

Projeto Final LP1

DOMINO

Tiago Martins
Docente – Rui Moreira
Versão 2.0

Domingo, 27 de Dezembro de 2016

Índice

Table of contents

Índice das estruturas de dados

Estruturas de dados

Lista das estruturas de dados com uma breve descrição:

ordena	4
peca	5
pecaint	6
pecasinit	7
seq	9

Índice dos ficheiros

Lista de ficheiros

Lista de todos os ficheiros com uma breve descrição:

main.c	10
projeto.c	11
projeto.h	103

Documentação da classe

Referência à estrutura ordena

```
#include <projeto.h>
```

Campos de Dados

- int tamanho
- int indice

Documentação dos campos e atributos

int indice

int tamanho

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

- projeto.h

Referência à estrutura peca

```
#include <projeto.h>
```

Campos de Dados

- char * str
- struct peca * pnext

Documentação dos campos e atributos

struct peca* pnext

char* str

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

- projeto.h

Referência à estrutura pecaint

```
#include <projeto.h>
```

Campos de Dados

- `int direito`
 - `int esquerdo`
 - `struct pecaint * pnext`
-

Documentação dos campos e atributos

`int direito`

`int esquerdo`

`struct pecaint* pnext`

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

- `projeto.h`

Referência à estrutura pecasinit

```
#include <projeto.h>
```

Campos de Dados

- **PECA** * pfirst
 - **PECAINT** * pfirstint
 - **SEQ** * seqf
 - **SEQ** * seqss
 - **SEQ** * baralhoaux
 - **SEQ** * seqssaux
 - **ORDENA** * ordenaseq
 - int npecasint
 - int npecas
 - int nseqf
 - int nseqss
 - int nbaralhoaux
 - int nseqssaux
 - int nordena
-

Documentação dos campos e atributos

SEQ* baralhoaux

int nbaralhoaux

int nordena

int npecas

int npecasint

int nseqf

int nseqss

int nseqssaux

ORDENA* ordenaseq

PECA* pfirst

PECAINT* pfirstint

SEQ* seqf

SEQ* seqss

SEQ* seqssaux

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

- projeto.h

Referência à estrutura seq

```
#include <projeto.h>
```

Campos de Dados

- `char * seqstr`
-

Documentação dos campos e atributos

`char* seqstr`

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

- `projeto.h`

Documentação do ficheiro

Referência ao ficheiro main.c

```
#include "projeto.h"
```

Funções

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

Documentação das funções

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

Função main.

função que contém a funcao que chama o **main()** do meu projeto

Parâmetros:

<i>int</i>	argc não está a ser usado mas contém o numero de posicoes do array de strings argv[] usadas,
<i>char</i>	*argv[] não está a ser usado, mas contém o numero de posições do array de strings

Retorna:

```
0
64 {
65
73     main_domino (argc, argv) ;
74     return 0;
75
76 }
```

Referência ao ficheiro projeto.c

```
#include "projeto.h"
```

Funções

- **int main_domino** (int argc, char *argv[])
Função main.
- **void save_jogo_bin** (PECASINIT p, char fname[])
Função save_jogo_bin.
- **void load_jogo_bin** (PECASINIT *p, char fname[])
Load do ficheiro binario.
- **void load_txt_jogo** (PECASINIT *p, char fname[])
Load do ficheiro txt.
- **void save_txt_jogo** (PECASINIT p, char fname[])
Função save_jogo_txt.
- **void create_array_baralhoaux** (PECASINIT *p, int n)
Criar array dinamico de sequencias.
- **void create_array_seqss** (PECASINIT *p, int n)
Criar array dinamico de sequencias.
- **void create_array_ordenaseq** (PECASINIT *p, int n)
Criar array dinamico de sequencias.
- **void create_array_seqssaux** (PECASINIT *p, int n)
Criar array dinamico de sequencias.
- **void create_array_seqf** (PECASINIT *p, int n)
Criar array dinamico de sequencias.
- **char * create_dyn_string** (char str[])
Criar string dinamica.
- **void inserir_pecaint** (PECASINIT *p, int dir, int esq)
Inserir pecas inteiras na lista ligada do baralho.
- **void inserir_ordenaseq** (PECASINIT *p, int tam, int index)
Inserir sequencias de pecas no array dinamico da funcao ordenar.
- **void inserir_peca** (PECASINIT *p, char pecanova[COLSTR])
Inserir pecas na lista ligada do baralho.
- **int preenchebaralhosstruct** (char pecass[][COLSTR], PECASINIT *p, int n)
- **int verificasequencia** (char seq[])
Função verificasequencia.
- **int * eliminarrep** (int *pv, PECASINIT *p)
Função eliminarrep.
- **int separarseqinvertidasstruct** (PECASINIT *p)
Função separarseqinvertidas struct.
- **int separarseqinvertidas** (char seqf[][COLSEQ], int numdeseq)
Função separarseqinvertidas.
- **void criapecasint** (int pecasi[][COL], int size)
Função criapecasint.
- **void criapecasstr** (char pecass[][COLSTR], int s)
Função criapecasstr.

- **int entregarbaralhos** (char pecass[][COLSTR], PECASINIT *p, int n)
Função entregarbaralhos.
- **void printpecasint** (int pecasi[][COL], int l, int c, int inicio)
Função printpecasint.
- **void printpecasintstruct** (PECASINIT p, int inicio, int fim)
Função printpecasintstruct.
- **void printpecasstrstruct** (PECASINIT p, int inicio, int fim)
Função printpecasintstruct.
- **void printpecasstr** (char pecass[][COLSTR], int inicio, int fim)
Função printpecasstr.
- **void printseqstr** (char seqss[][COLSEQ], int inicio, int fim)
Função printseqstr.
- **void printseqstrstruct** (PECASINIT p)
Função printpecasstrstruct.
- **void mostrarjogosstrstruct** (PECASINIT p, int njogos)
Função mostrarjogosstrstruct.
- **void mostrarjogosstr** (char baralhoss[][COLSTR], int njogos)
- **void mostrarjogosintstruct** (PECASINIT p, int njogos)
Função mostrarjogosstr.
- **void mostrarjogosint** (int baralhosi[][COL], int njogos)
Função mostrarjogosint.
- **void esvaziabaralhoint** (int baralhosi[][COL], int lin, int col)
Função esvaziabaralhoint.
- **void esvaziabaralhostrstruct** (PECASINIT *p)
Função esvaziabaralhostr.
- **void esvaziabaralhostr** (char baralhoss[][COLSTR], int size)
Função esvaziabaralhostr.
- **void esvaziaseqstr** (char seqss[][COLSEQ], int size)
Função esvaziaseqstr.
- **int convertestrtoint** (PECASINIT *p, int num)
Função convertestrtoint.
- **int converteinttostr** (PECASINIT *p, int num)
Função converteinttostr.
- **int remover** (PECASINIT *p, char pecass[][COLSTR], int num)
Função remover.
- **void remove_seqf** (PECASINIT *p)
- **void remove_seqss** (PECASINIT *p)
- **void remove_seqssaux** (PECASINIT *p)
- **void remove_pecas** (PECASINIT *p, char remove[])
Remover pecas dos jogos.
- **PECA * find_pecas_baralho** (PECASINIT *p, char aux[])
Procurar peca no baralho.
- **void inserir_seqss** (PECASINIT *p, char sequencia[])
Inserir sequencias no array dinamico das seqs auxiliares.
- **void inserir_seqssaux** (PECASINIT *p, char sequencia[])
Inserir sequencias no array dinamico das seqs auxiliares 2.

- void **inserir_seqf** (PECASINIT *p, char sequencia[])
Inserir sequencias no array dinamico das seqs.
- void **inserir_baralhoaux** (PECASINIT *p, char sequencia[])
Inserir sequencias de pecas no array dinamico.
- int **seq** (PECASINIT *p, int num)
Função seq.
- char * **inverterstr** (char str1[], char str2[])
Função inverterstr.
- int **prosubseq_ausar** (char seqf[][COLSEQ], int size, char subs[])
Função prosubseq_ausar.
- int **prosubseq** (char seqf[][COLSEQ], int size, char subs[])
Função prosubseq.
- int **strtoque** (char stra[][COLSEQ], char str[], char car)
Função strtoque.
- void **ordenarmatrizinteiros** (int m[][2], int size)
Função ordenarmatrizinteiros.
- void **ordenarsequencias** (char seqf[][COLSEQ], int size)
Função ordenarsequencias.
- void **ordenarsequenciasstruct** (PECASINIT *p)
Função ordenarsequencias struct.
- int **prosubseq_trocapadrao** (char seqf[][COLSEQ], int size, char subs[LIN], int l)
Função prosubseq_trocapadrao.
- void **trocapadrao** (char seqf[][COLSEQ], int size, char padrao[], char padraon[], char seqfpadrao[][COLSEQ], int *sizeseqfpadrao)
Função trocapadrao.
- int **tirartracosinvertidos** (char seqf[][COLSEQ], int numdeseq)
Função tirartracosinvertidos.
- int **seqcomseqinicial** (char baralhoss[][COLSTR], char seqf[][COLSEQ], int num, char seqinit[])
Função seqcomseqinicial.
- int **retiraseqinitrepetida** (char seqf[][COLSEQ], int size, char seqinit[])
Função retiraseqinitrepetida.
- int **jogoadois** (char baralhoss[][COLSTR], char seqf[][COLSEQ], int num, char seqinit[])
Função jogoadois.
- int **baralhoausar** (char baralhoss[][COLSTR], char baralhoaux[][COLSEQ], int numero)
Função baralhoausar.

