Imprime as pecas em string.

Imprime o array das 28 pecas em string

Parâmetros:

pecas[][COLSTRING]	Array do tipo char onde recebe as peças totais do jogo.

```
223                                             {
228     int i = 0;
229     for(i=0;i<28;i++){
230         printf("%s\n",pecas[i]);
231     }
232 }
```

void imprimirpecasint (char *pecas*[][COLSTRING], int *pecasint*[][COL])

Imprimir pecas em inteiros.

Imprime o array das 28 pecas em inteiros, guarda apenas os dois valores inteiros e ignora a barra. Ex: 1 3

Parâmetros:

pecas[][COLSTRING]	Array do tipo char onde recebe as peças totais do jogo.
pecasint[][COL]	Array do tipo int onde guarda as peças totais do jogo.

```
236                                                                 {
242     int i=0;
243     for(i=0;i<28;i++){
244         pecasint[i][0]=atoi(pecas[i]);
245         pecasint[i][1]=atoi(&pecas[i][2]);
246         printf("%d|%d \n", pecasint[i][0],pecasint[i][1]);
247     }
248 }
```

void inserir_peca_baralho (INICIARPECA * *b*, char *pecas*[COLSTRING])

Inserir pecas no baralho (lista ligada)

Remove peças de uma ou mais mãos, sendo que as peças removidas são substituidas por X|X, e é pedido ao utilizador para inserir tantas peças quantas removeu. Sendo tudo verificado para não remover peças que não existam e que não sejam adicionas peças repetidas.

Parâmetros:

INICIARPECA	* b , estrutura do tipo INICIARPECA
char	pecas[COLSTRING] peca nova a inserir

```
535 {
541     PECA *pnew = (PECA*)malloc(sizeof(PECA));
542     PECA *paux = NULL;
543 
544     pnew->pecastr = create dyn string(pecas);
545     pnew->pnext=NULL;
546 
547     paux=b->pfirst;
548 
549     if(b->pfirst==NULL)
550     {
551 
552         b->pfirst=pnew;
553         b->nbar++;
554     }
555     else
556     {
557         //insere na cauda
558 
559         while(paux->pnext!=NULL)
560         {
561 
562             paux=paux->pnext;
563 
564         }
565 
566         paux->pnext=pnew;
567         b->nbar++;
568     }
569 
570 }
```

void inserir_pecaint_baralho (INICIARPECA * *b*, int *dir*, int *esq*)

Inserir pecas do tipo int no baralho (lista ligada)

Remove peças de uma ou mais mãos, sendo que as peças removidas são substituidas por X|X, e é pedido ao utilizador para inserir tantas peças quantas removeu. Sendo tudo verificado para não remover peças que não existam e que não sejam adicionas peças repetidas.

Parâmetros:

INICIARPECA	* b , estrutura do tipo INICIARPECA
int	dir , valor do tipo inteiro, da parte direita da peça
int	esq , valor do tipo inteiro, da parte esquerda da peça

```
612 {
619     PECAINT *pnew = (PECAINT*)malloc(sizeof(PECAINT));
620     PECAINT *paux = NULL;
621 
622     pnew->dir = dir;
```

```
623     pnew->esq = esq;
624     pnew->pnextint=NULL;
625
626     paux=b->pfirstint;
627
628     if(b->pfirstint==NULL)
629     {
630
631         b->pfirstint=pnew;
632         b->npecasint++;
633
634     }
635     else
636     {
637         //insere na cauda
638         while(paux->pnextint!=NULL)
639         {
640             paux=paux->pnextint;
641         }
642
643         paux->pnextint=pnew;
644         b->npecasint++;
645     }
646 }
```

void inserir_seq_iniciarpeca (INICIARPECA * *b*, char *seq*[])

Inserir sequencias no array dinamico.

Inserir sequencias no array dinamico, caso não haja mais espaço para alocar, ele abre mais espaço recorrendo a outra função

Parâmetros:

INICIARPECA	* b , estrutura do tipo INICIARPECA
char	seq[], recebe sequencia a inserir no array dinamico

```
829 {
835     SEQUENCIA *paux=NULL;
836
837     paux=b->pfirstseq;
838
839     while(paux!=NULL&&paux->seqstring!=NULL && (paux - (b->pfirstseq)) < b->nseq)
840     {
841         paux++;
842     }
843
844     if((paux - (b->pfirstseq))==b->nseq)
845     {
846         create array pecas(b,b->nseq+1);
847         paux = b->pfirstseq + b->nseq-1;
848         paux->seqstring = create dyn string(seq);
849
850
851     }else
852     {
853
854         paux->seqstring=create_dyn_string(seq);
855
856     }
857
858 }
```

void inserir_sobrou_baralho (INICIARPECA * *b*, char *pecas*[COLSTRING])

Inserir pecas que sobraram do baralho (lista ligada)

Remove peças de uma ou mais mãos, sendo que as peças removidas são substituidas por X|X, e é pedido ao utilizador para inserir tantas peças quantas removeu. Sendo tudo verificado para não remover peças que não existam e que não sejam adicionas peças repetidas.

Parâmetros:

INICIARPECA	* b , estrutura do tipo INICIARPECA
char	pecas[COLSTRING] peca nova a inserir

```
574 {
580     SOBROU *pnew = (SOBROU*)malloc(sizeof(SOBROU));
581     SOBROU *paux = NULL;
582
583     pnew->sobroustr = create dyn string(pecas);
584     pnew->proximo=NULL;
585
586     paux=b->psobrou;
587
588     if(b->psobrou==NULL)
589     {
590
591         b->psobrou=pnew;
592         b->nsob++;
593     }
594     else
595     {
596         //insere na cauda
597
598         while(paux->proximo!=NULL)
599         {
600             paux=paux->proximo;
601         }
602
603         paux->proximo=pnew;
604         b->nsob++;
605
606         return;
607     }
608 }
```

void load_bin (INICIARPECA * *b*, char *fname*[])

Ler de um ficheiro binario.

Le as peças de uma ou mais maos e insere no jogo

Parâmetros:

INICIARPECA	* b , estrutura do tipo INICIARPECA
char	fname[], recebe o nome do ficheiro

```
1323 {
1329     FILE *fp=NULL;
1330
1331     int j=0;
1332     int size=0;
1333     char peca[50]="";
1334     int n = 0;
1335     if((fp=fopen(fname,"rb"))!=NULL)
1336     {
1337
1338         fread(&(b->nbar),sizeof(int),1,fp);
1339         n = b->nbar;
1340         b->nbar = 0;
1341
1342         //inserir pecas baralho
```

```
1343          for(j=0;j<n;j++)
1344          {
1345              fread(&size,sizeof(int),1,fp);
1346              fread(peca,sizeof(char),size,fp);
1347              printf("%s\n",peca);
1348              inserir_peca_baralho(b,peca);
1349          }
1350          fclose(fp);
1351      }
1352 }
```

void load_txt_jogo (INICIARPECA * *b*, char *fname*[])

Ler de um ficheiro de texto.

Le as peças de uma ou mais maos e insere no jogo

Parâmetros:

INICIARPECA	* b , estrutura do tipo INICIARPECA
char	fname[], recebe o nome do ficheiro

```
1283 {
1290      FILE *fp=NULL;
1291      int i=0;
1292      char peca[50];
1293      int nfields=0;
1294
1295
1296      if((fp=fopen(fname,"r"))==NULL)
1297      {
1298
1299          printf("... ERRO ...");
1300          return;
1301
1302      }
1303
1304
1305      for(i=0;i<7;i++)
1306      {
1307
1308          nfields=fscanf(fp,"%*[\n] %[^,] %*[,]",peca);
1309
1310          if(nfields>0)
1311          {
1312              inserir_peca_baralho(b,peca);
1313
1314          }
1315
1316      }
1317
1318
1319 }
```

int main_projetolp1primparte (int *argc*, char * *argv*[])

Função main_projectolp1primparte.

char pecas[LINSTRING][COLSTRING] - Array de strings para pecas

char baralho[LINSTRING][COLSTRING] - Array de strings de peças baralhadas, dependendo das mãos que o jogador pedir (4 mãos de 7 peças no máximo)

char sobrou[LINSTRING][COLSTRING] - Array de strings de pecas baralhadas, onde ficam as peças não utilizadas na mão do jogador

char arrayfinalcompleto[3000][150] - Array de strings de todas as sequencias possiveis das mãos do jogador

int opc=0 - Opção escolhida no menu inicial

int num=0 - Número de baralhos que o jogador escolhe (máximo 4)

int npecasremover=0 - Número de peças que o utilizador pretende remover da sua mão

int pecastotal=0 - Número de peças total que o jogador tem na mão, consoante o número de mãos escolhidas. Ex: 2 mão x 7 peças = 14 peças total

int npecas=0 - Número de peças que o utilizador pretende adicionar á sua mão

int y=0 - Tamanho do array de strings de todas as sequencias possiveis das mãos do jogador (arraycompleto)

criarpecas(pecas) - Criar todas peças do jogo

bool sair = false - Manter ciclo do menu enquanto for false, quando for true sai

char escolha - Usado para quando é escrito "S" ou "s" fecha programa, caso contrário continua

```
   44                                                          {
   46     char pecas[LINSTRING][COLSTRING];
   48     char baralho[LINSTRING][COLSTRING];
   50     char sobrou[LINSTRING][COLSTRING];
   52     char arrayfinalcompleto[3000][150];
   54     int opc=0;
   56     int num=0;
   58     int npecasremover=0;
   60     int pecastotal=0;
   62     int npecas=0;
   64     int y=0;
   66     criarpecas(pecas);
   68     bool sair = false;
   70     char escolha;
   71
   72     INICIARPECA b = {0,0,0,0,NULL,NULL,NULL,NULL};
   73
   74     create_array_pecas(&b,2);
   75
   76
   77     //Menu com ciclo. O utilizador escolhe a opção que pretende e através do switch
retorna para a função pretendida
   78     do{
   79     printf("**************** JOGO DO DOMINO ***************\n\n");
   80     printf("Escolha uma opcao\n");
   81     printf("1 ----> Listar Pecas Sring\n");
   82     printf("2 ----> Criar Mao\n");
   83     printf("3 ----> Remover Pecas\n");
   84     printf("4 ----> Adicionar Pecas\n");
   85     printf("5 ----> Listar baralho/mao string to int\n");
   86     printf("6 ----> Criar e ordenar sequencias\n");
   87     printf("7 ----> Procurar padrao\n");
   88     printf("8 ----> Criar sequencias com sequencia inicial\n");
   89     printf("9 ----> Criar sequencias com maos alternadas\n");
   90     printf("10 ----> Listar baralho/mao int to string\n");
   91     printf("11 ----> Guardar baralho/sequencias em ficheiro txt\n");
   92     printf("12 ----> Guardar sequencias em ficheiro bin\n");
   93     printf("13 ----> Ler de um ficheiro txt\n");
   94     printf("14 ----> Ler de um ficheiro binario\n");
   95
   96     printf("Opcao: ");
   97     scanf("%d", &opc);
   98
   99     switch(opc)
  100             {
```

```
101             case 1:
102                 imprimirpecas(pecas);
103                 break;
104             case 2:
105                 printf("Insira o numero de mãos a jogar:\n");
106                 scanf("%d",&num);
107                 baralhos(&b,pecas,baralho,sobrou,num);
108                 break;
109             case 3:
110                 printf("Quantas pecas pretende remover:\n");
111                 scanf("%d",&npecasremover);
112                 pecastotal=num*7;
113                 rempeca(&b,pecastotal,npecasremover);
114                 break;
115             case 4:
116                 printf("Quantas pecas pretende adicionar:\n");
117                 scanf("%d",&npecas);
118                 pecastotal=num*7;
119                 addpeca(&b,pecastotal,npecas);
120                 break;
121             case 5:
122                 pecastotal=num*7;
123                 convert mao string to int(&b,pecastotal);
124                 break;
125             case 6:
126                 y=ordenarseq(&b,num);
127                 break;
128             case 7:
129                 procurar_padrao(arrayfinalcompleto,y);
130                 break;
131             case 8:
132                 seq inicial(baralho);
133                 break;
134             case 9:
135                 retirar_mao_jogadores(baralho,num);
136                 break;
137             case 10:
138                 pecastotal=num*7;
139                 convert_mao_int_to_string(&b,pecastotal);
140                 break;
141             case 11:
142                 save_txt_jogo(b,"./wbarseq.txt");
143                 break;
144             case 12:
145                 save_bin(b,"./binwseq.bin");
146                 break;
147             case 13:
148                 load_txt_jogo(&b,"./rbar.txt");
149                 break;
150             case 14:
151                 load bin(&b,"binrbar.bin");
152                 break;
153
154             default:
155             printf("Escolha invalida!\n\n");
156             }
157             printf("Pretende sair? S --->sim \n");
158             scanf("%s",&escolha);
159             if(escolha=='S'||escolha=='s'){
160                 sair=true;
161             }else{
162                 sair=false;
163             }
164     }while(sair==false);
165     return 0;
166 }
```

int ordenarseq (INICIARPECA * *b*, int *num*)

Sequencias e ordenar sequencia por ordem decrescente.

Numa primeira parte, ele faz quatro verificações para juntar duas peças. Numa outra parte ele verifica se é possivel adicionar sequencias de mais de tres peças e vai juntando no arraydinamico.Tudo isto com verificações a ver se há repetidas, ou invertidas.

Parâmetros:

INICIARPECA	* b , estrutura do tipo INICIARPECA
num	Valor do tipo int, utilizado para definir o numero de mãos a ser usado pelo jogador

