Projeto lp1

Daniel Mendes 26385
Gonçalo Silva 26329
Docente Rui Moreira
Versão 1.0
Domingo, 27 de Novembro de 2016

Índice dos ficheiros

Lista de ficheiros

Lista de todos os ficheiros com uma breve descrição:

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

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** (char pecas[][**COLSTRING**], char baralho[][**COLSTRING**], char sobrou[][**COLSTRING**], int num)
  Criar baralhos/mãos aleatórios.
- void **rempeca** (char pecas[][**COLSTRING**], char baralho[][**COLSTRING**], int pecastotal, int npecasremover)
  Remover peças de uma ou mais mãos.
- void **addpeca** (char pecas[][**COLSTRING**], char baralho[][**COLSTRING**], char sobrou[][**COLSTRING**], int pecastotal, int npecas)
  Adicionar peças numa mão/mãos.
- int **ordenarseq** (char pecas[][**COLSTRING**], char baralho[][**COLSTRING**], char arrayfinalcompleto[][150], int num)
  Sequencias e ordenar sequencia por ordem decrescente.
- 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 (char *pecas*[][COLSTRING], char *baralho*[][COLSTRING], char *sobrou*[][COLSTRING], 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:

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

NG]	
baralho[][COLSTRING]	Array do tipo char, onde recebe 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
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 i=0 - Variavél usada nas funções seguintes

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

```
  369
{
  379     int aleatorio=0;
  381     int i=0;
  383     int j=0;
  385     int k=0;
  387     char novapeca[4];
  389     char invertnewpeca[4];
  390
  391
  392     aleatorio=pecastotal+npecas;
  393     // 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.
  394     if(aleatorio>0 && aleatorio<6){
  395         aleatorio=6-aleatorio;
  396     }else if(aleatorio>6 && aleatorio<13){
  397             aleatorio=13-aleatorio;
  398     }else if(aleatorio>13 && aleatorio<20){
  399         aleatorio=20-aleatorio;
  400     }else if(aleatorio>20 && aleatorio<27){
  401         aleatorio=27-aleatorio;
  402     }
  403     // 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
  404     for(j=pecastotal;j<=(pecastotal+aleatorio);j++){
  405
  406         strcpy(baralho[j],sobrou[j]);
  407     }
  408     // Imprime a mão/baralho mais as peças aleatorias
  409     for(i=0;i<j;i++){
  410         printf("%s\n",baralho[i]);
  411     }
  412     // Inserir peca (novapeca)
  413     int x=0,fim=0,h=0,cont=0;
  414     for(k=0;k<npecas;k++){
  415             printf("Insira a peca");
  416             scanf("%s",novapeca);
  417
  418             // Inverter novapeca
  419             int d=strlen(novapeca)-1;
  420             for(h=0;h<(strlen(novapeca));h++){
  421                 invertnewpeca[d]=novapeca[h];
  422                 d--;
  423             }
  424             // Verifica se a peca da mão(baralho[]) é igual a novapeca, ou á invertida
  425             for(i=0;i<=(pecastotal+aleatorio+x);i++){
```

```
  426
if((strcmp(baralho[i],novapeca)==0)||(strcmp(baralho[i],invertnewpeca)==0)){
  427                         cont++;
  428                     }
  429             }
  430             // Caso a peça exista, aumenta o cont, e entra no if
  431             if(cont>0){
  432                         printf("Essa peca ja existe!\n");
  433                         k--;
  434                         cont=0;
  435             }else{
  436                 //Caso contrario ele insere a nova peça na ultima posição da mão
(baralho[]) e imprime
  437                         cont=0;
  438                         x++;
  439                         fim=pecastotal+aleatorio+x;
  440                         strcpy(baralho[fim],novapeca);
  441
  442                         for(i=0;i<=fim;i++){
  443                             printf("%s\n",baralho[i]);
  444                         }
  445             }
  446     }
  447 }
```

void baralhos (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:

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]

```
  225
{
  234     srand((unsigned)time(NULL));
  236     int pecastotal=0;
  238     int i=0;
  240     int aleatorio;
  242     int array[28];
  244     int temp;
```

```
  245     // 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)
  246     pecastotal = num * 7;
  247     //Caso o numero de mãos escollhidas seja igual ou inferior a 4, entra neste if
  248     if(num<=4){
  249             // Cria array posicoes das pecas totais
  250             for(i=0;i<28;i++){
  251                 array[i] = i;
  252             }
  253             // Baralha essas 28 pecas e guarda em array, de forma aleatória
  254             for(i=0;i<28;i++){
  255                 temp=array[i];
  256                 aleatorio = rand() % 28;
  257                 array[i]=array[aleatorio];
  258                 array[aleatorio]=temp;
  259             }
  260             // Guarda num array baralho[] as peças aleatórias até ao numero de pecas
total definida pelo jogador e imprime
  261             printf("MAO:\n");
  262             for(i=0;i<pecastotal;i++){
  263                 strcpy(baralho[i],pecas[array[i]]);
  264                 printf("%s\n",baralho[i]);
  265             }
  266             // Guarda num array sobrou[] as peças aleatórias desde o numero de pecas
total definida pelo jogador até as 27 possiveis e imprime
  267             printf("SOBROU:\n");
  268             for(i=pecastotal;i<28;i++){
  269                 strcpy(sobrou[i],pecas[array[i]]);
  270                 printf("%s\n",sobrou[i]);
  271             }
  272     }else if(num>4){
  273         printf("Nao pode escolher mais que 4 baralhos!\n");
  274     }
  275 }
```

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

```
  144                                     {
  149     strcpy(pecas[0],"0|0");
  150     strcpy(pecas[1],"0|1");
  151     strcpy(pecas[2],"0|2");
  152     strcpy(pecas[3],"0|3");
  153     strcpy(pecas[4],"0|4");
  154     strcpy(pecas[5],"0|5");
  155     strcpy(pecas[6],"0|6");
  156     strcpy(pecas[7],"1|1");
  157     strcpy(pecas[8],"1|2");
  158     strcpy(pecas[9],"1|3");
  159     strcpy(pecas[10],"1|4");
  160     strcpy(pecas[11],"1|5");
  161     strcpy(pecas[12],"1|6");
  162     strcpy(pecas[13],"2|2");
  163     strcpy(pecas[14],"2|3");
  164     strcpy(pecas[15],"2|4");
  165     strcpy(pecas[16],"2|5");
```

```
166     strcpy(pecas[17],"2|6");
167     strcpy(pecas[18],"3|3");
168     strcpy(pecas[19],"3|4");
169     strcpy(pecas[20],"3|5");
170     strcpy(pecas[21],"3|6");
171     strcpy(pecas[22],"4|4");
172     strcpy(pecas[23],"4|5");
173     strcpy(pecas[24],"4|6");
174     strcpy(pecas[25],"5|5");
175     strcpy(pecas[26],"5|6");
176     strcpy(pecas[27],"6|6");
177
179 /*
180     strcpy(pecas[0],"2|5");
181     strcpy(pecas[1],"3|3");
182     strcpy(pecas[2],"2|2");
183     strcpy(pecas[3],"1|1");
184     strcpy(pecas[4],"4|4");
185     strcpy(pecas[5],"5|5");
186     strcpy(pecas[6],"6|6");
187     strcpy(pecas[7],"1|2");
188     strcpy(pecas[8],"6|3");
189     strcpy(pecas[9],"1|4");
190     strcpy(pecas[10],"1|5");
191     strcpy(pecas[11],"0|6");
192     strcpy(pecas[12],"2|1");
193     strcpy(pecas[13],"0|3");
194 */
195
196 }
```

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.

```
198                                             {
203     int i = 0;
204     for(i=0;i<28;i++){
205         printf("%s\n",pecas[i]);
206     }
207 }
```

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.

```
210                                                                 {
216     int i=0;
217     for(i=0;i<28;i++){
```

```
  218         pecasint[i][0]=atoi(pecas[i]);
  219         pecasint[i][1]=atoi(&pecas[i][2]);
  220         printf("%d|%d \n", pecasint[i][0],pecasint[i][1]);
  221     }
  222 }
```

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

Função main_projectolp1primparte.

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

int pecasint[LINSTRING][COL] - Array de inteiros 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     int pecasint[LINSTRING][COL];
   50     char baralho[LINSTRING][COLSTRING];
   52     char sobrou[LINSTRING][COLSTRING];
   54     char arrayfinalcompleto[3000][150];
   56     int opc=0;
   58     int num=0;
   60     int npecasremover=0;
   62     int pecastotal=0;
   64     int npecas=0;
   66     int y=0;
   68     criarpecas(pecas);
   70     bool sair = false;
   72     char escolha;
   73     //Menu com ciclo. O utilizador escolhe a opção que pretende e através do switch
retorna para a função pretendida
   74     do{
   75     printf("*************** JOGO DO DOMINO ***************\n\n");
   76     printf("Escolha uma opcao\n");
   77     printf("1 ----> Listar Pecas Sring\n");
   78     printf("2 ----> Criar Mao\n");
   79     printf("3 ----> Remover Pecas\n");
   80     printf("4 ----> Adicionar Pecas\n");
```

```
 81     printf("5 ----> Listar Pecas Int\n");
 82     printf("6 ----> Criar e ordenar sequencias\n");
 83     printf("7 ----> Procurar padrao\n");
 84     printf("8 ----> Criar sequencias com sequencia inicial\n");
 85     printf("9 ----> Criar sequencias com maos alternadas\n");
 86
 87     printf("Opcao: ");
 88     scanf("%d", &opc);
 89
 90     switch(opc)
 91             {
 92             case 1:
 93                 imprimirpecas(pecas);
 94                 break;
 95             case 2:
 96                 printf("Insira o numero de mãos a jogar:\n");
 97                 scanf("%d",&num);
 98                 baralhos(pecas,baralho,sobrou,num);
 99                 break;
100             case 3:
101                 printf("Quantas pecas pretende remover:\n");
102                 scanf("%d",&npecasremover);
103                 pecastotal=num*7;
104                 rempeca(pecas,baralho,pecastotal,npecasremover);
105                 break;
106             case 4:
107                 printf("Quantas pecas pretende adicionar:\n");
108                 scanf("%d",&npecas);
109                 pecastotal=num*7;
110                 addpeca(pecas,baralho,sobrou,pecastotal,npecas);
111                 break;
112             case 5:
113                 imprimirpecasint(pecas,pecasint);
114                 break;
115             case 6:
116                 y=ordenarseq(pecas,baralho,arrayfinalcompleto,num);
117                 break;
118             case 7:
119                 procurar_padrao(arrayfinalcompleto,y);
120                 break;
121             case 8:
122                 seq_inicial(baralho);
123                 break;
124             case 9:
125                 retirar_mao_jogadores(baralho,num);
126                 break;
127
128
129             default:
130             printf("Escolha invalida!\n\n");
131             }
132             printf("Pretende sair? S --->sim \n");
133             scanf("%s",&escolha);
134             if(escolha=='S'||escolha=='s'){
135                 sair=true;
136             }else{
137                 sair=false;
138             }
139     }while(sair==false);
140     return 0;
141 }
```

int ordenarseq (char *pecas*[][COLSTRING], char *baralho*[][COLSTRING], char *arrayfinalcompleto*[][150], 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 arrayfinalcompleto até o tamanho do arrayfinalcompleto (p) termine.Tudo isto com verificações a ver se há repetidas, ou invertidas.

Parâmetros:

pecas[][COLSTRING]	Array do tipo char onde recebe as peças totais do jogo.
baralho[][COLSTRING]	Array do tipo char, onde recebe as peças baralhadas da mão do jogador
arrayfinalcompleto[][150]	Array do tipo char, onde são guardadas as sequencias totais da mão do jogador
num	Valor do tipo int, utilizado para definir o numero de mãos a ser usado pelo jogador

Retorna:

y, tamanho do array total das sequencias (arrayfinalcompleto)

char seqduaspecas[3000][150] - Usado para guardar sequencias de duas peças

char arrayfinal[3000][150] - Usado para guardar sequencias de duas peças

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

```
  450
{
  459 int i=0,j=0,h=0,z=0,count=0,p=0;
  460 int x=0,k=0,y=0;
  461 char aux[3000][150];
  463 char seqduaspecas[3000][150];
  465 char arrayfinal[3000][150];
  466 char tempd[150][150];
  468 char invertnewpeca[150];
  470 char invertnewpecaaux[150];
  472 char invert[150];
  474 char inversopeca[150];
  476 char invertfinal[500];
  477
  478 num=num*7;
  479 //Juntar 2 pecas
  480     for(i=0;i<num;i++){
  481         for(j=0;j<num;j++){
  482             if(baralho[i][2]==baralho[j+1][0]){
  483                 //concatena
  484                 strcpy(aux[count], baralho[i]);
  485                 strcat(aux[count],"-");
  486                 strcat(aux[count], baralho[j+1]);
  487                 count++;
  488             }
  489             if(baralho[i][2]==baralho[j+1][2]){
  490                 // Inverter peca
  491                 int d=strlen(baralho[j+1])-1;
  492                 for(h=0;h<(strlen(baralho[j+1]));h++){
  493                     invertnewpeca[d]=baralho[j+1][h];
  494                     d--;
  495                 }
  496                 // concatena
  497                 strcpy(aux[count], baralho[i]);
  498                 strcat(aux[count],"-");
  499                 strcat(aux[count], invertnewpeca);
  500                 count++;
```

```
  501             }
  502             if(baralho[i][0]==baralho[j+1][2]){
  503                 // concatena
  504                 strcpy(aux[count], baralho[j+1]);
  505                 strcat(aux[count],"-");
  506                 strcat(aux[count], baralho[i]);
  507                 count++;
  508             }
  509             if(baralho[i][0]==baralho[j+1][0]){
  510                 // Inverter peca
  511                 int d=strlen(baralho[j+1])-1;
  512                 for(h=0;h<(strlen(baralho[j+1]));h++){
  513                     invertnewpeca[d]=baralho[j+1][h];
  514                     d--;
  515                 }
  516                 // concatena
  517                 strcpy(aux[count], invertnewpeca);
  518                 strcat(aux[count],"-");
  519                 strcat(aux[count], baralho[i]);
  520                 count++;
  521             }
  522         }
  523     }
  524     //Verificar se ha sequencias iguais e o inversos tambem
  525     for(x=0;x<count;x++){
  526                 // Inverter peca
  527                 int w=strlen(aux[x])-1;
  528                 for(k=0;k<(strlen(aux[x]));k++){
  529                     invertnewpecaaux[w]=aux[x][k];
  530                     w--;
  531                 }
  532     // Verifica se é igual e se for substitui por X|X
  533             for(i=x+1;i<count;i++){
  534
if((strcmp(aux[i],invertnewpecaaux)==0)||(strcmp(aux[i],aux[x])==0)||((aux[i][0]==aux[i]
[6])&&(aux[i][2]==aux[i][4]))){
  535                     strcpy(aux[i],"X|X");
  536                 }
  537             }
  538     }
  539     // Copia do aux que contem x|x e guarda num array final(seqduaspecas) e guarda num
array final (arrayfinal) as sequencias de duas pecas
  540      for(z=0;z<count;z++){
  541             if(strcmp(aux[z],"X|X")==0){
  542             }else{
  543                 strcpy(seqduaspecas[p],aux[z]);
  544                 strcpy(arrayfinal[p],aux[z]);
  545                 p++;
  546             }
  547     }
  548
  549 // Juntar 3 ou mais pecas
  550     for(i=0;i<p;i++){
  551         int tamlin=strlen(arrayfinal[i])-1;
  552         for(j=0;j<num;j++){
  553
  554             //Primeira verificação
  555             if(arrayfinal[i][tamlin]==baralho[j][0]){
  556                 int igual=0;
  557                 //inverte peca apenas para verificar
  558                 int w=strlen(baralho[j])-1;
  559                 for(k=0;k<(strlen(baralho[j]));k++){
  560                     inversopeca[w]=baralho[j][k];
  561                     w--;
  562                 }
  563                 //strtok da peca arrayfinal
  564                 char partidos[100][100];
  565                 char *palavra=NULL;
  566                 char umaseqpeca[100];
  567
  568                 strcpy(umaseqpeca,arrayfinal[i]);
```