Documentação das funções

int **baralhoausar** (char *baralhoss*[][COLSTR], char *baralhoaux*[][COLSEQ], int *numero*)

Função baralhoausar.

esta funcao recebe as pecas de um baralho e acrescenta as invertidas

Parâmetros:

<i>char</i>	baralhoss[][COLSTR] baralhos do jogador em string
<i>char</i>	baralhoaux[][COLSEQ] baralhos do jogador em string auxiliar na qual guardo também as peças invertidas
<i>int</i>	numero contém o numero do baralho ao qual vamos acrescentar as pecas invertidas também

Retorna:

retorna o numero de pecas do baralho do jogador com as pecas invertidas incluidas e retirando as repetidas

```

5858 {
5859     int i=0;
5860     char auxdir=' ';
5861     int invertidos=0;
5862     int k=0;
5863     int init=0;
5864
5865     esvaziaseqstr(baralhoaux,LINSEQ);
5866
5867     for(i=1; i<numero; i++)
5868     {
5869         init=init+7;
5870     }
5871
5872     for(i=init; i<(numero*7); i++)
5873     {
5874         strcpy(baralhoaux[k],baralhoss[i]);
5875         k++;
5876     }
5877
5878     invertidos=7;
5879
5880     //acrescentar pecas ao contrario
5881     i=0;
5882     while(i<7)
5883     {
5884         if(baralhoaux[i][0]==baralhoaux[i][2])
5885         {
5886             i++;
5887         }
5888         else
5889         {
5890             strcpy(baralhoaux[invertidos],baralhoaux[i]);
5891             auxdir=baralhoaux[invertidos][0];
5892             baralhoaux[invertidos][0]=baralhoaux[invertidos][2];
5893             baralhoaux[invertidos][2]=auxdir;
5894             invertidos++;
5895             i++;
5896         }
5897     }
5898
5899     return invertidos;
5900 }

```

int converteinttostr (PECASINIT * p, int num)

Função converteinttostr.

esta funcao converte as peças inteiras em string

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>int</i>	num numero de jogos a converter

Retorna:

retorna o numero de jogos a imprimir

```
3399 {
3406     //converte as peças inteiras em string
3407
3408     char aux[4];
3409     int i=0;
3410     num=num*7;
3411
3412     PECAINT *paux=NULL;
3413
3414     paux=p->pfirstint;
3415
3416     while (i<(p->npecasint))
3417     {
3418         aux[0]='0'+paux->direito;
3419         aux[1]='|';
3420         aux[2]='0'+paux->esquerdo;
3421
3422         inserir_pecas(p,aux);
3423
3424         paux=paux->pnext;
3425
3426         i++;
3427     }
3428
3429     return num;
3430
3431 }
```

int convertestrtoint (PECASINIT * p, int num)

Função convertestrtoint.

esta funcao converte as pecas do jogador de string para inteiros

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>int</i>	num numero de jogos a converter

Retorna:

retorna o numero de jogos a imprimir

```
3355 {
3356
3364     //converte as peças dos jogadores de strings para inteiros
3365
3366     char aux1[4];
3367     char aux2[4];
3368
3369     num=num*7;
3370
3371     PECA *paux=NULL;
3372
3373     paux=p->pfirst;
3374     while (paux!=NULL)
```

```

3375     {
3376
3377         strcpy(aux1,paux->str);
3378         strcpy(aux2,paux->str);
3379         aux1[1]='\0';
3380         aux2[0]=aux2[2];
3381         aux2[1]='\0';
3382         inserir_pecaint(p,atoi(aux1),atoi(aux2));
3383         paux=paux->pnext;
3384     }
3385
3386
3387
3388
3389     return num;
3390
3391
3392 }

```

void create_array_baralhoaux (PECASINIT * p, int n)

Criar array dinamico de sequencias.

função para colocar sequencias no array dinâmico

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>int</i>	n contem a qtde a alocar

```

1733 {
1734
1741     SEQ *pnew=NULL;
1742     SEQ *paux=NULL;
1743
1744     int i=0;
1745     int j=0;
1746
1747     if (p->baralhoaux==NULL || p->nbaralhoaux==0)
1748     {
1749
1750         pnew=(SEQ*)malloc(sizeof(SEQ)*n);
1751         p->nbaralhoaux=n;
1752
1753         for(i=0; i<n; i++)
1754         {
1755             (pnew+i)->seqstr=NULL;
1756         }
1757
1758         p->baralhoaux=pnew;
1759
1760     }
1761     else
1762     {
1763
1764         p->nbaralhoaux=n;
1765         pnew=(SEQ*)malloc(sizeof(SEQ)*n);
1766         paux=p->baralhoaux;
1767
1768         for(i=0; i<(p->nbaralhoaux-2); i++)
1769         {
1770             (pnew+i)->seqstr=(paux+i)->seqstr;
1771         }
1772
1773
1774
1775
1776     }

```

```

1777
1778         for (j=i; j<(p->nbaralhoaux); j++)
1779         {
1780
1781             (pnew+j)->seqstr=NULL;
1782
1783         }
1784         p->baralhoaux=pnew;
1785
1786     }
1787 }
1788
1789 }

```

void create_array_ordenaseq (PECASINIT * p, int n)

Criar array dinamico de sequencias.

função para colocar sequencias no array dinâmico

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>int</i>	n contem a qtde a alocar

```

1858 {
1859
1866     ORDENA *pnew=NULL;
1867     ORDENA *paux=NULL;
1868
1869     int i=0;
1870     int j=0;
1871
1872     if (p->ordenaseq==NULL || p->nordena==0)
1873     {
1874
1875         pnew= (ORDENA*) malloc (sizeof (ORDENA) *n) ;
1876         p->nordena=n;
1877
1878         for (i=0; i<n; i++)
1879         {
1880
1881             (pnew+i)->indice=0;
1882             (pnew+i)->tamanho=0;
1883
1884         }
1885
1886         p->ordenaseq=pnew;
1887     }
1888     else
1889     {
1890
1891
1892         p->nordena=n;
1893         pnew= (ORDENA*) malloc (sizeof (ORDENA) *n) ;
1894         paux=p->ordenaseq;
1895
1896         for (i=0; i<(p->nordena-2); i++)
1897         {
1898
1899             (pnew+i)->tamanho= (paux+i)->tamanho;
1900             (pnew+i)->indice= (paux+i)->indice;
1901
1902         }
1903
1904         for (j=i; j<(p->nordena); j++)
1905         {

```

```

1907
1908         (pnew+j)->tamanho=0;
1909         (pnew+j)->indice=0;
1910     }
1911
1912     p->ordenaseq=pnew;
1913
1914 }
1915
1916
1917 }

```

void create_array_seqf (PECASINIT * p, int n)

Criar array dinamico de sequencias.

função para colocar sequencias no array dinâmico

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>int</i>	n contem a qtde a alocar

```

1983 {
1984
1991     SEQ *pnew=NULL;
1992     SEQ *paux=NULL;
1993
1994     int i=0;
1995     int j=0;
1996
1997     if (p->seqf==NULL || p->nseqf==0)
1998     {
1999
2000         pnew=(SEQ*) malloc (sizeof (SEQ) *n) ;
2001         p->nseqf=n;
2002
2003         for(i=0; i<n; i++)
2004         {
2005
2006             (pnew+i)->seqstr=NULL;
2007
2008         }
2009
2010         p->seqf=pnew;
2011
2012     }
2013     else
2014     {
2015
2016         p->nseqf=n;
2017         pnew=(SEQ*) malloc (sizeof (SEQ) *n) ;
2018         paux=p->seqf;
2019
2020         for(i=0; i<(p->nseqf-2); i++)
2021         {
2022
2023             (pnew+i)->seqstr=(paux+i)->seqstr;
2024
2025         }
2026
2027         for(j=i; j<(p->nseqf); j++)
2028         {
2029
2030             (pnew+j)->seqstr=NULL;
2031
2032         }
2033     }

```

```

2034
2035         p->seqf=pnew;
2036
2037     }
2038
2039 }

```

void create_array_seqss (PECASINIT * p, int n)

Criar array dinamico de sequencias.

função para colocar sequencias no array dinâmico

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>int</i>	n contem a qtde a alocar

```

1795 {
1796
1803     SEQ *pnew=NULL;
1804     SEQ *paux=NULL;
1805
1806     int i=0;
1807     int j=0;
1808
1809     if (p->seqss==NULL || p->nseqss==0)
1810     {
1811
1812         pnew=(SEQ*)malloc(sizeof(SEQ)*n);
1813         p->nseqss=n;
1814
1815         for(i=0; i<n; i++)
1816         {
1817
1818             (pnew+i)->seqstr=NULL;
1819
1820         }
1821
1822         p->seqss=pnew;
1823
1824     }
1825     else
1826     {
1827
1828         p->nseqss=n;
1829         pnew=(SEQ*)malloc(sizeof(SEQ)*n);
1830         paux=p->seqss;
1831
1832         for(i=0; i<(p->nseqss-2); i++)
1833         {
1834
1835             (pnew+i)->seqstr=(paux+i)->seqstr;
1836
1837         }
1838
1839         for(j=i; j<(p->nseqss); j++)
1840         {
1841
1842             (pnew+j)->seqstr=NULL;
1843
1844         }
1845
1846         p->seqss=pnew;
1847
1848     }
1849 }
1850