char invertnewpeca[150] - Usado para guardar um peça inversa

char invertnewpecaaux[150] - Usado para guardar um peça inversa

char invert[150] - Usado para guardar um peça inversa

char inversopeca[150] - Usado para guardar um peça inversa

char invertfinal[500] - Usado para guardar um peça inversa

```
  861                                         {
  867 int i=0,j=0,h=0,z=0;
  868 int x=0,k=0;
  869 char aux[500];
  871 char invertnewpeca[150];
  873 char invertnewpecaaux[150];
  875 char invert[150];
  877 char inversopeca[150];
  879 char invertfinal[500];
  880
  881 num=num*7;
  882                 //Juntar 2 pecas
  883
  884                 PECA * paux = NULL;
  885                 PECA * paux2 = NULL;
  886                 paux = b->pfirst;
  887
  888                 //percorre lista ligada baralho, uso o paux que é o primeiro e depois
o paux2 do segundo para a frente ate NULL
  889                 while(paux!=NULL)
  890                 {
  891                     paux2 = paux->pnext;
  892                     while(paux2!=NULL)
  893                     {
  894                         if((paux->pecastr)[2]==(paux2->pecastr)[0])
  895                         {
  896                             //concatena
  897                             strcpy(aux, paux->pecastr);
  898                             strcat(aux,"-");
  899                             strcat(aux, paux2->pecastr);
  900                             inserir_seq_iniciarpeca(b,aux);
  901
  902                         }
  903                         if((paux->pecastr)[2]==(paux2->pecastr)[2]){
  904                             // Inverter peca
  905                             int d=strlen(paux2->pecastr)-1;
  906                             for(h=0;h<(strlen(paux2->pecastr));h++){
  907                                 invertnewpeca[d]=paux2->pecastr[h];
  908                                 d--;
  909                             }
  910
  911                             // concatena
  912                             strcpy(aux, paux->pecastr);
  913                             strcat(aux,"-");
  914                             strcat(aux, invertnewpeca);
  915                             inserir seq iniciarpeca(b,aux);
  916                         }
  917                         if((paux->pecastr)[0]==(paux2->pecastr)[2]){
```

```
  918                             // concatena
  919                             strcpy(aux, paux2->pecastr);
  920                             strcat(aux,"-");
  921                             strcat(aux, paux->pecastr);
  922                             inserir seq iniciarpeca(b,aux);
  923                         }
  924                         if((paux->pecastr)[0]==(paux2->pecastr)[0]){
  925                             // Inverter peca
  926                             int d=strlen(paux2->pecastr)-1;
  927                             for(h=0;h<(strlen(paux2->pecastr));h++){
  928                                 invertnewpeca[d]=(paux2->pecastr)[h];
  929                                 d--;
  930                             }
  931 
  932                             // concatena
  933                             strcpy(aux, invertnewpeca);
  934                             strcat(aux,"-");
  935                             strcat(aux, paux->pecastr);
  936                             inserir_seq_iniciarpeca(b,aux);
  937                         }
  938                         paux2 = paux2->pnext;
  939                     }
  940                     paux = paux->pnext;
  941                 }
  942 
  943 
  944     //Verificar se ha sequencias iguais e o inversos tambem
  945     for(x=0;x<b->nseq;x++){
  946                 // Inverter peca
  947                 int w=strlen((b->pfirstseq+x)->seqstring)-1;
  948                 for(k=0;k<(strlen((b->pfirstseq+x)->seqstring));k++){
  949                     invertnewpecaaux[w]=((b->pfirstseq+x)->seqstring)[k];
  950                     w--;
  951                 }
  952     // Verifica se é igual e se for substitui por X|X
  953             for(i=x+1;i<b->nseq;i++){
  954 
if((strcmp(((b->pfirstseq+i)->seqstring),invertnewpecaaux)==0)||(strcmp(((b->pfirstseq+i
)->seqstring),((b->pfirstseq+x)->seqstring))==0)||(((((b->pfirstseq+i)->seqstring)[0])==
(((b->pfirstseq+i)->seqstring)[6]))&&((((b->pfirstseq+i)->seqstring)[2])==(((b->pfirstse
q+i)->seqstring)[4])))){
  955                     strcpy(((b->pfirstseq+i)->seqstring),"X|X");
  956                 }
  957             }
  958     }
  959 
  960     //remover os X|X e passa os debaixo para cima
  961 
  962         for(i=0;i<b->nseq;i++)
  963         {
  964             if(strcmp(((b->pfirstseq+i)->seqstring),"X|X")==0)
  965             {
  966                 for(j=i;j<b->nseq-1;j++)
  967                 {
  968                     strcpy(((b->pfirstseq+j)->seqstring),((b->pfirstseq
+(j+1))->seqstring));
  969                 }
  970                 b->nseq--;
  971                 i=0;
  972             }
  973 
  974         }
  975 
  976 
  977 // Juntar 3 ou mais pecas
  978 
  979     for(i=0;i<b->nseq;i++){
  980 
  981             int tamlin=strlen((b->pfirstseq+i)->seqstring)-1;
  982             paux = b->pfirst;
  983             while(paux!=NULL)
```

```
  984             {
  985
  986                 //Primeira verificação
  987                 if(((b->pfirstseq+i)->seqstring)[tamlin]==(paux->pecastr)[0]){
  988                     int igual=0;
  989                     //inverte peca apenas para verificar
  990                     int w=strlen(paux->pecastr)-1;
  991                     for(k=0;k<(strlen(paux->pecastr));k++){
  992                         inversopeca[w]=(paux->pecastr)[k];
  993                         w--;
  994                     }
  995                     //strtok da peca
  996                     char partidos[100][100];
  997                     char *palavra=NULL;
  998                     char umaseqpeca[100];
  999
 1000                     strcpy(umaseqpeca,((b->pfirstseq+i)->seqstring));
 1001                     palavra = strtok (umaseqpeca,"-");
 1002                     int s=0;
 1003                     while (palavra != NULL)
 1004                     {
 1005                         strcpy(partidos[s++],palavra);
 1006                         palavra = strtok (NULL, "-");
 1007                     }
 1008
 1009                     //verifica se o arraypartidos ou o inverdo é igual a peca
 1010                     for (x=0;x<s;x++){
 1011
if(strcmp(partidos[x],(paux->pecastr))==0||strcmp(partidos[x],inversopeca)==0){
 1012                                     igual++;
 1013                             }
 1014                     }
 1015                     if(igual==0){
 1016                                 //concatena
 1017                                 strcpy(aux,((b->pfirstseq+i)->seqstring));
 1018                                 strcat(aux,"-");
 1019                                 strcat(aux, (paux->pecastr));
 1020                                 inserir seq iniciarpeca(b,aux);
 1021
 1022                     }
 1023                 }
 1024                 //Segunda verificação
 1025                 if(((b->pfirstseq+i)->seqstring)[tamlin]==(paux->pecastr)[2]){
 1026                     int igual=0;
 1027                     //inverte peca
 1028                     int w=strlen(paux->pecastr)-1;
 1029
 1030                     for(k=0;k<(strlen(paux->pecastr));k++){
 1031                         invert[w]=(paux->pecastr)[k];
 1032                         w--;
 1033                     }
 1034                     //strtok da peca
 1035                     char partidos[100][100];
 1036                     char *palavra=NULL;
 1037                     char umaseqpeca[100];
 1038
 1039                     strcpy(umaseqpeca,((b->pfirstseq+i)->seqstring));
 1040                     palavra = strtok (umaseqpeca,"-");
 1041                     int s=0;
 1042                     while (palavra != NULL)
 1043                     {
 1044                         strcpy(partidos[s++],palavra);
 1045                         palavra = strtok (NULL, "-");
 1046                     }
 1047
 1048                     //verifica se o arraypartidos ou o inverso é igual a peça
 1049                     for (x=0;x<s;x++){
 1050
if(strcmp(partidos[x],(paux->pecastr))==0||strcmp(partidos[x],invert)==0){
 1051                                     igual++;
 1052                             }
```

```
 1053                     }
 1054                     if(igual==0){
 1055                         //concatena
 1056                         strcpy(aux, ((b->pfirstseq+i)->seqstring));
 1057                         strcat(aux,"-");
 1058                         strcat(aux, invert);
 1059                         inserir_seq_iniciarpeca(b,aux);
 1060 
 1061                     }
 1062                 }
 1063 
 1064                 //Terceira verificação
 1065                 if(((b->pfirstseq+i)->seqstring)[0]==(paux->pecastr)[0]){
 1066                     int igual=0;
 1067 
 1068                     //inverte peca
 1069                     int w=strlen(paux->pecastr)-1;
 1070                     for(k=0;k<(strlen(paux->pecastr));k++){
 1071                         invert[w]=(paux->pecastr)[k];
 1072                         w--;
 1073                     }
 1074 
 1075                     //strtok da peca
 1076                     char partidos[100][100];
 1077                     char *palavra=NULL;
 1078                     char umaseqpeca[100];
 1079                     strcpy(umaseqpeca,((b->pfirstseq+i)->seqstring));
 1080                     palavra = strtok (umaseqpeca,"-");
 1081                     int s=0;
 1082                     while (palavra != NULL)
 1083                     {
 1084                         strcpy(partidos[s++],palavra);
 1085                         palavra = strtok (NULL, "-");
 1086                     }
 1087 
 1088                     //verifica se o arraypartidos ou o inverso é igual a peça
 1089                     for (x=0;x<s;x++){
 1090 
if(strcmp(partidos[x],(paux->pecastr))==0||strcmp(partidos[x],invert)==0){
 1091                                     igual++;
 1092                             }
 1093                     }
 1094                     if(igual==0){
 1095                         //concatena
 1096                         strcpy(aux,invert);
 1097                         strcat(aux,"-");
 1098                         strcat(aux,((b->pfirstseq+i)->seqstring));
 1099                         inserir_seq_iniciarpeca(b,aux);
 1100 
 1101                     }
 1102                 }
 1103                 //Quarta verificação
 1104                 if(((b->pfirstseq+i)->seqstring)[0]==(paux->pecastr)[2]){
 1105                     int igual=0;
 1106 
 1107                     //inverte peca
 1108                     int w=strlen(paux->pecastr)-1;
 1109                     for(k=0;k<(strlen(paux->pecastr));k++){
 1110                         invert[w]=(paux->pecastr)[k];
 1111                         w--;
 1112                     }
 1113                     //strtok da peca
 1114                     char partidos[100][100];
 1115                     char *palavra=NULL;
 1116                     char umaseqpeca[100];
 1117 
 1118                     strcpy(umaseqpeca,((b->pfirstseq+i)->seqstring));
 1119                     palavra = strtok (umaseqpeca,"-");
 1120                     int s=0;
 1121                     while (palavra != NULL)
 1122                     {
```

```
 1123                         strcpy(partidos[s++],palavra);
 1124                         palavra = strtok (NULL, "-");
 1125                     }
 1126
 1127                     //verifica se o arraypartidos ou o inverso é igual a peça
 1128                     for (x=0;x<s;x++){
 1129
if(strcmp(partidos[x],(paux->pecastr))==0||strcmp(partidos[x],invert)==0){
 1130                                     igual++;
 1131                                 }
 1132                     }
 1133                     if(igual==0){
 1134                         //concatena
 1135                         strcpy(aux,(paux->pecastr));
 1136                         strcat(aux,"-");
 1137                         strcat(aux,((b->pfirstseq+i)->seqstring));
 1138                         inserir seq iniciarpeca(b,aux);
 1139
 1140                     }
 1141                 }
 1142                paux = paux->pnext;
 1143             }
 1144         }
 1145
 1146     //verificar se ha sequencias iguais e inversos tambem
 1147
 1148     for(x=0;x<b->nseq;x++){
 1149                 // Inverter peca
 1150                 int w=strlen((b->pfirstseq+x)->seqstring)-1;
 1151                 for(k=0;k<(strlen((b->pfirstseq+x)->seqstring));k++){
 1152                     invertfinal[w]=((b->pfirstseq+x)->seqstring)[k];
 1153                     w--;
 1154                 }
 1155
 1156             // Verifica se e igual e se for substitui por X|X
 1157             for(i=x+1;i<b->nseq;i++){
 1158
if((strcmp(((b->pfirstseq+i)->seqstring),invertfinal)==0)||(strcmp(((b->pfirstseq+i)->se
qstring),((b->pfirstseq+x)->seqstring))==0)){
 1159                     strcpy(((b->pfirstseq+i)->seqstring),"X|X");
 1160                 }
 1161             }
 1162     }
 1163
 1164
 1165     //remover os X|X e passa os debaixo para cima
 1166
 1167         for(i=0;i<b->nseq;i++)
 1168         {
 1169             if(strcmp(((b->pfirstseq+i)->seqstring),"X|X")==0)
 1170             {
 1171                 for(j=i;j<b->nseq-1;j++)
 1172                 {
 1173                     strcpy(((b->pfirstseq+j)->seqstring),((b->pfirstseq
+(j+1))->seqstring));
 1174                 }
 1175                 b->nseq--;
 1176                 i=0;
 1177             }
 1178
 1179         }
 1180
 1181     //imprimir seq sem repeticoes
 1182     SEQUENCIA * pauxarray = NULL;
 1183     pauxarray = b->pfirstseq;
 1184
 1185     for(z=0;z<b->nseq;z++)
 1186     {
 1187         printf("%s\n",((pauxarray+z)->seqstring));
 1188     }
 1189     return 0;
```

```
1190 }
```

void procurar_padrao (char *arrayfinalcompleto*[][150], int *y*)

Funçao procurar e substituir padrão.

Nesta função é recebido uma sequencia aleatoria do arrayfinalcompleto, o utilizador insere uma sequencia a procurar,é verificada para ver se está inserida corretamente e se nao existe peças repetidas na mão do jogador. Depois é perguntado se pretende substituir esse padrao. Caso pretende ele faz verificaçoes dessa peça para ver se existe esse padrao de forma normal ou invertida e se for possivel troca o padrão.

Parâmetros:

arrayfinalcompleto [][150]	Array do tipo char, onde é recebido as sequencias totais da mão do jogador
y,tamanho	do array total das sequencias (arrayfinalcompleto)

```
1356                                                                {
1362     //Guarda todas as posicoes num array para depois baralhar esses i's
1363     int z=0;
1364     int auxp[50];
1365     int ptemp=0;
1366     int aleatorio=0;
1367     int b=0,i=0;
1368     char strproc[100];
1369     int tamarraycompleto=y;
1370
1371     for(z=0;z<tamarraycompleto;z++){
1372         auxp[z]=z;
1373     }
1374
1375     //baralho esse numeros do auxp
1376     for(b=0;b<tamarraycompleto;b++){
1377         ptemp=auxp[b];
1378         aleatorio=rand()%(tamarraycompleto-1);
1379         auxp[b]=auxp[aleatorio];
1380         auxp[aleatorio]=ptemp;
1381     }
1382     //vai buscar uma sequencia do arrayfinalcompleto dependendo do num de sequencias
que o utilizador quer
1383         strcpy(strproc,arrayfinalcompleto[auxp[0]]);
1384         printf("A sequencia escolhida foi:\n %s \n",strproc);
1385
1386
1387     char sub[50];
1388     char str2[100][100];
1389     //declaracoes partir arrayfinalcompleto
1390     char partidosstr1[100][100];
1391     char *s1=NULL;
1392     char str1apartir[100];
1393     int u=0;
1394     //declaracoes partir subtring
1395     char partidosstr2[100][100];
1396     char *s2=NULL;
1397     char str2apartir[100];
1398     int l=0;
1399     //declaracoes partir subtring inversa
1400     char partidosstr2inv[100][100];
1401     char *s2inv=NULL;
1402     char str2invapartir[100];
1403     int inv=0;
1404     //declaracao inverter subtring
1405     char invertsub[100];
1406     //declaracoes comparar a ver se sao iguais
1407     int x=0,g=0,count=0,r=0,pos=0;;
1408     int v[100];
```

```
 1409
 1410
 1411     do{
 1412
 1413     printf("Insira uma subsequencia para procurar:\n");
 1414     scanf("%s",sub);
 1415
 1416                 //Strtok do arrayfinalcompleto
 1417                 strcpy(str1apartir,strproc);
 1418                 s1 = strtok (str1apartir,"-");
 1419
 1420                 while (s1!= NULL)
 1421                 {
 1422                     strcpy(partidosstr1[u++],s1);
 1423                     s1= strtok (NULL, "-");
 1424                 }
 1425
 1426                 //Strtok da substring escolhida
 1427                 strcpy(str2[0],sub);
 1428                 strcpy(str2apartir,str2[0]);
 1429                 s2 = strtok (str2apartir,"-");
 1430                 while (s2!= NULL)
 1431                 {
 1432                     strcpy(partidosstr2[l++],s2);
 1433                     s2= strtok (NULL, "-");
 1434                 }
 1435
 1436                 // Inverter substring
 1437                     int w=strlen(sub)-1;
 1438                     int k=0;
 1439                     for(k=0;k<(strlen(sub));k++){
 1440                         invertsub[w]=sub[k];
 1441                         w--;
 1442                     }
 1443
 1444                 //Strtok da substring invertida
 1445
 1446                 strcpy(str2invapartir,invertsub);
 1447                 s2inv = strtok (str2invapartir,"-");
 1448                 while (s2inv!= NULL)
 1449                 {
 1450                     strcpy(partidosstr2inv[inv++],s2inv);
 1451                     s2inv= strtok (NULL, "-");
 1452                 }
 1453
 1454                 //Verifica se a invertida é igual a string original (partidosstr1)
 1455                 for(x=0;x<u;x++){
 1456                     if((strcmp(partidosstr1[x],partidosstr2inv[g])==0)){
 1457                         count++;
 1458                         g++;
 1459                       if(count==g && count==l){
 1460                         //pos- é o valor da peca final menos o tam da sub -1, para
retornar a primeira posicao onde encontrou
 1461                         pos=(x-(l-1));
 1462                         v[r]=pos;
 1463                         r++;
 1464                         count=0;
 1465                         g=0;
 1466                         pos=0;
 1467                         strcpy(sub,invertsub);
 1468                       }
 1469                     }else{
 1470                         count=0;
 1471                         g=0;
 1472                         pos=0;
 1473                     }
 1474                 }
 1475                 //Verifica se a string original partida (partidosstr1) é igual a
substring inserida partida (partidosstr2)
 1476                 for(x=0;x<u;x++){
 1477                     if(strcmp(partidosstr1[x],partidosstr2[g])==0){
```

```
 1478                          count++;
 1479                          g++;
 1480                        if(count==g && count==l){
 1481                          //pos- é o valor da peca final menos o tam da sub -1, para
retornar a primeira posicao onde encontrou
 1482                          pos=(x-(l-1));
 1483                          v[r]=pos;
 1484                          r++;
 1485                          count=0;
 1486                          g=0;
 1487                          pos=0;
 1488                        }
 1489                      }else{
 1490                          count=0;
 1491                          g=0;
 1492                          pos=0;
 1493                      }
 1494                  }
 1495                  if(r==0){
 1496                      printf("Nao foi encontado a subsequencia %s, na sequencia %s!\n
",sub,strproc);
 1497                      u=0;
 1498                      l=0;
 1499
 1500                  }else{
 1501                      for(i=0;i<r;i++){
 1502                          printf(" Posicao: %d\n",v[i]);
 1503                      }
 1504                  }
 1505     }while(r==0);
 1506                  //Substituir Padrão
 1507                  int opcao=0,j=0,q=0;
 1508                  char aux[100][100];
 1509                  char addseqpeca[50];
 1510                  printf("Pretende substituir essa sequencia por outra peca/seq:\n
sim->1 , nao->0\n");
 1511                  scanf("%d",&opcao);
 1512                  if(opcao==1){
 1513                      int continua=1;
 1514                      while(continua>0){
 1515                          continua=0;
 1516                          printf("Insira a seq/peca que deseja substituir");
 1517                          scanf("%s",addseqpeca);
 1518
 1519                          //Strtok da peça adicionada (addseqpeca)
 1520                          char partidasaddseqpeca[100][100];
 1521                          char *s3=NULL;
 1522                          char addseqpecaapartir[100];
 1523                          strcpy(addseqpecaapartir,addseqpeca);
 1524                          s3 = strtok (addseqpecaapartir,"-");
 1525                          int b=0;
 1526
 1527                          while (s3!= NULL)
 1528                          {
 1529                              strcpy(partidasaddseqpeca[b++],s3);
 1530                              s3= strtok (NULL, "-");
 1531                          }
 1532
 1533                          //Verifica se essa seq adicionada é possivel inserir
 1534                          char invert[4];
 1535                          int cont=0,n=0,m=0;
 1536                          for(n=0;n<u;n++){
 1537                              for(m=0;m<b;m++){
 1538                                      // Inverter addseqpeca
 1539                                      int d=strlen(partidasaddseqpeca[m])-1;
 1540                                      int h=0;
 1541
for(h=0;h<(strlen(partidasaddseqpeca[m]));h++){
 1542                                          invert[d]=partidasaddseqpeca[m][h];
 1543                                          d--;
 1544                                      }
```

```
 1545                                    //verifica se a peça adicionada existe na string
original , ou se a peça adicionada invertida existe na string original
 1546
if((strcmp(partidasaddseqpeca[m],partidosstr1[n])==0)||(strcmp(partidosstr1[n],invert)==
0)){
 1547                                        cont++;
 1548                                    }
 1549                                }
 1550                            }
 1551                            if(cont>0){
 1552
 1553                                    printf("Essa peca ja existe!\n");
 1554                                    continua++;
 1555                                    cont=0;
 1556                            }else{
 1557                                    //verifica se o padrao encontrado está no meio,
á frente ou atrás.
 1558
if(((sub[0]==addseqpeca[0])&&(sub[strlen(sub)-1]==addseqpeca[strlen(addseqpeca)-1]))||((
addseqpeca[0]==sub[0])&&(sub[strlen(sub)-1]==strproc[strlen(strproc)-1])&&(sub[strlen(su
b)-3]==strproc[strlen(strproc)-3]))||((addseqpeca[strlen(addseqpeca)-1]==sub[strlen(sub)
-1])&&(sub[0]==strproc[0])&&(sub[2]==strproc[2]))){
 1559                                        cont=0;
 1560                                        //Adiciona no aux as primeiras pecas, as
novas e depois o resto
 1561                                            for(i=0;i<v[0];i++){
 1562                                                strcpy(aux[q],partidosstr1[i]);
 1563                                                    q++;
 1564                                            }
 1565                                            int c=0;
 1566                                            for(c=0;c<b;c++){
 1567
strcat(aux[q],partidasaddseqpeca[c]);
 1568                                                q++;
 1569                                            }
 1570                                            for(j=(v[0]+l);j<u;j++){
 1571
strcat(aux[q],partidosstr1[j]);
 1572                                                    q++;
 1573                                            }
 1574                                }else{
 1575                                    printf("Pecas inseridas nao coincidem!\n");
 1576                                    continua++;
 1577                                }
 1578                            }
 1579                            for(m=0;m<q;m++){
 1580                                printf("%s",aux[m]);
 1581                                if(m!=(q-1)){
 1582                                    printf("-");
 1583                                }
 1584                            }
 1585                            printf("\n");
 1586                }
 1587                }else if(opcao==0){
 1588
 1589                }else{
 1590                    printf("opcao errada!");
 1591                }
 1592 }
```

void rempeca (INICIARPECA * *b*, int *pecastotal*, int *npecasremover*)

Remover peças de uma ou mais mãos.

Remove peças de uma ou mais mãos, sendo que as peças removidas são substituidas por X|X, e é pedido ao utilizador para inserir tantas peças quantas removeu. Sendo tudo verificado para não remover peças que não existam e que não sejam adicionas peças repetidas.

Parâmetros:

INICIARPECA	* b , estrutura do tipo INICIARPECAr
pecastotal	Valor do tipo int, utilizado para definir as peças totais usadas pelo jogador
npecasremover	Valor do tipo int, utilizado para definir o numero de peças a ser removido pelo jogador

char pecaremover[4] - Usado para guardar a peça inserida pelo jogador, para depois remover

int j=0 - Variavél usada nas funções seguintes

int h=0 - Variavél usada nas funções seguintes

char newpeca[4] - Usado para guardar a peça a ser inserida pelo jogador, para depois adicionar

char invertnewpeca[4] - Usado para guardar a peça invertida da newpeca inserida

```
  419                                                                 {
  427     char pecaremover[4];
  429     int j=0;
  431     int h=0;
  433     char newpeca[4];
  435     char invertnewpecarem[4]={};
  436
  437     PECA * pauxbar = NULL;
  438
  439     //Caso o numero de peças a remover seja maior que as peças totais, entra neste if
  440     if(npecasremover>pecastotal){
  441         printf("Numero de pecas a remover e superior ao baralho!\n");
  442     // Caso o numero de peças a remover seja menor que as peças totais, entra neste
if
  443     }else if(npecasremover<pecastotal){
  444         // Insere peca a remover
  445         for(j=0;j<npecasremover;j++){
  446             printf("Insira a peca a remover:");
  447             scanf("%s",pecaremover);
  448
  449             PECA * pauxfind = NULL;
  450             PECA * paux = NULL;
  451             PECA * pant = NULL;
  452
  453             pauxfind = find peca baralho(b,pecaremover);
  454
  455             if(pauxfind != NULL){
  456                 paux = b->pfirst;
  457                 //percorre todas as peças
  458                 while(paux!=NULL && strcmp(paux->pecastr,pecaremover)!=0){
  459                         pant=paux;
  460                         paux=paux->pnext;
  461                 }
  462                 //remover na cabeça
  463                 if(paux==b->pfirst)
  464                 {
  465                     b->pfirst=paux->pnext;
  466                     b->nbar--;
  467                 }else if(paux->pnext==NULL)
  468                 {
  469                     //remover na cauda
  470                     pant->pnext=NULL;
  471                     b->nbar--;
  472                 }else{
  473                     //se for no meio
  474                     pant->pnext=paux->pnext;
  475                     b->nbar--;
  476                 }
  477
  478             }else{
  479                printf("Essa peca nao existe, insira outra:\n");
  480                   j--;
  481             }
  482         }
  483
```

```
  484         //Imprime mão já com as peças removidas
  485         pauxbar = b->pfirst;
  486         while(pauxbar!=NULL)
  487         {
  488             printf("%s\n",pauxbar->pecastr);
  489             pauxbar=pauxbar->pnext;
  490         }
  491
  492         //Foram removidas peças logo insiro novas para completar mão
  493         for(j=0;j<npecasremover;j++){
  494             printf("Insira uma nova peca:");
  495             scanf("%s",newpeca);
  496
  497             // Inverter peça inserida
  498             int d=strlen(newpeca)-1;
  499             for(h=0;h<(strlen(newpeca));h++){
  500                 invertnewpecarem[d]=newpeca[h];
  501                 d--;
  502             }
  503             // Verifica se a peca da mão(baralho[]) é igual a novapeca, ou á invertida
  504             PECA * pauxfindrem = NULL;
  505             PECA * pauxfindinvrem = NULL;
  506
  507             pauxfindrem = find_peca_baralho(b,newpeca);
  508
  509             pauxfindinvrem = find_peca_baralho(b,invertnewpecarem);
  510
  511             if(pauxfindrem != NULL || pauxfindinvrem != NULL){
  512                         printf("A peca ja existe!\n");
  513                         j--;
  514
  515             }else{
  516                         pauxbar = b->pfirst;
  517
  518                         //Caso contrario ele insere a nova peça na ultima posição da
mão (baralho[]) e imprime
  519
  520                         inserir peca baralho(b,newpeca);
  521
  522                         //imprime baralho todo
  523                         while(pauxbar!=NULL)
  524                         {
  525                             printf("%s\n",pauxbar->pecastr);
  526                             pauxbar=pauxbar->pnext;
  527                         }
  528             }
  529         }
  530     }
  531 }
```

void retirar_mao_jogadores (char *baralho*[][COLSTRING], int *num*)

Cria as sequencias possiveis apartir de uma sequencia inicial com 2 ou mais mãos.