```
  569                 palavra = strtok (umaseqpeca,"-");
  570                 int s=0;
  571                 while (palavra != NULL)
  572                 {
  573                     strcpy(partidos[s++],palavra);
  574                     palavra = strtok (NULL, "-");
  575                 }
  576 
  577                 //verifica se o arraypartidos ou o inverdo é igual a peca
  578                 for (x=0;x<s;x++){
  579 
if(strcmp(partidos[x],baralho[j])==0||strcmp(partidos[x],inversopeca)==0){
  580                                 igual++;
  581                         }
  582                 }
  583                 if(igual==0){
  584                             //concatena para arrayfinal
  585                             strcpy(arrayfinal[p], arrayfinal[i]);
  586                             strcat(arrayfinal[p],"-");
  587                             strcat(arrayfinal[p], baralho[j]);
  588                             p++;
  589                 }
  590             }
  591             //Segunda verificação
  592             if(arrayfinal[i][tamlin]==baralho[j][2]){
  593                 int igual=0;
  594                 //inverte peca
  595                 int w=strlen(baralho[j])-1;
  596 
  597                 for(k=0;k<(strlen(baralho[j]));k++){
  598                     invert[w]=baralho[j][k];
  599                     w--;
  600                 }
  601                 //strtok da peca arrayfinal
  602                 char partidos[100][100];
  603                 char *palavra=NULL;
  604                 char umaseqpeca[100];
  605 
  606                 strcpy(umaseqpeca,arrayfinal[i]);
  607                 palavra = strtok (umaseqpeca,"-");
  608                 int s=0;
  609                 while (palavra != NULL)
  610                 {
  611                     strcpy(partidos[s++],palavra);
  612                     palavra = strtok (NULL, "-");
  613                 }
  614 
  615                 //verifica se o arraypartidos ou o inverso é igual a peça
  616                 for (x=0;x<s;x++){
  617 
if(strcmp(partidos[x],baralho[j])==0||strcmp(partidos[x],invert)==0){
  618                                 igual++;
  619                         }
  620                 }
  621                 if(igual==0){
  622                     //concatena
  623                     strcpy(arrayfinal[p], arrayfinal[i]);
  624                     strcat(arrayfinal[p],"-");
  625                     strcat(arrayfinal[p], invert);
  626                     p++;
  627                 }
  628             }
  629 
  630             //Terceira verificação
  631             if(arrayfinal[i][0]==baralho[j][0]){
  632                 int igual=0;
  633 
  634                 //inverte peca
  635                 int w=strlen(baralho[j])-1;
  636                 for(k=0;k<(strlen(baralho[j]));k++){
  637                     invert[w]=baralho[j][k];
```

```
  638                     w--;
  639                 }
  640 
  641                 //strtok da peca arrayfinal
  642                 char partidos[100][100];
  643                 char *palavra=NULL;
  644                 char umaseqpeca[100];
  645                 strcpy(umaseqpeca,arrayfinal[i]);
  646                 palavra = strtok (umaseqpeca,"-");
  647                 int s=0;
  648                 while (palavra != NULL)
  649                 {
  650                     strcpy(partidos[s++],palavra);
  651                     palavra = strtok (NULL, "-");
  652                 }
  653 
  654                 //verifica se o arraypartidos ou o inverso é igual a peça
  655                 for (x=0;x<s;x++){
  656 
if(strcmp(partidos[x],baralho[j])==0||strcmp(partidos[x],invert)==0){
  657                                 igual++;
  658                         }
  659                 }
  660                 if(igual==0){
  661                     //concatena
  662                     strcpy(arrayfinal[p], invert);
  663                     strcat(arrayfinal[p],"-");
  664                     strcat(arrayfinal[p], arrayfinal[i]);
  665                     p++;
  666                 }
  667             }
  668             //Quarta verificação
  669             if(arrayfinal[i][0]==baralho[j][2]){
  670                 int igual=0;
  671 
  672                 //inverte peca
  673                 int w=strlen(baralho[j])-1;
  674                 for(k=0;k<(strlen(baralho[j]));k++){
  675                     invert[w]=baralho[j][k];
  676                     w--;
  677                 }
  678                 //strtok da peca arrayfinal
  679                 char partidos[100][100];
  680                 char *palavra=NULL;
  681                 char umaseqpeca[100];
  682 
  683                 strcpy(umaseqpeca,arrayfinal[i]);
  684                 palavra = strtok (umaseqpeca,"-");
  685                 int s=0;
  686                 while (palavra != NULL)
  687                 {
  688                     strcpy(partidos[s++],palavra);
  689                     palavra = strtok (NULL, "-");
  690                 }
  691 
  692                 //verifica se o arraypartidos ou o inverso é igual a peça
  693                 for (x=0;x<s;x++){
  694 
if(strcmp(partidos[x],baralho[j])==0||strcmp(partidos[x],invert)==0){
  695                                 igual++;
  696                         }
  697                 }
  698                 if(igual==0){
  699                     //concatena
  700                     strcpy(arrayfinal[p], baralho[j]);
  701                     strcat(arrayfinal[p],"-");
  702                     strcat(arrayfinal[p], arrayfinal[i]);
  703                     p++;
  704                 }
  705             }
  706         }
```

```
  707     }
  708     //verificar se ha sequencias iguais e inversos tambem
  709     for(x=0;x<p;x++){
  710                 // Inverter peca
  711                 int w=strlen(arrayfinal[x])-1;
  712                 for(k=0;k<(strlen(arrayfinal[x]));k++){
  713                     invertfinal[w]=arrayfinal[x][k];
  714                     w--;
  715                 }
  716
  717             // Verifica se e igual e se for substitui por X|X
  718             for(i=x+1;i<p;i++){
  719
if((strcmp(arrayfinal[i],invertfinal)==0)||(strcmp(arrayfinal[i],arrayfinal[x])==0)){
  720                     strcpy(arrayfinal[i],"X|X");
  721                 }
  722             }
  723     }
  724     // Copia do aux que contem x|x e guarda num array final(arrayfinalcompleto) as
sequencias de tres ou mais pecas */
  725      for(z=0;z<p;z++){
  726             if(strcmp(arrayfinal[z],"X|X")==0){
  727             }else{
  728                 strcpy(arrayfinalcompleto[y],arrayfinal[z]);
  729                 y++;
  730             }
  731     }
  732
  733         // Ordenar de forma decrescente
  734         for( i=0; i<y; i++ ){
  735                   for( j=(i+1); j<y; j++ ){
  736
if((strlen(arrayfinalcompleto[i]))<(strlen(arrayfinalcompleto[j]))){
  737                           strcpy(tempd[0],arrayfinalcompleto[i]);
  738                           strcpy(arrayfinalcompleto[i],arrayfinalcompleto[j]);
  739                           strcpy(arrayfinalcompleto[j],tempd[0]);
  740                       }
  741                   }
  742         }
  743
  744         for(k=0;k<y;k++){
  745               printf("%s\n",arrayfinalcompleto[k]);
  746         }
  747     return y;
  748 }
```

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)

```
  750                                                                 {
  756     //Guarda todas as posicoes num array para depois baralhar esses i's
  757     int z=0;
  758     int auxp[50];
  759     int ptemp=0;
```

```
  760     int aleatorio=0;
  761     int b=0,i=0;
  762     char strproc[100];
  763     int tamarraycompleto=y;
  764 
  765     for(z=0;z<tamarraycompleto;z++){
  766         auxp[z]=z;
  767     }
  768 
  769     //baralho esse numeros do auxp
  770     for(b=0;b<tamarraycompleto;b++){
  771         ptemp=auxp[b];
  772         aleatorio=rand()%(tamarraycompleto-1);
  773         auxp[b]=auxp[aleatorio];
  774         auxp[aleatorio]=ptemp;
  775     }
  776     //vai buscar uma sequencia do arrayfinalcompleto dependendo do num de sequencias
que o utilizador quer
  777         strcpy(strproc,arrayfinalcompleto[auxp[0]]);
  778         printf("A sequencia escolhida foi:\n %s \n",strproc);
  779 
  780 
  781     char sub[50];
  782     char str2[100][100];
  783     //declaracoes partir arrayfinalcompleto
  784     char partidosstr1[100][100];
  785     char *s1=NULL;
  786     char str1apartir[100];
  787     int u=0;
  788     //declaracoes partir subtring
  789     char partidosstr2[100][100];
  790     char *s2=NULL;
  791     char str2apartir[100];
  792     int l=0;
  793     //declaracoes partir subtring inversa
  794     char partidosstr2inv[100][100];
  795     char *s2inv=NULL;
  796     char str2invapartir[100];
  797     int inv=0;
  798     //declaracao inverter subtring
  799     char invertsub[100];
  800     //declaracoes comparar a ver se sao iguais
  801     int x=0,g=0,count=0,r=0,pos=0;;
  802     int v[100];
  803 
  804 
  805     do{
  806 
  807     printf("Insira uma subsequencia para procurar:\n");
  808     scanf("%s",sub);
  809 
  810                 //Strtok do arrayfinalcompleto
  811                 strcpy(str1apartir,strproc);
  812                 s1 = strtok (str1apartir,"-");
  813 
  814                 while (s1!= NULL)
  815                 {
  816                     strcpy(partidosstr1[u++],s1);
  817                     s1= strtok (NULL, "-");
  818                 }
  819 
  820                 //Strtok da substring escolhida
  821                 strcpy(str2[0],sub);
  822                 strcpy(str2apartir,str2[0]);
  823                 s2 = strtok (str2apartir,"-");
  824                 while (s2!= NULL)
  825                 {
  826                     strcpy(partidosstr2[l++],s2);
  827                     s2= strtok (NULL, "-");
  828                 }
  829 
```

```
  830                 // Inverter substring
  831                     int w=strlen(sub)-1;
  832                     int k=0;
  833                     for(k=0;k<(strlen(sub));k++){
  834                         invertsub[w]=sub[k];
  835                         w--;
  836                     }
  837
  838                 //Strtok da substring invertida
  839
  840                 strcpy(str2invapartir,invertsub);
  841                 s2inv = strtok (str2invapartir,"-");
  842                 while (s2inv!= NULL)
  843                 {
  844                     strcpy(partidosstr2inv[inv++],s2inv);
  845                     s2inv= strtok (NULL, "-");
  846                 }
  847
  848                 //Verifica se a invertida é igual a string original (partidosstr1)
  849                 for(x=0;x<u;x++){
  850                     if((strcmp(partidosstr1[x],partidosstr2inv[g])==0)){
  851                         count++;
  852                         g++;
  853                       if(count==g && count==l){
  854                         //pos- é o valor da peca final menos o tam da sub -1, para
retornar a primeira posicao onde encontrou
  855                         pos=(x-(l-1));
  856                         v[r]=pos;
  857                         r++;
  858                         count=0;
  859                         g=0;
  860                         pos=0;
  861                         strcpy(sub,invertsub);
  862                       }
  863                     }else{
  864                         count=0;
  865                         g=0;
  866                         pos=0;
  867                     }
  868                 }
  869                 //Verifica se a string original partida (partidosstr1) é igual a
substring inserida partida (partidosstr2)
  870                 for(x=0;x<u;x++){
  871                     if(strcmp(partidosstr1[x],partidosstr2[g])==0){
  872                         count++;
  873                         g++;
  874                       if(count==g && count==l){
  875                         //pos- é o valor da peca final menos o tam da sub -1, para
retornar a primeira posicao onde encontrou
  876                         pos=(x-(l-1));
  877                         v[r]=pos;
  878                         r++;
  879                         count=0;
  880                         g=0;
  881                         pos=0;
  882                       }
  883                     }else{
  884                         count=0;
  885                         g=0;
  886                         pos=0;
  887                     }
  888                 }
  889                 if(r==0){
  890                     printf("Nao foi encontado a subsequencia %s, na sequencia %s!\n
",sub,strproc);
  891                     u=0;
  892                     l=0;
  893
  894                 }else{
  895                     for(i=0;i<r;i++){
  896                         printf(" Posicao: %d\n",v[i]);
```

```
  897                     }
  898                 }
  899     }while(r==0);
  900                 //Substituir Padrão
  901                 int opcao=0,j=0,q=0;
  902                 char aux[100][100];
  903                 char addseqpeca[50];
  904                 printf("Pretende substituir essa sequencia por outra peca/seq:\n
sim->1 , nao->0\n");
  905                 scanf("%d",&opcao);
  906                 if(opcao==1){
  907                     int continua=1;
  908                     while(continua>0){
  909                         continua=0;
  910                         printf("Insira a seq/peca que deseja substituir");
  911                         scanf("%s",addseqpeca);
  912 
  913                         //Strtok da peça adicionada (addseqpeca)
  914                         char partidasaddseqpeca[100][100];
  915                         char *s3=NULL;
  916                         char addseqpecaapartir[100];
  917                         strcpy(addseqpecaapartir,addseqpeca);
  918                         s3 = strtok (addseqpecaapartir,"-");
  919                         int b=0;
  920 
  921                         while (s3!= NULL)
  922                         {
  923                             strcpy(partidasaddseqpeca[b++],s3);
  924                             s3= strtok (NULL, "-");
  925                         }
  926 
  927                         //Verifica se essa seq adicionada é possivel inserir
  928                         char invert[4];
  929                         int cont=0,n=0,m=0;
  930                         for(n=0;n<u;n++){
  931                             for(m=0;m<b;m++){
  932                                     // Inverter addseqpeca
  933                                     int d=strlen(partidasaddseqpeca[m])-1;
  934                                     int h=0;
  935 
for(h=0;h<(strlen(partidasaddseqpeca[m]));h++){
  936                                         invert[d]=partidasaddseqpeca[m][h];
  937                                         d--;
  938                                     }
  939                                     //verifica se a peça adicionada existe na string
original , ou se a peça adicionada invertida existe na string original
  940 
if((strcmp(partidasaddseqpeca[m],partidosstr1[n])==0)||(strcmp(partidosstr1[n],invert)==
0)){
  941                                         cont++;
  942                                     }
  943                             }
  944                         }
  945                         if(cont>0){
  946 
  947                                     printf("Essa peca ja existe!\n");
  948                                     continua++;
  949                                     cont=0;
  950                         }else{
  951                                     //verifica se o padrao encontrado está no meio,
á frente ou atrás.
  952 
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]))){
  953                                         cont=0;
  954                                         //Adiciona no aux as primeiras pecas, as
novas e depois o resto
  955                                             for(i=0;i<v[0];i++){
  956                                                 strcpy(aux[q],partidosstr1[i]);
```