```

```
1851 }
```

void create_array_seqssaux (PECASINIT * p, int n)

Criar array dinamico de sequencias.

função para colocar sequencias no array dinâmico

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>int</i>	n contem a qtde a alocar

```
1922 {
1923
1930     SEQ *pnew=NULL;
1931     SEQ *paux=NULL;
1932
1933     int i=0;
1934     int j=0;
1935
1936     if (p->seqssaux==NULL || p->nseqssaux==0)
1937     {
1938
1939         pnew=(SEQ*) malloc (sizeof (SEQ) *n) ;
1940         p->nseqssaux=n;
1941
1942         for (i=0; i<n; i++)
1943         {
1944
1945             (pnew+i)->seqstr=NULL;
1946
1947         }
1948
1949         p->seqssaux=pnew;
1950
1951     }
1952     else
1953     {
1954
1955         p->nseqssaux=n;
1956         pnew=(SEQ*) malloc (sizeof (SEQ) *n) ;
1957         paux=p->seqssaux;
1958
1959         for (i=0; i<(p->nseqssaux-2); i++)
1960         {
1961
1962             (pnew+i)->seqstr=(paux+i)->seqstr;
1963
1964
1965         }
1966
1967         for (j=i; j<(p->nseqssaux); j++)
1968         {
1969
1970             (pnew+j)->seqstr=NULL;
1971
1972         }
1973
1974         p->seqssaux=pnew;
1975
1976     }
1977
1978 }
```

char* create_dyn_string (char str[])

Criar string dinamica.

função para criar string dinamica

Parâmetros:

<i>char</i>	str[] string a tornar dinamica
2045 {	
2046	
2052 char *paux=NULL;	
2053 int slen = strlen(str)+1;	
2054	
2055 paux=(char*)malloc(sizeof(char)*slen);	
2056	
2057 strcpy(paux,str);	
2058	
2059 return paux;	
2060	
2061 }	

void criapecasint (int *pecas*[][COL], int *size*)

Função criapecasint.

função que cria as pecas em inteiros

Parâmetros:

<i>char</i>	pecas[][COLSTR] array que guarda as pecas em int
<i>int</i>	size contém o numero de pecas a ser guardadas
2662 {	
2663	
2671 int dominomax=6;	
2672 int cont=0;	
2673 int linha=0;	
2674 int coluna=0;	
2675	
2676	
2677 //percorre 28 vezes para fornecer as 28 peças	
2678 for(linha=0; linha<size; linha++)	
2679 {	
2680	
2681 for(coluna=0; coluna<COL; coluna++)	
2682 {	
2683	
2684 if (coluna==0)	
2685 {	
2686 //Comeca com o maior lado esquerdo na peça e vai decrementando	
2687 pecasi[linha][coluna]=dominomax;	
2688 //printf("[%d][%d] = %d ",linha,coluna,pecasi[linha][coluna]);	
2689 }	
2690 if (coluna==1)	
2691 {	
2692 //comeca com o menor lado direito e vai incrementando	
2693 pecasi[linha][coluna]=cont;	
2694 //printf("%d = [%d][%d]\n",pecasi[linha][coluna],linha,coluna);	
2695 cont++;	
2696 }	
2697 if (cont>dominomax)	
2698 {	
2699 dominomax--;	
2700 cont=0;	
2701 }	
2702	
2703	
2704	


```

2705
2706         }
2707
2708     }
2709
2710 }
2711
2712 }

```

void criapecasstr (char *pecass*[][COLSTR], int *s*)

Função criapecasstr.

função que cria as pecas em strings

Parâmetros:

<i>char</i>	<i>pecass</i> [][COLSTR] array que guarda as pecas em string
<i>int</i>	<i>s</i> contém o numero de pecas a ser guardadas

```

2717 {
2718
2725     //defino as pecas 28 em string estáticamente
2726     strcpy(pecass[0], "6|0");
2727     strcpy(pecass[1], "6|1");
2728     strcpy(pecass[2], "6|2");
2729     strcpy(pecass[3], "6|3");
2730     strcpy(pecass[4], "6|4");
2731     strcpy(pecass[5], "6|5");
2732     strcpy(pecass[6], "6|6");
2733     strcpy(pecass[7], "5|0");
2734     strcpy(pecass[8], "5|1");
2735     strcpy(pecass[9], "5|2");
2736     strcpy(pecass[10], "5|3");
2737     strcpy(pecass[11], "5|4");
2738     strcpy(pecass[12], "5|5");
2739     strcpy(pecass[13], "4|0");
2740     strcpy(pecass[14], "4|1");
2741     strcpy(pecass[15], "4|2");
2742     strcpy(pecass[16], "4|3");
2743     strcpy(pecass[17], "4|4");
2744     strcpy(pecass[18], "3|0");
2745     strcpy(pecass[19], "3|1");
2746     strcpy(pecass[20], "3|2");
2747     strcpy(pecass[21], "3|3");
2748     strcpy(pecass[22], "2|0");
2749     strcpy(pecass[23], "2|1");
2750     strcpy(pecass[24], "2|2");
2751     strcpy(pecass[25], "1|0");
2752     strcpy(pecass[26], "1|1");
2753     strcpy(pecass[27], "0|0");
2754
2755 }

```

int* eliminarep (int * *pv*, PECASINIT * *p*)

Função eliminarep.

função eliminarep, serve para eliminar as sequencias repetidas, recebe os indices das sequencias e a quantidade de sequencias e remove as repetidas que são as que contém -1 e aloca espaco para as que não sao repetidas e retorna essa quantidade para imprimir só as que não são repetidas

Parâmetros:

<i>int</i>	*pv apontador para um array de inteiros que contém os índices das sequencias
<i>int</i>	PECASINIT *p estrutura do tipo PECASINIT *p

Retorna:

retorna o novo tamanho do array de inteiros para depois apenas imprimir as sequências que possuem os índices com sequencias que não são repetidas

```

2355 {
2356
2363     int i=0;
2364     int j=0;
2365     int cont=0;
2366     int k=0;
2367     int *vaux=NULL;
2368
2369     for(i=0; i<(p->nseqss); i++)
2370     {
2371         if (*(pv+i) != -1)
2372         {
2373
2374             for(j=0; j<(p->nseqss); j++)
2375             {
2376
2377                 if (*(pv+i) == *(pv+j))
2378                 {
2379
2380                     cont++;
2381
2382                 }
2383
2384                 if (*(pv+i) == *(pv+j) && cont > 1)
2385                 {
2386
2387                     *(pv+j) = -1;
2388
2389                 }
2390
2391             }
2392
2393             cont=0;
2394
2395         }
2396
2397     }
2398
2399     for(i=0; i<(p->nseqss); i++)
2400     {
2401
2402         if (*(pv+i) == -1)
2403         {
2404
2405             cont++;
2406
2407         }
2408
2409     }
2410
2411     vaux = (int*) malloc(sizeof(int) * (p->nseqss) - cont);
2412
2413     for(i=0; i<(p->nseqss); i++)
2414     {
2415
2416         //printf("%d ", *(pv+i));
2417         if (*(pv+i) != -1)
2418         {
2419
2420             *(vaux+k) = *(pv+i);
2421             k++;
2422         }
2423

```

```

2424     }
2425
2426     p->nseqss=(p->nseqss-cont);
2427
2428     return vaux;
2429
2430 }

```

int entregarbaralhos (char pecass[][COLSTR], PECASINIT * p, int n)

Função entregarbaralhos.

função que entrega os baralhos aos jogadores aleatoriamente sem pecas repetidas para cada jogador (baralhos de 7 pecas e máximo de 4 jogadores)

Parâmetros:

<i>pecass[][COLSTR]</i>	array de string que contém as pecas todas
<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>int</i>	n numero de jogos entregues

```

2763 {
2764
2772     int pecas=0;
2773     //int verifica=0;
2774     int aux=0;
2775     int verify=0;
2776     PECA *paux=NULL;
2777     paux=p->pfirst;
2778
2779     srand( (unsigned)time(NULL) );
2780
2781     //Entrega 4 baralhos
2782     if (n>4)
2783     {
2784
2785         printf("Nao e possivel entregar mais de 4 baralhos\n");
2786
2787     }
2788     else
2789     {
2790         n=n*7;
2791
2792         for(pecas=0; pecas<n; pecas++)
2793         {
2794             //guardo um valor aleatório entre 0 e 28
2795             aux=rand() % 28 + 0;
2796
2797             if(pecas!=0)
2798             {
2799                 //ciclo para verificar se já existe, senao volta a ter outro numero
2800                 paux=p->pfirst;
2801
2802                 while(paux!=NULL)
2803                 {
2804
2805                     if(strcmp(paux->str,pecass[aux])==0)
2806                     {
2807
2808                         pecas--;
2809                         verify--;
2810
2811                     }
2812
2813                     paux=paux->pnext;