A partir de uma sequencia inicial inserida pelo utilizador, ele vai guardando num array mixpecas de forma alternada as mãos pedidas pelo jogador, e vai inserido do array mixpecas de forma alternada na sequencia inicial e guardado no arrayfinalmixpecascompleto.

Parâmetros:

baralho[][COLSTRING]	Array do tipo char, onde recebe as peças baralhadas da mão do jogador

char mixpecas[3000][150] - guarda varios baralhos de forma alternada

```
 1852                                                                {
 1858      char mixpecas[3000][150];
 1859
```

```
 1860     //Ver quantos baralhos sao, e guardar num array mix as pecas dos jogadores
alternadamente
 1861     if(num==2){
 1862         int i=0;
 1863         int j=7;
 1864         int m=0;
 1865
 1866         while(i<7 || j<14){
 1867             if(i<7){
 1868                 strcpy(mixpecas[m++],baralho[i]);
 1869             }
 1870             if(j<14){
 1871                 strcpy(mixpecas[m++],baralho[j]);
 1872             }
 1873             i++;
 1874             j++;
 1875         }
 1876     }
 1877     if(num==3){
 1878         int i=0;
 1879         int j=7;
 1880         int k=14;
 1881         int m=0;
 1882
 1883         while(i<7 || j<14 || k<21 ){
 1884             if(i<7){
 1885                 strcpy(mixpecas[m++],baralho[i]);
 1886             }
 1887             if(j<14){
 1888                 strcpy(mixpecas[m++],baralho[j]);
 1889             }
 1890             if(k<21){
 1891                 strcpy(mixpecas[m++],baralho[k]);
 1892             }
 1893             i++;
 1894             j++;
 1895             k++;
 1896         }
 1897     }
 1898
 1899     if(num==4){
 1900         int i=0;
 1901         int j=7;
 1902         int k=14;
 1903         int s=21;
 1904         int m=0;
 1905
 1906         while(i<7 || j<14 || k<21 || s<28){
 1907             if(i<7){
 1908                 strcpy(mixpecas[m++],baralho[i]);
 1909             }
 1910             if(j<14){
 1911                 strcpy(mixpecas[m++],baralho[j]);
 1912             }
 1913             if(k<21){
 1914                 strcpy(mixpecas[m++],baralho[k]);
 1915             }
 1916             if(s<28){
 1917                 strcpy(mixpecas[m++],baralho[s]);
 1918             }
 1919             i++;
 1920             j++;
 1921             k++;
 1922             s++;
 1923         }
 1924     }
 1925
 1926     printf("ARRAY MIX:\n");
 1927     int v=0;
 1928     int pecasmix=num*7;
 1929     for(v=0;v<pecasmix;v++){
```

```
 1930         printf("%s\n",mixpecas[v]);
 1931     }
 1932
 1933     //Comeca com uma sequencia inicial e vai metendo peca a peca do mixpecas
 1934         char seqcomecar[3000];
 1935         char arrayfinalmixpecas[3000][150];
 1936         char arrayfinalmixpecascompleto[3000][150];
 1937         int i=0,j=0,count=0,x=0,y=0,p=1,z=0,contador=0;
 1938         char inversazero[150];
 1939         char inversoum[150];
 1940         char inversodois[150];
 1941         char inversotres[150];
 1942         char inversoquatro[150];
 1943         char invertfinal[150];
 1944
 1945         printf("Insira uma sequencia a comecar:\n");
 1946         scanf("%s",seqcomecar);
 1947
 1948         strcpy(arrayfinalmixpecas[0],seqcomecar);
 1949
 1950                 //Strtok da sequencia inicial
 1951                 char partidosseqcomecar[100][100];
 1952                 char *s=NULL;
 1953                 char seqapartir[100];
 1954
 1955                 strcpy(seqapartir,seqcomecar);
 1956                 s = strtok (seqapartir,"-");
 1957                 int k=0;
 1958
 1959                 while (s!= NULL)
 1960                 {
 1961                     strcpy(partidosseqcomecar[k++],s);
 1962                     s= strtok (NULL, "-");
 1963                 }
 1964
 1965                 //Verificar se as pecas da sequencia inicial sao iguais as pecas do
baralho mix
 1966                 for(x=0;x<pecasmix;x++){
 1967                     for(y=0;y<k;y++){
 1968                             // Inverter pecas baralho
 1969                             int d=strlen(mixpecas[x])-1;
 1970                             int h=0;
 1971                             for(h=0;h<(strlen(mixpecas[x]));h++){
 1972                                     inversazero[d]=mixpecas[x][h];
 1973                                     d--;
 1974                             }
 1975                             //Verifica se as pecas da seq inicial coincidem
 1976                             int tam=strlen(seqcomecar)-1;
 1977                             int i=0;
 1978                             for(i=0;i<tam;i++){
 1979                                 if(seqcomecar[i]=='-'){
 1980                                     if(seqcomecar[i-1]!=seqcomecar[i+1]){
 1981                                         contador++;
 1982                                     }
 1983                                 }
 1984                             }
 1985
if((strcmp(mixpecas[x],partidosseqcomecar[y])==0)||(strcmp(inversazero,partidosseqcomeca
r[y])==0)){
 1986                                 count++;
 1987                            }
 1988                     }
 1989                 }
 1990                 if(count>0){
 1991                     printf("As pecas da sequencia que inseriu, ja existem no
baralho!\n");
 1992                     count=0;
 1993                 }else if(contador>0){
 1994                     printf("A sequencia que inseriu esta errada!\n ");
 1995                     contador=0;
 1996                 }else{
```

```
 1997                     //Juntar pecas
 1998                     for(i=0;i<p;i++){
 1999                         int tamlin=strlen(arrayfinalmixpecas[i])-1;
 2000                         for(j=0;j<pecasmix;j++){
 2001                             //Primeira verificação
 2002                             if(arrayfinalmixpecas[i][tamlin]==mixpecas[j][0]){
 2003                                 int igual=0;
 2004                                 //inverte peca apenas para verificar
 2005                                 int w=strlen(mixpecas[j])-1;
 2006                                 for(k=0;k<(strlen(mixpecas[j]));k++){
 2007                                     inversoum[w]=mixpecas[j][k];
 2008                                     w--;
 2009                                 }
 2010                                 //strtok da peca arrayfinalmixpecas
 2011                                 char partidos[100][100];
 2012                                 char *palavra=NULL;
 2013                                 char umaseqpeca[100];
 2014 
 2015                                 strcpy(umaseqpeca,arrayfinalmixpecas[i]);
 2016                                 palavra = strtok (umaseqpeca,"-");
 2017                                 int s=0;
 2018                                 while (palavra != NULL)
 2019                                 {
 2020                                     strcpy(partidos[s++],palavra);
 2021                                     palavra = strtok (NULL, "-");
 2022                                 }
 2023 
 2024                                 //verifica se o arrayfinalmixpecas ou o inverso e
igual a peca
 2025                                 for (x=0;x<s;x++){
 2026 
if(strcmp(partidos[x],mixpecas[j])==0||strcmp(partidos[x],inversoum)==0){
 2027                                                         igual++;
 2028                                             }
 2029                                 }
 2030                                 if(igual==0){
 2031                                     //concatena para arrayfinalmixpecas
 2032                                             strcpy(arrayfinalmixpecas[p],
arrayfinalmixpecas[i]);
 2033                                             strcat(arrayfinalmixpecas[p],"-");
 2034                                             strcat(arrayfinalmixpecas[p],
mixpecas[j]);
 2035                                             p++;
 2036                                 }
 2037                             }
 2038                             //Segunda verificação
 2039                             if(arrayfinalmixpecas[i][tamlin]==mixpecas[j][2]){
 2040                                 int igual=0;
 2041                                 //inverte peca
 2042                                 int w=strlen(mixpecas[j])-1;
 2043                                 for(k=0;k<(strlen(mixpecas[j]));k++){
 2044                                     inversodois[w]=mixpecas[j][k];
 2045                                     w--;
 2046                                 }
 2047                                 //strtok da peca arrayfinalmixpecas
 2048                                 char partidos[100][100];
 2049                                 char *palavra=NULL;
 2050                                 char umaseqpeca[100];
 2051 
 2052                                 strcpy(umaseqpeca,arrayfinalmixpecas[i]);
 2053                                 palavra = strtok (umaseqpeca,"-");
 2054                                 int s=0;
 2055                                 while (palavra != NULL)
 2056                                 {
 2057                                     strcpy(partidos[s++],palavra);
 2058                                     palavra = strtok (NULL, "-");
 2059                                 }
 2060                                 //verifica se o partidos ou o inverso e igual a peca
 2061                                 for (x=0;x<s;x++){
 2062 
if(strcmp(partidos[x],mixpecas[j])==0||strcmp(partidos[x],inversodois)==0){
```

```
 2063                                                    igual++;
 2064                                                }
 2065                                    }
 2066                                    if(igual==0){
 2067                                        //concatena
 2068                                        strcpy(arrayfinalmixpecas[p],
arrayfinalmixpecas[i]);
 2069                                        strcat(arrayfinalmixpecas[p],"-");
 2070                                        strcat(arrayfinalmixpecas[p], inversodois);
 2071                                        p++;
 2072                                    }
 2073                                }
 2074                                //Terceira verificação
 2075                                if(arrayfinalmixpecas[i][0]==mixpecas[j][0]){
 2076                                    int igual=0;
 2077                                    //inverte peca
 2078                                    int w=strlen(mixpecas[j])-1;
 2079                                    for(k=0;k<(strlen(mixpecas[j]));k++){
 2080                                        inversotres[w]=mixpecas[j][k];
 2081                                        w--;
 2082                                    }
 2083                                    //strtok da peca arrayfinal
 2084                                    char partidos[100][100];
 2085                                    char *palavra=NULL;
 2086                                    char umaseqpeca[100];
 2087
 2088                                    strcpy(umaseqpeca,arrayfinalmixpecas[i]);
 2089                                    palavra = strtok (umaseqpeca,"-");
 2090                                    int s=0;
 2091                                    while (palavra != NULL)
 2092                                    {
 2093                                        strcpy(partidos[s++],palavra);
 2094                                        palavra = strtok (NULL, "-");
 2095                                    }
 2096                                    //verifica se o partidos ou o inverso é igual a peca
 2097                                    for (x=0;x<s;x++){
 2098
if(strcmp(partidos[x],mixpecas[j])==0||strcmp(partidos[x],inversotres)==0){
 2099                                                    igual++;
 2100                                                }
 2101                                    }
 2102                                    if(igual==0){
 2103                                        //concatena
 2104                                        strcpy(arrayfinalmixpecas[p], inversotres);
 2105                                        strcat(arrayfinalmixpecas[p],"-");
 2106                                        strcat(arrayfinalmixpecas[p],
arrayfinalmixpecas[i]);
 2107                                        p++;
 2108                                    }
 2109                                }
 2110                                //Quarta verificação
 2111                                if(arrayfinalmixpecas[i][0]==mixpecas[j][2]){
 2112                                    int igual=0;
 2113
 2114                                    //inverte peca
 2115                                    int w=strlen(mixpecas[j])-1;
 2116                                    for(k=0;k<(strlen(mixpecas[j]));k++){
 2117                                        inversoquatro[w]=mixpecas[j][k];
 2118                                        w--;
 2119                                    }
 2120                                    //strtok da peca arrayfinal
 2121                                    char partidos[100][100];
 2122                                    char *palavra=NULL;
 2123                                    char umaseqpeca[100];
 2124
 2125                                    strcpy(umaseqpeca,arrayfinalmixpecas[i]);
 2126                                    palavra = strtok (umaseqpeca,"-");
 2127                                    int s=0;
 2128                                    while (palavra != NULL)
 2129                                    {
 2130                                        strcpy(partidos[s++],palavra);
```

```
 2131                                    palavra = strtok (NULL, "-");
 2132                                }
 2133                                //verifica se o partidos ou o inverso é igual a peca
 2134                                for (x=0;x<s;x++){
 2135
if(strcmp(partidos[x],mixpecas[j])==0||strcmp(partidos[x],inversoquatro)==0){
 2136                                                        igual++;
 2137                                            }
 2138                                }
 2139                                if(igual==0){
 2140                                    //concatena
 2141                                    strcpy(arrayfinalmixpecas[p], mixpecas[j]);
 2142                                    strcat(arrayfinalmixpecas[p],"-");
 2143                                    strcat(arrayfinalmixpecas[p],
arrayfinalmixpecas[i]);
 2144                                    p++;
 2145                                }
 2146                            }
 2147                        }
 2148                    }
 2149                //verificar se ha seq iguais e o inversos tambem
 2150                    for(x=0;x<p;x++){
 2151                                // Inverter peca
 2152                                int w=strlen(arrayfinalmixpecas[x])-1;
 2153                                for(k=0;k<(strlen(arrayfinalmixpecas[x]));k++){
 2154                                    invertfinal[w]=arrayfinalmixpecas[x][k];
 2155                                    w--;
 2156                                }
 2157
 2158                    // Verifica se e igual e se for substitui por X|X
 2159                            for(i=x+1;i<p;i++){
 2160
if((strcmp(arrayfinalmixpecas[i],invertfinal)==0)||(strcmp(arrayfinalmixpecas[i],arrayfi
nalmixpecas[x])==0)){
 2161                                    strcpy(arrayfinalmixpecas[i],"X|X");
 2162                                }
 2163                            }
 2164                    }
 2165                    // Copia do arrayfinalmixpecas que contem x|x e guarda num array
final(arrayfinalmixpecascompleto) as sequencias possiveis
 2166                     for(z=0;z<p;z++){
 2167                            if(strcmp(arrayfinalmixpecas[z],"X|X")==0){
 2168                            }else{
 2169
strcpy(arrayfinalmixpecascompleto[y],arrayfinalmixpecas[z]);
 2170                                y++;
 2171                            }
 2172                    }
 2173                }
 2174                for(i=0;i<y;i++){
 2175                    printf("%s\n",arrayfinalmixpecascompleto[i]);
 2176                }
 2177 }
```

void save_bin (INICIARPECA *b*, char *fnome*[])

Guardar num ficheiro binario.

Guarda as peças utlizadas na mão e guarda as sequencias

Parâmetros:

INICIARPECA	b , estrutura do tipo INICIARPECA
char	fnome[], recebe o nome do ficheiro

```
 1241                                        {
 1247     //nplayers tamnome nome\0 cardid carsuit cardpoints
 1248     //2        7  daniel\0      A      E         2
```

```
1249     PECA * paux = NULL;
1250     FILE * fp = NULL;
1251     int size = 0, i = 0,sizebar = 0;
1252 
1253     if((fp=fopen(fnome,"wb"))!=NULL)
1254     {
1255         //numero de pecas
1256         fwrite(&b.nbar,sizeof(int),1,fp);
1257 
1258         //gravar baralho
1259         paux = b.pfirst;
1260 
1261         while(paux!=NULL)
1262         {
1263             sizebar=strlen((b.pfirst)->pecastr)+1;
1264             fwrite(&sizebar,sizeof(int),1,fp);
1265             fwrite(paux->pecastr,sizeof(char),sizebar,fp);
1266             paux = paux->pnext;
1267         }
1268          //numero de sequencias
1269         fwrite(&b.nseq,sizeof(int),1,fp);
1270 
1271         //gravar sequencias
1272         for(i=0;i<b.nseq;i++){
1273             size=strlen((b.pfirstseq+i)->seqstring)+1;
1274             fwrite(&size,sizeof(int),1,fp);
1275             fwrite((b.pfirstseq+i)->seqstring,sizeof(char),size,fp);
1276         }
1277         fclose(fp);
1278     }
1279 }
```

void save_txt_jogo (INICIARPECA *b*, char *fname*[])

Guardar num ficheiro de texto.

Guarda as peças utlizadas na mão, guarda as peças que sobraram e guarda as sequencias

Parâmetros:

INICIARPECA	b , estrutura do tipo INICIARPECA
char	fname[], recebe o nome do ficheiro

```
1194 {
1201     FILE *fp = NULL;
1202     PECA * pauxbar = NULL;
1203     SOBROU * pauxsob = NULL;
1204     int i=0;
1205 
1206     if ((fp = fopen(fname, "w")) == NULL)
1207     {
1208 
1209         printf("save txt jogo(): Erro abrir ficheiro %s\n",fname);
1210         return;
1211 
1212     }
1213 
1214     pauxbar= b.pfirst;
1215 
1216     fprintf(fp,"Mão do jogador:\n");
1217     while(pauxbar!=NULL)
1218     {
1219         fprintf(fp,"%s\n",pauxbar->pecastr);
1220         pauxbar = pauxbar->pnext;
1221     }
1222 
1223     pauxsob = b.psobrou;
1224     fprintf(fp,"\nPeças que sobram:\n");
```

```
1225     while(pauxsob!=NULL)
1226     {
1227         fprintf(fp,"%s\n",pauxsob->sobroustr);
1228         pauxsob = pauxsob->proximo;
1229     }
1230
1231     fprintf(fp,"\nSequencias possiveis:\n");
1232     for(i=0;i<b.nseq;i++)
1233     {
1234         fprintf(fp,"%s\n",(b.pfirstseq+i)->seqstring);
1235     }
1236     fclose(fp);
1237
1238 }
```

void seq_inicial (char *baralho*[][COLSTRING])

Faz sequencias a partir de uma sequencia inicial.