```
  957                                                         q++;
  958                                                 }
  959                                                 int c=0;
  960                                                 for(c=0;c<b;c++){
  961
strcat(aux[q],partidasaddseqpeca[c]);
  962                                                     q++;
  963                                                 }
  964                                                 for(j=(v[0]+l);j<u;j++){
  965
strcat(aux[q],partidosstr1[j]);
  966                                                         q++;
  967                                                 }
  968                                 }else{
  969                                     printf("Pecas inseridas nao coincidem!\n");
  970                                     continua++;
  971                                 }
  972                             }
  973                             for(m=0;m<q;m++){
  974                                 printf("%s",aux[m]);
  975                                 if(m!=(q-1)){
  976                                     printf("-");
  977                                 }
  978                             }
  979                             printf("\n");
  980                 }
  981                 }else if(opcao==0){
  982
  983                 }else{
  984                     printf("opcao errada!");
  985                 }
  986 }
```

void rempeca (char *pecas*[][COLSTRING], char *baralho*[][COLSTRING], 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:

pecas[][COLSTRING]	Array do tipo char onde recebe as peças totais do jogo.
baralho[][COLSTRING]	Array do tipo char, onde recebe as peças baralhadas da mão do jogador
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 i=0 - Variavél usada nas funções seguintes

int j=0 - Variavél usada nas funções seguintes

int k=0 - Variavél usada nas funções seguintes

int y=0 - Variavél usada nas funções seguintes

int h=0 - Variavél usada nas funções seguintes

int aux[npecasremover] - Usado para guardar as posições onde as peças foram removidas para depois usar para adicionar as peças novas nessa posição

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

```
  277
{
  286     char pecaremover[4];
  288     int i=0;
  290     int j=0;
  292     int k=0;
  294     int y=0;
  296     int h=0;
  298     int aux[npecasremover];
  300     char newpeca[4];
  302     char invertnewpeca[4];
  303     //Caso o numero de peças a remover seja maior que as peças totais, entra neste if
  304     if(npecasremover>pecastotal){
  305         printf("Numero de pecas a remover e superior ao baralho!\n");
  306     // Caso o numero de peças a remover seja menor que as peças totais, entra neste
if
  307     }else if(npecasremover<pecastotal){
  308         int count=0;
  309         // Insere peca a remover
  310         for(j=0;j<npecasremover;j++){
  311             printf("Insira a peca a remover:");
  312             scanf("%s",pecaremover);
  313             // Verifica se a peça do baralho/mão é igual á peça a remover, se for
substitui por X|X
  314             for(i=0;i<pecastotal;i++){
  315                 if(strcmp(baralho[i],pecaremover)==0){
  316                     strcpy(baralho[i],"X|X");
  317                     aux[k]=i;
  318                     k++;
  319                     count++;
  320                 }
  321             }
  322             // Caso a peça do baralho/mão não seja igual á peça a remover, count mantém
a zero, e entra neste if
  323             if(count==0){
  324                 printf("Essa peca nao existe, insira outra:\n");
  325                     j--;
  326             }
  327             count=0;
  328         }
  329         //Imprime mão já com as peças removidas (com pecas X|X)
  330         for(i=0;i<pecastotal;i++){
  331                 printf("%s\n",baralho[i]);
  332         }
  333         //No lugar onde foram removidas as peças, insere novas peças
  334         for(j=0;j<npecasremover;j++){
  335             printf("Insira uma nova peca:");
  336             scanf("%s",newpeca);
  337
  338             // Inverter peça inserida
  339             int d=strlen(newpeca)-1;
  340             for(h=0;h<(strlen(newpeca));h++){
  341                 invertnewpeca[d]=newpeca[h];
  342                 d--;
  343             }
  344             // Verifica se as peças da mão são iguais ás novas peças inseridas e ao
seu inverso
  345             for(i=0;i<pecastotal;i++){
  346
if((strcmp(baralho[i],newpeca)==0)||(strcmp(baralho[i],invertnewpeca)==0)){
  347                         count++;
  348                 }
  349             }
  350             // Caso sejam as peças verificadas iguais, entra neste if
  351             if(count>0){
  352                 printf("Essa peca existe!\n");
  353                 j--;
  354                 count=0;
```

```
  355             }else{
  356                 // Caso as peças verificadas não sejam iguais, ele copia a nova peça
inserida para a posição do baralho/mão onde foi removida
  357                 strcpy(baralho[aux[y]],newpeca);
  358                 y++;
  359             }
  360         }
  361         // Imprime as peças da mão do jogador
  362         for(i=0;i<pecastotal;i++){
  363                 printf("%s\n",baralho[i]);
  364         }
  365     }
  366 }
```

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

```
 1246                                                                {
 1252      char mixpecas[3000][150];
 1253
 1254     //Ver quantos baralhos sao, e guardar num array mix as pecas dos jogadores
alternadamente
 1255     if(num==2){
 1256         int i=0;
 1257         int j=7;
 1258         int m=0;
 1259
 1260         while(i<7 || j<14){
 1261             if(i<7){
 1262                 strcpy(mixpecas[m++],baralho[i]);
 1263             }
 1264             if(j<14){
 1265                 strcpy(mixpecas[m++],baralho[j]);
 1266             }
 1267             i++;
 1268             j++;
 1269         }
 1270     }
 1271     if(num==3){
 1272         int i=0;
 1273         int j=7;
 1274         int k=14;
 1275         int m=0;
 1276
 1277         while(i<7 || j<14 || k<21 ){
 1278             if(i<7){
 1279                 strcpy(mixpecas[m++],baralho[i]);
 1280             }
 1281             if(j<14){
 1282                 strcpy(mixpecas[m++],baralho[j]);
 1283             }
 1284             if(k<21){
 1285                 strcpy(mixpecas[m++],baralho[k]);
 1286             }
 1287             i++;
 1288             j++;
```

```
1289            k++;
1290        }
1291    }
1292
1293    if(num==4){
1294        int i=0;
1295        int j=7;
1296        int k=14;
1297        int s=21;
1298        int m=0;
1299
1300        while(i<7 || j<14 || k<21 || s<28){
1301            if(i<7){
1302                strcpy(mixpecas[m++],baralho[i]);
1303            }
1304            if(j<14){
1305                strcpy(mixpecas[m++],baralho[j]);
1306            }
1307            if(k<21){
1308                strcpy(mixpecas[m++],baralho[k]);
1309            }
1310            if(s<28){
1311                strcpy(mixpecas[m++],baralho[s]);
1312            }
1313            i++;
1314            j++;
1315            k++;
1316            s++;
1317        }
1318    }
1319
1320    printf("ARRAY MIX:\n");
1321    int v=0;
1322    int pecasmix=num*7;
1323    for(v=0;v<pecasmix;v++){
1324        printf("%s\n",mixpecas[v]);
1325    }
1326
1327    //Comeca com uma sequencia inicial e vai metendo peca a peca do mixpecas
1328        char seqcomecar[3000];
1329        char arrayfinalmixpecas[3000][150];
1330        char arrayfinalmixpecascompleto[3000][150];
1331        int i=0,j=0,count=0,x=0,y=0,p=1,z=0,contador=0;
1332        char inversazero[150];
1333        char inversoum[150];
1334        char inversodois[150];
1335        char inversotres[150];
1336        char inversoquatro[150];
1337        char invertfinal[150];
1338
1339        printf("Insira uma sequencia a comecar:\n");
1340        scanf("%s",seqcomecar);
1341
1342        strcpy(arrayfinalmixpecas[0],seqcomecar);
1343
1344                //Strtok da sequencia inicial
1345                char partidosseqcomecar[100][100];
1346                char *s=NULL;
1347                char seqapartir[100];
1348
1349                strcpy(seqapartir,seqcomecar);
1350                s = strtok (seqapartir,"-");
1351                int k=0;
1352
1353                while (s!= NULL)
1354                {
1355                    strcpy(partidosseqcomecar[k++],s);
1356                    s= strtok (NULL, "-");
1357                }
1358
```

```
 1359                 //Verificar se as pecas da sequencia inicial sao iguais as pecas do
baralho mix
 1360                 for(x=0;x<pecasmix;x++){
 1361                     for(y=0;y<k;y++){
 1362                             // Inverter pecas baralho
 1363                             int d=strlen(mixpecas[x])-1;
 1364                             int h=0;
 1365                             for(h=0;h<(strlen(mixpecas[x]));h++){
 1366                                     inversazero[d]=mixpecas[x][h];
 1367                                     d--;
 1368                             }
 1369                             //Verifica se as pecas da seq inicial coincidem
 1370                             int tam=strlen(seqcomecar)-1;
 1371                             int i=0;
 1372                             for(i=0;i<tam;i++){
 1373                                 if(seqcomecar[i]=='-'){
 1374                                     if(seqcomecar[i-1]!=seqcomecar[i+1]){
 1375                                         contador++;
 1376                                     }
 1377                                 }
 1378                             }
 1379
if((strcmp(mixpecas[x],partidosseqcomecar[y])==0)||(strcmp(inversazero,partidosseqcomeca
r[y])==0)){
 1380                             count++;
 1381                         }
 1382                     }
 1383                 }
 1384                 if(count>0){
 1385                     printf("As pecas da sequencia que inseriu, ja existem no
baralho!\n");
 1386                     count=0;
 1387                 }else if(contador>0){
 1388                     printf("A sequencia que inseriu esta errada!\n ");
 1389                     contador=0;
 1390                 }else{
 1391                     //Juntar pecas
 1392                     for(i=0;i<p;i++){
 1393                         int tamlin=strlen(arrayfinalmixpecas[i])-1;
 1394                         for(j=0;j<pecasmix;j++){
 1395                             //Primeira verificação
 1396                             if(arrayfinalmixpecas[i][tamlin]==mixpecas[j][0]){
 1397                                 int igual=0;
 1398                                 //inverte peca apenas para verificar
 1399                                 int w=strlen(mixpecas[j])-1;
 1400                                 for(k=0;k<(strlen(mixpecas[j]));k++){
 1401                                     inversoum[w]=mixpecas[j][k];
 1402                                     w--;
 1403                                 }
 1404                                 //strtok da peca arrayfinalmixpecas
 1405                                 char partidos[100][100];
 1406                                 char *palavra=NULL;
 1407                                 char umaseqpeca[100];
 1408
 1409                                 strcpy(umaseqpeca,arrayfinalmixpecas[i]);
 1410                                 palavra = strtok (umaseqpeca,"-");
 1411                                 int s=0;
 1412                                 while (palavra != NULL)
 1413                                 {
 1414                                     strcpy(partidos[s++],palavra);
 1415                                     palavra = strtok (NULL, "-");
 1416                                 }
 1417
 1418                                 //verifica se o arrayfinalmixpecas ou o inverso e
igual a peca
 1419                                 for (x=0;x<s;x++){
 1420
if(strcmp(partidos[x],mixpecas[j])==0||strcmp(partidos[x],inversoum)==0){
 1421                                                         igual++;
 1422                                             }
 1423                                 }
```

```
 1424                                    if(igual==0){
 1425                                        //concatena para arrayfinalmixpecas
 1426                                                strcpy(arrayfinalmixpecas[p],
arrayfinalmixpecas[i]);
 1427                                                strcat(arrayfinalmixpecas[p],"-");
 1428                                                strcat(arrayfinalmixpecas[p],
mixpecas[j]);
 1429                                                p++;
 1430                                    }
 1431                                }
 1432                                //Segunda verificação
 1433                                if(arrayfinalmixpecas[i][tamlin]==mixpecas[j][2]){
 1434                                    int igual=0;
 1435                                    //inverte peca
 1436                                    int w=strlen(mixpecas[j])-1;
 1437                                    for(k=0;k<(strlen(mixpecas[j]));k++){
 1438                                        inversodois[w]=mixpecas[j][k];
 1439                                        w--;
 1440                                    }
 1441                                    //strtok da peca arrayfinalmixpecas
 1442                                    char partidos[100][100];
 1443                                    char *palavra=NULL;
 1444                                    char umaseqpeca[100];
 1445
 1446                                    strcpy(umaseqpeca,arrayfinalmixpecas[i]);
 1447                                    palavra = strtok (umaseqpeca,"-");
 1448                                    int s=0;
 1449                                    while (palavra != NULL)
 1450                                    {
 1451                                        strcpy(partidos[s++],palavra);
 1452                                        palavra = strtok (NULL, "-");
 1453                                    }
 1454                                    //verifica se o partidos ou o inverso e igual a peca
 1455                                    for (x=0;x<s;x++){
 1456
if(strcmp(partidos[x],mixpecas[j])==0||strcmp(partidos[x],inversodois)==0){
 1457                                                        igual++;
 1458                                            }
 1459                                    }
 1460                                    if(igual==0){
 1461                                        //concatena
 1462                                        strcpy(arrayfinalmixpecas[p],
arrayfinalmixpecas[i]);
 1463                                        strcat(arrayfinalmixpecas[p],"-");
 1464                                        strcat(arrayfinalmixpecas[p], inversodois);
 1465                                        p++;
 1466                                    }
 1467                                }
 1468                                //Terceira verificação
 1469                                if(arrayfinalmixpecas[i][0]==mixpecas[j][0]){
 1470                                    int igual=0;
 1471                                    //inverte peca
 1472                                    int w=strlen(mixpecas[j])-1;
 1473                                    for(k=0;k<(strlen(mixpecas[j]));k++){
 1474                                        inversotres[w]=mixpecas[j][k];
 1475                                        w--;
 1476                                    }
 1477                                    //strtok da peca arrayfinal
 1478                                    char partidos[100][100];
 1479                                    char *palavra=NULL;
 1480                                    char umaseqpeca[100];
 1481
 1482                                    strcpy(umaseqpeca,arrayfinalmixpecas[i]);
 1483                                    palavra = strtok (umaseqpeca,"-");
 1484                                    int s=0;
 1485                                    while (palavra != NULL)
 1486                                    {
 1487                                        strcpy(partidos[s++],palavra);
 1488                                        palavra = strtok (NULL, "-");
 1489                                    }
 1490                                    //verifica se o partidos ou o inverso é igual a peca
```