```

```

2814
2815         }
2816
2817     }
2818     else
2819     {
2820         //na primeira vez insere a peca sem problema
2821         inserir_pecas(p,pecas[aux]);
2822
2823     }
2824
2825     //verifica se não é a primeira peca, se não está repetida.
2826     if (verify==0 && pecas!=0)
2827     {
2828         //se a peca a inserir é diferente de todas das outras já inseridas,
2829         inserir_pecas(p,pecas[aux]);
2830
2831     }
2832     verify=0;
2833
2834 }
2835
2836 }
2837 }
2838
2839 return n;
2840
2841 }

```

void esvaziabaralhoint (int *baralhosi*[][COL], int *lin*, int *col*)

Função esvaziabaralhoint.

esta funcao coloca o baralho de inteiros vazio

Parâmetros:

<i>int</i>	baralhosi[][COL] baralhos de pecas dos jogadores em inteiros
<i>int</i>	lin linhas do baralho a esvaziar
<i>int</i>	col colunas do baralho a esvaziar

```

3250 {
3251
3252     //esta funcao coloca o baralho de inteiros vazio
3253     int i=0;
3254     int j=0;
3255
3256     for(i=0; i<lin; i++)
3257     {
3258         for(j=0; j<col; j++)
3259         {
3260             baralhosi[i][j]=9;
3261         }
3262     }
3263 }
3264
3265 }
3266
3267 }
3268 }
3269 }
3270 }
3271 }
3272 }
3273 }
3274 }
3275 }
3276 }

```

void esvaziabaralhostr (char *baralhoss*[][COLSTR], int *size*)

Função esvaziabaralhostr.

esta funcao coloca o baralho de strings vazio

Parâmetros:

<i>char</i>	baralhoss[][COLSTR] baralhos de pecas dos jogadores em strings
<i>int</i>	size tamanho do baralho

```

3308 {
3309
3316     //esta funcao coloca o baralho de strings vazio
3317     int i=0;
3318
3319     for(i=0; i<size; i++)
3320     {
3321
3322         strcpy(baralhoss[i], "9|9");
3323
3324     }
3325
3326 }
```

void esvaziabaralhostruct (PECASINIT * p)

Função esvaziabaralhostruct.

esta funcao coloca o baralho de strings vazio

Parâmetros:

<i>char</i>	baralhoss[][COLSTR] baralhos de pecas dos jogadores em strings
<i>int</i>	size tamanho do baralho

```

3280 {
3281
3288     //esta funcao coloca o baralho de strings vazio
3289     PECA *paux=NULL;
3290
3291     paux=p->pfirst;
3292
3293     while (paux!=NULL)
3294     {
3295
3296         paux->str=NULL;
3297         p->npecas--;
3298         paux=paux->pnext;
3299
3300     }
3301
3302 }
```

void esvaziaseqstr (char seqss[][COLSEQ], int size)

Função esvaziaseqstr.

esta funcao coloca o array das sequencias vazias

Parâmetros:

<i>char</i>	seqss[][COLSEQ] array de strings das sequencias
<i>int</i>	size numero de sequencias inseridas

```

3332 {
3333
3340     //esvazia as sequencias de peças
```

```

3341     int i=0;
3342
3343     for(i=0; i<size; i++)
3344     {
3345
3346         strcpy(seqss[i], "9|9");
3347
3348     }
3349
3350 }

```

PECA* find_peca_baralho (PECASINIT * *p*, char *aux*[])

Procurar peca no baralho.

função para procurar e retornar a peca se encontrar e null se nao encontrar

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>char</i>	aux[] peca a procurar

```

3662 {
3663
3670     PECA *paux=NULL;
3671     paux=p->pfirst;
3672
3673
3674     while (paux!=NULL)
3675     {
3676
3677         if (strcmp(paux->str, aux)==0)
3678         {
3679
3680             return paux;
3681
3682         }
3683
3684         paux=paux->pnext;
3685
3686     }
3687
3688     return NULL;
3689
3690 }

```

void inserir_baralhoaux (PECASINIT * *p*, char *sequencia*[])

Inserir sequencias de pecas no array dinamico.

função para inserir pecas no array dinâmico

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>char</i>	sequencia[] contem a sequencia a inserir

```

3809 {
3810
3817     SEQ *paux=NULL;
3818     paux=p->baralhoaux;
3819
3820     while (paux!=NULL && paux->seqstr!=NULL && (paux - (p->baralhoaux)) <
p->nbaralhoaux)
3821     {
3822

```

```

3823     paux++;
3824 }
3825 }
3826 if ( (paux - (p->baralhoaux)) == p->nbaralhoaux)
3827 {
3828     create_array_baralhoaux(p, p->nbaralhoaux+2);
3829     paux = p->baralhoaux + p->nbaralhoaux-2;
3830     paux->seqstr = create_dyn_string(sequencia);
3831 }else
3832 {
3833     paux->seqstr = create_dyn_string(sequencia);
3834 }
3835 }
3836 }
3837 }
3838 }
3839 }
3840 }
3841 }

```

void inserir_ordenaseq (PECASINIT * p, int tam, int index)

Inserir sequencias de pecas no array dinamico da função ordenar.

função para inserir pecas no array dinâmico

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>int</i>	tam strlen das sequencias
<i>int</i>	index indice da sequencia

```

2116 {
2117     ORDENA *paux=NULL;
2118     paux=p->ordenaseq;
2119     while (paux!=NULL && paux->tamanho!=0 && (paux - (p->ordenaseq)) < p->nordena)
2120     {
2121         paux++;
2122     }
2123     if ( (paux - (p->ordenaseq)) == p->nordena)
2124     {
2125         create_array_ordenaseq(p, p->nordena+2);
2126         paux = p->ordenaseq + p->nordena-2;
2127         paux->tamanho=tam;
2128         paux->indice=index;
2129     }else
2130     {
2131         paux->tamanho=tam;
2132         paux->indice=index;
2133     }
2134 }
2135 }
2136 }
2137 }
2138 }
2139 }
2140 }
2141 }
2142 }
2143 }
2144 }
2145 }
2146 }
2147 }
2148 }
2149 }
2150 }
2151 }

```

void inserir_peca (PECASINIT * p, char pecanova[COLSTR])

Inserir pecas na lista ligada do baralho.

função para inserir pecas nas listas ligadas

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>char</i>	pecanova[COLSTR] que contem a peca nova a inserir

```

2157 {
2158
2165     PECA *pnew = (PECA*)malloc(sizeof(PECA));
2166     PECA *paux = NULL;
2167
2168     pnew->str = create_dyn_string(pecanova);
2169     pnew->pnext=NULL;
2170
2171     paux=p->pfirst;
2172
2173     if (p->pfirst==NULL)
2174     {
2175
2176         p->pfirst=pnew;
2177         p->npecas++;
2178         //printf("PRIMEIRA %s ----   INSERIDAS: %d \n",pnew->str,p->npecas);
2179
2180     }
2181     else
2182     {
2183
2184         //cauda
2185         while (paux->pnext!=NULL)
2186         {
2187
2188             paux=paux->pnext;
2189
2190         }
2191
2192         paux->pnext=pnew;
2193         p->npecas++;
2194         //printf("CAUDA %s ----   INSERIDAS: %d \n",pnew->str,p->npecas);
2195         return;
2196
2197     }
2198
2199 }

```

void inserir_pecaint (PECASINIT * *p*, int *dir*, int *esq*)

Inserir pecas inteiras na lista ligada do baralho.

função para inserir pecas inteiras nas listas ligadas

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>int</i>	dir que contem lado direito da peca nova a inserir
<i>int</i>	dir que contem lado esquerdo da peca nova a inserir

```

2067 {
2068
2076     PECAINT *pnew = (PECAINT*)malloc(sizeof(PECAINT));
2077     PECAINT *paux = NULL;
2078
2079     pnew->direito=dir;
2080     pnew->esquerdo=esq;
2081     pnew->pnext=NULL;
2082
2083     paux=p->pfirstint;

```



```

2084
2085     if (p->pfirstint==NULL)
2086     {
2087
2088         p->pfirstint=pnew;
2089         p->npecasint++;
2090
2091     }
2092     else
2093     {
2094
2095         //cauda
2096         while (paux->pnext!=NULL)
2097         {
2098
2099             paux=paux->pnext;
2100
2101         }
2102
2103         paux->pnext=pnew;
2104         p->npecasint++;
2105
2106         return;
2107     }
2108 }
2109
2110 }

```

void inserir_seqf (PECASINIT * p, char sequencia[])

Inserir sequencias no array dinamico das seqs.

função para inserir sequencias no array dinamico

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>char</i>	sequencia[] contem sequencia a inserir

```

3772 {
3773
3780     SEQ *paux=NULL;
3781     paux=p->seqf;
3782
3783     while (paux!=NULL && paux->seqstr!=NULL && (paux - (p->seqf)) < p->nseqf)
3784     {
3785
3786         paux++;
3787
3788     }
3789
3790     if ( (paux - (p->seqf)) == p->nseqf)
3791     {
3792
3793         create_array_seqf(p,p->nseqf+2);
3794         paux = p->seqf+p->nseqf-2;
3795         paux->seqstr=create_dyn_string(sequencia);
3796
3797     }else
3798     {
3799
3800         paux->seqstr=create_dyn_string(sequencia);
3801
3802     }
3803
3804 }