O utilizador escolhe uma sequencia inicial, e é realizado uma verificação para ver se é possivel encaixar essa sequencia com as peças da mão/baralho

Parâmetros:

baralho[][COLSTRING]	Array do tipo char, onde recebe as peças baralhadas da mão do jogador

```
1594                                        {
1599         char seqcomecar[3000];
1600         char arrayseqinicial[3000][150];
1601         char arrayseqinicialcompleto[3000][150];
1602         int i=0,j=0,count=0,x=0,y=0,p=1,z=0,contador=0;
1603         char inversazero[150];
1604         char inversoum[150];
1605         char inversodois[150];
1606         char inversotres[150];
1607         char inversoquatro[150];
1608         char invertfinal[150];
1609
1610         printf("Insira uma sequencia a comecar:\n");
1611         scanf("%s",seqcomecar);
1612
1613         strcpy(arrayseqinicial[0],seqcomecar);
1614
1615                 //Strtok da sequencia inicial
1616                 char partidosseqcomecar[100][100];
1617                 char *s=NULL;
1618                 char seqapartir[100];
1619                 strcpy(seqapartir,seqcomecar);
1620                 s = strtok (seqapartir,"-");
1621                 int k=0;
1622                 while (s!= NULL)
1623                 {
1624                     strcpy(partidosseqcomecar[k++],s);
1625                     s= strtok (NULL, "-");
1626                 }
1627                 //Verificar se as pecas da sequencia inicial sao iguais as pecas do
baralho
1628                 for(x=0;x<7;x++){
1629                     for(y=0;y<k;y++){
1630                             // Inverter pecas baralho
1631                             int d=strlen(baralho[x])-1;
1632                             int h=0;
1633                             for(h=0;h<(strlen(baralho[x]));h++){
1634                                     inversazero[d]=baralho[x][h];
1635                                     d--;
1636                             }
1637                             //Verifica se as pecas da seq inicial coincidem
```

```
 1638                                int tam=strlen(seqcomecar)-1;
 1639                                int i=0;
 1640                                for(i=0;i<tam;i++){
 1641                                    if(seqcomecar[i]=='-'){
 1642                                        if(seqcomecar[i-1]!=seqcomecar[i+1]){
 1643                                            contador++;
 1644                                        }
 1645                                    }
 1646                                }
 1647
 1648
if((strcmp(baralho[x],partidosseqcomecar[y])==0)||(strcmp(inversazero,partidosseqcomecar
[y])==0)){
 1649                                count++;
 1650                            }
 1651                        }
 1652                }
 1653                if(count>0){
 1654                    printf("As pecas da sequencia que inseriu, ja existem no
baralho!\n");
 1655                    count=0;
 1656                }else if(contador>0){
 1657                    printf("A sequencia que inseriu esta errada!\n ");
 1658                    contador=0;
 1659                }else{
 1660
 1661                    // Juntar pecas
 1662                    for(i=0;i<p;i++){
 1663                        int tamlin=strlen(arrayseqinicial[i])-1;
 1664                        for(j=0;j<7;j++){
 1665
 1666                            //Primeira verificação
 1667                            if(arrayseqinicial[i][tamlin]==baralho[j][0]){
 1668                                int igual=0;
 1669                                //inverte peca apenas para verificar
 1670                                int w=strlen(baralho[j])-1;
 1671                                for(k=0;k<(strlen(baralho[j]));k++){
 1672                                    inversoum[w]=baralho[j][k];
 1673                                    w--;
 1674                                }
 1675                                //strtok da peca arrayfinal
 1676                                char partidos[100][100];
 1677                                char *palavra=NULL;
 1678                                char umaseqpeca[100];
 1679
 1680                                strcpy(umaseqpeca,arrayseqinicial[i]);
 1681                                palavra = strtok (umaseqpeca,"-");
 1682                                int s=0;
 1683                                while (palavra != NULL)
 1684                                {
 1685                                    strcpy(partidos[s++],palavra);
 1686                                    palavra = strtok (NULL, "-");
 1687                                }
 1688
 1689                                //verifica se o partidos ou o inverso é igual a peça
 1690                                for (x=0;x<s;x++){
 1691
if(strcmp(partidos[x],baralho[j])==0||strcmp(partidos[x],inversoum)==0){
 1692                                                    igual++;
 1693                                            }
 1694                                }
 1695                                if(igual==0){
 1696                                            //concatena para arrayseqinicial
 1697                                            strcpy(arrayseqinicial[p],
arrayseqinicial[i]);
 1698                                            strcat(arrayseqinicial[p],"-");
 1699                                            strcat(arrayseqinicial[p],
baralho[j]);
 1700                                            p++;
 1701                                }
 1702
```

```
 1703                                }
 1704                                //Segunda verificação
 1705                                if(arrayseqinicial[i][tamlin]==baralho[j][2]){
 1706                                    int igual=0;
 1707                                    //inverte peca
 1708                                    int w=strlen(baralho[j])-1;
 1709                                    for(k=0;k<(strlen(baralho[j]));k++){
 1710                                        inversodois[w]=baralho[j][k];
 1711                                        w--;
 1712                                    }
 1713                                    //strtok da peca arrayseqinicial
 1714                                    char partidos[100][100];
 1715                                    char *palavra=NULL;
 1716                                    char umaseqpeca[100];
 1717
 1718                                    strcpy(umaseqpeca,arrayseqinicial[i]);
 1719                                    palavra = strtok (umaseqpeca,"-");
 1720                                    int s=0;
 1721                                    while (palavra != NULL)
 1722                                    {
 1723                                        strcpy(partidos[s++],palavra);
 1724                                        palavra = strtok (NULL, "-");
 1725                                    }
 1726
 1727                                    //verifica se o partidos ou o inverso é igual a peça
 1728                                    for (x=0;x<s;x++){
 1729
if(strcmp(partidos[x],baralho[j])==0||strcmp(partidos[x],inversodois)==0){
 1730                                                    igual++;
 1731                                            }
 1732                                    }
 1733                                    if(igual==0){
 1734                                        //concatena
 1735                                        strcpy(arrayseqinicial[p],
arrayseqinicial[i]);
 1736                                        strcat(arrayseqinicial[p],"-");
 1737                                        strcat(arrayseqinicial[p], inversodois);
 1738                                        p++;
 1739                                    }
 1740                                }
 1741                                //Terceira verificação
 1742                                if(arrayseqinicial[i][0]==baralho[j][0]){
 1743                                    int igual=0;
 1744
 1745                                    //inverte peca
 1746                                    int w=strlen(baralho[j])-1;
 1747                                    for(k=0;k<(strlen(baralho[j]));k++){
 1748                                        inversotres[w]=baralho[j][k];
 1749                                        w--;
 1750                                    }
 1751                                    //strtok da peca arrayfinal
 1752                                    char partidos[100][100];
 1753                                    char *palavra=NULL;
 1754                                    char umaseqpeca[100];
 1755
 1756                                    strcpy(umaseqpeca,arrayseqinicial[i]);
 1757                                    palavra = strtok (umaseqpeca,"-");
 1758                                    int s=0;
 1759                                    while (palavra != NULL)
 1760                                    {
 1761                                        strcpy(partidos[s++],palavra);
 1762                                        palavra = strtok (NULL, "-");
 1763                                    }
 1764
 1765                                    //verifica se o partidos ou o inverso é igual a peca
 1766                                    for (x=0;x<s;x++){
 1767
if(strcmp(partidos[x],baralho[j])==0||strcmp(partidos[x],inversotres)==0){
 1768                                                    igual++;
 1769                                            }
 1770                                    }
```

```
 1771                                    if(igual==0){
 1772                                        //concatena
 1773                                        strcpy(arrayseqinicial[p], inversotres);
 1774                                        strcat(arrayseqinicial[p],"-");
 1775                                        strcat(arrayseqinicial[p],
arrayseqinicial[i]);
 1776                                        p++;
 1777                                    }
 1778                                }
 1779                                //Quarta verificação
 1780                                if(arrayseqinicial[i][0]==baralho[j][2]){
 1781                                    int igual=0;
 1782
 1783                                    //inverte peca
 1784                                    int w=strlen(baralho[j])-1;
 1785                                    for(k=0;k<(strlen(baralho[j]));k++){
 1786                                        inversoquatro[w]=baralho[j][k];
 1787                                        w--;
 1788                                    }
 1789
 1790                                    //strtok da peca arrayfinal
 1791                                    char partidos[100][100];
 1792                                    char *palavra=NULL;
 1793                                    char umaseqpeca[100];
 1794
 1795                                    strcpy(umaseqpeca,arrayseqinicial[i]);
 1796                                    palavra = strtok (umaseqpeca,"-");
 1797                                    int s=0;
 1798                                    while (palavra != NULL)
 1799                                    {
 1800                                        strcpy(partidos[s++],palavra);
 1801                                        palavra = strtok (NULL, "-");
 1802                                    }
 1803
 1804                                    //verifica se o arraypartidos ou o invero e igual a
peca
 1805                                    for (x=0;x<s;x++){
 1806
if(strcmp(partidos[x],baralho[j])==0||strcmp(partidos[x],inversoquatro)==0){
 1807                                                            igual++;
 1808                                                }
 1809                                    }
 1810
 1811                                    if(igual==0){
 1812                                        //concatena
 1813                                        strcpy(arrayseqinicial[p], baralho[j]);
 1814                                        strcat(arrayseqinicial[p],"-");
 1815                                        strcat(arrayseqinicial[p],
arrayseqinicial[i]);
 1816                                        p++;
 1817                                    }
 1818                                }
 1819                            }
 1820                        }
 1821                    //verificar se ha sequencias iguais e o inversos tambem
 1822                        for(x=0;x<p;x++){
 1823                                    // Inverter peca
 1824                                    int w=strlen(arrayseqinicial[x])-1;
 1825
 1826                                    for(k=0;k<(strlen(arrayseqinicial[x]));k++){
 1827                                        invertfinal[w]=arrayseqinicial[x][k];
 1828                                        w--;
 1829                                    }
 1830                                // Verifica se e igual e se for substitui por X|X
 1831                                        for(i=x+1;i<p;i++){
 1832
if((strcmp(arrayseqinicial[i],invertfinal)==0)||(strcmp(arrayseqinicial[i],arrayseqinici
al[x])==0)){
 1833                                        strcpy(arrayseqinicial[i],"X|X");
 1834                                    }
 1835                                }
```

```
 1836                     }
 1837                     // Copia do aux que contem x|x e guarda num array
final(arrayseqinicialcompleto) as sequencias das pecas
 1838                      for(z=0;z<p;z++){
 1839                             if(strcmp(arrayseqinicial[z],"X|X")==0){
 1840                             }else{
 1841
strcpy(arrayseqinicialcompleto[y],arrayseqinicial[z]);
 1842                                 y++;
 1843                             }
 1844                     }
 1845                 }
 1846                 for(i=0;i<y;i++){
 1847                     printf("%s\n",arrayseqinicialcompleto[i]);
 1848                 }
 1849 }
```

Referência ao ficheiro projetolp1primparte.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <time.h>
```

Estruturas de Dados

- struct **sequencia**
- struct **pecaint**
- struct **sobrou**
- struct **peca**
- struct **iniciarpeca**

Macros

- #define **COLSTRING** 4
- #define **COL** 2
- #define **LINSTRING** 28

Definições de tipos

- typedef struct **sequencia SEQUENCIA**
- typedef struct **pecaint PECAINT**
- typedef struct **sobrou SOBROU**
- typedef struct **peca PECA**
- typedef struct **iniciarpeca INICIARPECA**

Funções

- int **main_projetolp1primparte** (int argc, char *argv[])
  Função main_projectolp1primparte.
- void **criarpecas** (char pecas[][**COLSTRING**])
  Criar peças string.
- void **imprimirpecas** (char pecas[][**COLSTRING**])
  Imprime as pecas em string.
- void **baralhos** (**INICIARPECA** *b, char pecas[][**COLSTRING**], char baralho[][**COLSTRING**], char **sobrou**[][**COLSTRING**], int)
  Criar baralhos/mãos aleatórios.
- void **rempeca** (**INICIARPECA** *b, int pecastotal, int npecasremover)
  Remover peças de uma ou mais mãos.
- void **addpeca** (**INICIARPECA** *b, int pecastotal, int npecas)
  Adicionar peças numa mão/mãos.
- void **imprimirpecasint** (char pecas[][**COLSTRING**], int pecasint[][**COL**])
  Imprimir pecas em inteiros.
- int **ordenarseq** (**INICIARPECA** *b, int num)
  Sequencias e ordenar sequencia por ordem decrescente.
- void **procurar_padrao** (char arrayfinalcompleto[][150], int)
  Funçao procurar e substituir padrão.
- void **seq_inicial** (char baralho[][**COLSTRING**])

Faz sequencias a partir de uma sequencia inicial.

- void **retirar_mao_jogadores** (char baralho[][**COLSTRING**], int)
  Cria as sequencias possiveis apartir de uma sequencia inicial com 2 ou mais mãos.
- char * **create_dyn_string** (char str[])
  Criar um array dinamico para strings.
- void **inserir_peca_baralho** (**INICIARPECA** *b, char pecas[**COLSTRING**])
  Inserir pecas no baralho (lista ligada)
- void **inserir_sobrou_baralho** (**INICIARPECA** *b, char pecas[**COLSTRING**])
  Inserir pecas que sobraram do baralho (lista ligada)
- **PECA** * **find_peca_baralho** (**INICIARPECA** *b, char novapeca[])
  Procurar peça no baralho.
- void **inserir_pecaint_baralho** (**INICIARPECA** *b, int dir, int esq)
  Inserir pecas do tipo int no baralho (lista ligada)
- void **convert_mao_string_to_int** (**INICIARPECA** *b, int pecastotal)
  Converter peças string para int.
- void **convert_mao_int_to_string** (**INICIARPECA** *b, int pecastotal)
  Converter peças int para string.
- void **create_array_pecas** (**INICIARPECA** *b, unsigned int n)
  Criar array dinamico para pecas.
- void **inserir_seq_iniciarpeca** (**INICIARPECA** *b, char seq[])
  Inserir sequencias no array dinamico.
- void **save_txt_jogo** (**INICIARPECA** b, char fname[])
  Guardar num ficheiro de texto.
- void **save_bin** (**INICIARPECA** b, char fnome[])
  Guardar num ficheiro binario.
- void **load_txt_jogo** (**INICIARPECA** *b, char fname[])
  Ler de um ficheiro de texto.
- void **load_bin** (**INICIARPECA** *b, char fname[])
  Ler de um ficheiro binario.

Documentação das macros

#define COL 2

#define COLSTRING 4

#define LINSTRING 28

Documentação dos tipos

typedef struct iniciarpeca INICIARPECA

typedef struct peca PECA

typedef struct pecaint PECAINT

typedef struct sequencia SEQUENCIA

typedef struct sobrou SOBROU

Documentação das funções

void addpeca (INICIARPECA * *b*, int *pecastotal*, int *npecas*)

Adicionar peças numa mão/mãos.

Adicionar peças na mão, sendo que as peças inseridas sao adicionadas á mão inicial e completada com peças aleatorias para preencher as mãos. Ex: o jogador tem uma mão de 7 peças, quer adicionar mais duas, ou seja adiciona essas duas, e sao adicionadas automaticamente 5 peças aleatorias, para completar duas mãos (14 peças)

Parâmetros:

INICIARPECA	* b , estrutura do tipo INICIARPECA
pecastotal	Valor do tipo int, utilizado para definir as peças totais usadas pelo jogador
npecas	Valor do tipo int, utilizado para definir o numero de peças a ser adicionado pelo jogador

int aleatorio=0 - Variavél onde é guardada o numero de peças total mais as peças adicionadas pelo jogador. Posteriormente, este valor é subtraido conforme as mãos usadas

int j=0 - Variavél usada nas funções seguintes

int k=0 - Variavél usada nas funções seguintes

char novapeca[4] - Usado para guardar a peça a ser inserida pelo jogador, para depois adicionar

char invertnewpeca[4] - Usado para guardar a peça invertida da novapeca inserida

```
  666                                                         {
  674     int aleatorio=0;
  676     int j=0;
  678     int k=0;
  680     char novapeca[4];
  682     char invertnewpeca[4]={};
  683
  684     SOBROU * pauxsob = NULL;
  685     PECA * pauxbar = NULL;
  686
  687     aleatorio=pecastotal+npecas;
  688     // Verificar se a soma da numero de pecas total com as peças que quer adicionar
está no primeiro baralho, entre 0 e 6. Se sim, subtrai o valor aleatorio por 6 e guarda em
aleatório o numero de peças que vao ser lançadas aleatóriamente. E assim sucessivamente para
os outros.
  689     if(aleatorio>0 && aleatorio<6){
  690         aleatorio=6-aleatorio;
  691     }else if(aleatorio>6 && aleatorio<13){
```

```
  692             aleatorio=13-aleatorio;
  693     }else if(aleatorio>13 && aleatorio<20){
  694         aleatorio=20-aleatorio;
  695     }else if(aleatorio>20 && aleatorio<27){
  696         aleatorio=27-aleatorio;
  697     }
  698     // Por exemplo, caso o utilizador adiciona 5 pecas, entao 5+7 (7 do baralho original)
=12 logo 2 pecas sao aleatorias e as outras 5 manual
  699
  700     pauxsob = b->psobrou;
  701     //inserir as pecas do sobrou para o baralho
  702     for(j=pecastotal;j<=(pecastotal+aleatorio);j++){
  703         inserir_peca_baralho(b,pauxsob->sobroustr);
  704         pauxsob=pauxsob->proximo;
  705     }
  706     // Imprime a mão/baralho mais as peças aleatorias
  707     pauxbar = b->pfirst;
  708     while(pauxbar!=NULL)
  709     {
  710         printf("%s\n",pauxbar->pecastr);
  711         pauxbar=pauxbar->pnext;
  712     }
  713
  714
  715     // Inserir peca (novapeca)
  716     int h=0;
  717
  718     for(k=0;k<npecas;k++){
  719             printf("Insira a peca");
  720             scanf("%s",novapeca);
  721
  722             // Inverter novapeca
  723             int d=strlen(novapeca)-1;
  724             for(h=0;h<(strlen(novapeca));h++){
  725                 invertnewpeca[d]=novapeca[h];
  726                 d--;
  727             }
  728             // Verifica se a peca da mão(baralho[]) é igual a novapeca, ou á invertida
  729             PECA * pauxfind = NULL;
  730             PECA * pauxfindinv = NULL;
  731
  732             pauxfind = find_peca_baralho(b,novapeca);
  733
  734             pauxfindinv = find peca baralho(b,invertnewpeca);
  735
  736             if(pauxfind != NULL || pauxfindinv != NULL){
  737                         printf("A peca ja existe!\n");
  738                         k--;
  739
  740             }else{
  741                         pauxbar = b->pfirst;
  742
  743                         //Caso contrario ele insere a nova peça na ultima posição da
mão (baralho[]) e imprime
  744
  745                         inserir_peca_baralho(b,novapeca);
  746
  747                         //imprime baralho todo
  748                         while(pauxbar!=NULL)
  749                         {
  750                             printf("%s\n",pauxbar->pecastr);
  751                             pauxbar=pauxbar->pnext;
  752                         }
  753             }
  754     }
  755 }
```

void baralhos (INICIARPECA * *b*, char *pecas*[][COLSTRING], char *baralho*[][COLSTRING], char *sobrou*[][COLSTRING], int)

Criar baralhos/mãos aleatórios.

Cria um array de peças baralhadas (baralho), sendo esta a mão do jogador. E guarda noutro array (sobrou) as peças não utilizadas na mão.

Parâmetros:

INICIARPECA	* b , estrutura do tipo INICIARPECA
pecas[][COLSTRING]	Array do tipo char onde recebe as peças totais do jogo.
baralho[][COLSTRING]	Array do tipo char, onde irão ser guardadas as peças baralhadas da mão do jogador
sobrou[][COLSTRING]	Array do tipo char, onde irão ser guardadas as peças que sobraram, ou seja, que não foram usadas na mão do jogador
num	Valor do tipo int, utilizado para definir o numero de mãos a ser usado pelo jogador

srand((unsigned)time(NULL)) - Função srand(),responsável por alimentar o rand() e gerar números aleatórios

int pecastotal=0 - Número de peças total que o jogador tem na mão

int i=0 - Variavél usada nos ciclos

int aleatorio - Variavél que guarda número gerado aleatório

int array[28] - Array com as posicoes de 0 a 27 das pecas das mãos

int temp - Variavél temporária para guardas conteudo de array[i]