```
 1491                                   for (x=0;x<s;x++){
 1492
if(strcmp(partidos[x],mixpecas[j])==0||strcmp(partidos[x],inversotres)==0){
 1493                                                           igual++;
 1494                                               }
 1495                                   }
 1496                                   if(igual==0){
 1497                                       //concatena
 1498                                       strcpy(arrayfinalmixpecas[p], inversotres);
 1499                                       strcat(arrayfinalmixpecas[p],"-");
 1500                                       strcat(arrayfinalmixpecas[p],
arrayfinalmixpecas[i]);
 1501                                       p++;
 1502                                   }
 1503                               }
 1504                               //Quarta verificação
 1505                               if(arrayfinalmixpecas[i][0]==mixpecas[j][2]){
 1506                                   int igual=0;
 1507
 1508                                   //inverte peca
 1509                                   int w=strlen(mixpecas[j])-1;
 1510                                   for(k=0;k<(strlen(mixpecas[j]));k++){
 1511                                       inversoquatro[w]=mixpecas[j][k];
 1512                                       w--;
 1513                                   }
 1514                                   //strtok da peca arrayfinal
 1515                                   char partidos[100][100];
 1516                                   char *palavra=NULL;
 1517                                   char umaseqpeca[100];
 1518
 1519                                   strcpy(umaseqpeca,arrayfinalmixpecas[i]);
 1520                                   palavra = strtok (umaseqpeca,"-");
 1521                                   int s=0;
 1522                                   while (palavra != NULL)
 1523                                   {
 1524                                       strcpy(partidos[s++],palavra);
 1525                                       palavra = strtok (NULL, "-");
 1526                                   }
 1527                                   //verifica se o partidos ou o inverso é igual a peca
 1528                                   for (x=0;x<s;x++){
 1529
if(strcmp(partidos[x],mixpecas[j])==0||strcmp(partidos[x],inversoquatro)==0){
 1530                                                           igual++;
 1531                                               }
 1532                                   }
 1533                                   if(igual==0){
 1534                                       //concatena
 1535                                       strcpy(arrayfinalmixpecas[p], mixpecas[j]);
 1536                                       strcat(arrayfinalmixpecas[p],"-");
 1537                                       strcat(arrayfinalmixpecas[p],
arrayfinalmixpecas[i]);
 1538                                       p++;
 1539                                   }
 1540                               }
 1541                           }
 1542                       }
 1543                   //verificar se ha seq iguais e o inversos tambem
 1544                       for(x=0;x<p;x++){
 1545                                   // Inverter peca
 1546                                   int w=strlen(arrayfinalmixpecas[x])-1;
 1547                                   for(k=0;k<(strlen(arrayfinalmixpecas[x]));k++){
 1548                                       invertfinal[w]=arrayfinalmixpecas[x][k];
 1549                                       w--;
 1550                                   }
 1551
 1552                       // Verifica se e igual e se for substitui por X|X
 1553                                   for(i=x+1;i<p;i++){
 1554
if((strcmp(arrayfinalmixpecas[i],invertfinal)==0)||(strcmp(arrayfinalmixpecas[i],arrayfi
nalmixpecas[x])==0)){
 1555                                       strcpy(arrayfinalmixpecas[i],"X|X");
```

```
 1556                                    }
 1557                                }
 1558                    }
 1559                    // Copia do arrayfinalmixpecas que contem x|x e guarda num array
final(arrayfinalmixpecascompleto) as sequencias possiveis
 1560                     for(z=0;z<p;z++){
 1561                            if(strcmp(arrayfinalmixpecas[z],"X|X")==0){
 1562                            }else{
 1563
strcpy(arrayfinalmixpecascompleto[y],arrayfinalmixpecas[z]);
 1564                                y++;
 1565                            }
 1566                    }
 1567                }
 1568                for(i=0;i<y;i++){
 1569                    printf("%s\n",arrayfinalmixpecascompleto[i]);
 1570                }
 1571 }
```

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

```
  988                                           {
  993         char seqcomecar[3000];
  994         char arrayseqinicial[3000][150];
  995         char arrayseqinicialcompleto[3000][150];
  996         int i=0,j=0,count=0,x=0,y=0,p=1,z=0,contador=0;
  997         char inversazero[150];
  998         char inversoum[150];
  999         char inversodois[150];
 1000         char inversotres[150];
 1001         char inversoquatro[150];
 1002         char invertfinal[150];
 1003
 1004         printf("Insira uma sequencia a comecar:\n");
 1005         scanf("%s",seqcomecar);
 1006
 1007         strcpy(arrayseqinicial[0],seqcomecar);
 1008
 1009                 //Strtok da sequencia inicial
 1010                 char partidosseqcomecar[100][100];
 1011                 char *s=NULL;
 1012                 char seqapartir[100];
 1013                 strcpy(seqapartir,seqcomecar);
 1014                 s = strtok (seqapartir,"-");
 1015                 int k=0;
 1016                 while (s!= NULL)
 1017                 {
 1018                     strcpy(partidosseqcomecar[k++],s);
 1019                     s= strtok (NULL, "-");
 1020                 }
 1021                 //Verificar se as pecas da sequencia inicial sao iguais as pecas do
baralho
 1022                 for(x=0;x<7;x++){
 1023                     for(y=0;y<k;y++){
 1024                             // Inverter pecas baralho
 1025                             int d=strlen(baralho[x])-1;
 1026                             int h=0;
 1027                             for(h=0;h<(strlen(baralho[x]));h++){
```

```
 1028                                       inversazero[d]=baralho[x][h];
 1029                                       d--;
 1030                                   }
 1031                                   //Verifica se as pecas da seq inicial coincidem
 1032                                   int tam=strlen(seqcomecar)-1;
 1033                                   int i=0;
 1034                                   for(i=0;i<tam;i++){
 1035                                       if(seqcomecar[i]=='-'){
 1036                                           if(seqcomecar[i-1]!=seqcomecar[i+1]){
 1037                                               contador++;
 1038                                           }
 1039                                       }
 1040                                   }
 1041
 1042
if((strcmp(baralho[x],partidosseqcomecar[y])==0)||(strcmp(inversazero,partidosseqcomecar
[y])==0)){
 1043                                   count++;
 1044                               }
 1045                           }
 1046                       }
 1047                       if(count>0){
 1048                           printf("As pecas da sequencia que inseriu, ja existem no
baralho!\n");
 1049                           count=0;
 1050                       }else if(contador>0){
 1051                           printf("A sequencia que inseriu esta errada!\n ");
 1052                           contador=0;
 1053                       }else{
 1054
 1055                           // Juntar pecas
 1056                           for(i=0;i<p;i++){
 1057                               int tamlin=strlen(arrayseqinicial[i])-1;
 1058                               for(j=0;j<7;j++){
 1059
 1060                                   //Primeira verificação
 1061                                   if(arrayseqinicial[i][tamlin]==baralho[j][0]){
 1062                                       int igual=0;
 1063                                       //inverte peca apenas para verificar
 1064                                       int w=strlen(baralho[j])-1;
 1065                                       for(k=0;k<(strlen(baralho[j]));k++){
 1066                                           inversoum[w]=baralho[j][k];
 1067                                           w--;
 1068                                       }
 1069                                       //strtok da peca arrayfinal
 1070                                       char partidos[100][100];
 1071                                       char *palavra=NULL;
 1072                                       char umaseqpeca[100];
 1073
 1074                                       strcpy(umaseqpeca,arrayseqinicial[i]);
 1075                                       palavra = strtok (umaseqpeca,"-");
 1076                                       int s=0;
 1077                                       while (palavra != NULL)
 1078                                       {
 1079                                           strcpy(partidos[s++],palavra);
 1080                                           palavra = strtok (NULL, "-");
 1081                                       }
 1082
 1083                                       //verifica se o partidos ou o inverso é igual a peça
 1084                                       for (x=0;x<s;x++){
 1085
if(strcmp(partidos[x],baralho[j])==0||strcmp(partidos[x],inversoum)==0){
 1086                                                               igual++;
 1087                                                   }
 1088                                       }
 1089                                       if(igual==0){
 1090                                                   //concatena para arrayseqinicial
 1091                                                   strcpy(arrayseqinicial[p],
arrayseqinicial[i]);
 1092                                                   strcat(arrayseqinicial[p],"-");
```

```
 1093                                            strcat(arrayseqinicial[p],
baralho[j]);
 1094                                            p++;
 1095                                        }
 1096
 1097                                    }
 1098                                    //Segunda verificação
 1099                                    if(arrayseqinicial[i][tamlin]==baralho[j][2]){
 1100                                        int igual=0;
 1101                                        //inverte peca
 1102                                        int w=strlen(baralho[j])-1;
 1103                                        for(k=0;k<(strlen(baralho[j]));k++){
 1104                                            inversodois[w]=baralho[j][k];
 1105                                            w--;
 1106                                        }
 1107                                        //strtok da peca arrayseqinicial
 1108                                        char partidos[100][100];
 1109                                        char *palavra=NULL;
 1110                                        char umaseqpeca[100];
 1111
 1112                                        strcpy(umaseqpeca,arrayseqinicial[i]);
 1113                                        palavra = strtok (umaseqpeca,"-");
 1114                                        int s=0;
 1115                                        while (palavra != NULL)
 1116                                        {
 1117                                            strcpy(partidos[s++],palavra);
 1118                                            palavra = strtok (NULL, "-");
 1119                                        }
 1120
 1121                                        //verifica se o partidos ou o inverso é igual a peça
 1122                                        for (x=0;x<s;x++){
 1123
if(strcmp(partidos[x],baralho[j])==0||strcmp(partidos[x],inversodois)==0){
 1124                                                        igual++;
 1125                                                }
 1126                                        }
 1127                                        if(igual==0){
 1128                                            //concatena
 1129                                            strcpy(arrayseqinicial[p],
arrayseqinicial[i]);
 1130                                            strcat(arrayseqinicial[p],"-");
 1131                                            strcat(arrayseqinicial[p], inversodois);
 1132                                            p++;
 1133                                        }
 1134                                    }
 1135                                    //Terceira verificação
 1136                                    if(arrayseqinicial[i][0]==baralho[j][0]){
 1137                                        int igual=0;
 1138
 1139                                        //inverte peca
 1140                                        int w=strlen(baralho[j])-1;
 1141                                        for(k=0;k<(strlen(baralho[j]));k++){
 1142                                            inversotres[w]=baralho[j][k];
 1143                                            w--;
 1144                                        }
 1145                                        //strtok da peca arrayfinal
 1146                                        char partidos[100][100];
 1147                                        char *palavra=NULL;
 1148                                        char umaseqpeca[100];
 1149
 1150                                        strcpy(umaseqpeca,arrayseqinicial[i]);
 1151                                        palavra = strtok (umaseqpeca,"-");
 1152                                        int s=0;
 1153                                        while (palavra != NULL)
 1154                                        {
 1155                                            strcpy(partidos[s++],palavra);
 1156                                            palavra = strtok (NULL, "-");
 1157                                        }
 1158
 1159                                        //verifica se o partidos ou o inverso é igual a peca
 1160                                        for (x=0;x<s;x++){
```

```
 1161
if(strcmp(partidos[x],baralho[j])==0||strcmp(partidos[x],inversotres)==0){
 1162                                                igual++;
 1163                                            }
 1164                                    }
 1165                                    if(igual==0){
 1166                                        //concatena
 1167                                        strcpy(arrayseqinicial[p], inversotres);
 1168                                        strcat(arrayseqinicial[p],"-");
 1169                                        strcat(arrayseqinicial[p],
arrayseqinicial[i]);
 1170                                        p++;
 1171                                    }
 1172                                }
 1173                                //Quarta verificação
 1174                                if(arrayseqinicial[i][0]==baralho[j][2]){
 1175                                    int igual=0;
 1176
 1177                                    //inverte peca
 1178                                    int w=strlen(baralho[j])-1;
 1179                                    for(k=0;k<(strlen(baralho[j]));k++){
 1180                                        inversoquatro[w]=baralho[j][k];
 1181                                        w--;
 1182                                    }
 1183
 1184                                    //strtok da peca arrayfinal
 1185                                    char partidos[100][100];
 1186                                    char *palavra=NULL;
 1187                                    char umaseqpeca[100];
 1188
 1189                                    strcpy(umaseqpeca,arrayseqinicial[i]);
 1190                                    palavra = strtok (umaseqpeca,"-");
 1191                                    int s=0;
 1192                                    while (palavra != NULL)
 1193                                    {
 1194                                        strcpy(partidos[s++],palavra);
 1195                                        palavra = strtok (NULL, "-");
 1196                                    }
 1197
 1198                                    //verifica se o arraypartidos ou o invero e igual a
peca
 1199                                    for (x=0;x<s;x++){
 1200
if(strcmp(partidos[x],baralho[j])==0||strcmp(partidos[x],inversoquatro)==0){
 1201                                                igual++;
 1202                                            }
 1203                                    }
 1204
 1205                                    if(igual==0){
 1206                                        //concatena
 1207                                        strcpy(arrayseqinicial[p], baralho[j]);
 1208                                        strcat(arrayseqinicial[p],"-");
 1209                                        strcat(arrayseqinicial[p],
arrayseqinicial[i]);
 1210                                        p++;
 1211                                    }
 1212                                }
 1213                            }
 1214                        }
 1215                    //verificar se ha sequencias iguais e o inversos tambem
 1216                        for(x=0;x<p;x++){
 1217                                    // Inverter peca
 1218                                    int w=strlen(arrayseqinicial[x])-1;
 1219
 1220                                    for(k=0;k<(strlen(arrayseqinicial[x]));k++){
 1221                                        invertfinal[w]=arrayseqinicial[x][k];
 1222                                        w--;
 1223                                    }
 1224                        // Verifica se e igual e se for substitui por X|X
 1225                                for(i=x+1;i<p;i++){
```