```

void inserir_seqss (PECASINIT * p, char sequencia[])

Inserir sequencias no array dinamico das seqs auxiliares.

função para inserir sequencias no array dinamico

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>char</i>	sequencia[] contem sequencia a inserir

```

3697 {
3698
3705     SEQ *paux=NULL;
3706     paux=p->seqss;
3707
3708     while (paux!=NULL && paux->seqstr!=NULL && (paux - (p->seqss)) < p->nseqss)
3709     {
3710
3711         paux++;
3712
3713     }
3714
3715     if ( (paux- (p->seqss)) ==p->nseqss)
3716     {
3717
3718         create_array_seqss(p,p->nseqss+2);
3719         paux = p->seqss+p->nseqss-2;
3720         paux->seqstr=create_dyn_string(sequencia);
3721
3722     }else
3723     {
3724
3725         paux->seqstr=create_dyn_string(sequencia);
3726
3727     }
3728
3729 }
```

void inserir_seqssaux (PECASINIT * p, char sequencia[])

Inserir sequencias no array dinamico das seqs auxiliares 2.

função para inserir sequencias no array dinamico

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>char</i>	sequencia[] contem sequencia a inserir

```

3735 {
3736
3743     SEQ *paux=NULL;
3744     paux=p->seqssaux;
3745
3746     while (paux!=NULL && paux->seqstr!=NULL && (paux - (p->seqssaux)) < p->nseqssaux)
3747     {
3748
3749         paux++;
3750
3751     }
3752
3753     if ( (paux- (p->seqssaux)) ==p->nseqssaux)
3754     {
3755
3756         create_array_seqssaux(p,p->nseqssaux+2);
```

```

3757     paux = p->seqssaux+p->nseqssaux-2;
3758     paux->seqstr=create_dyn_string(sequencia);
3759
3760 }else
3761 {
3762
3763     paux->seqstr=create_dyn_string(sequencia);
3764
3765 }
3766
3767 }

```

char* inverterstr (char str1[], char str2[])

Função inverterstr.

função para inverter uma sequencia

Parâmetros:

<i>char</i>	str1[] string a inverter
<i>char</i>	str2[] string invertida a retornar

Retorna:

retorna a string invertida

```

4150 {
4151
4159     //funcao que retorna uma sequencia de peças invertida
4160     int invertidas=0;
4161
4162     for(invertidas=0; str1[invertidas]!='\0'; invertidas++)
4163     {
4164
4165         str2[invertidas]=str1[strlen(str1)-1-invertidas];
4166
4167     }
4168
4169     return str2;
4170
4171 }

```

int jogoadois (char baralhoss[][COLSTR], char seqf[][COLSEQ], int num, char seqinit[])

Função jogoadois.

esta funcao é igual à função das sequencias, só que nestas posso começar inicialmente com uma sequencia on nao e os jogadores jogam à vez de cada baralho

Parâmetros:

<i>char</i>	baralhoss[][COLSTR] baralhos do jogador em string
<i>char</i>	seqf[][COLSEQ] array final onde vou guardar as sequencias todas possiveis
<i>int</i>	num numero de jogos a calcular sequencias
<i>char</i>	seqinit[] sequencia inicial

Retorna:

retorna o numero de sequencias que encontrou

VERIFICA QUANTIDADE DE CORRESPONDENCIAS

JUNTA AS CORRESPONDENCIAS

```

5491 {

```

```

5492
5501 //esta função é igual à função de sequencias com a particularidade que os jogadores
jogam alternadamente para fazerem sequencias
5502
5503 char seqss[LINSEQ][COLSEQ];
5504 char baralhoaux[LINSEQ][COLSEQ];
5505 char invertidas[LINSEQ][COLSEQ];
5506 char vinvertidas[LINSEQ][COLSEQ];
5507 char auxinv[COLSTR];
5508 char invfinal[COLSTR];
5509 int i=0;
5510 int j=0;
5511 int k=0;
5512 int tam=LINSEQ;
5513 int cont=0;
5514 int fimdastring=0;
5515 int decont=LINSEQ-1;
5516 int iguais=0;
5517 char aux[2000];
5518 char seqinitinvertida[200];
5519 char *s;
5520 char *inv;
5521 char *invseq=NULL;
5522 int contadorfinal=0;
5523 int contapecasvinvertidas=0;
5524 int numero=1;
5525 int tambaralhosaux=0;
5526 /*a variavel ok acrescentada para apenas passar para o array secundário o que foi
passado para o final sem concatenar
5527 peças mal*/
5528 int ok=0;
5529
5530 esvaziaseqstr(seqss,LINSEQ);
5531 esvaziaseqstr(baralhoaux,LINSEQ);
5532
5533 //se nao tiver uma sequencia inicial, comecao com o primeiro jogo
5534
5535 tambaralhosaux=baralhoausar(baralhoss,baralhoaux,numero);
5536
5537 if(strcmp(seqinit,"9|9")==0)
5538 {
5539
5540     for(i=0; i<tambaralhosaux; i++)
5541     {
5542
5543         strcpy(seqss[i],baralhoaux[i]);
5544
5545
5546     }
5547
5548     tambaralhosaux=baralhoausar(baralhoss,baralhoaux,++numero);
5549
5550 }
5551 else
5552 {
5553
5554     strcpy(seqss[1],seqinit);
5555     for(i=0; seqinit[i]!='\0'; i++)
5556     {
5557
5558         seqinitinvertida[i]=seqinit[strlen(seqinit)-1-i];
5559
5560     }
5561     strcpy(seqss[0],seqinitinvertida);
5562
5563 }
5564
5565
5568 do
5569 {
5570

```

```

5571         cont=0;
5572         for(i=0; i<tambaralhosaux; i++)
5573         {
5574             for(j=0; j<tam; j++)
5575             {
5576                 if(k!=0)
5577                 {
5578                     strcpy(aux,seqss[j]);
5579                     s = strtok (aux,"-");
5580                     while (s!= NULL)
5581                     {
5582                         if(strcmp(baralhoaux[i],s)==0 || (s[2]==baralhoaux[i][0] &&
5583 s[0]==baralhoaux[i][2]))
5584                         {
5585                             iguais++;
5586                         }
5587                         s = strtok (NULL, "-");
5588                     }
5589                     strcpy(aux,"");
5590                 }
5591                 if(iguais==0)
5592                 {
5593                     /*Ve o tamanho da string*/
5594                     for(fimdastring=0; seqss[j][fimdastring]!='\0'; fimdastring++);
5595                     fimdastring--;
5596                     /*compara as strings iniciasi e invertidas com aquelas que vamos
5597 aumentar*/
5598                     if(seqss[j][fimdastring]==baralhoaux[i][0] && i!=j)
5599                     {
5600                         cont++;
5601                         strcpy(seqf[contadorfinal],seqss[j]);
5602                         strcat(seqf[contadorfinal],"-");
5603                         strcat(seqf[contadorfinal],baralhoaux[i]);
5604                         contadorfinal++;
5605                         ok++;
5606                     }
5607                 }
5608                 iguais=0;
5609             }
5610         }
5611         for(i=0; i<tambaralhosaux; i++)
5612         {
5613             for(j=0; j<ok; j++)
5614             {

```

```

5642         if(k!=0)
5643         {
5644
5645             strcpy(aux,seqss[j]);
5646
5647             s = strtok (aux,"-");
5648
5649             while (s!= NULL)
5650             {
5651
5652                 if(strcmp(baralhoaux[i],s)==0 || (s[2]==baralhoaux[i][0] &&
s[0]==baralhoaux[i][2]))
5653                 {
5654
5655                     iguais++;
5656
5657                 }
5658
5659                 s = strtok (NULL, "-");
5660
5661             }
5662
5663             strcpy(aux,"");
5664
5665             if(iguais==0)
5666             {
5667
5668                 for(fimdastring=0; seqss[j][fimdastring]!='\0';
fimdastring++);
5669                 fimdastring--;
5670
5671
5672                 if(seqss[j][fimdastring]==baralhoaux[i][0] && i!=j)
5673                 {
5674
5675                     strcpy(seqss[decont],seqss[j]);
5676                     strcat(seqss[decont],"-");
5677                     strcat(seqss[decont],baralhoaux[i]);
5678                     decont--;
5679
5680                 }
5681             }
5682
5683         }
5684
5685         else
5686         {
5687
5688             for(fimdastring=0; seqss[j][fimdastring]!='\0'; fimdastring++);
5689             fimdastring--;
5690
5691             if(seqss[j][fimdastring]==baralhoaux[i][0] && i!=j)
5692             {
5693
5694                 strcpy(seqss[decont],seqss[j]);
5695                 strcat(seqss[decont],"-");
5696                 strcat(seqss[decont],baralhoaux[i]);
5697                 decont--;
5698
5699             }
5700
5701         }
5702
5703         iguais=0;
5704
5705     }
5706
5707 }
5708
5709 //Agora limpa o array seqss em cima e assa de baixo para cima para continuar
a juntar