```
  251
{
  261     srand((unsigned)time(NULL));
  263     int pecastotal=0;
  265     int i=0;
  267     int aleatorio;
  269     int array[28];
  271     int temp;
  272
  273     PECA * pauxbar = NULL;
  274     SOBROU * pauxsob = NULL;
  275
  276     // Número de peças total que o jogador tem na mão, numero de mãos (1,2,3 ou 4)
multiplicado pelas pecas possiveis de cada mão (7)
  277     pecastotal = num * 7;
  278     //Caso o numero de mãos escollhidas seja igual ou inferior a 4, entra neste if
  279     if(num<=4){
  280             // Cria array posicoes das pecas totais
  281             for(i=0;i<28;i++){
  282                 array[i] = i;
  283             }
  284             // Baralha essas 28 pecas e guarda em array, de forma aleatória
  285             for(i=0;i<28;i++){
  286                 temp=array[i];
  287                 aleatorio = rand() % 28;
  288                 array[i]=array[aleatorio];
  289                 array[aleatorio]=temp;
  290             }
  291             // Guarda num array baralho[] as peças aleatórias até ao numero de pecas
total definida pelo jogador e imprime
  292             printf("MAO:\n");
  293             for(i=0;i<pecastotal;i++){
  294                 //strcpy(baralho[i],pecas[array[i]]);
  295                 //printf("%s\n",baralho[i]);
  296                 inserir_peca_baralho(b,pecas[array[i]]);
  297             }
  298
  299                 //imprime mao do baralho
```

```
  300                 pauxbar=b->pfirst;
  301
  302                 while(pauxbar!=NULL)
  303                 {
  304                     printf("%s\n",pauxbar->pecastr);
  305                      pauxbar=pauxbar->pnext;
  306                 }
  307             // Guarda num array sobrou[] as peças aleatórias desde o numero de pecas
total definida pelo jogador até as 27 possiveis e imprime
  308             printf("SOBROU:\n");
  309             for(i=pecastotal;i<28;i++){
  310                 //strcpy(sobrou[i],pecas[array[i]]);
  311                 //printf("%s\n",sobrou[i]);
  312
  313                 inserir sobrou baralho(b,pecas[array[i]]);
  314
  315             }
  316
  317             //imprime sobras do baralho
  318             pauxsob=b->psobrou;
  319
  320                 while(pauxsob!=NULL)
  321                 {
  322                     printf("%s\n",pauxsob->sobroustr);
  323                      pauxsob=pauxsob->proximo;
  324                 }
  325
  326     }else if(num>4){
  327         printf("Nao pode escolher mais que 4 baralhos!\n");
  328     }
  329 }
```

void convert_mao_int_to_string (INICIARPECA * *b*, int *pecastotal*)

Converter peças int para string.

Converte peças inteiro para strings

Parâmetros:

INICIARPECA	* b , estrutura do tipo INICIARPECA
pecastotal	Valor do tipo int, utilizado para definir as peças totais usadas pelo jogador

```
  370 {
  377     PECA * paux = NULL;
  378     PECA * pauxbar = NULL;
  379     PECAINT * pauxint = NULL;
  380     char str[4];
  381     int aux1 = 0 ,aux2 = 0,i = 0;
  382     int tamanhobar=b->nbar;
  383
  384     paux = b->pfirst;
  385     //percorro o baralho inicial e meto tudo vazio
  386     while(paux!=NULL)
  387     {
  388             paux->pecastr=NULL;
  389             b->nbar--;
  390             paux = paux->pnext;
  391     }
  392
  393
  394     //percorro a lista das pecas int e transformo e coloco de volta no baralho original
(b->pfirst)
  395     pauxint = b->pfirstint;
  396     for(i=0;i<tamanhobar;i++)
  397     {
  398         aux1=pauxint->dir;
  399         aux2=pauxint->esq;
```

```
400         str[0]=aux1+'0';
401         str[1]='|';
402         str[2]=aux2+'0';
403 
404         inserir peca baralho(b,str);
405 
406         pauxint = pauxint->pnextint;
407     }
408 
409     pauxbar = b->pfirst;
410 
411     //imprime baralho todo
412     while(pauxbar!=NULL)
413     {
414         printf("%s\n",pauxbar->pecastr);
415         pauxbar=pauxbar->pnext;
416     }
417 }
```

void convert_mao_string_to_int (INICIARPECA * *b*, int *pecastotal*)

Converter peças string para int.

Converte peças strings para inteiro

Parâmetros:

INICIARPECA	* b , estrutura do tipo INICIARPECA
pecastotal	Valor do tipo int, utilizado para definir as peças totais usadas pelo jogador

```
337 {
338     PECA * paux = NULL;
339     PECAINT * pauxbar = NULL;
340     char aux1[4];
341     char aux2[4];
342 
343     paux = b->pfirst;
344 
345     while(paux!=NULL)
346     {
347         strcpy(aux1,paux->pecastr);
348         strcpy(aux2,paux->pecastr);
349         aux1[1]='\0';
350         aux2[0]=aux2[2];
351         aux2[1]='\0';
352 
353         inserir pecaint baralho(b,atoi(aux1),atoi(aux2));
354 
355         paux = paux->pnext;
356     }
357 
358     pauxbar = b->pfirstint;
359 
360     //imprime baralho todo
361     while(pauxbar!=NULL)
362     {
363         printf("%d|%d\n",pauxbar->dir,pauxbar->esq);
364         pauxbar=pauxbar->pnextint;
365     }
366 }
```

void create_array_pecas (INICIARPECA * *b*, unsigned int *n*)

Criar array dinamico para pecas.

Remove peças de uma ou mais mãos, sendo que as peças removidas são substituidas por X|X, e é pedido ao utilizador para inserir tantas peças quantas removeu. Sendo tudo verificado para não remover peças que não existam e que não sejam adicionas peças repetidas.

Parâmetros:

INICIARPECA	* b , estrutura do tipo INICIARPECA
int	n, quantidade que vamos alocar

```
781 {
787     SEQUENCIA * paux = NULL;
788     SEQUENCIA * pnew = NULL;
789
790     int i=0;
791     int j=0;
792
793     if(b->pfirstseq==NULL && b->nseq==0)
794     {
795
796         pnew=(SEQUENCIA*)malloc(sizeof(SEQUENCIA)*n);
797         b->nseq = n;
798
799         for(i=0;i<n;i++)
800         {
801             (pnew+i)->seqstring = NULL;
802
803         }
804         b->pfirstseq=pnew;
805     }else{
806
807             b->nseq = n;
808             pnew=(SEQUENCIA*)malloc(sizeof(SEQUENCIA)*n);
809             paux=b->pfirstseq;
810
811
812         //copiar para o novo espaço
813         for(i=0;i<(b->nseq - 1);i++)
814         {
815             (pnew+i)->seqstring = (paux+i)->seqstring;
816         }
817         for(j=i;j<n;j++)
818         {
819             (pnew+j)->seqstring = NULL;
820         }
821
822         b->pfirstseq=pnew;
823
824     }
825 }
```

char* create_dyn_string (char *str*[])

Criar um array dinamico para strings.

Criar um array dinamico para strings

Parâmetros:

char	str[], recebe uma string

```
650 {
655     char *paux=NULL;
656     int slen = strlen(str)+1;
657
658     paux=(char*)malloc(sizeof(char)*slen);
659
660     strcpy(paux,str);
661
662     return paux;
```

```
663 }
```

void criarpecas (char *pecas*[][COLSTRING])

Criar peças string.

São criadas todas as peças possiveis que um jogo tem e são copiadas através do strcpy para dentro do array pecas

Parâmetros:

pecas[][COLSTRING]	Array do tipo char guarda as peças totais do jogo.

O que está comentado - Peças teste para usar no ponto R7, visto que pecas aleatorias de 2 ou mais mãos, ele não tem memória para gerar as sequencias

```
169                                             {
174     strcpy(pecas[0],"0|0");
175     strcpy(pecas[1],"0|1");
176     strcpy(pecas[2],"0|2");
177     strcpy(pecas[3],"0|3");
178     strcpy(pecas[4],"0|4");
179     strcpy(pecas[5],"0|5");
180     strcpy(pecas[6],"0|6");
181     strcpy(pecas[7],"1|1");
182     strcpy(pecas[8],"1|2");
183     strcpy(pecas[9],"1|3");
184     strcpy(pecas[10],"1|4");
185     strcpy(pecas[11],"1|5");
186     strcpy(pecas[12],"1|6");
187     strcpy(pecas[13],"2|2");
188     strcpy(pecas[14],"2|3");
189     strcpy(pecas[15],"2|4");
190     strcpy(pecas[16],"2|5");
191     strcpy(pecas[17],"2|6");
192     strcpy(pecas[18],"3|3");
193     strcpy(pecas[19],"3|4");
194     strcpy(pecas[20],"3|5");
195     strcpy(pecas[21],"3|6");
196     strcpy(pecas[22],"4|4");
197     strcpy(pecas[23],"4|5");
198     strcpy(pecas[24],"4|6");
199     strcpy(pecas[25],"5|5");
200     strcpy(pecas[26],"5|6");
201     strcpy(pecas[27],"6|6");
202
204 /*
205     strcpy(pecas[0],"2|5");
206     strcpy(pecas[1],"3|3");
207     strcpy(pecas[2],"2|2");
208     strcpy(pecas[3],"1|1");
209     strcpy(pecas[4],"4|4");
210     strcpy(pecas[5],"5|5");
211     strcpy(pecas[6],"6|6");
212     strcpy(pecas[7],"1|2");
213     strcpy(pecas[8],"6|3");
214     strcpy(pecas[9],"1|4");
215     strcpy(pecas[10],"1|5");
216     strcpy(pecas[11],"0|6");
217     strcpy(pecas[12],"2|1");
218     strcpy(pecas[13],"0|3");
219 */
220
221 }
```

PECA* find_peca_baralho (INICIARPECA * *b*, char *novapeca*[])

Procurar peça no baralho.

Verificar se a peça existe no baralho, caso haja retorna a peça, senão retorna NULL

Parâmetros:

INICIARPECA	* b , estrutura do tipo INICIARPECA
char	novapeca[], recebe a nova peça para verificar se existe no baralho

```
759 {
765     PECA *paux = NULL;
766     paux = b->pfirst;
767
768     while(paux !=NULL){
769         if( strcmp(paux->pecastr,novapeca)==0 ){
770             return paux;
771         }else{
772             paux=paux->pnext;
773         }
774     }
775
776     return NULL;
777 }
```

void imprimirpecas (char *pecas*[][COLSTRING])

Imprime as pecas em string.

Imprime o array das 28 pecas em string

Parâmetros:

pecas[][COLSTRING]	Array do tipo char onde recebe as peças totais do jogo.

```
223                                             {
228     int i = 0;
229     for(i=0;i<28;i++){
230         printf("%s\n",pecas[i]);
231     }
232 }
```

void imprimirpecasint (char *pecas*[][COLSTRING], int *pecasint*[][COL])

Imprimir pecas em inteiros.

Imprime o array das 28 pecas em inteiros, guarda apenas os dois valores inteiros e ignora a barra. Ex: 1 3

Parâmetros:

pecas[][COLSTRING]	Array do tipo char onde recebe as peças totais do jogo.
pecasint[][COL]	Array do tipo int onde guarda as peças totais do jogo.

```
236                                                                 {
242     int i=0;
243     for(i=0;i<28;i++){
244         pecasint[i][0]=atoi(pecas[i]);
245         pecasint[i][1]=atoi(&pecas[i][2]);
```

```
246         printf("%d|%d \n", pecasint[i][0],pecasint[i][1]);
247     }
248 }
```

void inserir_peca_baralho (INICIARPECA * *b*, char *pecas*[COLSTRING])

Inserir pecas no baralho (lista ligada)

Remove peças de uma ou mais mãos, sendo que as peças removidas são substituidas por X|X, e é pedido ao utilizador para inserir tantas peças quantas removeu. Sendo tudo verificado para não remover peças que não existam e que não sejam adicionas peças repetidas.

Parâmetros:

INICIARPECA	* b , estrutura do tipo INICIARPECA
char	pecas[COLSTRING] peca nova a inserir

```
535 {
541     PECA *pnew = (PECA*)malloc(sizeof(PECA));
542     PECA *paux = NULL;
543 
544     pnew->pecastr = create_dyn_string(pecas);
545     pnew->pnext=NULL;
546 
547     paux=b->pfirst;
548 
549     if(b->pfirst==NULL)
550     {
551 
552         b->pfirst=pnew;
553         b->nbar++;
554     }
555     else
556     {
557         //insere na cauda
558 
559         while(paux->pnext!=NULL)
560         {
561 
562             paux=paux->pnext;
563 
564         }
565 
566         paux->pnext=pnew;
567         b->nbar++;
568     }
569 
570 }
```

void inserir_pecaint_baralho (INICIARPECA * *b*, int *dir*, int *esq*)

Inserir pecas do tipo int no baralho (lista ligada)

Remove peças de uma ou mais mãos, sendo que as peças removidas são substituidas por X|X, e é pedido ao utilizador para inserir tantas peças quantas removeu. Sendo tudo verificado para não remover peças que não existam e que não sejam adicionas peças repetidas.

Parâmetros:

INICIARPECA	* b , estrutura do tipo INICIARPECA
int	dir , valor do tipo inteiro, da parte direita da peça
int	esq , valor do tipo inteiro, da parte esquerda da peça

```
612 {
```

```
619     PECAINT *pnew = (PECAINT*)malloc(sizeof(PECAINT));
620     PECAINT *paux = NULL;
621
622     pnew->dir = dir;
623     pnew->esq = esq;
624     pnew->pnextint=NULL;
625
626     paux=b->pfirstint;
627
628     if(b->pfirstint==NULL)
629     {
630
631         b->pfirstint=pnew;
632         b->npecasint++;
633
634     }
635     else
636     {
637         //insere na cauda
638         while(paux->pnextint!=NULL)
639         {
640             paux=paux->pnextint;
641         }
642
643         paux->pnextint=pnew;
644         b->npecasint++;
645     }
646 }
```

void inserir_seq_iniciarpeca (INICIARPECA * *b*, char *seq*[])

Inserir sequencias no array dinamico.

Inserir sequencias no array dinamico, caso não haja mais espaço para alocar, ele abre mais espaço recorrendo a outra função

Parâmetros:

INICIARPECA	* b , estrutura do tipo INICIARPECA
char	seq[], recebe sequencia a inserir no array dinamico

```
829 {
835     SEQUENCIA *paux=NULL;
836
837     paux=b->pfirstseq;
838
839     while(paux!=NULL&&paux->seqstring!=NULL && (paux - (b->pfirstseq)) < b->nseq)
840     {
841         paux++;
842     }
843
844     if((paux - (b->pfirstseq))==b->nseq)
845     {
846         create_array_pecas(b,b->nseq+1);
847         paux = b->pfirstseq + b->nseq-1;
848         paux->seqstring = create dyn string(seq);
849
850
851     }else
852     {
853
854         paux->seqstring=create_dyn_string(seq);
855
856     }
857
858 }
```

void inserir_sobrou_baralho (INICIARPECA * *b*, char *pecas*[COLSTRING])

Inserir pecas que sobraram do baralho (lista ligada)

Remove peças de uma ou mais mãos, sendo que as peças removidas são substituidas por X|X, e é pedido ao utilizador para inserir tantas peças quantas removeu. Sendo tudo verificado para não remover peças que não existam e que não sejam adicionas peças repetidas.

Parâmetros:

INICIARPECA	* b , estrutura do tipo INICIARPECA
char	pecas[COLSTRING] peca nova a inserir

```
574 {
580     SOBROU *pnew = (SOBROU*)malloc(sizeof(SOBROU));
581     SOBROU *paux = NULL;
582
583     pnew->sobroustr = create dyn string(pecas);
584     pnew->proximo=NULL;
585
586     paux=b->psobrou;
587
588     if(b->psobrou==NULL)
589     {
590
591         b->psobrou=pnew;
592         b->nsob++;
593     }
594     else
595     {
596         //insere na cauda
597
598         while(paux->proximo!=NULL)
599         {
600             paux=paux->proximo;
601         }
602
603         paux->proximo=pnew;
604         b->nsob++;
605
606         return;
607     }
608 }
```

void load_bin (INICIARPECA * *b*, char *fname*[])

Ler de um ficheiro binario.

Le as peças de uma ou mais maos e insere no jogo

Parâmetros:

INICIARPECA	* b , estrutura do tipo INICIARPECA
char	fname[], recebe o nome do ficheiro

```
1323 {
1329     FILE *fp=NULL;
1330
1331     int j=0;
1332     int size=0;
1333     char peca[50]="";
1334     int n = 0;
1335     if((fp=fopen(fname,"rb"))!=NULL)
1336     {
1337
```

```
1338          fread(&(b->nbar),sizeof(int),1,fp);
1339          n = b->nbar;
1340          b->nbar = 0;
1341
1342          //inserir pecas baralho
1343          for(j=0;j<n;j++)
1344          {
1345              fread(&size,sizeof(int),1,fp);
1346              fread(peca,sizeof(char),size,fp);
1347              printf("%s\n",peca);
1348              inserir peca baralho(b,peca);
1349          }
1350          fclose(fp);
1351      }
1352 }
```

void load_txt_jogo (INICIARPECA * *b*, char *fname*[])

Ler de um ficheiro de texto.

Le as peças de uma ou mais maos e insere no jogo

Parâmetros:

INICIARPECA	* b , estrutura do tipo INICIARPECA
char	fname[], recebe o nome do ficheiro

```
1283 {
1290      FILE *fp=NULL;
1291      int i=0;
1292      char peca[50];
1293      int nfields=0;
1294
1295
1296      if((fp=fopen(fname,"r"))==NULL)
1297      {
1298
1299          printf("... ERRO ...");
1300          return;
1301
1302      }
1303
1304
1305      for(i=0;i<7;i++)
1306      {
1307
1308          nfields=fscanf(fp,"%*[\n] %[^,] %*[,]",peca);
1309
1310          if(nfields>0)
1311          {
1312              inserir_peca_baralho(b,peca);
1313
1314          }
1315
1316      }
1317
1318
1319 }
```

int main_projetolp1primparte (int *argc*, char * *argv*[])

Função main_projectolp1primparte.

char pecas[LINSTRING][COLSTRING] - Array de strings para pecas

char baralho[LINSTRING][COLSTRING] - Array de strings de peças baralhadas, dependendo das mãos que o jogador pedir (4 mãos de 7 peças no máximo)

char sobrou[LINSTRING][COLSTRING] - Array de strings de pecas baralhadas, onde ficam as peças não utilizadas na mão do jogador

char arrayfinalcompleto[3000][150] - Array de strings de todas as sequencias possiveis das mãos do jogador

int opc=0 - Opção escolhida no menu inicial

int num=0 - Número de baralhos que o jogador escolhe (máximo 4)

int npecasremover=0 - Número de peças que o utilizador pretende remover da sua mão

int pecastotal=0 - Número de peças total que o jogador tem na mão, consoante o número de mãos escolhidas. Ex: 2 mão x 7 peças = 14 peças total

int npecas=0 - Número de peças que o utilizador pretende adicionar á sua mão

int y=0 - Tamanho do array de strings de todas as sequencias possiveis das mãos do jogador (arraycompleto)

criarpecas(pecas) - Criar todas peças do jogo

bool sair = false - Manter ciclo do menu enquanto for false, quando for true sai

char escolha - Usado para quando é escrito "S" ou "s" fecha programa, caso contrário continua

```
   44                                                      {
   46     char pecas[LINSTRING][COLSTRING];
   48     char baralho[LINSTRING][COLSTRING];
   50     char sobrou[LINSTRING][COLSTRING];
   52     char arrayfinalcompleto[3000][150];
   54     int opc=0;
   56     int num=0;
   58     int npecasremover=0;
   60     int pecastotal=0;
   62     int npecas=0;
   64     int y=0;
   66     criarpecas(pecas);
   68     bool sair = false;
   70     char escolha;
   71
   72     INICIARPECA b = {0,0,0,0,NULL,NULL,NULL,NULL};
   73
   74     create array pecas(&b,2);
   75
   76
   77     //Menu com ciclo. O utilizador escolhe a opção que pretende e através do switch
retorna para a função pretendida
   78     do{
   79     printf("*************** JOGO DO DOMINO ***************\n\n");
   80     printf("Escolha uma opcao\n");
   81     printf("1 ----> Listar Pecas Sring\n");
   82     printf("2 ----> Criar Mao\n");
   83     printf("3 ----> Remover Pecas\n");
   84     printf("4 ----> Adicionar Pecas\n");
   85     printf("5 ----> Listar baralho/mao string to int\n");
   86     printf("6 ----> Criar e ordenar sequencias\n");
   87     printf("7 ----> Procurar padrao\n");
   88     printf("8 ----> Criar sequencias com sequencia inicial\n");
   89     printf("9 ----> Criar sequencias com maos alternadas\n");
   90     printf("10 ----> Listar baralho/mao int to string\n");
   91     printf("11 ----> Guardar baralho/sequencias em ficheiro txt\n");
   92     printf("12 ----> Guardar sequencias em ficheiro bin\n");
   93     printf("13 ----> Ler de um ficheiro txt\n");
   94     printf("14 ----> Ler de um ficheiro binario\n");
   95
   96     printf("Opcao: ");
   97     scanf("%d", &opc);
```