```
 1226
if((strcmp(arrayseqinicial[i],invertfinal)==0)||(strcmp(arrayseqinicial[i],arrayseqinici
al[x])==0)){
 1227                                    strcpy(arrayseqinicial[i],"X|X");
 1228                                }
 1229                            }
 1230                        }
 1231                        // Copia do aux que contem x|x e guarda num array
final(arrayseqinicialcompleto) as sequencias das pecas
 1232                         for(z=0;z<p;z++){
 1233                                if(strcmp(arrayseqinicial[z],"X|X")==0){
 1234                                }else{
 1235
strcpy(arrayseqinicialcompleto[y],arrayseqinicial[z]);
 1236                                    y++;
 1237                                }
 1238                        }
 1239                }
 1240                for(i=0;i<y;i++){
 1241                    printf("%s\n",arrayseqinicialcompleto[i]);
 1242                }
 1243 }
```

Referência ao ficheiro projetolp1primparte.h

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

Macros

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

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** (char pecas[][**COLSTRING**], char baralho[][**COLSTRING**], char sobrou[][**COLSTRING**], int)
  Criar baralhos/mãos aleatórios.
- void **rempeca** (char pecas[][**COLSTRING**], char baralho[][**COLSTRING**], int, int)
  Remover peças de uma ou mais mãos.
- void **addpeca** (char pecas[][**COLSTRING**], char baralho[][**COLSTRING**], char sobrou[][**COLSTRING**], int, int)
  Adicionar peças numa mão/mãos.
- void **imprimirpecasint** (char pecas[][**COLSTRING**], int pecasint[][**COL**])
  Imprimir pecas em inteiros.
- int **ordenarseq** (char pecas[][**COLSTRING**], char baralho[][**COLSTRING**], char arrayfinalcompleto[][150], int)
  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.

Documentação das macros

#define COL 2

#define COLSTRING 4

#define LINSTRING 28

Documentação das funções

void addpeca (char *pecas*[][COLSTRING], char *baralho*[][COLSTRING], char *sobrou*[][COLSTRING], int , int)

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:

pecas[][COLSTRING]	Array do tipo char onde recebe as peças totais do jogo.
baralho[][COLSTRING]	Array do tipo char, onde recebe 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
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 i=0 - Variavél usada nas funções seguintes

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

```
  369
{
  379     int aleatorio=0;
  381     int i=0;
  383     int j=0;
  385     int k=0;
  387     char novapeca[4];
  389     char invertnewpeca[4];
  390
  391
  392     aleatorio=pecastotal+npecas;
  393     // 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.
  394     if(aleatorio>0 && aleatorio<6){
  395         aleatorio=6-aleatorio;
  396     }else if(aleatorio>6 && aleatorio<13){
  397             aleatorio=13-aleatorio;
  398     }else if(aleatorio>13 && aleatorio<20){
  399         aleatorio=20-aleatorio;
  400     }else if(aleatorio>20 && aleatorio<27){
  401         aleatorio=27-aleatorio;
  402     }
  403     // 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
  404     for(j=pecastotal;j<=(pecastotal+aleatorio);j++){
  405
  406         strcpy(baralho[j],sobrou[j]);
  407     }
  408     // Imprime a mão/baralho mais as peças aleatorias
  409     for(i=0;i<j;i++){
  410         printf("%s\n",baralho[i]);
  411     }
  412     // Inserir peca (novapeca)
  413     int x=0,fim=0,h=0,cont=0;
  414     for(k=0;k<npecas;k++){
  415             printf("Insira a peca");
  416             scanf("%s",novapeca);
  417
  418             // Inverter novapeca
  419             int d=strlen(novapeca)-1;
  420             for(h=0;h<(strlen(novapeca));h++){
  421                 invertnewpeca[d]=novapeca[h];
  422                 d--;
  423             }
  424             // Verifica se a peca da mão(baralho[]) é igual a novapeca, ou á invertida
  425             for(i=0;i<=(pecastotal+aleatorio+x);i++){
  426
if((strcmp(baralho[i],novapeca)==0)||(strcmp(baralho[i],invertnewpeca)==0)){
  427                         cont++;
  428                      }
  429             }
  430             // Caso a peça exista, aumenta o cont, e entra no if
  431             if(cont>0){
  432                         printf("Essa peca ja existe!\n");
  433                         k--;
  434                         cont=0;
  435             }else{
  436                 //Caso contrario ele insere a nova peça na ultima posição da mão
(baralho[]) e imprime
  437                         cont=0;
  438                         x++;
  439                         fim=pecastotal+aleatorio+x;
  440                         strcpy(baralho[fim],novapeca);
  441
  442                         for(i=0;i<=fim;i++){
  443                             printf("%s\n",baralho[i]);
  444                         }
  445             }
  446     }
  447 }
```

void baralhos (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:

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]

```
  225
{
  234     srand((unsigned)time(NULL));
  236     int pecastotal=0;
  238     int i=0;
  240     int aleatorio;
  242     int array[28];
  244     int temp;
  245     // 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)
  246     pecastotal = num * 7;
  247     //Caso o numero de mãos escollhidas seja igual ou inferior a 4, entra neste if
  248     if(num<=4){
  249             // Cria array posicoes das pecas totais
  250             for(i=0;i<28;i++){
  251                 array[i] = i;
  252             }
  253             // Baralha essas 28 pecas e guarda em array, de forma aleatória
  254             for(i=0;i<28;i++){
  255                 temp=array[i];
  256                 aleatorio = rand() % 28;
  257                 array[i]=array[aleatorio];
  258                 array[aleatorio]=temp;
  259             }
  260             // Guarda num array baralho[] as peças aleatórias até ao numero de pecas
total definida pelo jogador e imprime
  261             printf("MAO:\n");
  262             for(i=0;i<pecastotal;i++){
  263                 strcpy(baralho[i],pecas[array[i]]);
  264                 printf("%s\n",baralho[i]);
  265             }
  266             // Guarda num array sobrou[] as peças aleatórias desde o numero de pecas
total definida pelo jogador até as 27 possiveis e imprime
  267             printf("SOBROU:\n");
  268             for(i=pecastotal;i<28;i++){
  269                 strcpy(sobrou[i],pecas[array[i]]);
  270                 printf("%s\n",sobrou[i]);
  271             }
  272     }else if(num>4){
  273         printf("Nao pode escolher mais que 4 baralhos!\n");
  274     }
  275 }
```

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

```
144                                             {
149     strcpy(pecas[0],"0|0");
150     strcpy(pecas[1],"0|1");
151     strcpy(pecas[2],"0|2");
152     strcpy(pecas[3],"0|3");
153     strcpy(pecas[4],"0|4");
154     strcpy(pecas[5],"0|5");
155     strcpy(pecas[6],"0|6");
156     strcpy(pecas[7],"1|1");
157     strcpy(pecas[8],"1|2");
158     strcpy(pecas[9],"1|3");
159     strcpy(pecas[10],"1|4");
160     strcpy(pecas[11],"1|5");
161     strcpy(pecas[12],"1|6");
162     strcpy(pecas[13],"2|2");
163     strcpy(pecas[14],"2|3");
164     strcpy(pecas[15],"2|4");
165     strcpy(pecas[16],"2|5");
166     strcpy(pecas[17],"2|6");
167     strcpy(pecas[18],"3|3");
168     strcpy(pecas[19],"3|4");
169     strcpy(pecas[20],"3|5");
170     strcpy(pecas[21],"3|6");
171     strcpy(pecas[22],"4|4");
172     strcpy(pecas[23],"4|5");
173     strcpy(pecas[24],"4|6");
174     strcpy(pecas[25],"5|5");
175     strcpy(pecas[26],"5|6");
176     strcpy(pecas[27],"6|6");
177
179 /*
180     strcpy(pecas[0],"2|5");
181     strcpy(pecas[1],"3|3");
182     strcpy(pecas[2],"2|2");
183     strcpy(pecas[3],"1|1");
184     strcpy(pecas[4],"4|4");
185     strcpy(pecas[5],"5|5");
186     strcpy(pecas[6],"6|6");
187     strcpy(pecas[7],"1|2");
188     strcpy(pecas[8],"6|3");
189     strcpy(pecas[9],"1|4");
190     strcpy(pecas[10],"1|5");
191     strcpy(pecas[11],"0|6");
192     strcpy(pecas[12],"2|1");
193     strcpy(pecas[13],"0|3");
194 */
195
196 }
```

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.

```
198                                             {
203     int i = 0;
204     for(i=0;i<28;i++){
205         printf("%s\n",pecas[i]);
206     }
207 }
```

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.

```
210                                                                     {
216     int i=0;
217     for(i=0;i<28;i++){
218         pecasint[i][0]=atoi(pecas[i]);
219         pecasint[i][1]=atoi(&pecas[i][2]);
220         printf("%d|%d \n", pecasint[i][0],pecasint[i][1]);
221     }
222 }
```

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

Função main_projectolp1primparte.

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

int pecasint[LINSTRING][COL] - Array de inteiros 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     int pecasint[LINSTRING][COL];
   50     char baralho[LINSTRING][COLSTRING];
   52     char sobrou[LINSTRING][COLSTRING];
   54     char arrayfinalcompleto[3000][150];
   56     int opc=0;
   58     int num=0;
   60     int npecasremover=0;
   62     int pecastotal=0;
   64     int npecas=0;
   66     int y=0;
   68     criarpecas(pecas);
   70     bool sair = false;
   72     char escolha;
   73     //Menu com ciclo. O utilizador escolhe a opção que pretende e através do switch
retorna para a função pretendida
   74     do{
   75     printf("*************** JOGO DO DOMINO ***************\n\n");
   76     printf("Escolha uma opcao\n");
   77     printf("1 ----> Listar Pecas Sring\n");
   78     printf("2 ----> Criar Mao\n");
   79     printf("3 ----> Remover Pecas\n");
   80     printf("4 ----> Adicionar Pecas\n");
   81     printf("5 ----> Listar Pecas Int\n");
   82     printf("6 ----> Criar e ordenar sequencias\n");
   83     printf("7 ----> Procurar padrao\n");
   84     printf("8 ----> Criar sequencias com sequencia inicial\n");
   85     printf("9 ----> Criar sequencias com maos alternadas\n");
   86
   87     printf("Opcao: ");
   88     scanf("%d", &opc);
   89
   90     switch(opc)
   91             {
   92             case 1:
   93                 imprimirpecas(pecas);
   94                 break;
   95             case 2:
   96                 printf("Insira o numero de mãos a jogar:\n");
   97                 scanf("%d",&num);
   98                 baralhos(pecas,baralho,sobrou,num);
   99                 break;
  100             case 3:
  101                 printf("Quantas pecas pretende remover:\n");
  102                 scanf("%d",&npecasremover);
  103                 pecastotal=num*7;
  104                 rempeca(pecas,baralho,pecastotal,npecasremover);
  105                 break;
  106             case 4:
  107                 printf("Quantas pecas pretende adicionar:\n");
  108                 scanf("%d",&npecas);
  109                 pecastotal=num*7;
  110                 addpeca(pecas,baralho,sobrou,pecastotal,npecas);
  111                 break;
  112             case 5:
  113                 imprimirpecasint(pecas,pecasint);
  114                 break;
  115             case 6:
  116                 y=ordenarseq(pecas,baralho,arrayfinalcompleto,num);
  117                 break;
  118             case 7:
```

```
 119                 procurar_padrao(arrayfinalcompleto,y);
 120                 break;
 121             case 8:
 122                 seq_inicial(baralho);
 123                 break;
 124             case 9:
 125                 retirar_mao_jogadores(baralho,num);
 126                 break;
 127
 128
 129             default:
 130             printf("Escolha invalida!\n\n");
 131             }
 132             printf("Pretende sair? S --->sim \n");
 133             scanf("%s",&escolha);
 134             if(escolha=='S'||escolha=='s'){
 135                 sair=true;
 136             }else{
 137                 sair=false;
 138             }
 139     }while(sair==false);
 140     return 0;
 141 }
```

int ordenarseq (char *pecas*[][COLSTRING], char *baralho*[][COLSTRING], char *arrayfinalcompleto*[][150], int)

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 arrayfinalcompleto até o tamanho do arrayfinalcompleto (p) termine.Tudo isto com verificações a ver se há repetidas, ou invertidas.

Parâmetros:

pecas[][COLSTRING]	Array do tipo char onde recebe as peças totais do jogo.
baralho[][COLSTRING]	Array do tipo char, onde recebe as peças baralhadas da mão do jogador
arrayfinalcompleto[][150]	Array do tipo char, onde são guardadas as sequencias totais da mão do jogador
num	Valor do tipo int, utilizado para definir o numero de mãos a ser usado pelo jogador

Retorna:

y, tamanho do array total das sequencias (arrayfinalcompleto)