```

```

5710         i=0;
5711
5712         while (strcmp(seqss[i],"9|9")!=0)
5713         {
5714
5715             strcpy(seqss[i],"9|9");
5716             i++;
5717         }
5718
5719         decont=LINSEQ-1;
5720         i=0;
5721
5722         //apaga array em baixo
5723
5724         while (strcmp(seqss[decont],"9|9")!=0)
5725         {
5726
5727             strcpy(seqss[i],seqss[decont]);
5728             strcpy(seqss[decont],"9|9");
5729             decont--;
5730             i++;
5731         }
5732
5733         tam=(LINSEQ-decont-1);
5734         decont=LINSEQ-1;
5735         k++;
5736
5737         //aqui faço com que o baralho auxiliar a usar seja usado alternadamente ou seja,
5738         //na primeira vez
5739         //comparo o baralhoaux do jogador 1 mas na segunda vez comparo com o baralho
5740         //auxiliar do jogador dois de modo
5741         // a fazer sequencias com os baralhos dos jogadores alternadamente
5742
5743         if(numero<num)
5744         {
5745
5746             tambaralhosaux=baralhoausar(baralhoss,baralhoaux,++numero);
5747
5748         }
5749         else
5750         {
5751
5752             numero=1;
5753             tambaralhosaux=baralhoausar(baralhoss,baralhoaux,numero);
5754
5755         }
5756     }
5757 }
5758
5759 while(cont>0);
5760
5761 //verifica se contem invertidas e normais e retira-as
5762 cont=0;
5763 i=0;
5764
5765 while(strcmp(seqf[cont],"9|9")!=0)
5766 {
5767     strcpy(seqss[i],seqf[cont]);
5768     cont++;
5769     i++;
5770 }
5771
5772 esvaziaseqstr(invertidas,LINSEQ);
5773
5774 for(i=0; i<cont; i++)
5775 {
5776
5777     strcpy(invertidas[i],seqss[i]);

```

```

5779     }
5780
5781
5782     j=0;
5783     for(i=0; i<cont; i++)
5784     {
5785
5786         invseq = strtok (invertidas[i], "-");
5787
5788         while (invseq != NULL)
5789         {
5790
5791             strcpy(vinvertidas[j], invseq);
5792             j++;
5793             invseq = strtok (NULL, "-");
5794
5795         }
5796         contapecasvinvertidas=j;
5797         j=0;
5798         inv = strtok (seqss[i], "-");
5799
5800         while (inv != NULL)
5801         {
5802
5803             strcpy(auxinv, inv);
5804
5805             invfinal[2]=auxinv[0];
5806             invfinal[1]=auxinv[1];
5807             invfinal[0]=auxinv[2];
5808
5809             if(invfinal[2]!=invfinal[0])
5810             {
5811
5812                 for(k=0; k<contapecasvinvertidas; k++)
5813                 {
5814
5815                     if(strcmp(invfinal, vinvertidas[k])==0)
5816                     {
5817
5818                         strcpy(seqf[i], "-");
5819
5820                     }
5821
5822                 }
5823
5824             }
5825
5826             inv = strtok (NULL, "-");
5827
5828         }
5829         esvaziaseqstr(vinvertidas, LINSEQ);
5830         j=0;
5831     }
5832
5833     contadorfinal=0;
5834     i=0;
5835
5836     while(i<LINSEQ)
5837     {
5838
5839         if(strcmp(seqf[i], "9|9")!=0)
5840         {
5841
5842             contadorfinal++;
5843
5844         }
5845
5846         i++;
5847     }
5848
5849     return contadorfinal;

```



```
5850
5851 }
```

void load_jogo_bin (PECASINIT * p, char fname[])

Load do ficheiro binario.

função para carregar um ficheiro binario

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>char</i>	fname[] nome do ficheiro a carregar

```
1561 {
1562     //SE QUISERMOS GUARDAR TODA A ESTRUTURA NUM FICHEIRO
1563     //fwrite(&t,sizeof(t),1,fp);
1564     FILE *fp=NULL;
1565
1566     char nome[50];
1567     int size=0;
1568     int i=0;
1569     int n=0;
1570
1571     if((fp=fopen(fname,"rb"))!=NULL)
1572     {
1573         fread(&(p->npecas),sizeof(int),1,fp);
1574         n=p->npecas;
1575         p->npecas=0;
1576
1577         for(i=0;i<n;i++)
1578         {
1579             fread(&size,sizeof(int),1,fp);
1580             fread(nome,sizeof(char),size,fp);
1581             inserir_pecas(p,nome);
1582
1583         }
1584         fclose(fp);
1585     }
1586 }
```

void load_txt_jogo (PECASINIT * p, char fname[])

Load do ficheiro txt.

função para carregar um ficheiro txt

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>char</i>	fname[] nome do ficheiro a carregar

```
1605 {
1606
1607     FILE *fp=NULL;
1608     char nome[50];
1609     char ignora[50];
1610     char njogos[1];
```

```

1617     int i=0;
1618     int n=0;
1619     /*%[:] -> ler até ':'
1620     %*[:] -> ignorar ':'
1621     %[^,] -> ler até ','
1622     %*[,] -> ignorar ','
1623     %[^:] -> ler até ':'
1624     %*[:] -> ignorar ':'
1625     %d    -> ler inteiro*/
1626
1627     if((fp=fopen(fname,"r"))==NULL)
1628     {
1629
1630         printf("... ERRO ...");
1631         return;
1632     }
1633
1634     //gravo o numero de jogos
1635     fscanf(fp,"%[^ ]",njogos);
1636     n=atoi(njogos);
1637
1638     //nao gravo a palavra jogos
1639     fscanf(fp,"%[^-] %*[-] %*[\n]",ignora);
1640     //pecas
1641
1642
1643     for(i=0;i<n*7;i++)
1644     {
1645
1646         fscanf(fp,"%[^\\n] %*[" ]",nome);
1647         inserir_pecas(p,nome);
1648     }
1649
1650 }
1651
1652
1653 }

```

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

Função main.

função que contém o menu do programa e as respectivas chamadas as funcoes, neste momento estou também a fazer algumas verificações para verificar se as subsequências a procurar são válidas, se os padrões a substituir são válidos. Estas verificações passarão posteriormente a ser realizadas dentro de funções

Parâmetros:

<i>int</i>	argc não está a ser usado mas contém o numero de posicoes do array de strings argv[] usadas,
<i>char</i>	*argv[] não está a ser usado, mas contém o numero de posições do array de strings

Retorna:

0

MENU

```

66 {
67     PECASINIT p= {NULL,NULL,NULL,0,0,0};
68
69     int num=1;
70     int jogosaaimprimir=0;
71     int numdeseq=0;
72     int pecasint[LIN][COL];
73     int baralhosint[LIN][COL];

```

```

81     char pecasstr[LIN][COLSTR];
82     char baralhosstr[LIN][COLSTR];
83     char seqstr[LINSEQ][COLSEQ];
84     char subseq[LINSEQ];
85     char padrao[LINSEQ]="4|3";
86     char padraonovo[LINSEQ]="0|9-9|9-9|3";
87     char seqinicial[LINSEQ]="9|9";
88     int tam=0;
89
90
91     //var menu
92     char seqinicialaux[LINSEQ];
93     char seqinicialpartida[LINSEQ][COLSEQ];
94     char pecasstrinv[COLSTR];
95     char pecasstraux[LIN][COLSTR];
96     char op='0';
97     char op1='0';
98     char op2='0';
99     char op3='0';
100    char op4='0';
101    int i=0;
102    int j=0;
103    int r=0;
104    int cont=0;
105    char subseqinv[COLSEQ];
106    int contseq=0;
107    int matrizprocurasub[LINSEQ][2];
108    int k=0;
109    char padraonovoinv[LINSEQ];
110    char padraoinv[COLSEQ];
111    char seqfpadrao[LINSEQ][COLSEQ];
112    int sizeseqfpadrao=0;
113    char subseqaux[COLSEQ];
114    char auxinv[COLSEQ];
115    int tamsubseq=0;
116    char seqpadraoaux[LINSEQ][COLSEQ];
117    char padraoaux[COLSEQ];
118    char baralhosstrauxcominv[LINSEQ][COLSEQ];
119    char baralhosstrauxcominvnovo[LINSEQ][COLSEQ];
120    char baralhosstrauxcominvnovopadraonovo[LINSEQ][COLSEQ];
121    char pecastrauxinvpadroes[LINSEQ][COLSTR];
122    char padraototal[COLSEQ];
123
124    //cria pecas todas
125    criapecasint(pecasint,LIN);
126    criapecasstr(pecasstr,LIN);
127    criapecasstr(pecasstraux,LIN);
128
129    //limpa baralhos
130    esvaziabaralhoint(baralhosint,LIN,COL);
131    esvaziabaralhostr(baralhosstr,LIN);
132    esvaziaseqstr(seqstr,LINSEQ);
133    esvaziaseqstr(seqfpadrao,LINSEQ);
134    esvaziaseqstr(seqinicialpartida,LINSEQ);
135    esvaziaseqstr(baralhosstrauxcominv,LINSEQ);
136    esvaziaseqstr(baralhosstrauxcominvnovo,LINSEQ);
137    esvaziabaralhostr(pecastrauxinvpadroes,LINSEQ);
138    esvaziaseqstr(baralhosstrauxcominvnovopadraonovo,LINSEQ);
139
140    create_array_seqf(&p,2);
141    create_array_seqss(&p,2);
142    create_array_seqssaux(&p,2);
143    create_array_baralhoaux(&p,2);
144
145    char filename[]="./lp1.txt";
146
147
148    while(1)
149    {
150
151        switch(op)