```
 98
 99     switch(opc)
100             {
101             case 1:
102                 imprimirpecas(pecas);
103                 break;
104             case 2:
105                 printf("Insira o numero de mãos a jogar:\n");
106                 scanf("%d",&num);
107                 baralhos(&b,pecas,baralho,sobrou,num);
108                 break;
109             case 3:
110                 printf("Quantas pecas pretende remover:\n");
111                 scanf("%d",&npecasremover);
112                 pecastotal=num*7;
113                 rempeca(&b,pecastotal,npecasremover);
114                 break;
115             case 4:
116                 printf("Quantas pecas pretende adicionar:\n");
117                 scanf("%d",&npecas);
118                 pecastotal=num*7;
119                 addpeca(&b,pecastotal,npecas);
120                 break;
121             case 5:
122                 pecastotal=num*7;
123                 convert_mao_string_to_int(&b,pecastotal);
124                 break;
125             case 6:
126                 y=ordenarseq(&b,num);
127                 break;
128             case 7:
129                 procurar padrao(arrayfinalcompleto,y);
130                 break;
131             case 8:
132                 seq_inicial(baralho);
133                 break;
134             case 9:
135                 retirar mao jogadores(baralho,num);
136                 break;
137             case 10:
138                 pecastotal=num*7;
139                 convert_mao_int_to_string(&b,pecastotal);
140                 break;
141             case 11:
142                 save_txt_jogo(b,"./wbarseq.txt");
143                 break;
144             case 12:
145                 save_bin(b,"./binwseq.bin");
146                 break;
147             case 13:
148                 load txt jogo(&b,"./rbar.txt");
149                 break;
150             case 14:
151                 load_bin(&b,"binrbar.bin");
152                 break;
153
154             default:
155             printf("Escolha invalida!\n\n");
156             }
157             printf("Pretende sair? S --->sim \n");
158             scanf("%s",&escolha);
159             if(escolha=='S'||escolha=='s'){
160                 sair=true;
161             }else{
162                 sair=false;
163             }
164     }while(sair==false);
165     return 0;
166 }
```

int ordenarseq (INICIARPECA * *b*, int *num*)

Sequencias e ordenar sequencia por ordem decrescente.

Numa primeira parte, ele faz quatro verificações para juntar duas peças. Numa outra parte ele verifica se é possivel adicionar sequencias de mais de tres peças e vai juntando no arraydinamico.Tudo isto com verificações a ver se há repetidas, ou invertidas.

Parâmetros:

INICIARPECA	* b , estrutura do tipo INICIARPECA
num	Valor do tipo int, utilizado para definir o numero de mãos a ser usado pelo jogador

char invertnewpeca[150] - Usado para guardar um peça inversa

char invertnewpecaaux[150] - Usado para guardar um peça inversa

char invert[150] - Usado para guardar um peça inversa

char inversopeca[150] - Usado para guardar um peça inversa

char invertfinal[500] - Usado para guardar um peça inversa

```
  861                                         {
  867 int i=0,j=0,h=0,z=0;
  868 int x=0,k=0;
  869 char aux[500];
  871 char invertnewpeca[150];
  873 char invertnewpecaaux[150];
  875 char invert[150];
  877 char inversopeca[150];
  879 char invertfinal[500];
  880
  881 num=num*7;
  882                 //Juntar 2 pecas
  883
  884                 PECA * paux = NULL;
  885                 PECA * paux2 = NULL;
  886                 paux = b->pfirst;
  887
  888                 //percorre lista ligada baralho, uso o paux que é o primeiro e depois
o paux2 do segundo para a frente ate NULL
  889                 while(paux!=NULL)
  890                 {
  891                     paux2 = paux->pnext;
  892                     while(paux2!=NULL)
  893                     {
  894                         if((paux->pecastr)[2]==(paux2->pecastr)[0])
  895                         {
  896                             //concatena
  897                             strcpy(aux, paux->pecastr);
  898                             strcat(aux,"-");
  899                             strcat(aux, paux2->pecastr);
  900                             inserir seq iniciarpeca(b,aux);
  901
  902                         }
  903                         if((paux->pecastr)[2]==(paux2->pecastr)[2]){
  904                             // Inverter peca
  905                             int d=strlen(paux2->pecastr)-1;
  906                             for(h=0;h<(strlen(paux2->pecastr));h++){
  907                                 invertnewpeca[d]=paux2->pecastr[h];
  908                                 d--;
  909                             }
  910
  911                             // concatena
  912                             strcpy(aux, paux->pecastr);
  913                             strcat(aux,"-");
```

```
  914                             strcat(aux, invertnewpeca);
  915                             inserir_seq_iniciarpeca(b,aux);
  916                         }
  917                         if((paux->pecastr)[0]==(paux2->pecastr)[2]){
  918                             // concatena
  919                             strcpy(aux, paux2->pecastr);
  920                             strcat(aux,"-");
  921                             strcat(aux, paux->pecastr);
  922                             inserir seq iniciarpeca(b,aux);
  923                         }
  924                         if((paux->pecastr)[0]==(paux2->pecastr)[0]){
  925                             // Inverter peca
  926                             int d=strlen(paux2->pecastr)-1;
  927                             for(h=0;h<(strlen(paux2->pecastr));h++){
  928                                 invertnewpeca[d]=(paux2->pecastr)[h];
  929                                 d--;
  930                             }
  931 
  932                             // concatena
  933                             strcpy(aux, invertnewpeca);
  934                             strcat(aux,"-");
  935                             strcat(aux, paux->pecastr);
  936                             inserir seq iniciarpeca(b,aux);
  937                         }
  938                         paux2 = paux2->pnext;
  939                     }
  940                     paux = paux->pnext;
  941                 }
  942 
  943 
  944     //Verificar se ha sequencias iguais e o inversos tambem
  945     for(x=0;x<b->nseq;x++){
  946                 // Inverter peca
  947                 int w=strlen((b->pfirstseq+x)->seqstring)-1;
  948                 for(k=0;k<(strlen((b->pfirstseq+x)->seqstring));k++){
  949                     invertnewpecaaux[w]=((b->pfirstseq+x)->seqstring)[k];
  950                     w--;
  951                 }
  952     // Verifica se é igual e se for substitui por X|X
  953             for(i=x+1;i<b->nseq;i++){
  954 
if((strcmp(((b->pfirstseq+i)->seqstring),invertnewpecaaux)==0)||(strcmp(((b->pfirstseq+i
)->seqstring),((b->pfirstseq+x)->seqstring))==0)||(((((b->pfirstseq+i)->seqstring)[0])==
(((b->pfirstseq+i)->seqstring)[6]))&&((((b->pfirstseq+i)->seqstring)[2])==(((b->pfirstse
q+i)->seqstring)[4])))){
  955                     strcpy(((b->pfirstseq+i)->seqstring),"X|X");
  956                 }
  957             }
  958     }
  959 
  960     //remover os X|X e passa os debaixo para cima
  961 
  962         for(i=0;i<b->nseq;i++)
  963         {
  964             if(strcmp(((b->pfirstseq+i)->seqstring),"X|X")==0)
  965             {
  966                 for(j=i;j<b->nseq-1;j++)
  967                 {
  968                     strcpy(((b->pfirstseq+j)->seqstring),((b->pfirstseq
+(j+1))->seqstring));
  969                 }
  970                 b->nseq--;
  971                 i=0;
  972             }
  973 
  974         }
  975 
  976 
  977 // Juntar 3 ou mais pecas
  978 
  979     for(i=0;i<b->nseq;i++){
```

```
  980
  981             int tamlin=strlen((b->pfirstseq+i)->seqstring)-1;
  982             paux = b->pfirst;
  983             while(paux!=NULL)
  984             {
  985
  986                 //Primeira verificação
  987                 if(((b->pfirstseq+i)->seqstring)[tamlin]==(paux->pecastr)[0]){
  988                     int igual=0;
  989                     //inverte peca apenas para verificar
  990                     int w=strlen(paux->pecastr)-1;
  991                     for(k=0;k<(strlen(paux->pecastr));k++){
  992                         inversopeca[w]=(paux->pecastr)[k];
  993                         w--;
  994                     }
  995                     //strtok da peca
  996                     char partidos[100][100];
  997                     char *palavra=NULL;
  998                     char umaseqpeca[100];
  999
 1000                     strcpy(umaseqpeca,((b->pfirstseq+i)->seqstring));
 1001                     palavra = strtok (umaseqpeca,"-");
 1002                     int s=0;
 1003                     while (palavra != NULL)
 1004                     {
 1005                         strcpy(partidos[s++],palavra);
 1006                         palavra = strtok (NULL, "-");
 1007                     }
 1008
 1009                     //verifica se o arraypartidos ou o inverdo é igual a peca
 1010                     for (x=0;x<s;x++){
 1011
if(strcmp(partidos[x],(paux->pecastr))==0||strcmp(partidos[x],inversopeca)==0){
 1012                                     igual++;
 1013                             }
 1014                     }
 1015                     if(igual==0){
 1016                                 //concatena
 1017                                 strcpy(aux,((b->pfirstseq+i)->seqstring));
 1018                                 strcat(aux,"-");
 1019                                 strcat(aux, (paux->pecastr));
 1020                                 inserir_seq_iniciarpeca(b,aux);
 1021
 1022                     }
 1023                 }
 1024                 //Segunda verificação
 1025                 if(((b->pfirstseq+i)->seqstring)[tamlin]==(paux->pecastr)[2]){
 1026                     int igual=0;
 1027                     //inverte peca
 1028                     int w=strlen(paux->pecastr)-1;
 1029
 1030                     for(k=0;k<(strlen(paux->pecastr));k++){
 1031                         invert[w]=(paux->pecastr)[k];
 1032                         w--;
 1033                     }
 1034                     //strtok da peca
 1035                     char partidos[100][100];
 1036                     char *palavra=NULL;
 1037                     char umaseqpeca[100];
 1038
 1039                     strcpy(umaseqpeca,((b->pfirstseq+i)->seqstring));
 1040                     palavra = strtok (umaseqpeca,"-");
 1041                     int s=0;
 1042                     while (palavra != NULL)
 1043                     {
 1044                         strcpy(partidos[s++],palavra);
 1045                         palavra = strtok (NULL, "-");
 1046                     }
 1047
 1048                     //verifica se o arraypartidos ou o inverso é igual a peça
 1049                     for (x=0;x<s;x++){
```

```
 1050
if(strcmp(partidos[x],(paux->pecastr))==0||strcmp(partidos[x],invert)==0){
 1051                                     igual++;
 1052                                 }
 1053                     }
 1054                     if(igual==0){
 1055                         //concatena
 1056                         strcpy(aux, ((b->pfirstseq+i)->seqstring));
 1057                         strcat(aux,"-");
 1058                         strcat(aux, invert);
 1059                         inserir seq iniciarpeca(b,aux);
 1060
 1061                     }
 1062                 }
 1063
 1064                 //Terceira verificação
 1065                 if(((b->pfirstseq+i)->seqstring)[0]==(paux->pecastr)[0]){
 1066                     int igual=0;
 1067
 1068                     //inverte peca
 1069                     int w=strlen(paux->pecastr)-1;
 1070                     for(k=0;k<(strlen(paux->pecastr));k++){
 1071                         invert[w]=(paux->pecastr)[k];
 1072                         w--;
 1073                     }
 1074
 1075                     //strtok da peca
 1076                     char partidos[100][100];
 1077                     char *palavra=NULL;
 1078                     char umaseqpeca[100];
 1079                     strcpy(umaseqpeca,((b->pfirstseq+i)->seqstring));
 1080                     palavra = strtok (umaseqpeca,"-");
 1081                     int s=0;
 1082                     while (palavra != NULL)
 1083                     {
 1084                         strcpy(partidos[s++],palavra);
 1085                         palavra = strtok (NULL, "-");
 1086                     }
 1087
 1088                     //verifica se o arraypartidos ou o inverso é igual a peça
 1089                     for (x=0;x<s;x++){
 1090
if(strcmp(partidos[x],(paux->pecastr))==0||strcmp(partidos[x],invert)==0){
 1091                                     igual++;
 1092                                 }
 1093                     }
 1094                     if(igual==0){
 1095                         //concatena
 1096                         strcpy(aux,invert);
 1097                         strcat(aux,"-");
 1098                         strcat(aux,((b->pfirstseq+i)->seqstring));
 1099                         inserir_seq_iniciarpeca(b,aux);
 1100
 1101                     }
 1102                 }
 1103                 //Quarta verificação
 1104                 if(((b->pfirstseq+i)->seqstring)[0]==(paux->pecastr)[2]){
 1105                     int igual=0;
 1106
 1107                     //inverte peca
 1108                     int w=strlen(paux->pecastr)-1;
 1109                     for(k=0;k<(strlen(paux->pecastr));k++){
 1110                         invert[w]=(paux->pecastr)[k];
 1111                         w--;
 1112                     }
 1113                     //strtok da peca
 1114                     char partidos[100][100];
 1115                     char *palavra=NULL;
 1116                     char umaseqpeca[100];
 1117
 1118                     strcpy(umaseqpeca,((b->pfirstseq+i)->seqstring));
```

```
 1119                     palavra = strtok (umaseqpeca,"-");
 1120                     int s=0;
 1121                     while (palavra != NULL)
 1122                     {
 1123                         strcpy(partidos[s++],palavra);
 1124                         palavra = strtok (NULL, "-");
 1125                     }
 1126
 1127                     //verifica se o arraypartidos ou o inverso é igual a peça
 1128                     for (x=0;x<s;x++){
 1129
if(strcmp(partidos[x],(paux->pecastr))==0||strcmp(partidos[x],invert)==0){
 1130                                     igual++;
 1131                                 }
 1132                     }
 1133                     if(igual==0){
 1134                         //concatena
 1135                         strcpy(aux,(paux->pecastr));
 1136                         strcat(aux,"-");
 1137                         strcat(aux,((b->pfirstseq+i)->seqstring));
 1138                         inserir seq iniciarpeca(b,aux);
 1139
 1140                     }
 1141                 }
 1142                paux = paux->pnext;
 1143             }
 1144         }
 1145
 1146     //verificar se ha sequencias iguais e inversos tambem
 1147
 1148     for(x=0;x<b->nseq;x++){
 1149                 // Inverter peca
 1150                 int w=strlen((b->pfirstseq+x)->seqstring)-1;
 1151                 for(k=0;k<(strlen((b->pfirstseq+x)->seqstring));k++){
 1152                     invertfinal[w]=((b->pfirstseq+x)->seqstring)[k];
 1153                     w--;
 1154                 }
 1155
 1156             // Verifica se e igual e se for substitui por X|X
 1157             for(i=x+1;i<b->nseq;i++){
 1158
if((strcmp(((b->pfirstseq+i)->seqstring),invertfinal)==0)||(strcmp(((b->pfirstseq+i)->se
qstring),((b->pfirstseq+x)->seqstring))==0)){
 1159                     strcpy(((b->pfirstseq+i)->seqstring),"X|X");
 1160                 }
 1161             }
 1162     }
 1163
 1164
 1165     //remover os X|X e passa os debaixo para cima
 1166
 1167         for(i=0;i<b->nseq;i++)
 1168         {
 1169             if(strcmp(((b->pfirstseq+i)->seqstring),"X|X")==0)
 1170             {
 1171                 for(j=i;j<b->nseq-1;j++)
 1172                 {
 1173                     strcpy(((b->pfirstseq+j)->seqstring),((b->pfirstseq
+(j+1))->seqstring));
 1174                 }
 1175                 b->nseq--;
 1176                 i=0;
 1177             }
 1178
 1179         }
 1180
 1181     //imprimir seq sem repeticoes
 1182     SEQUENCIA * pauxarray = NULL;
 1183     pauxarray = b->pfirstseq;
 1184
 1185     for(z=0;z<b->nseq;z++)
```

```
 1186     {
 1187         printf("%s\n",((pauxarray+z)->seqstring));
 1188     }
 1189     return 0;
 1190 }
```

void procurar_padrao (char *arrayfinalcompleto*[][150], int)

Funçao procurar e substituir padrão.

Nesta função é recebido uma sequencia aleatoria do arrayfinalcompleto, o utilizador insere uma sequencia a procurar,é verificada para ver se está inserida corretamente e se nao existe peças repetidas na mão do jogador. Depois é perguntado se pretende substituir esse padrao. Caso pretende ele faz verificaçoes dessa peça para ver se existe esse padrao de forma normal ou invertida e se for possivel troca o padrão.

Parâmetros:

arrayfinalcompleto [][150]	Array do tipo char, onde é recebido as sequencias totais da mão do jogador
y,tamanho	do array total das sequencias (arrayfinalcompleto)

```
 1356                                                                {
 1362     //Guarda todas as posicoes num array para depois baralhar esses i's
 1363     int z=0;
 1364     int auxp[50];
 1365     int ptemp=0;
 1366     int aleatorio=0;
 1367     int b=0,i=0;
 1368     char strproc[100];
 1369     int tamarraycompleto=y;
 1370
 1371     for(z=0;z<tamarraycompleto;z++){
 1372         auxp[z]=z;
 1373     }
 1374
 1375     //baralho esse numeros do auxp
 1376     for(b=0;b<tamarraycompleto;b++){
 1377         ptemp=auxp[b];
 1378         aleatorio=rand()%(tamarraycompleto-1);
 1379         auxp[b]=auxp[aleatorio];
 1380         auxp[aleatorio]=ptemp;
 1381     }
 1382     //vai buscar uma sequencia do arrayfinalcompleto dependendo do num de sequencias
que o utilizador quer
 1383         strcpy(strproc,arrayfinalcompleto[auxp[0]]);
 1384         printf("A sequencia escolhida foi:\n %s \n",strproc);
 1385
 1386
 1387     char sub[50];
 1388     char str2[100][100];
 1389     //declaracoes partir arrayfinalcompleto
 1390     char partidosstr1[100][100];
 1391     char *s1=NULL;
 1392     char str1apartir[100];
 1393     int u=0;
 1394     //declaracoes partir subtring
 1395     char partidosstr2[100][100];
 1396     char *s2=NULL;
 1397     char str2apartir[100];
 1398     int l=0;
 1399     //declaracoes partir subtring inversa
 1400     char partidosstr2inv[100][100];
 1401     char *s2inv=NULL;
 1402     char str2invapartir[100];
 1403     int inv=0;
 1404     //declaracao inverter subtring
```