char seqduaspecas[3000][150] - Usado para guardar sequencias de duas peças

char arrayfinal[3000][150] - Usado para guardar sequencias de duas peças

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

```
 450
{
 459 int i=0,j=0,h=0,z=0,count=0,p=0;
 460 int x=0,k=0,y=0;
```

```
  461 char aux[3000][150];
  463 char seqduaspecas[3000][150];
  465 char arrayfinal[3000][150];
  466 char tempd[150][150];
  468 char invertnewpeca[150];
  470 char invertnewpecaaux[150];
  472 char invert[150];
  474 char inversopeca[150];
  476 char invertfinal[500];
  477
  478 num=num*7;
  479 //Juntar 2 pecas
  480     for(i=0;i<num;i++){
  481         for(j=0;j<num;j++){
  482             if(baralho[i][2]==baralho[j+1][0]){
  483                 //concatena
  484                 strcpy(aux[count], baralho[i]);
  485                 strcat(aux[count],"-");
  486                 strcat(aux[count], baralho[j+1]);
  487                 count++;
  488             }
  489             if(baralho[i][2]==baralho[j+1][2]){
  490                 // Inverter peca
  491                 int d=strlen(baralho[j+1])-1;
  492                 for(h=0;h<(strlen(baralho[j+1]));h++){
  493                     invertnewpeca[d]=baralho[j+1][h];
  494                     d--;
  495                 }
  496                 // concatena
  497                 strcpy(aux[count], baralho[i]);
  498                 strcat(aux[count],"-");
  499                 strcat(aux[count], invertnewpeca);
  500                 count++;
  501             }
  502             if(baralho[i][0]==baralho[j+1][2]){
  503                 // concatena
  504                 strcpy(aux[count], baralho[j+1]);
  505                 strcat(aux[count],"-");
  506                 strcat(aux[count], baralho[i]);
  507                 count++;
  508             }
  509             if(baralho[i][0]==baralho[j+1][0]){
  510                 // Inverter peca
  511                 int d=strlen(baralho[j+1])-1;
  512                 for(h=0;h<(strlen(baralho[j+1]));h++){
  513                     invertnewpeca[d]=baralho[j+1][h];
  514                     d--;
  515                 }
  516                 // concatena
  517                 strcpy(aux[count], invertnewpeca);
  518                 strcat(aux[count],"-");
  519                 strcat(aux[count], baralho[i]);
  520                 count++;
  521             }
  522         }
  523     }
  524     //Verificar se ha sequencias iguais e o inversos tambem
  525     for(x=0;x<count;x++){
  526                 // Inverter peca
  527                 int w=strlen(aux[x])-1;
  528                 for(k=0;k<(strlen(aux[x]));k++){
  529                     invertnewpecaaux[w]=aux[x][k];
  530                     w--;
  531                 }
  532     // Verifica se é igual e se for substitui por X|X
  533             for(i=x+1;i<count;i++){
  534
if((strcmp(aux[i],invertnewpecaaux)==0)||(strcmp(aux[i],aux[x])==0)||((aux[i][0]==aux[i]
[6])&&(aux[i][2]==aux[i][4]))){
  535                     strcpy(aux[i],"X|X");
  536                 }
```

```
  537             }
  538     }
  539     // Copia do aux que contem x|x e guarda num array final(seqduaspecas) e guarda num
array final (arrayfinal) as sequencias de duas pecas
  540      for(z=0;z<count;z++){
  541             if(strcmp(aux[z],"X|X")==0){
  542             }else{
  543                 strcpy(seqduaspecas[p],aux[z]);
  544                 strcpy(arrayfinal[p],aux[z]);
  545                 p++;
  546             }
  547     }
  548
  549 // Juntar 3 ou mais pecas
  550     for(i=0;i<p;i++){
  551         int tamlin=strlen(arrayfinal[i])-1;
  552         for(j=0;j<num;j++){
  553
  554             //Primeira verificação
  555             if(arrayfinal[i][tamlin]==baralho[j][0]){
  556                 int igual=0;
  557                 //inverte peca apenas para verificar
  558                 int w=strlen(baralho[j])-1;
  559                 for(k=0;k<(strlen(baralho[j]));k++){
  560                     inversopeca[w]=baralho[j][k];
  561                     w--;
  562                 }
  563                 //strtok da peca arrayfinal
  564                 char partidos[100][100];
  565                 char *palavra=NULL;
  566                 char umaseqpeca[100];
  567
  568                 strcpy(umaseqpeca,arrayfinal[i]);
  569                 palavra = strtok (umaseqpeca,"-");
  570                 int s=0;
  571                 while (palavra != NULL)
  572                 {
  573                     strcpy(partidos[s++],palavra);
  574                     palavra = strtok (NULL, "-");
  575                 }
  576
  577                 //verifica se o arraypartidos ou o inverdo é igual a peca
  578                 for (x=0;x<s;x++){
  579
if(strcmp(partidos[x],baralho[j])==0||strcmp(partidos[x],inversopeca)==0){
  580                                     igual++;
  581                             }
  582                 }
  583                 if(igual==0){
  584                             //concatena para arrayfinal
  585                             strcpy(arrayfinal[p], arrayfinal[i]);
  586                             strcat(arrayfinal[p],"-");
  587                             strcat(arrayfinal[p], baralho[j]);
  588                             p++;
  589                 }
  590             }
  591             //Segunda verificação
  592             if(arrayfinal[i][tamlin]==baralho[j][2]){
  593                 int igual=0;
  594                 //inverte peca
  595                 int w=strlen(baralho[j])-1;
  596
  597                 for(k=0;k<(strlen(baralho[j]));k++){
  598                     invert[w]=baralho[j][k];
  599                     w--;
  600                 }
  601                 //strtok da peca arrayfinal
  602                 char partidos[100][100];
  603                 char *palavra=NULL;
  604                 char umaseqpeca[100];
  605
```

```
606                 strcpy(umaseqpeca,arrayfinal[i]);
607                 palavra = strtok (umaseqpeca,"-");
608                 int s=0;
609                 while (palavra != NULL)
610                 {
611                     strcpy(partidos[s++],palavra);
612                     palavra = strtok (NULL, "-");
613                 }
614 
615                 //verifica se o arraypartidos ou o inverso é igual a peça
616                 for (x=0;x<s;x++){
617 
if(strcmp(partidos[x],baralho[j])==0||strcmp(partidos[x],invert)==0){
618                                 igual++;
619                         }
620                 }
621                 if(igual==0){
622                     //concatena
623                     strcpy(arrayfinal[p], arrayfinal[i]);
624                     strcat(arrayfinal[p],"-");
625                     strcat(arrayfinal[p], invert);
626                     p++;
627                 }
628             }
629 
630             //Terceira verificação
631             if(arrayfinal[i][0]==baralho[j][0]){
632                 int igual=0;
633 
634                 //inverte peca
635                 int w=strlen(baralho[j])-1;
636                 for(k=0;k<(strlen(baralho[j]));k++){
637                     invert[w]=baralho[j][k];
638                     w--;
639                 }
640 
641                 //strtok da peca arrayfinal
642                 char partidos[100][100];
643                 char *palavra=NULL;
644                 char umaseqpeca[100];
645                 strcpy(umaseqpeca,arrayfinal[i]);
646                 palavra = strtok (umaseqpeca,"-");
647                 int s=0;
648                 while (palavra != NULL)
649                 {
650                     strcpy(partidos[s++],palavra);
651                     palavra = strtok (NULL, "-");
652                 }
653 
654                 //verifica se o arraypartidos ou o inverso é igual a peça
655                 for (x=0;x<s;x++){
656 
if(strcmp(partidos[x],baralho[j])==0||strcmp(partidos[x],invert)==0){
657                                 igual++;
658                         }
659                 }
660                 if(igual==0){
661                     //concatena
662                     strcpy(arrayfinal[p], invert);
663                     strcat(arrayfinal[p],"-");
664                     strcat(arrayfinal[p], arrayfinal[i]);
665                     p++;
666                 }
667             }
668             //Quarta verificação
669             if(arrayfinal[i][0]==baralho[j][2]){
670                 int igual=0;
671 
672                 //inverte peca
673                 int w=strlen(baralho[j])-1;
674                 for(k=0;k<(strlen(baralho[j]));k++){
```

```
  675                     invert[w]=baralho[j][k];
  676                     w--;
  677                 }
  678                 //strtok da peca arrayfinal
  679                 char partidos[100][100];
  680                 char *palavra=NULL;
  681                 char umaseqpeca[100];
  682
  683                 strcpy(umaseqpeca,arrayfinal[i]);
  684                 palavra = strtok (umaseqpeca,"-");
  685                 int s=0;
  686                 while (palavra != NULL)
  687                 {
  688                     strcpy(partidos[s++],palavra);
  689                     palavra = strtok (NULL, "-");
  690                 }
  691
  692                 //verifica se o arraypartidos ou o inverso é igual a peça
  693                 for (x=0;x<s;x++){
  694
if(strcmp(partidos[x],baralho[j])==0||strcmp(partidos[x],invert)==0){
  695                                 igual++;
  696                         }
  697                 }
  698                 if(igual==0){
  699                     //concatena
  700                     strcpy(arrayfinal[p], baralho[j]);
  701                     strcat(arrayfinal[p],"-");
  702                     strcat(arrayfinal[p], arrayfinal[i]);
  703                     p++;
  704                 }
  705             }
  706         }
  707     }
  708     //verificar se ha sequencias iguais e inversos tambem
  709     for(x=0;x<p;x++){
  710                 // Inverter peca
  711                 int w=strlen(arrayfinal[x])-1;
  712                 for(k=0;k<(strlen(arrayfinal[x]));k++){
  713                     invertfinal[w]=arrayfinal[x][k];
  714                     w--;
  715                 }
  716
  717             // Verifica se e igual e se for substitui por X|X
  718             for(i=x+1;i<p;i++){
  719
if((strcmp(arrayfinal[i],invertfinal)==0)||(strcmp(arrayfinal[i],arrayfinal[x])==0)){
  720                     strcpy(arrayfinal[i],"X|X");
  721                 }
  722             }
  723     }
  724     // Copia do aux que contem x|x e guarda num array final(arrayfinalcompleto) as
sequencias de tres ou mais pecas */
  725      for(z=0;z<p;z++){
  726             if(strcmp(arrayfinal[z],"X|X")==0){
  727             }else{
  728                 strcpy(arrayfinalcompleto[y],arrayfinal[z]);
  729                 y++;
  730             }
  731     }
  732
  733         // Ordenar de forma decrescente
  734         for( i=0; i<y; i++ ){
  735                   for( j=(i+1); j<y; j++ ){
  736
if((strlen(arrayfinalcompleto[i]))<(strlen(arrayfinalcompleto[j]))){
  737                             strcpy(tempd[0],arrayfinalcompleto[i]);
  738                             strcpy(arrayfinalcompleto[i],arrayfinalcompleto[j]);
  739                             strcpy(arrayfinalcompleto[j],tempd[0]);
  740                         }
  741                   }
```

```
  742         }
  743
  744         for(k=0;k<y;k++){
  745               printf("%s\n",arrayfinalcompleto[k]);
  746         }
  747     return y;
  748 }
```

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)

```
  750                                                                 {
  756     //Guarda todas as posicoes num array para depois baralhar esses i's
  757     int z=0;
  758     int auxp[50];
  759     int ptemp=0;
  760     int aleatorio=0;
  761     int b=0,i=0;
  762     char strproc[100];
  763     int tamarraycompleto=y;
  764
  765     for(z=0;z<tamarraycompleto;z++){
  766         auxp[z]=z;
  767     }
  768
  769     //baralho esse numeros do auxp
  770     for(b=0;b<tamarraycompleto;b++){
  771         ptemp=auxp[b];
  772         aleatorio=rand()%(tamarraycompleto-1);
  773         auxp[b]=auxp[aleatorio];
  774         auxp[aleatorio]=ptemp;
  775     }
  776     //vai buscar uma sequencia do arrayfinalcompleto dependendo do num de sequencias
que o utilizador quer
  777         strcpy(strproc,arrayfinalcompleto[auxp[0]]);
  778         printf("A sequencia escolhida foi:\n %s \n",strproc);
  779
  780
  781     char sub[50];
  782     char str2[100][100];
  783     //declaracoes partir arrayfinalcompleto
  784     char partidosstr1[100][100];
  785     char *s1=NULL;
  786     char str1apartir[100];
  787     int u=0;
  788     //declaracoes partir subtring
  789     char partidosstr2[100][100];
  790     char *s2=NULL;
  791     char str2apartir[100];
  792     int l=0;
  793     //declaracoes partir subtring inversa
  794     char partidosstr2inv[100][100];
  795     char *s2inv=NULL;
  796     char str2invapartir[100];
```

```
  797     int inv=0;
  798     //declaracao inverter subtring
  799     char invertsub[100];
  800     //declaracoes comparar a ver se sao iguais
  801     int x=0,g=0,count=0,r=0,pos=0;;
  802     int v[100];
  803
  804
  805     do{
  806
  807     printf("Insira uma subsequencia para procurar:\n");
  808     scanf("%s",sub);
  809
  810                 //Strtok do arrayfinalcompleto
  811                 strcpy(str1apartir,strproc);
  812                 s1 = strtok (str1apartir,"-");
  813
  814                 while (s1!= NULL)
  815                 {
  816                     strcpy(partidosstr1[u++],s1);
  817                     s1= strtok (NULL, "-");
  818                 }
  819
  820                 //Strtok da substring escolhida
  821                 strcpy(str2[0],sub);
  822                 strcpy(str2apartir,str2[0]);
  823                 s2 = strtok (str2apartir,"-");
  824                 while (s2!= NULL)
  825                 {
  826                     strcpy(partidosstr2[l++],s2);
  827                     s2= strtok (NULL, "-");
  828                 }
  829
  830                 // Inverter substring
  831                     int w=strlen(sub)-1;
  832                     int k=0;
  833                     for(k=0;k<(strlen(sub));k++){
  834                         invertsub[w]=sub[k];
  835                         w--;
  836                     }
  837
  838                 //Strtok da substring invertida
  839
  840                 strcpy(str2invapartir,invertsub);
  841                 s2inv = strtok (str2invapartir,"-");
  842                 while (s2inv!= NULL)
  843                 {
  844                     strcpy(partidosstr2inv[inv++],s2inv);
  845                     s2inv= strtok (NULL, "-");
  846                 }
  847
  848                 //Verifica se a invertida é igual a string original (partidosstr1)
  849                 for(x=0;x<u;x++){
  850                     if((strcmp(partidosstr1[x],partidosstr2inv[g])==0)){
  851                         count++;
  852                         g++;
  853                       if(count==g && count==l){
  854                         //pos- é o valor da peca final menos o tam da sub -1, para
retornar a primeira posicao onde encontrou
  855                         pos=(x-(l-1));
  856                         v[r]=pos;
  857                         r++;
  858                         count=0;
  859                         g=0;
  860                         pos=0;
  861                         strcpy(sub,invertsub);
  862                       }
  863                     }else{
  864                         count=0;
  865                         g=0;
  866                         pos=0;
```

```
  867                     }
  868                 }
  869                 //Verifica se a string original partida (partidosstr1) é igual a
substring inserida partida (partidosstr2)
  870                 for(x=0;x<u;x++){
  871                     if(strcmp(partidosstr1[x],partidosstr2[g])==0){
  872                         count++;
  873                         g++;
  874                       if(count==g && count==l){
  875                         //pos- é o valor da peca final menos o tam da sub -1, para
retornar a primeira posicao onde encontrou
  876                         pos=(x-(l-1));
  877                         v[r]=pos;
  878                         r++;
  879                         count=0;
  880                         g=0;
  881                         pos=0;
  882                       }
  883                     }else{
  884                         count=0;
  885                         g=0;
  886                         pos=0;
  887                     }
  888                 }
  889                 if(r==0){
  890                     printf("Nao foi encontado a subsequencia %s, na sequencia %s!\n
",sub,strproc);
  891                     u=0;
  892                     l=0;
  893
  894                 }else{
  895                     for(i=0;i<r;i++){
  896                         printf(" Posicao: %d\n",v[i]);
  897                     }
  898                 }
  899     }while(r==0);
  900                 //Substituir Padrão
  901                 int opcao=0,j=0,q=0;
  902                 char aux[100][100];
  903                 char addseqpeca[50];
  904                 printf("Pretende substituir essa sequencia por outra peca/seq:\n
sim->1 , nao->0\n");
  905                 scanf("%d",&opcao);
  906                 if(opcao==1){
  907                     int continua=1;
  908                     while(continua>0){
  909                         continua=0;
  910                         printf("Insira a seq/peca que deseja substituir");
  911                         scanf("%s",addseqpeca);
  912
  913                         //Strtok da peça adicionada (addseqpeca)
  914                         char partidasaddseqpeca[100][100];
  915                         char *s3=NULL;
  916                         char addseqpecaapartir[100];
  917                         strcpy(addseqpecaapartir,addseqpeca);
  918                         s3 = strtok (addseqpecaapartir,"-");
  919                         int b=0;
  920
  921                         while (s3!= NULL)
  922                         {
  923                             strcpy(partidasaddseqpeca[b++],s3);
  924                             s3= strtok (NULL, "-");
  925                         }
  926
  927                         //Verifica se essa seq adicionada é possivel inserir
  928                         char invert[4];
  929                         int cont=0,n=0,m=0;
  930                         for(n=0;n<u;n++){
  931                             for(m=0;m<b;m++){
  932                                     // Inverter addseqpeca
  933                                     int d=strlen(partidasaddseqpeca[m])-1;
```

```
  934                                     int h=0;
  935
for(h=0;h<(strlen(partidasaddseqpeca[m]));h++){
  936                                         invert[d]=partidasaddseqpeca[m][h];
  937                                         d--;
  938                                     }
  939                                     //verifica se a peça adicionada existe na string
original , ou se a peça adicionada invertida existe na string original
  940
if((strcmp(partidasaddseqpeca[m],partidosstr1[n])==0)||(strcmp(partidosstr1[n],invert)==
0)){
  941                                         cont++;
  942                                     }
  943                                 }
  944                             }
  945                             if(cont>0){
  946
  947                                     printf("Essa peca ja existe!\n");
  948                                     continua++;
  949                                     cont=0;
  950                             }else{
  951                                     //verifica se o padrao encontrado está no meio,
á frente ou atrás.
  952
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]))){
  953                                         cont=0;
  954                                         //Adiciona no aux as primeiras pecas, as
novas e depois o resto
  955                                             for(i=0;i<v[0];i++){
  956                                                 strcpy(aux[q],partidosstr1[i]);
  957                                                     q++;
  958                                             }
  959                                             int c=0;
  960                                             for(c=0;c<b;c++){
  961
strcat(aux[q],partidasaddseqpeca[c]);
  962                                                 q++;
  963                                             }
  964                                             for(j=(v[0]+l);j<u;j++){
  965
strcat(aux[q],partidosstr1[j]);
  966                                                     q++;
  967                                             }
  968                                 }else{
  969                                     printf("Pecas inseridas nao coincidem!\n");
  970                                     continua++;
  971                                 }
  972                             }
  973                             for(m=0;m<q;m++){
  974                                 printf("%s",aux[m]);
  975                                 if(m!=(q-1)){
  976                                     printf("-");
  977                                 }
  978                             }
  979                             printf("\n");
  980                 }
  981                 }else if(opcao==0){
  982
  983                 }else{
  984                     printf("opcao errada!");
  985                 }
  986 }
```

void rempeca (char *pecas*[][COLSTRING], char *baralho*[][COLSTRING], int , int)