```

```

154     {
155
156     case '0':
157
158         printf("--DOMINO--\n\n");
159         printf("1- Ler Jogos do ficheiro\n");
160         printf("2- Inserir manualmente\n");
161         printf("3- Ler Jogos de um ficheiro binario\n");
162
163
164         scanf("%c",&op1);
165         system("CLS");
166
167         switch(op1)
168         {
169
170             case '0':
171
172                 op='0';
173
174                 break;
175
176             case '1':
177
178                 load_txt_jogo(&p,filename);
179                 mostrarjogosstrstruct(p,7);
180                 op='2';
181                 scanf("%c",&op2);
182
183                 break;
184
185             break;
186
187             case '2':
188
189                 printf("Quantos jogos pretende:\n");
190                 scanf("%d",&num);
191                 system("CLS");
192
193                 if (num<1||num>4)
194                 {
195
196                     printf("O NUMERO DE JOGOS TEM DE SER ENTRE 1 e 4!!\n");
197                     op='0';
198
199                 }
200                 else
201                 {
202
203                     op='1';
204
205                 }
206
207             break;
208
209             case '3':
210
211                 load_jogo_bin(&p,"DOMINO.BIN");
212                 mostrarjogosstrstruct(p,7);
213                 op='3';
214                 scanf("%c",&op2);
215
216                 break;
217
218             break;
219
220             default:
221
222                 printf("OPCAO INVALIDA\n");
223                 op='0';
224

```

```

225             break;
226         }
227     }
228
229
230     break;
231
232
233     case '1':
234
235         printf("--MENU--\n\n");
236         printf("1- Criar jogo(s) alaeatoriamente\n");
237         printf("2- Preencher jogo(s)\n");
238         printf("0- Voltar atras\n");
239         scanf(" %c",&op2);
240         system("CLS");
241
242
243         switch(op2)
244         {
245
246             case '1':
247
248                 p.npecas=0;
249                 p.pfirst=NULL;
250                 jogosaimprimir=entregarbaralhos(pecasstr,&p,num);
251
252                 /*inserir peca(&p,"2|6");
253                 inserir_pecas(&p,"6|6");
254                 inserir_pecas(&p,"6|4");
255                 inserir_pecas(&p,"4|5");
256                 inserir_pecas(&p,"5|0");
257                 inserir_pecas(&p,"0|3");
258                 inserir_pecas(&p,"3|1");*/
259                 mostrarjogosstrstruct(p,7);
260                 //mostrarjogosstrstruct(p,jogosaimprimir);
261                 /*
262                 strcpy(baralhosstr[0],"6|4");
263                 strcpy(baralhosstr[1],"4|1");
264                 strcpy(baralhosstr[2],"2|2");
265                 strcpy(baralhosstr[3],"6|6");
266                 strcpy(baralhosstr[4],"5|1");
267                 strcpy(baralhosstr[5],"4|0");
268                 strcpy(baralhosstr[6],"3|2");
269                 */
270                 op='2';
271
272                 break;
273
274             case '2':
275
276                 p.npecas=0;
277                 p.pfirst=NULL;
278                 jogosaimprimir=preenchebaralhosstrstruct(pecasstr,&p,num);
279                 mostrarjogosstrstruct(p,jogosaimprimir);
280                 op='2';
281
282                 break;
283
284             case '0':
285
286                 op='0';
287
288                 break;
289
290             default:
291
292                 printf("OPCAO INVALIDA\n");
293                 op='1';
294
295                 break;

```

```

296
297     }
298
299
300
301     break;
302
303     case '2':
304
305         printf("--MENU--\n\n");
306         printf("1- Mostrar sequencias possiveis, (decrecente)\n");
307         printf("2- Mostrar sequencias possiveis, (decrecente), com sequencia
inicial\n");
308         printf("3- Mostrar sequencias possiveis, (decrecente) com jogadores a
jogar alternadamente\n");
309         printf("4- Alterar Pecas\n");
310         printf("5- Converter String para Inteiro\n");
311         printf("0- Voltar ao Menu anterior\n\n");
312
313         scanf(" %c",&op2);
314         system("CLS");
315
316         switch(op2)
317         {
318
319             case '1':
320
321                 numdeseq=seq(&p,num);
322                 ordernarsequenciasstruct(&p);
323                 separarseqinvertidasstruct(&p);
324                 printseqstrstruct(p);
325                 /*numdeseq=seq(baralhosstr,seqstr,num);
326                 ordernarsequencias(seqstr,numdeseq);
327                 numdeseq=tirartracosinvertidos(seqstr,numdeseq);
328                 numdeseq=separarseqinvertidas(seqstr,numdeseq);
329                 printseqstr(seqstr,0,numdeseq);
330                 printf("\n");
331
332                 mostrarjogosstr(baralhosstr,jogosaimprimir);
333                 contseq=1;
334                 */
335                 op='3';
336
337                 break;
338
339             case '2':
340
341                 mostrarjogosstr(baralhosstr,jogosaimprimir);
342                 esvaziaseqstr(seqinicialpartida,LINSEQ);
343                 esvaziaseqstr(baralhosstrauxcominv,LINSEQ);
344                 cont=0;
345                 printf("Insira a sequencia inicial: (SO PODE ESCOLHER PECAS DAS
SEGUINTEs)\n\n");
346                 //copia as pecas iniciais para o baralho vazio
347
348                 for(i=0; i<LIN; i++)
349                 {
350
351                     strcpy(baralhosstrauxcominv[i],pecasstr[i]);
352
353                 }
354
355                 //copio as respectivas invertidas para o baralho auxiliar
356
357                 for(j=0; j<LIN; j++)
358                 {
359
360                     if(pecasstr[j][0]!=pecasstr[j][2])
361                     {
362

```

```

363 strcpy(baralhosstrauxcominv[i++], inverterstr(pecasstr[j], auxinv));
364
365         }
366
367     }
368
369
370     for(i=0; strcmp(baralhosstrauxcominv[i], "9|9")!=0; i++)
371     {
372
373         for(j=0; strcmp(baralhosstr[j], "9|9")!=0; j++)
374         {
375
376             if(strcmp(baralhosstrauxcominv[i], baralhosstr[j])==0 ||
strcmp(baralhosstrauxcominv[i], inverterstr(baralhosstr[j], pecasstrinv))==0)
377             {
378
379                 strcpy(baralhosstrauxcominv[i], "-");
380
381             }
382
383
384         }
385
386     }
387
388
389     scanf("%s", seqinicial);
390
391     if(verificasequencia(seqinicial)==1)
392     {
393
394         k=1;
395
396     }
397
398     //partimos a sequencia inicial partida e guardamos no array auxiliar
399     strcpy(seqinicialaux, seqinicial);
400     tamsubseq=strtoque(seqinicialpartida, seqinicialaux, '-');
401
402     //verifico de o padrao novo possui pecas repetidas ou nao
403
404     for(i=0; strcmp(seqinicialpartida[i], "9|9")!=0; i++)
405     {
406
407         for(j=0; strcmp(baralhosstrauxcominv[j], "9|9")!=0; j++)
408         {
409             //printf("%s ==
%s\n", seqpadraoaux[i], baralhosstrauxcominvnovopadraonovo[j]);
410
411             if(strcmp(seqinicialpartida[i], baralhosstrauxcominv[j])==0)
412             {
413
414                 strcpy(baralhosstrauxcominv[j], "-");
415                 cont++;
416
417                 for(r=0; strcmp(baralhosstrauxcominv[r], "9|9")!=0; r++)
418                 {
419
420                     if(strcmp(inverterstr(seqinicialpartida[i], pecasstrinv), baralhosstrauxcominv[r])==0)
421                     {
422
423                         strcpy(baralhosstrauxcominv[r], "-");
424
425                     }
426
427                 }
428

```

```

429
430         }
431
432
433     }
434
435
436     }
437
438
439     if(cont!=tamsubseq || k==1)
440     {
441
442         printf("A sequequencia nao e valida!!\n\n");
443
444         //volta a carregar o pecasstraux com as pecas todas e esvazia o
array da sequencia inicial partida
445         for(i=0; i<LIN; i++)
446         {
447
448             strcpy(pecasstraux[i],pecasstr[i]);
449
450         }
451         k=0;
452         op2='2';
453
454
455     }
456     else
457     {
458
459         numdeseq=seqcomseqinicial(baralhosstr,seqstr,num,seqinicial);
460         ordenarsequencias(seqstr,numdeseq);
461         numdeseq=tirartracosinvertidos(seqstr,numdeseq);
462         numdeseq=separarseqinvertidas(seqstr,numdeseq);
463         tam=retiraseqinitrepetida(seqstr,numdeseq,seqinicial);
464         ordenarsequencias(seqstr,numdeseq);
465         numdeseq=numdeseq-tam;
466         printseqstr(seqstr,0,numdeseq);
467
468         mostrarjogosstr(baralhosstr,jogosaimprimir);
469         contseq=2;
470         op='3';
471
472     }
473
474     break;
475
476     case '0':
477
478         op='1';
479
480         break;
481
482     case '3':
483
484         if(num<2)
485         {
486
487             printf("O numero de baralhos para esta opcao tem de ser no minimo
(2)");
488             op='0';
489
490         }
491         else
492         {
493
494             op='4';
495
496         }
497