```
 1405     char invertsub[100];
 1406     //declaracoes comparar a ver se sao iguais
 1407     int x=0,g=0,count=0,r=0,pos=0;;
 1408     int v[100];
 1409
 1410
 1411     do{
 1412
 1413     printf("Insira uma subsequencia para procurar:\n");
 1414     scanf("%s",sub);
 1415
 1416                 //Strtok do arrayfinalcompleto
 1417                 strcpy(str1apartir,strproc);
 1418                 s1 = strtok (str1apartir,"-");
 1419
 1420                 while (s1!= NULL)
 1421                 {
 1422                     strcpy(partidosstr1[u++],s1);
 1423                     s1= strtok (NULL, "-");
 1424                 }
 1425
 1426                 //Strtok da substring escolhida
 1427                 strcpy(str2[0],sub);
 1428                 strcpy(str2apartir,str2[0]);
 1429                 s2 = strtok (str2apartir,"-");
 1430                 while (s2!= NULL)
 1431                 {
 1432                     strcpy(partidosstr2[l++],s2);
 1433                     s2= strtok (NULL, "-");
 1434                 }
 1435
 1436                 // Inverter substring
 1437                     int w=strlen(sub)-1;
 1438                     int k=0;
 1439                     for(k=0;k<(strlen(sub));k++){
 1440                         invertsub[w]=sub[k];
 1441                         w--;
 1442                     }
 1443
 1444                 //Strtok da substring invertida
 1445
 1446                 strcpy(str2invapartir,invertsub);
 1447                 s2inv = strtok (str2invapartir,"-");
 1448                 while (s2inv!= NULL)
 1449                 {
 1450                     strcpy(partidosstr2inv[inv++],s2inv);
 1451                     s2inv= strtok (NULL, "-");
 1452                 }
 1453
 1454                 //Verifica se a invertida é igual a string original (partidosstr1)
 1455                 for(x=0;x<u;x++){
 1456                     if((strcmp(partidosstr1[x],partidosstr2inv[g])==0)){
 1457                         count++;
 1458                         g++;
 1459                       if(count==g && count==l){
 1460                         //pos- é o valor da peca final menos o tam da sub -1, para
retornar a primeira posicao onde encontrou
 1461                         pos=(x-(l-1));
 1462                         v[r]=pos;
 1463                         r++;
 1464                         count=0;
 1465                         g=0;
 1466                         pos=0;
 1467                         strcpy(sub,invertsub);
 1468                       }
 1469                     }else{
 1470                         count=0;
 1471                         g=0;
 1472                         pos=0;
 1473                     }
 1474                 }
```

```
 1475                 //Verifica se a string original partida (partidosstr1) é igual a
substring inserida partida (partidosstr2)
 1476                 for(x=0;x<u;x++){
 1477                     if(strcmp(partidosstr1[x],partidosstr2[g])==0){
 1478                         count++;
 1479                         g++;
 1480                       if(count==g && count==l){
 1481                         //pos- é o valor da peca final menos o tam da sub -1, para
retornar a primeira posicao onde encontrou
 1482                         pos=(x-(l-1));
 1483                         v[r]=pos;
 1484                         r++;
 1485                         count=0;
 1486                         g=0;
 1487                         pos=0;
 1488                       }
 1489                     }else{
 1490                         count=0;
 1491                         g=0;
 1492                         pos=0;
 1493                     }
 1494                 }
 1495                 if(r==0){
 1496                     printf("Nao foi encontado a subsequencia %s, na sequencia %s!\n
",sub,strproc);
 1497                     u=0;
 1498                     l=0;
 1499
 1500                 }else{
 1501                     for(i=0;i<r;i++){
 1502                         printf(" Posicao: %d\n",v[i]);
 1503                     }
 1504                 }
 1505     }while(r==0);
 1506                 //Substituir Padrão
 1507                 int opcao=0,j=0,q=0;
 1508                 char aux[100][100];
 1509                 char addseqpeca[50];
 1510                 printf("Pretende substituir essa sequencia por outra peca/seq:\n
sim->1 , nao->0\n");
 1511                 scanf("%d",&opcao);
 1512                 if(opcao==1){
 1513                     int continua=1;
 1514                     while(continua>0){
 1515                         continua=0;
 1516                         printf("Insira a seq/peca que deseja substituir");
 1517                         scanf("%s",addseqpeca);
 1518
 1519                         //Strtok da peça adicionada (addseqpeca)
 1520                         char partidasaddseqpeca[100][100];
 1521                         char *s3=NULL;
 1522                         char addseqpecaapartir[100];
 1523                         strcpy(addseqpecaapartir,addseqpeca);
 1524                         s3 = strtok (addseqpecaapartir,"-");
 1525                         int b=0;
 1526
 1527                         while (s3!= NULL)
 1528                         {
 1529                             strcpy(partidasaddseqpeca[b++],s3);
 1530                             s3= strtok (NULL, "-");
 1531                         }
 1532
 1533                         //Verifica se essa seq adicionada é possivel inserir
 1534                         char invert[4];
 1535                         int cont=0,n=0,m=0;
 1536                         for(n=0;n<u;n++){
 1537                             for(m=0;m<b;m++){
 1538                                     // Inverter addseqpeca
 1539                                     int d=strlen(partidasaddseqpeca[m])-1;
 1540                                     int h=0;
```

```
 1541
for(h=0;h<(strlen(partidasaddseqpeca[m]));h++){
 1542                                        invert[d]=partidasaddseqpeca[m][h];
 1543                                        d--;
 1544                                    }
 1545                                    //verifica se a peça adicionada existe na string
original , ou se a peça adicionada invertida existe na string original
 1546
if((strcmp(partidasaddseqpeca[m],partidosstr1[n])==0)||(strcmp(partidosstr1[n],invert)==
0)){
 1547                                        cont++;
 1548                                    }
 1549                                }
 1550                            }
 1551                            if(cont>0){
 1552
 1553                                    printf("Essa peca ja existe!\n");
 1554                                    continua++;
 1555                                    cont=0;
 1556                            }else{
 1557                                    //verifica se o padrao encontrado está no meio,
á frente ou atrás.
 1558
if(((sub[0]==addseqpeca[0])&&(sub[strlen(sub)-1]==addseqpeca[strlen(addseqpeca)-1]))||((
addseqpeca[0]==sub[0])&&(sub[strlen(sub)-1]==strproc[strlen(strproc)-1])&&(sub[strlen(su
b)-3]==strproc[strlen(strproc)-3]))||((addseqpeca[strlen(addseqpeca)-1]==sub[strlen(sub)
-1])&&(sub[0]==strproc[0])&&(sub[2]==strproc[2]))){
 1559                                        cont=0;
 1560                                        //Adiciona no aux as primeiras pecas, as
novas e depois o resto
 1561                                            for(i=0;i<v[0];i++){
 1562                                                strcpy(aux[q],partidosstr1[i]);
 1563                                                    q++;
 1564                                            }
 1565                                            int c=0;
 1566                                            for(c=0;c<b;c++){
 1567
strcat(aux[q],partidasaddseqpeca[c]);
 1568                                                q++;
 1569                                            }
 1570                                            for(j=(v[0]+l);j<u;j++){
 1571
strcat(aux[q],partidosstr1[j]);
 1572                                                    q++;
 1573                                            }
 1574                                }else{
 1575                                    printf("Pecas inseridas nao coincidem!\n");
 1576                                    continua++;
 1577                                }
 1578                            }
 1579                            for(m=0;m<q;m++){
 1580                                printf("%s",aux[m]);
 1581                                if(m!=(q-1)){
 1582                                    printf("-");
 1583                                }
 1584                            }
 1585                            printf("\n");
 1586                }
 1587                }else if(opcao==0){
 1588
 1589                }else{
 1590                    printf("opcao errada!");
 1591                }
 1592 }
```

void rempeca (INICIARPECA * *b*, int *pecastotal*, int *npecasremover*)

Remover peças de uma ou mais mãos.

Remove peças de uma ou mais mãos, sendo que as peças removidas são substituidas por X|X, e é pedido ao utilizador para inserir tantas peças quantas removeu. Sendo tudo verificado para não remover peças que não existam e que não sejam adicionas peças repetidas.

Parâmetros:

INICIARPECA	* b , estrutura do tipo INICIARPECAr
pecastotal	Valor do tipo int, utilizado para definir as peças totais usadas pelo jogador
npecasremover	Valor do tipo int, utilizado para definir o numero de peças a ser removido pelo jogador

char pecaremover[4] - Usado para guardar a peça inserida pelo jogador, para depois remover

int j=0 - Variavél usada nas funções seguintes

int h=0 - Variavél usada nas funções seguintes

char newpeca[4] - Usado para guardar a peça a ser inserida pelo jogador, para depois adicionar

char invertnewpeca[4] - Usado para guardar a peça invertida da newpeca inserida

```
  419                                                                 {
  427     char pecaremover[4];
  429     int j=0;
  431     int h=0;
  433     char newpeca[4];
  435     char invertnewpecarem[4]={};
  436
  437     PECA * pauxbar = NULL;
  438
  439     //Caso o numero de peças a remover seja maior que as peças totais, entra neste if
  440     if(npecasremover>pecastotal){
  441         printf("Numero de pecas a remover e superior ao baralho!\n");
  442     // Caso o numero de peças a remover seja menor que as peças totais, entra neste
if
  443     }else if(npecasremover<pecastotal){
  444         // Insere peca a remover
  445         for(j=0;j<npecasremover;j++){
  446             printf("Insira a peca a remover:");
  447             scanf("%s",pecaremover);
  448
  449             PECA * pauxfind = NULL;
  450             PECA * paux = NULL;
  451             PECA * pant = NULL;
  452
  453             pauxfind = find_peca_baralho(b,pecaremover);
  454
  455             if(pauxfind != NULL){
  456                 paux = b->pfirst;
  457                 //percorre todas as peças
  458                 while(paux!=NULL && strcmp(paux->pecastr,pecaremover)!=0){
  459                         pant=paux;
  460                         paux=paux->pnext;
  461                 }
  462                 //remover na cabeça
  463                 if(paux==b->pfirst)
  464                 {
  465                     b->pfirst=paux->pnext;
  466                     b->nbar--;
  467                 }else if(paux->pnext==NULL)
  468                 {
  469                     //remover na cauda
  470                     pant->pnext=NULL;
  471                     b->nbar--;
  472                 }else{
  473                     //se for no meio
  474                     pant->pnext=paux->pnext;
  475                     b->nbar--;
  476                 }
```

```
 477
 478             }else{
 479                printf("Essa peca nao existe, insira outra:\n");
 480                   j--;
 481             }
 482         }
 483
 484         //Imprime mão já com as peças removidas
 485         pauxbar = b->pfirst;
 486         while(pauxbar!=NULL)
 487         {
 488             printf("%s\n",pauxbar->pecastr);
 489             pauxbar=pauxbar->pnext;
 490         }
 491
 492         //Foram removidas peças logo insiro novas para completar mão
 493         for(j=0;j<npecasremover;j++){
 494             printf("Insira uma nova peca:");
 495             scanf("%s",newpeca);
 496
 497             // Inverter peça inserida
 498             int d=strlen(newpeca)-1;
 499             for(h=0;h<(strlen(newpeca));h++){
 500                 invertnewpecarem[d]=newpeca[h];
 501                 d--;
 502             }
 503             // Verifica se a peca da mão(baralho[]) é igual a novapeca, ou á invertida
 504             PECA * pauxfindrem = NULL;
 505             PECA * pauxfindinvrem = NULL;
 506
 507             pauxfindrem = find_peca_baralho(b,newpeca);
 508
 509             pauxfindinvrem = find_peca_baralho(b,invertnewpecarem);
 510
 511             if(pauxfindrem != NULL || pauxfindinvrem != NULL){
 512                         printf("A peca ja existe!\n");
 513                         j--;
 514
 515             }else{
 516                         pauxbar = b->pfirst;
 517
 518                         //Caso contrario ele insere a nova peça na ultima posição da
mão (baralho[]) e imprime
 519
 520                         inserir_peca_baralho(b,newpeca);
 521
 522                         //imprime baralho todo
 523                         while(pauxbar!=NULL)
 524                         {
 525                             printf("%s\n",pauxbar->pecastr);
 526                             pauxbar=pauxbar->pnext;
 527                         }
 528             }
 529         }
 530     }
 531 }
```

void retirar_mao_jogadores (char *baralho*[][COLSTRING], int)

Cria as sequencias possiveis apartir de uma sequencia inicial com 2 ou mais mãos.

A partir de uma sequencia inicial inserida pelo utilizador, ele vai guardando num array mixpecas de forma alternada as mãos pedidas pelo jogador, e vai inserido do array mixpecas de forma alternada na sequencia inicial e guardado no arrayfinalmixpecascompleto.

Parâmetros:

baralho[][COLSTRING]	Array do tipo char, onde recebe as peças baralhadas da mão do jogador

char mixpecas[3000][150] - guarda varios baralhos de forma alternada

```
1852                                                                {
1858      char mixpecas[3000][150];
1859
1860     //Ver quantos baralhos sao, e guardar num array mix as pecas dos jogadores
alternadamente
1861     if(num==2){
1862         int i=0;
1863         int j=7;
1864         int m=0;
1865
1866         while(i<7 || j<14){
1867             if(i<7){
1868                 strcpy(mixpecas[m++],baralho[i]);
1869             }
1870             if(j<14){
1871                 strcpy(mixpecas[m++],baralho[j]);
1872             }
1873             i++;
1874             j++;
1875         }
1876     }
1877     if(num==3){
1878         int i=0;
1879         int j=7;
1880         int k=14;
1881         int m=0;
1882
1883         while(i<7 || j<14 || k<21 ){
1884             if(i<7){
1885                 strcpy(mixpecas[m++],baralho[i]);
1886             }
1887             if(j<14){
1888                 strcpy(mixpecas[m++],baralho[j]);
1889             }
1890             if(k<21){
1891                 strcpy(mixpecas[m++],baralho[k]);
1892             }
1893             i++;
1894             j++;
1895             k++;
1896         }
1897     }
1898
1899     if(num==4){
1900         int i=0;
1901         int j=7;
1902         int k=14;
1903         int s=21;
1904         int m=0;
1905
1906         while(i<7 || j<14 || k<21 || s<28){
1907             if(i<7){
1908                 strcpy(mixpecas[m++],baralho[i]);
1909             }
1910             if(j<14){
1911                 strcpy(mixpecas[m++],baralho[j]);
1912             }
1913             if(k<21){
1914                 strcpy(mixpecas[m++],baralho[k]);
1915             }
1916             if(s<28){
1917                 strcpy(mixpecas[m++],baralho[s]);
1918             }
1919             i++;
1920             j++;
```

```
 1921             k++;
 1922             s++;
 1923         }
 1924     }
 1925
 1926     printf("ARRAY MIX:\n");
 1927     int v=0;
 1928     int pecasmix=num*7;
 1929     for(v=0;v<pecasmix;v++){
 1930         printf("%s\n",mixpecas[v]);
 1931     }
 1932
 1933     //Comeca com uma sequencia inicial e vai metendo peca a peca do mixpecas
 1934         char seqcomecar[3000];
 1935         char arrayfinalmixpecas[3000][150];
 1936         char arrayfinalmixpecascompleto[3000][150];
 1937         int i=0,j=0,count=0,x=0,y=0,p=1,z=0,contador=0;
 1938         char inversazero[150];
 1939         char inversoum[150];
 1940         char inversodois[150];
 1941         char inversotres[150];
 1942         char inversoquatro[150];
 1943         char invertfinal[150];
 1944
 1945         printf("Insira uma sequencia a comecar:\n");
 1946         scanf("%s",seqcomecar);
 1947
 1948         strcpy(arrayfinalmixpecas[0],seqcomecar);
 1949
 1950                 //Strtok da sequencia inicial
 1951                 char partidosseqcomecar[100][100];
 1952                 char *s=NULL;
 1953                 char seqapartir[100];
 1954
 1955                 strcpy(seqapartir,seqcomecar);
 1956                 s = strtok (seqapartir,"-");
 1957                 int k=0;
 1958
 1959                 while (s!= NULL)
 1960                 {
 1961                     strcpy(partidosseqcomecar[k++],s);
 1962                     s= strtok (NULL, "-");
 1963                 }
 1964
 1965                 //Verificar se as pecas da sequencia inicial sao iguais as pecas do
baralho mix
 1966                 for(x=0;x<pecasmix;x++){
 1967                     for(y=0;y<k;y++){
 1968                             // Inverter pecas baralho
 1969                             int d=strlen(mixpecas[x])-1;
 1970                             int h=0;
 1971                             for(h=0;h<(strlen(mixpecas[x]));h++){
 1972                                     inversazero[d]=mixpecas[x][h];
 1973                                     d--;
 1974                             }
 1975                             //Verifica se as pecas da seq inicial coincidem
 1976                             int tam=strlen(seqcomecar)-1;
 1977                             int i=0;
 1978                             for(i=0;i<tam;i++){
 1979                                 if(seqcomecar[i]=='-'){
 1980                                     if(seqcomecar[i-1]!=seqcomecar[i+1]){
 1981                                         contador++;
 1982                                     }
 1983                                 }
 1984                             }
 1985
if((strcmp(mixpecas[x],partidosseqcomecar[y])==0)||(strcmp(inversazero,partidosseqcomeca
r[y])==0)){
 1986                             count++;
 1987                         }
 1988                     }
```

```
 1989                 }
 1990                 if(count>0){
 1991                     printf("As pecas da sequencia que inseriu, ja existem no
baralho!\n");
 1992                     count=0;
 1993                 }else if(contador>0){
 1994                     printf("A sequencia que inseriu esta errada!\n ");
 1995                     contador=0;
 1996                 }else{
 1997                     //Juntar pecas
 1998                     for(i=0;i<p;i++){
 1999                         int tamlin=strlen(arrayfinalmixpecas[i])-1;
 2000                         for(j=0;j<pecasmix;j++){
 2001                             //Primeira verificação
 2002                             if(arrayfinalmixpecas[i][tamlin]==mixpecas[j][0]){
 2003                                 int igual=0;
 2004                                 //inverte peca apenas para verificar
 2005                                 int w=strlen(mixpecas[j])-1;
 2006                                 for(k=0;k<(strlen(mixpecas[j]));k++){
 2007                                     inversoum[w]=mixpecas[j][k];
 2008                                     w--;
 2009                                 }
 2010                                 //strtok da peca arrayfinalmixpecas
 2011                                 char partidos[100][100];
 2012                                 char *palavra=NULL;
 2013                                 char umaseqpeca[100];
 2014
 2015                                 strcpy(umaseqpeca,arrayfinalmixpecas[i]);
 2016                                 palavra = strtok (umaseqpeca,"-");
 2017                                 int s=0;
 2018                                 while (palavra != NULL)
 2019                                 {
 2020                                     strcpy(partidos[s++],palavra);
 2021                                     palavra = strtok (NULL, "-");
 2022                                 }
 2023
 2024                                 //verifica se o arrayfinalmixpecas ou o inverso e
igual a peca
 2025                                 for (x=0;x<s;x++){
 2026
if(strcmp(partidos[x],mixpecas[j])==0||strcmp(partidos[x],inversoum)==0){
 2027                                                         igual++;
 2028                                             }
 2029                                 }
 2030                                 if(igual==0){
 2031                                     //concatena para arrayfinalmixpecas
 2032                                             strcpy(arrayfinalmixpecas[p],
arrayfinalmixpecas[i]);
 2033                                             strcat(arrayfinalmixpecas[p],"-");
 2034                                             strcat(arrayfinalmixpecas[p],
mixpecas[j]);
 2035                                             p++;
 2036                                 }
 2037                             }
 2038                             //Segunda verificação
 2039                             if(arrayfinalmixpecas[i][tamlin]==mixpecas[j][2]){
 2040                                 int igual=0;
 2041                                 //inverte peca
 2042                                 int w=strlen(mixpecas[j])-1;
 2043                                 for(k=0;k<(strlen(mixpecas[j]));k++){
 2044                                     inversodois[w]=mixpecas[j][k];
 2045                                     w--;
 2046                                 }
 2047                                 //strtok da peca arrayfinalmixpecas
 2048                                 char partidos[100][100];
 2049                                 char *palavra=NULL;
 2050                                 char umaseqpeca[100];
 2051
 2052                                 strcpy(umaseqpeca,arrayfinalmixpecas[i]);
 2053                                 palavra = strtok (umaseqpeca,"-");
 2054                                 int s=0;
```