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:

pecas[][COLSTRING]	Array do tipo char onde recebe as peças totais do jogo.
baralho[][COLSTRING]	Array do tipo char, onde recebe as peças baralhadas da mão do jogador
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 i=0 - Variavél usada nas funções seguintes

int j=0 - Variavél usada nas funções seguintes

int k=0 - Variavél usada nas funções seguintes

int y=0 - Variavél usada nas funções seguintes

int h=0 - Variavél usada nas funções seguintes

int aux[npecasremover] - Usado para guardar as posições onde as peças foram removidas para depois usar para adicionar as peças novas nessa posição

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

```
  277
{
  286     char pecaremover[4];
  288     int i=0;
  290     int j=0;
  292     int k=0;
  294     int y=0;
  296     int h=0;
  298     int aux[npecasremover];
  300     char newpeca[4];
  302     char invertnewpeca[4];
  303     //Caso o numero de peças a remover seja maior que as peças totais, entra neste if
  304     if(npecasremover>pecastotal){
  305         printf("Numero de pecas a remover e superior ao baralho!\n");
  306     // Caso o numero de peças a remover seja menor que as peças totais, entra neste
if
  307     }else if(npecasremover<pecastotal){
  308         int count=0;
  309         // Insere peca a remover
  310         for(j=0;j<npecasremover;j++){
  311             printf("Insira a peca a remover:");
  312             scanf("%s",pecaremover);
  313             // Verifica se a peça do baralho/mão é igual á peça a remover, se for
substitui por X|X
  314             for(i=0;i<pecastotal;i++){
  315                 if(strcmp(baralho[i],pecaremover)==0){
  316                     strcpy(baralho[i],"X|X");
  317                     aux[k]=i;
  318                     k++;
  319                     count++;
  320                 }
  321             }
  322             // Caso a peça do baralho/mão não seja igual á peça a remover, count mantém
a zero, e entra neste if
  323             if(count==0){
```

```
  324                 printf("Essa peca nao existe, insira outra:\n");
  325                     j--;
  326             }
  327             count=0;
  328         }
  329         //Imprime mão já com as peças removidas (com pecas X|X)
  330         for(i=0;i<pecastotal;i++){
  331                 printf("%s\n",baralho[i]);
  332         }
  333         //No lugar onde foram removidas as peças, insere novas peças
  334         for(j=0;j<npecasremover;j++){
  335             printf("Insira uma nova peca:");
  336             scanf("%s",newpeca);
  337
  338             // Inverter peça inserida
  339             int d=strlen(newpeca)-1;
  340             for(h=0;h<(strlen(newpeca));h++){
  341                 invertnewpeca[d]=newpeca[h];
  342                 d--;
  343             }
  344             // Verifica se as peças da mão são iguais ás novas peças inseridas e ao
seu inverso
  345             for(i=0;i<pecastotal;i++){
  346
if((strcmp(baralho[i],newpeca)==0)||(strcmp(baralho[i],invertnewpeca)==0)){
  347                         count++;
  348                 }
  349             }
  350             // Caso sejam as peças verificadas iguais, entra neste if
  351             if(count>0){
  352                 printf("Essa peca existe!\n");
  353                 j--;
  354                 count=0;
  355             }else{
  356                 // Caso as peças verificadas não sejam iguais, ele copia a nova peça
inserida para a posição do baralho/mão onde foi removida
  357                 strcpy(baralho[aux[y]],newpeca);
  358                 y++;
  359             }
  360         }
  361         // Imprime as peças da mão do jogador
  362         for(i=0;i<pecastotal;i++){
  363                 printf("%s\n",baralho[i]);
  364         }
  365     }
  366 }
```

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

```
 1246                                                                {
 1252      char mixpecas[3000][150];
 1253
 1254     //Ver quantos baralhos sao, e guardar num array mix as pecas dos jogadores
alternadamente
 1255     if(num==2){
```

```
1256         int i=0;
1257         int j=7;
1258         int m=0;
1259 
1260         while(i<7 || j<14){
1261             if(i<7){
1262                 strcpy(mixpecas[m++],baralho[i]);
1263             }
1264             if(j<14){
1265                 strcpy(mixpecas[m++],baralho[j]);
1266             }
1267             i++;
1268             j++;
1269         }
1270     }
1271     if(num==3){
1272         int i=0;
1273         int j=7;
1274         int k=14;
1275         int m=0;
1276 
1277         while(i<7 || j<14 || k<21 ){
1278             if(i<7){
1279                 strcpy(mixpecas[m++],baralho[i]);
1280             }
1281             if(j<14){
1282                 strcpy(mixpecas[m++],baralho[j]);
1283             }
1284             if(k<21){
1285                 strcpy(mixpecas[m++],baralho[k]);
1286             }
1287             i++;
1288             j++;
1289             k++;
1290         }
1291     }
1292 
1293     if(num==4){
1294         int i=0;
1295         int j=7;
1296         int k=14;
1297         int s=21;
1298         int m=0;
1299 
1300         while(i<7 || j<14 || k<21 || s<28){
1301             if(i<7){
1302                 strcpy(mixpecas[m++],baralho[i]);
1303             }
1304             if(j<14){
1305                 strcpy(mixpecas[m++],baralho[j]);
1306             }
1307             if(k<21){
1308                 strcpy(mixpecas[m++],baralho[k]);
1309             }
1310             if(s<28){
1311                 strcpy(mixpecas[m++],baralho[s]);
1312             }
1313             i++;
1314             j++;
1315             k++;
1316             s++;
1317         }
1318     }
1319 
1320     printf("ARRAY MIX:\n");
1321     int v=0;
1322     int pecasmix=num*7;
1323     for(v=0;v<pecasmix;v++){
1324         printf("%s\n",mixpecas[v]);
1325     }
1326 
```

```
 1327     //Comeca com uma sequencia inicial e vai metendo peca a peca do mixpecas
 1328         char seqcomecar[3000];
 1329         char arrayfinalmixpecas[3000][150];
 1330         char arrayfinalmixpecascompleto[3000][150];
 1331         int i=0,j=0,count=0,x=0,y=0,p=1,z=0,contador=0;
 1332         char inversazero[150];
 1333         char inversoum[150];
 1334         char inversodois[150];
 1335         char inversotres[150];
 1336         char inversoquatro[150];
 1337         char invertfinal[150];
 1338
 1339         printf("Insira uma sequencia a comecar:\n");
 1340         scanf("%s",seqcomecar);
 1341
 1342         strcpy(arrayfinalmixpecas[0],seqcomecar);
 1343
 1344                 //Strtok da sequencia inicial
 1345                 char partidosseqcomecar[100][100];
 1346                 char *s=NULL;
 1347                 char seqapartir[100];
 1348
 1349                 strcpy(seqapartir,seqcomecar);
 1350                 s = strtok (seqapartir,"-");
 1351                 int k=0;
 1352
 1353                 while (s!= NULL)
 1354                 {
 1355                     strcpy(partidosseqcomecar[k++],s);
 1356                     s= strtok (NULL, "-");
 1357                 }
 1358
 1359                 //Verificar se as pecas da sequencia inicial sao iguais as pecas do
baralho mix
 1360                 for(x=0;x<pecasmix;x++){
 1361                     for(y=0;y<k;y++){
 1362                             // Inverter pecas baralho
 1363                             int d=strlen(mixpecas[x])-1;
 1364                             int h=0;
 1365                             for(h=0;h<(strlen(mixpecas[x]));h++){
 1366                                     inversazero[d]=mixpecas[x][h];
 1367                                     d--;
 1368                             }
 1369                             //Verifica se as pecas da seq inicial coincidem
 1370                             int tam=strlen(seqcomecar)-1;
 1371                             int i=0;
 1372                             for(i=0;i<tam;i++){
 1373                                 if(seqcomecar[i]=='-'){
 1374                                     if(seqcomecar[i-1]!=seqcomecar[i+1]){
 1375                                         contador++;
 1376                                     }
 1377                                 }
 1378                             }
 1379
if((strcmp(mixpecas[x],partidosseqcomecar[y])==0)||(strcmp(inversazero,partidosseqcomeca
r[y])==0)){
 1380                                 count++;
 1381                             }
 1382                     }
 1383                 }
 1384                 if(count>0){
 1385                     printf("As pecas da sequencia que inseriu, ja existem no
baralho!\n");
 1386                     count=0;
 1387                 }else if(contador>0){
 1388                     printf("A sequencia que inseriu esta errada!\n ");
 1389                     contador=0;
 1390                 }else{
 1391                     //Juntar pecas
 1392                     for(i=0;i<p;i++){
 1393                         int tamlin=strlen(arrayfinalmixpecas[i])-1;
```

```
 1394                            for(j=0;j<pecasmix;j++){
 1395                                //Primeira verificação
 1396                                if(arrayfinalmixpecas[i][tamlin]==mixpecas[j][0]){
 1397                                    int igual=0;
 1398                                    //inverte peca apenas para verificar
 1399                                    int w=strlen(mixpecas[j])-1;
 1400                                    for(k=0;k<(strlen(mixpecas[j]));k++){
 1401                                        inversoum[w]=mixpecas[j][k];
 1402                                        w--;
 1403                                    }
 1404                                    //strtok da peca arrayfinalmixpecas
 1405                                    char partidos[100][100];
 1406                                    char *palavra=NULL;
 1407                                    char umaseqpeca[100];
 1408
 1409                                    strcpy(umaseqpeca,arrayfinalmixpecas[i]);
 1410                                    palavra = strtok (umaseqpeca,"-");
 1411                                    int s=0;
 1412                                    while (palavra != NULL)
 1413                                    {
 1414                                        strcpy(partidos[s++],palavra);
 1415                                        palavra = strtok (NULL, "-");
 1416                                    }
 1417
 1418                                    //verifica se o arrayfinalmixpecas ou o inverso e
igual a peca
 1419                                    for (x=0;x<s;x++){
 1420
if(strcmp(partidos[x],mixpecas[j])==0||strcmp(partidos[x],inversoum)==0){
 1421                                                igual++;
 1422                                            }
 1423                                    }
 1424                                    if(igual==0){
 1425                                        //concatena para arrayfinalmixpecas
 1426                                                strcpy(arrayfinalmixpecas[p],
arrayfinalmixpecas[i]);
 1427                                                strcat(arrayfinalmixpecas[p],"-");
 1428                                                strcat(arrayfinalmixpecas[p],
mixpecas[j]);
 1429                                                p++;
 1430                                    }
 1431                                }
 1432                                //Segunda verificação
 1433                                if(arrayfinalmixpecas[i][tamlin]==mixpecas[j][2]){
 1434                                    int igual=0;
 1435                                    //inverte peca
 1436                                    int w=strlen(mixpecas[j])-1;
 1437                                    for(k=0;k<(strlen(mixpecas[j]));k++){
 1438                                        inversodois[w]=mixpecas[j][k];
 1439                                        w--;
 1440                                    }
 1441                                    //strtok da peca arrayfinalmixpecas
 1442                                    char partidos[100][100];
 1443                                    char *palavra=NULL;
 1444                                    char umaseqpeca[100];
 1445
 1446                                    strcpy(umaseqpeca,arrayfinalmixpecas[i]);
 1447                                    palavra = strtok (umaseqpeca,"-");
 1448                                    int s=0;
 1449                                    while (palavra != NULL)
 1450                                    {
 1451                                        strcpy(partidos[s++],palavra);
 1452                                        palavra = strtok (NULL, "-");
 1453                                    }
 1454                                    //verifica se o partidos ou o inverso e igual a peca
 1455                                    for (x=0;x<s;x++){
 1456
if(strcmp(partidos[x],mixpecas[j])==0||strcmp(partidos[x],inversodois)==0){
 1457                                                igual++;
 1458                                            }
 1459                                    }
```

```
 1460                                    if(igual==0){
 1461                                        //concatena
 1462                                        strcpy(arrayfinalmixpecas[p],
arrayfinalmixpecas[i]);
 1463                                        strcat(arrayfinalmixpecas[p],"-");
 1464                                        strcat(arrayfinalmixpecas[p], inversodois);
 1465                                        p++;
 1466                                    }
 1467                                }
 1468                                //Terceira verificação
 1469                                if(arrayfinalmixpecas[i][0]==mixpecas[j][0]){
 1470                                    int igual=0;
 1471                                    //inverte peca
 1472                                    int w=strlen(mixpecas[j])-1;
 1473                                    for(k=0;k<(strlen(mixpecas[j]));k++){
 1474                                        inversotres[w]=mixpecas[j][k];
 1475                                        w--;
 1476                                    }
 1477                                    //strtok da peca arrayfinal
 1478                                    char partidos[100][100];
 1479                                    char *palavra=NULL;
 1480                                    char umaseqpeca[100];
 1481
 1482                                    strcpy(umaseqpeca,arrayfinalmixpecas[i]);
 1483                                    palavra = strtok (umaseqpeca,"-");
 1484                                    int s=0;
 1485                                    while (palavra != NULL)
 1486                                    {
 1487                                        strcpy(partidos[s++],palavra);
 1488                                        palavra = strtok (NULL, "-");
 1489                                    }
 1490                                    //verifica se o partidos ou o inverso é igual a peca
 1491                                    for (x=0;x<s;x++){
 1492
if(strcmp(partidos[x],mixpecas[j])==0||strcmp(partidos[x],inversotres)==0){
 1493                                                        igual++;
 1494                                            }
 1495                                    }
 1496                                    if(igual==0){
 1497                                        //concatena
 1498                                        strcpy(arrayfinalmixpecas[p], inversotres);
 1499                                        strcat(arrayfinalmixpecas[p],"-");
 1500                                        strcat(arrayfinalmixpecas[p],
arrayfinalmixpecas[i]);
 1501                                        p++;
 1502                                    }
 1503                                }
 1504                                //Quarta verificação
 1505                                if(arrayfinalmixpecas[i][0]==mixpecas[j][2]){
 1506                                    int igual=0;
 1507
 1508                                    //inverte peca
 1509                                    int w=strlen(mixpecas[j])-1;
 1510                                    for(k=0;k<(strlen(mixpecas[j]));k++){
 1511                                        inversoquatro[w]=mixpecas[j][k];
 1512                                        w--;
 1513                                    }
 1514                                    //strtok da peca arrayfinal
 1515                                    char partidos[100][100];
 1516                                    char *palavra=NULL;
 1517                                    char umaseqpeca[100];
 1518
 1519                                    strcpy(umaseqpeca,arrayfinalmixpecas[i]);
 1520                                    palavra = strtok (umaseqpeca,"-");
 1521                                    int s=0;
 1522                                    while (palavra != NULL)
 1523                                    {
 1524                                        strcpy(partidos[s++],palavra);
 1525                                        palavra = strtok (NULL, "-");
 1526                                    }
 1527                                    //verifica se o partidos ou o inverso é igual a peca
```

```
 1528                                for (x=0;x<s;x++){
 1529
if(strcmp(partidos[x],mixpecas[j])==0||strcmp(partidos[x],inversoquatro)==0){
 1530                                                    igual++;
 1531                                            }
 1532                                }
 1533                                if(igual==0){
 1534                                    //concatena
 1535                                    strcpy(arrayfinalmixpecas[p], mixpecas[j]);
 1536                                    strcat(arrayfinalmixpecas[p],"-");
 1537                                    strcat(arrayfinalmixpecas[p],
arrayfinalmixpecas[i]);
 1538                                    p++;
 1539                                }
 1540                            }
 1541                        }
 1542                    }
 1543                //verificar se ha seq iguais e o inversos tambem
 1544                    for(x=0;x<p;x++){
 1545                                // Inverter peca
 1546                                int w=strlen(arrayfinalmixpecas[x])-1;
 1547                                for(k=0;k<(strlen(arrayfinalmixpecas[x]));k++){
 1548                                    invertfinal[w]=arrayfinalmixpecas[x][k];
 1549                                    w--;
 1550                                }
 1551
 1552                    // Verifica se e igual e se for substitui por X|X
 1553                            for(i=x+1;i<p;i++){
 1554
if((strcmp(arrayfinalmixpecas[i],invertfinal)==0)||(strcmp(arrayfinalmixpecas[i],arrayfi
nalmixpecas[x])==0)){
 1555                                    strcpy(arrayfinalmixpecas[i],"X|X");
 1556                                }
 1557                            }
 1558                    }
 1559                    // Copia do arrayfinalmixpecas que contem x|x e guarda num array
final(arrayfinalmixpecascompleto) as sequencias possiveis
 1560                     for(z=0;z<p;z++){
 1561                            if(strcmp(arrayfinalmixpecas[z],"X|X")==0){
 1562                            }else{
 1563
strcpy(arrayfinalmixpecascompleto[y],arrayfinalmixpecas[z]);
 1564                                y++;
 1565                            }
 1566                    }
 1567                }
 1568                for(i=0;i<y;i++){
 1569                    printf("%s\n",arrayfinalmixpecascompleto[i]);
 1570                }
 1571 }
```