```
 2055                                    while (palavra != NULL)
 2056                                    {
 2057                                        strcpy(partidos[s++],palavra);
 2058                                        palavra = strtok (NULL, "-");
 2059                                    }
 2060                                    //verifica se o partidos ou o inverso e igual a peca
 2061                                    for (x=0;x<s;x++){
 2062
if(strcmp(partidos[x],mixpecas[j])==0||strcmp(partidos[x],inversodois)==0){
 2063                                                    igual++;
 2064                                            }
 2065                                    }
 2066                                    if(igual==0){
 2067                                        //concatena
 2068                                        strcpy(arrayfinalmixpecas[p],
arrayfinalmixpecas[i]);
 2069                                        strcat(arrayfinalmixpecas[p],"-");
 2070                                        strcat(arrayfinalmixpecas[p], inversodois);
 2071                                        p++;
 2072                                    }
 2073                                }
 2074                                //Terceira verificação
 2075                                if(arrayfinalmixpecas[i][0]==mixpecas[j][0]){
 2076                                    int igual=0;
 2077                                    //inverte peca
 2078                                    int w=strlen(mixpecas[j])-1;
 2079                                    for(k=0;k<(strlen(mixpecas[j]));k++){
 2080                                        inversotres[w]=mixpecas[j][k];
 2081                                        w--;
 2082                                    }
 2083                                    //strtok da peca arrayfinal
 2084                                    char partidos[100][100];
 2085                                    char *palavra=NULL;
 2086                                    char umaseqpeca[100];
 2087
 2088                                    strcpy(umaseqpeca,arrayfinalmixpecas[i]);
 2089                                    palavra = strtok (umaseqpeca,"-");
 2090                                    int s=0;
 2091                                    while (palavra != NULL)
 2092                                    {
 2093                                        strcpy(partidos[s++],palavra);
 2094                                        palavra = strtok (NULL, "-");
 2095                                    }
 2096                                    //verifica se o partidos ou o inverso é igual a peca
 2097                                    for (x=0;x<s;x++){
 2098
if(strcmp(partidos[x],mixpecas[j])==0||strcmp(partidos[x],inversotres)==0){
 2099                                                    igual++;
 2100                                            }
 2101                                    }
 2102                                    if(igual==0){
 2103                                        //concatena
 2104                                        strcpy(arrayfinalmixpecas[p], inversotres);
 2105                                        strcat(arrayfinalmixpecas[p],"-");
 2106                                        strcat(arrayfinalmixpecas[p],
arrayfinalmixpecas[i]);
 2107                                        p++;
 2108                                    }
 2109                                }
 2110                                //Quarta verificação
 2111                                if(arrayfinalmixpecas[i][0]==mixpecas[j][2]){
 2112                                    int igual=0;
 2113
 2114                                    //inverte peca
 2115                                    int w=strlen(mixpecas[j])-1;
 2116                                    for(k=0;k<(strlen(mixpecas[j]));k++){
 2117                                        inversoquatro[w]=mixpecas[j][k];
 2118                                        w--;
 2119                                    }
 2120                                    //strtok da peca arrayfinal
 2121                                    char partidos[100][100];
```

```
 2122                                    char *palavra=NULL;
 2123                                    char umaseqpeca[100];
 2124
 2125                                    strcpy(umaseqpeca,arrayfinalmixpecas[i]);
 2126                                    palavra = strtok (umaseqpeca,"-");
 2127                                    int s=0;
 2128                                    while (palavra != NULL)
 2129                                    {
 2130                                        strcpy(partidos[s++],palavra);
 2131                                        palavra = strtok (NULL, "-");
 2132                                    }
 2133                                    //verifica se o partidos ou o inverso é igual a peca
 2134                                    for (x=0;x<s;x++){
 2135
if(strcmp(partidos[x],mixpecas[j])==0||strcmp(partidos[x],inversoquatro)==0){
 2136                                                            igual++;
 2137                                                }
 2138                                    }
 2139                                    if(igual==0){
 2140                                        //concatena
 2141                                        strcpy(arrayfinalmixpecas[p], mixpecas[j]);
 2142                                        strcat(arrayfinalmixpecas[p],"-");
 2143                                        strcat(arrayfinalmixpecas[p],
arrayfinalmixpecas[i]);
 2144                                        p++;
 2145                                    }
 2146                                }
 2147                            }
 2148                        }
 2149                    //verificar se ha seq iguais e o inversos tambem
 2150                        for(x=0;x<p;x++){
 2151                                    // Inverter peca
 2152                                    int w=strlen(arrayfinalmixpecas[x])-1;
 2153                                    for(k=0;k<(strlen(arrayfinalmixpecas[x]));k++){
 2154                                        invertfinal[w]=arrayfinalmixpecas[x][k];
 2155                                        w--;
 2156                                    }
 2157
 2158                        // Verifica se e igual e se for substitui por X|X
 2159                                for(i=x+1;i<p;i++){
 2160
if((strcmp(arrayfinalmixpecas[i],invertfinal)==0)||(strcmp(arrayfinalmixpecas[i],arrayfi
nalmixpecas[x])==0)){
 2161                                        strcpy(arrayfinalmixpecas[i],"X|X");
 2162                                    }
 2163                                }
 2164                        }
 2165                        // Copia do arrayfinalmixpecas que contem x|x e guarda num array
final(arrayfinalmixpecascompleto) as sequencias possiveis
 2166                         for(z=0;z<p;z++){
 2167                                if(strcmp(arrayfinalmixpecas[z],"X|X")==0){
 2168                                }else{
 2169
strcpy(arrayfinalmixpecascompleto[y],arrayfinalmixpecas[z]);
 2170                                    y++;
 2171                                }
 2172                        }
 2173                    }
 2174                    for(i=0;i<y;i++){
 2175                        printf("%s\n",arrayfinalmixpecascompleto[i]);
 2176                    }
 2177 }
```

void save_bin (INICIARPECA *b*, char *fnome*[])

Guardar num ficheiro binario.

Guarda as peças utlizadas na mão e guarda as sequencias

Parâmetros:

INICIARPECA	b , estrutura do tipo INICIARPECA
char	fnome[], recebe o nome do ficheiro

```
1241                                        {
1247    //nplayers tamnome nome\0 cardid carsuit cardpoints
1248    //2        7  daniel\0      A      E         2
1249    PECA * paux = NULL;
1250    FILE * fp = NULL;
1251    int size = 0, i = 0,sizebar = 0;
1252
1253    if((fp=fopen(fnome,"wb"))!=NULL)
1254    {
1255        //numero de pecas
1256        fwrite(&b.nbar,sizeof(int),1,fp);
1257
1258        //gravar baralho
1259        paux = b.pfirst;
1260
1261        while(paux!=NULL)
1262        {
1263            sizebar=strlen((b.pfirst)->pecastr)+1;
1264            fwrite(&sizebar,sizeof(int),1,fp);
1265            fwrite(paux->pecastr,sizeof(char),sizebar,fp);
1266            paux = paux->pnext;
1267        }
1268         //numero de sequencias
1269        fwrite(&b.nseq,sizeof(int),1,fp);
1270
1271        //gravar sequencias
1272        for(i=0;i<b.nseq;i++){
1273            size=strlen((b.pfirstseq+i)->seqstring)+1;
1274            fwrite(&size,sizeof(int),1,fp);
1275            fwrite((b.pfirstseq+i)->seqstring,sizeof(char),size,fp);
1276        }
1277        fclose(fp);
1278    }
1279 }
```

void save_txt_jogo (INICIARPECA *b*, char *fname*[])

Guardar num ficheiro de texto.

Guarda as peças utlizadas na mão, guarda as peças que sobraram e guarda as sequencias

Parâmetros:

INICIARPECA	b , estrutura do tipo INICIARPECA
char	fname[], recebe o nome do ficheiro

```
1194 {
1201    FILE *fp = NULL;
1202    PECA * pauxbar = NULL;
1203    SOBROU * pauxsob = NULL;
1204    int i=0;
1205
1206    if ((fp = fopen(fname, "w")) == NULL)
1207    {
1208
1209        printf("save txt jogo(): Erro abrir ficheiro %s\n",fname);
1210        return;
1211
1212    }
1213
1214    pauxbar= b.pfirst;
1215
```

```
 1216     fprintf(fp,"Mão do jogador:\n");
 1217     while(pauxbar!=NULL)
 1218     {
 1219         fprintf(fp,"%s\n",pauxbar->pecastr);
 1220         pauxbar = pauxbar->pnext;
 1221     }
 1222
 1223     pauxsob = b.psobrou;
 1224     fprintf(fp,"\nPeças que sobram:\n");
 1225     while(pauxsob!=NULL)
 1226     {
 1227         fprintf(fp,"%s\n",pauxsob->sobroustr);
 1228         pauxsob = pauxsob->proximo;
 1229     }
 1230
 1231     fprintf(fp,"\nSequencias possiveis:\n");
 1232     for(i=0;i<b.nseq;i++)
 1233     {
 1234         fprintf(fp,"%s\n",(b.pfirstseq+i)->seqstring);
 1235     }
 1236     fclose(fp);
 1237
 1238 }
```

void seq_inicial (char *baralho*[][COLSTRING])

Faz sequencias a partir de uma sequencia inicial.

O utilizador escolhe uma sequencia inicial, e é realizado uma verificação para ver se é possivel encaixar essa sequencia com as peças da mão/baralho

Parâmetros:

baralho[][COLSTRING]	Array do tipo char, onde recebe as peças baralhadas da mão do jogador

```
 1594                                            {
 1599         char seqcomecar[3000];
 1600         char arrayseqinicial[3000][150];
 1601         char arrayseqinicialcompleto[3000][150];
 1602         int i=0,j=0,count=0,x=0,y=0,p=1,z=0,contador=0;
 1603         char inversazero[150];
 1604         char inversoum[150];
 1605         char inversodois[150];
 1606         char inversotres[150];
 1607         char inversoquatro[150];
 1608         char invertfinal[150];
 1609
 1610         printf("Insira uma sequencia a comecar:\n");
 1611         scanf("%s",seqcomecar);
 1612
 1613         strcpy(arrayseqinicial[0],seqcomecar);
 1614
 1615                 //Strtok da sequencia inicial
 1616                 char partidosseqcomecar[100][100];
 1617                 char *s=NULL;
 1618                 char seqapartir[100];
 1619                 strcpy(seqapartir,seqcomecar);
 1620                 s = strtok (seqapartir,"-");
 1621                 int k=0;
 1622                 while (s!= NULL)
 1623                 {
 1624                     strcpy(partidosseqcomecar[k++],s);
 1625                     s= strtok (NULL, "-");
 1626                 }
 1627                 //Verificar se as pecas da sequencia inicial sao iguais as pecas do
baralho
 1628                 for(x=0;x<7;x++){
```

```
 1629                       for(y=0;y<k;y++){
 1630                               // Inverter pecas baralho
 1631                               int d=strlen(baralho[x])-1;
 1632                               int h=0;
 1633                               for(h=0;h<(strlen(baralho[x]));h++){
 1634                                       inversazero[d]=baralho[x][h];
 1635                                       d--;
 1636                               }
 1637                               //Verifica se as pecas da seq inicial coincidem
 1638                               int tam=strlen(seqcomecar)-1;
 1639                               int i=0;
 1640                               for(i=0;i<tam;i++){
 1641                                   if(seqcomecar[i]=='-'){
 1642                                       if(seqcomecar[i-1]!=seqcomecar[i+1]){
 1643                                           contador++;
 1644                                       }
 1645                                   }
 1646                               }
 1647
 1648
if((strcmp(baralho[x],partidosseqcomecar[y])==0)||(strcmp(inversazero,partidosseqcomecar
[y])==0)){
 1649                                   count++;
 1650                              }
 1651                       }
 1652                   }
 1653                   if(count>0){
 1654                       printf("As pecas da sequencia que inseriu, ja existem no
baralho!\n");
 1655                       count=0;
 1656                   }else if(contador>0){
 1657                       printf("A sequencia que inseriu esta errada!\n ");
 1658                       contador=0;
 1659                   }else{
 1660
 1661                       // Juntar pecas
 1662                       for(i=0;i<p;i++){
 1663                           int tamlin=strlen(arrayseqinicial[i])-1;
 1664                           for(j=0;j<7;j++){
 1665
 1666                               //Primeira verificação
 1667                               if(arrayseqinicial[i][tamlin]==baralho[j][0]){
 1668                                   int igual=0;
 1669                                   //inverte peca apenas para verificar
 1670                                   int w=strlen(baralho[j])-1;
 1671                                   for(k=0;k<(strlen(baralho[j]));k++){
 1672                                       inversoum[w]=baralho[j][k];
 1673                                       w--;
 1674                                   }
 1675                                   //strtok da peca arrayfinal
 1676                                   char partidos[100][100];
 1677                                   char *palavra=NULL;
 1678                                   char umaseqpeca[100];
 1679
 1680                                   strcpy(umaseqpeca,arrayseqinicial[i]);
 1681                                   palavra = strtok (umaseqpeca,"-");
 1682                                   int s=0;
 1683                                   while (palavra != NULL)
 1684                                   {
 1685                                       strcpy(partidos[s++],palavra);
 1686                                       palavra = strtok (NULL, "-");
 1687                                   }
 1688
 1689                                   //verifica se o partidos ou o inverso é igual a peça
 1690                                   for (x=0;x<s;x++){
 1691
if(strcmp(partidos[x],baralho[j])==0||strcmp(partidos[x],inversoum)==0){
 1692                                                           igual++;
 1693                                                   }
 1694                                   }
 1695                                   if(igual==0){
```

```
 1696                                            //concatena para arrayseqinicial
 1697                                            strcpy(arrayseqinicial[p],
arrayseqinicial[i]);
 1698                                            strcat(arrayseqinicial[p],"-");
 1699                                            strcat(arrayseqinicial[p],
baralho[j]);
 1700                                            p++;
 1701                                        }
 1702
 1703                                    }
 1704                                    //Segunda verificação
 1705                                    if(arrayseqinicial[i][tamlin]==baralho[j][2]){
 1706                                        int igual=0;
 1707                                        //inverte peca
 1708                                        int w=strlen(baralho[j])-1;
 1709                                        for(k=0;k<(strlen(baralho[j]));k++){
 1710                                            inversodois[w]=baralho[j][k];
 1711                                            w--;
 1712                                        }
 1713                                        //strtok da peca arrayseqinicial
 1714                                        char partidos[100][100];
 1715                                        char *palavra=NULL;
 1716                                        char umaseqpeca[100];
 1717
 1718                                        strcpy(umaseqpeca,arrayseqinicial[i]);
 1719                                        palavra = strtok (umaseqpeca,"-");
 1720                                        int s=0;
 1721                                        while (palavra != NULL)
 1722                                        {
 1723                                            strcpy(partidos[s++],palavra);
 1724                                            palavra = strtok (NULL, "-");
 1725                                        }
 1726
 1727                                        //verifica se o partidos ou o inverso é igual a peça
 1728                                        for (x=0;x<s;x++){
 1729
if(strcmp(partidos[x],baralho[j])==0||strcmp(partidos[x],inversodois)==0){
 1730                                                        igual++;
 1731                                                }
 1732                                        }
 1733                                        if(igual==0){
 1734                                            //concatena
 1735                                            strcpy(arrayseqinicial[p],
arrayseqinicial[i]);
 1736                                            strcat(arrayseqinicial[p],"-");
 1737                                            strcat(arrayseqinicial[p], inversodois);
 1738                                            p++;
 1739                                        }
 1740                                    }
 1741                                    //Terceira verificação
 1742                                    if(arrayseqinicial[i][0]==baralho[j][0]){
 1743                                        int igual=0;
 1744
 1745                                        //inverte peca
 1746                                        int w=strlen(baralho[j])-1;
 1747                                        for(k=0;k<(strlen(baralho[j]));k++){
 1748                                            inversotres[w]=baralho[j][k];
 1749                                            w--;
 1750                                        }
 1751                                        //strtok da peca arrayfinal
 1752                                        char partidos[100][100];
 1753                                        char *palavra=NULL;
 1754                                        char umaseqpeca[100];
 1755
 1756                                        strcpy(umaseqpeca,arrayseqinicial[i]);
 1757                                        palavra = strtok (umaseqpeca,"-");
 1758                                        int s=0;
 1759                                        while (palavra != NULL)
 1760                                        {
 1761                                            strcpy(partidos[s++],palavra);
 1762                                            palavra = strtok (NULL, "-");
```

```
 1763                                    }
 1764
 1765                                    //verifica se o partidos ou o inverso é igual a peca
 1766                                    for (x=0;x<s;x++){
 1767
if(strcmp(partidos[x],baralho[j])==0||strcmp(partidos[x],inversotres)==0){
 1768                                                            igual++;
 1769                                                }
 1770                                    }
 1771                                    if(igual==0){
 1772                                        //concatena
 1773                                        strcpy(arrayseqinicial[p], inversotres);
 1774                                        strcat(arrayseqinicial[p],"-");
 1775                                        strcat(arrayseqinicial[p],
arrayseqinicial[i]);
 1776                                        p++;
 1777                                    }
 1778                                }
 1779                                //Quarta verificação
 1780                                if(arrayseqinicial[i][0]==baralho[j][2]){
 1781                                    int igual=0;
 1782
 1783                                    //inverte peca
 1784                                    int w=strlen(baralho[j])-1;
 1785                                    for(k=0;k<(strlen(baralho[j]));k++){
 1786                                        inversoquatro[w]=baralho[j][k];
 1787                                        w--;
 1788                                    }
 1789
 1790                                    //strtok da peca arrayfinal
 1791                                    char partidos[100][100];
 1792                                    char *palavra=NULL;
 1793                                    char umaseqpeca[100];
 1794
 1795                                    strcpy(umaseqpeca,arrayseqinicial[i]);
 1796                                    palavra = strtok (umaseqpeca,"-");
 1797                                    int s=0;
 1798                                    while (palavra != NULL)
 1799                                    {
 1800                                        strcpy(partidos[s++],palavra);
 1801                                        palavra = strtok (NULL, "-");
 1802                                    }
 1803
 1804                                    //verifica se o arraypartidos ou o invero e igual a
peca
 1805                                    for (x=0;x<s;x++){
 1806
if(strcmp(partidos[x],baralho[j])==0||strcmp(partidos[x],inversoquatro)==0){
 1807                                                            igual++;
 1808                                                }
 1809                                    }
 1810
 1811                                    if(igual==0){
 1812                                        //concatena
 1813                                        strcpy(arrayseqinicial[p], baralho[j]);
 1814                                        strcat(arrayseqinicial[p],"-");
 1815                                        strcat(arrayseqinicial[p],
arrayseqinicial[i]);
 1816                                        p++;
 1817                                    }
 1818                                }
 1819                            }
 1820                        }
 1821                    //verificar se ha sequencias iguais e o inversos tambem
 1822                        for(x=0;x<p;x++){
 1823                                    // Inverter peca
 1824                                    int w=strlen(arrayseqinicial[x])-1;
 1825
 1826                                    for(k=0;k<(strlen(arrayseqinicial[x]));k++){
 1827                                        invertfinal[w]=arrayseqinicial[x][k];
 1828                                        w--;
```

```
 1829                                    }
 1830                         // Verifica se e igual e se for substitui por X|X
 1831                                 for(i=x+1;i<p;i++){
 1832
if((strcmp(arrayseqinicial[i],invertfinal)==0)||(strcmp(arrayseqinicial[i],arrayseqinici
al[x])==0)){
 1833                                        strcpy(arrayseqinicial[i],"X|X");
 1834                                    }
 1835                                 }
 1836                         }
 1837                         // Copia do aux que contem x|x e guarda num array
final(arrayseqinicialcompleto) as sequencias das pecas
 1838                          for(z=0;z<p;z++){
 1839                                 if(strcmp(arrayseqinicial[z],"X|X")==0){
 1840                                 }else{
 1841
strcpy(arrayseqinicialcompleto[y],arrayseqinicial[z]);
 1842                                     y++;
 1843                                 }
 1844                         }
 1845                 }
 1846                 for(i=0;i<y;i++){
 1847                     printf("%s\n",arrayseqinicialcompleto[i]);
 1848                 }
 1849 }
```

Índice

INDEX