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

```
  988                                           {
  993         char seqcomecar[3000];
  994         char arrayseqinicial[3000][150];
  995         char arrayseqinicialcompleto[3000][150];
  996         int i=0,j=0,count=0,x=0,y=0,p=1,z=0,contador=0;
```

```
  997         char inversazero[150];
  998         char inversoum[150];
  999         char inversodois[150];
 1000         char inversotres[150];
 1001         char inversoquatro[150];
 1002         char invertfinal[150];
 1003
 1004         printf("Insira uma sequencia a comecar:\n");
 1005         scanf("%s",seqcomecar);
 1006
 1007         strcpy(arrayseqinicial[0],seqcomecar);
 1008
 1009                 //Strtok da sequencia inicial
 1010                 char partidosseqcomecar[100][100];
 1011                 char *s=NULL;
 1012                 char seqapartir[100];
 1013                 strcpy(seqapartir,seqcomecar);
 1014                 s = strtok (seqapartir,"-");
 1015                 int k=0;
 1016                 while (s!= NULL)
 1017                 {
 1018                     strcpy(partidosseqcomecar[k++],s);
 1019                     s= strtok (NULL, "-");
 1020                 }
 1021                 //Verificar se as pecas da sequencia inicial sao iguais as pecas do
baralho
 1022                 for(x=0;x<7;x++){
 1023                     for(y=0;y<k;y++){
 1024                             // Inverter pecas baralho
 1025                             int d=strlen(baralho[x])-1;
 1026                             int h=0;
 1027                             for(h=0;h<(strlen(baralho[x]));h++){
 1028                                     inversazero[d]=baralho[x][h];
 1029                                     d--;
 1030                             }
 1031                             //Verifica se as pecas da seq inicial coincidem
 1032                             int tam=strlen(seqcomecar)-1;
 1033                             int i=0;
 1034                             for(i=0;i<tam;i++){
 1035                                 if(seqcomecar[i]=='-'){
 1036                                     if(seqcomecar[i-1]!=seqcomecar[i+1]){
 1037                                         contador++;
 1038                                     }
 1039                                 }
 1040                             }
 1041
 1042
if((strcmp(baralho[x],partidosseqcomecar[y])==0)||(strcmp(inversazero,partidosseqcomecar
[y])==0)){
 1043                                 count++;
 1044                             }
 1045                     }
 1046                 }
 1047                 if(count>0){
 1048                     printf("As pecas da sequencia que inseriu, ja existem no
baralho!\n");
 1049                     count=0;
 1050                 }else if(contador>0){
 1051                     printf("A sequencia que inseriu esta errada!\n ");
 1052                     contador=0;
 1053                 }else{
 1054
 1055                     // Juntar pecas
 1056                     for(i=0;i<p;i++){
 1057                         int tamlin=strlen(arrayseqinicial[i])-1;
 1058                         for(j=0;j<7;j++){
 1059
 1060                             //Primeira verificação
 1061                             if(arrayseqinicial[i][tamlin]==baralho[j][0]){
 1062                                 int igual=0;
 1063                                 //inverte peca apenas para verificar
```

```
 1064                                    int w=strlen(baralho[j])-1;
 1065                                    for(k=0;k<(strlen(baralho[j]));k++){
 1066                                        inversoum[w]=baralho[j][k];
 1067                                        w--;
 1068                                    }
 1069                                    //strtok da peca arrayfinal
 1070                                    char partidos[100][100];
 1071                                    char *palavra=NULL;
 1072                                    char umaseqpeca[100];
 1073
 1074                                    strcpy(umaseqpeca,arrayseqinicial[i]);
 1075                                    palavra = strtok (umaseqpeca,"-");
 1076                                    int s=0;
 1077                                    while (palavra != NULL)
 1078                                    {
 1079                                        strcpy(partidos[s++],palavra);
 1080                                        palavra = strtok (NULL, "-");
 1081                                    }
 1082
 1083                                    //verifica se o partidos ou o inverso é igual a peça
 1084                                    for (x=0;x<s;x++){
 1085
if(strcmp(partidos[x],baralho[j])==0||strcmp(partidos[x],inversoum)==0){
 1086                                                        igual++;
 1087                                                }
 1088                                    }
 1089                                    if(igual==0){
 1090                                                //concatena para arrayseqinicial
 1091                                                strcpy(arrayseqinicial[p],
arrayseqinicial[i]);
 1092                                                strcat(arrayseqinicial[p],"-");
 1093                                                strcat(arrayseqinicial[p],
baralho[j]);
 1094                                                p++;
 1095                                    }
 1096
 1097                                }
 1098                                //Segunda verificação
 1099                                if(arrayseqinicial[i][tamlin]==baralho[j][2]){
 1100                                    int igual=0;
 1101                                    //inverte peca
 1102                                    int w=strlen(baralho[j])-1;
 1103                                    for(k=0;k<(strlen(baralho[j]));k++){
 1104                                        inversodois[w]=baralho[j][k];
 1105                                        w--;
 1106                                    }
 1107                                    //strtok da peca arrayseqinicial
 1108                                    char partidos[100][100];
 1109                                    char *palavra=NULL;
 1110                                    char umaseqpeca[100];
 1111
 1112                                    strcpy(umaseqpeca,arrayseqinicial[i]);
 1113                                    palavra = strtok (umaseqpeca,"-");
 1114                                    int s=0;
 1115                                    while (palavra != NULL)
 1116                                    {
 1117                                        strcpy(partidos[s++],palavra);
 1118                                        palavra = strtok (NULL, "-");
 1119                                    }
 1120
 1121                                    //verifica se o partidos ou o inverso é igual a peça
 1122                                    for (x=0;x<s;x++){
 1123
if(strcmp(partidos[x],baralho[j])==0||strcmp(partidos[x],inversodois)==0){
 1124                                                        igual++;
 1125                                                }
 1126                                    }
 1127                                    if(igual==0){
 1128                                        //concatena
 1129                                        strcpy(arrayseqinicial[p],
arrayseqinicial[i]);
```

```
 1130                                        strcat(arrayseqinicial[p],"-");
 1131                                        strcat(arrayseqinicial[p], inversodois);
 1132                                        p++;
 1133                                    }
 1134                                }
 1135                                //Terceira verificação
 1136                                if(arrayseqinicial[i][0]==baralho[j][0]){
 1137                                    int igual=0;
 1138
 1139                                    //inverte peca
 1140                                    int w=strlen(baralho[j])-1;
 1141                                    for(k=0;k<(strlen(baralho[j]));k++){
 1142                                        inversotres[w]=baralho[j][k];
 1143                                        w--;
 1144                                    }
 1145                                    //strtok da peca arrayfinal
 1146                                    char partidos[100][100];
 1147                                    char *palavra=NULL;
 1148                                    char umaseqpeca[100];
 1149
 1150                                    strcpy(umaseqpeca,arrayseqinicial[i]);
 1151                                    palavra = strtok (umaseqpeca,"-");
 1152                                    int s=0;
 1153                                    while (palavra != NULL)
 1154                                    {
 1155                                        strcpy(partidos[s++],palavra);
 1156                                        palavra = strtok (NULL, "-");
 1157                                    }
 1158
 1159                                    //verifica se o partidos ou o inverso é igual a peca
 1160                                    for (x=0;x<s;x++){
 1161
if(strcmp(partidos[x],baralho[j])==0||strcmp(partidos[x],inversotres)==0){
 1162                                                        igual++;
 1163                                                }
 1164                                    }
 1165                                    if(igual==0){
 1166                                        //concatena
 1167                                        strcpy(arrayseqinicial[p], inversotres);
 1168                                        strcat(arrayseqinicial[p],"-");
 1169                                        strcat(arrayseqinicial[p],
arrayseqinicial[i]);
 1170                                        p++;
 1171                                    }
 1172                                }
 1173                                //Quarta verificação
 1174                                if(arrayseqinicial[i][0]==baralho[j][2]){
 1175                                    int igual=0;
 1176
 1177                                    //inverte peca
 1178                                    int w=strlen(baralho[j])-1;
 1179                                    for(k=0;k<(strlen(baralho[j]));k++){
 1180                                        inversoquatro[w]=baralho[j][k];
 1181                                        w--;
 1182                                    }
 1183
 1184                                    //strtok da peca arrayfinal
 1185                                    char partidos[100][100];
 1186                                    char *palavra=NULL;
 1187                                    char umaseqpeca[100];
 1188
 1189                                    strcpy(umaseqpeca,arrayseqinicial[i]);
 1190                                    palavra = strtok (umaseqpeca,"-");
 1191                                    int s=0;
 1192                                    while (palavra != NULL)
 1193                                    {
 1194                                        strcpy(partidos[s++],palavra);
 1195                                        palavra = strtok (NULL, "-");
 1196                                    }
 1197
```

```
 1198                                    //verifica se o arraypartidos ou o invero e igual a
peca
 1199                                    for (x=0;x<s;x++){
 1200
if(strcmp(partidos[x],baralho[j])==0||strcmp(partidos[x],inversoquatro)==0){
 1201                                                            igual++;
 1202                                                    }
 1203                                    }
 1204
 1205                                    if(igual==0){
 1206                                        //concatena
 1207                                        strcpy(arrayseqinicial[p], baralho[j]);
 1208                                        strcat(arrayseqinicial[p],"-");
 1209                                        strcat(arrayseqinicial[p],
arrayseqinicial[i]);
 1210                                        p++;
 1211                                    }
 1212                                }
 1213                            }
 1214                        }
 1215                    //verificar se ha sequencias iguais e o inversos tambem
 1216                        for(x=0;x<p;x++){
 1217                                    // Inverter peca
 1218                                    int w=strlen(arrayseqinicial[x])-1;
 1219
 1220                                    for(k=0;k<(strlen(arrayseqinicial[x]));k++){
 1221                                        invertfinal[w]=arrayseqinicial[x][k];
 1222                                        w--;
 1223                                    }
 1224                        // Verifica se e igual e se for substitui por X|X
 1225                                for(i=x+1;i<p;i++){
 1226
if((strcmp(arrayseqinicial[i],invertfinal)==0)||(strcmp(arrayseqinicial[i],arrayseqinici
al[x])==0)){
 1227                                        strcpy(arrayseqinicial[i],"X|X");
 1228                                    }
 1229                                }
 1230                        }
 1231                        // Copia do aux que contem x|x e guarda num array
final(arrayseqinicialcompleto) as sequencias das pecas
 1232                         for(z=0;z<p;z++){
 1233                                if(strcmp(arrayseqinicial[z],"X|X")==0){
 1234                                }else{
 1235
strcpy(arrayseqinicialcompleto[y],arrayseqinicial[z]);
 1236                                    y++;
 1237                                }
 1238                        }
 1239                    }
 1240                    for(i=0;i<y;i++){
 1241                        printf("%s\n",arrayseqinicialcompleto[i]);
 1242                    }
 1243 }
```