```



```

498             break;
499
500
501         case '4':
502
503             jogosaimprimir=remover(&p,pecasstr,num);
504             mostrarjogosstrstruct(p,jogosaimprimir);
505
506             break;
507
508         case '5':
509
510             printf("String para inteiro || Esvazia o baralho (string) || Inteiro
para string\n");
511             jogosaimprimir=convertestrtoint(&p,num);
512             mostrarjogosintstruct(p,jogosaimprimir);
513
514             esvaziabaralhostruct(&p);
515             mostrarjogosstrstruct(p,jogosaimprimir);
516
517             jogosaimprimir=converteinttostr(&p,num);
518             mostrarjogosstrstruct(p,jogosaimprimir);
519
520             break;
521
522         default:
523
524             printf("OPCAO INVALIDA\n");
525             mostrarjogosstr(baralhosstr,jogosaimprimir);
526
527             break;
528     }
529
530
531
532
533     break;
534
535     case '3':
536
537         printf("\n\n--MENU--\n\n");
538         printf("1- Procurar subsequencias\n");
539         printf("2- Substituir padroes nas sequencias\n");
540         printf("3- Gravar para ficheiro txt\n");
541         printf("4- Gravar para ficheiro binario\n");
542
543
544         printf("0- Voltar ao menu principal\n\n");
545
546         scanf(" %c",&op3);
547         system("CLS");
548
549         switch(op3)
550         {
551
552             case '4':
553
554                 save_jogo_bin(p,"DOMINO.BIN");
555
556                 break;
557
558             case '1':
559
560                 if(contseq==1)
561                 {
562
563                     //numdeseq=seq(baralhosstr,seqstr,num);
564                     ordernarsequencias(seqstr,numdeseq);
565                     numdeseq=tirartracosinvertidos(seqstr,numdeseq);
566                     numdeseq=separarseqinvertidas(seqstr,numdeseq);
567                     printseqstr(seqstr,0,numdeseq);

```

```

568             printf("\n");
569
570             mostrarjogosstr(baralhosstr, jogosaimprimir);
571
572         }
573
574         if(contseq==2)
575         {
576
577             numdeseq=seqcomseqinicial(baralhosstr, seqstr, num, seqinicial);
578             ordenarsequencias(seqstr, numdeseq);
579             numdeseq=tirartracosinvertidos(seqstr, numdeseq);
580             numdeseq=separarseqinvertidas(seqstr, numdeseq);
581             tam=retiraseqinitrepetida(seqstr, numdeseq, seqinicial);
582             ordenarsequencias(seqstr, numdeseq);
583             numdeseq=numdeseq-tam;
584             printseqstr(seqstr, 0, numdeseq);
585
586             mostrarjogosstr(baralhosstr, jogosaimprimir);
587
588         }
589
590
591         k=0;
592         printf("Qual a sub sequencia a procurar:\n");
593         scanf("%s", subseq);
594
595         //fazemos uma copia da subseq
596         strcpy(subseqaux, subseq);
597         //reinicia os contadores e variavies
598         esvaziaseqstr(seqinicialpartida, LINSEQ);
599         esvaziaseqstr(baralhosstrauxcominv, LINSEQ);
600         cont=0;
601
602
603         //copia para array auxiliar
604         for(i=0; strcmp(baralhosstr[i], "9|9")!=0; i++)
605         {
606
607             strcpy(baralhosstrauxcominv[i], baralhosstr[i]);
608
609         }
610
611         //copia invertidas para array auxiliar
612         for(j=0; strcmp(baralhosstr[j], "9|9")!=0; j++)
613         {
614             if(baralhosstr[j][0]!=baralhosstr[j][2])
615             {
616
617                 strcpy(baralhosstrauxcominv[i++], inverterstr(baralhosstr[j], auxinv));
618
619             }
620
621
622         }
623
624         //se tivermos a usar uma sequencia inicial ao procurar uma subsequencia
625         temos de também poder procurar pelas peças usadas na sequencia inicial
626
627         if(strcmp(seqinicial, "9|9")!=0)
628         {
629
630             strtok(seqinicialpartida, seqinicial, '-');
631
632             for(j=0; strcmp(seqinicialpartida[j], "9|9")!=0; j++)
633             {
634
635                 strcpy(baralhosstrauxcominv[i++], seqinicialpartida[j]);
636
637             }

```

```

637
638         for(j=0; strcmp(seqinicialpartida[j],"9|9")!=0; j++)
639         {
640
641             if(seqinicialpartida[j][0]!=seqinicialpartida[j][2])
642             {
643
644
645         strcpy(baralhosstrauxcominv[i++],inverterstr(baralhosstrauxcominv[j],auxinv));
646
647             }
648
649         }
650
651     }
652
653     //esvaziamos o array de strings da sequencia inicial partida para
654     podermos partir a subsequencia
655
656     esvaziaseqstr(seqinicialpartida,LINSEQ);
657
658     tamsubseq=strtoque(seqinicialpartida,subseqaux,'-');
659
660     //verifico de o padrao novo possui pecas repetidas ou nao
661
662     for(i=0; strcmp(seqinicialpartida[i],"9|9")!=0; i++)
663     {
664
665         for(j=0; strcmp(baralhosstrauxcominv[j],"9|9")!=0; j++)
666         {
667             //printf("%s ==
668             %s\n",seqpadraoaux[i],baralhosstrauxcominvnovopadraonovo[j]);
669
670             if(strcmp(seqinicialpartida[i],baralhosstrauxcominv[j])==0)
671             {
672
673                 strcpy(baralhosstrauxcominv[j],"-");
674                 cont++;
675
676                 for(r=0; strcmp(baralhosstrauxcominv[r],"9|9")!=0; r++)
677                 {
678
679                     if(strcmp(inverterstr(seqinicialpartida[i],pecasstrinv),baralhosstrauxcominv[r])==0)
680                     {
681
682                         strcpy(baralhosstrauxcominv[r],"-");
683
684                     }
685
686                 }
687
688             }
689
690         }
691
692     }
693
694     if(k==1 || tamsubseq!=cont)
695     {
696
697         printf("A sequequencia nao e valida!!\n\n");
698
699         //volta a carregar o pecasstraus com as pecas todas e esvazia o
700         array da sequencia inicial partida
701

```

```

702         for(i=0; i<LIN; i++)
703         {
704
705             strcpy(pecasstraux[i], pecasstr[i]);
706
707         }
708
709         esvaziaseqstr(seginicialpartida, LINSEQ);
710         esvaziaseqstr(baralhosstrauxcominv, LINSEQ);
711         cont=0;
712         k=0;
713         op2='2';
714
715
716     }
717
718
719     if(verificasequencia(subseq)==0)
720     {
721
722         //procura sub-sequencia normal na sequencia e se encontrar guarda
723         //a primeira posicao de uma matriz
724         //e na segunda posicao da matriz guarda a primeira peça em que fez
725         //o match
726         for(i=0; i<numdeseq; i++)
727         {
728
729             if(procsubseq_ausar(seqstr, i, subseq) != -1)
730             {
731
732                 matrizprocurasub[k][0]=i;
733                 matrizprocurasub[k][1]=procsubseq_ausar(seqstr, i, subseq);
734                 k++;
735             }
736
737         }
738
739         //procura sub-sequencia invertida na sequencia e se encontrar
740         //a primeira posicao de uma matriz
741         //e na segunda posicao da matriz guarda a primeira peça em que fez
742         //o match
743         inverterstr(subseq, subseqinv);
744         for(i=0; i<numdeseq; i++)
745         {
746
747             if(procsubseq_ausar(seqstr, i, subseqinv) != -1)
748             {
749
750                 matrizprocurasub[k][0]=i;
751                 matrizprocurasub[k][1]=procsubseq_ausar(seqstr, i, subseqinv);
752                 k++;
753             }
754
755         }
756
757     }
758
759     ordenarmatrizinteiros(matrizprocurasub, k);
760
761     for(i=0; i<k; i++)
762     {
763
764         printf("[%d] %s --->
765         %d\n", matrizprocurasub[i][0], seqstr[matrizprocurasub[i][0]], matrizprocurasub[i][1]);

```

```

766         }
767
768         //esvaziaseqstr(seqfpadrao,LINSEQ);
769
770     }
771     else
772     {
773
774         op3='1';
775
776     }
777
778     break;
779
780 case '2':
781
782     if(contseq==1)
783     {
784
785         //numdeseq=seq(baralhosstr,seqstr,num);
786         ordenarsequencias(seqstr,numdeseq);
787         numdeseq=tirartracosinvertidos(seqstr,numdeseq);
788         numdeseq=separarseqinvertidas(seqstr,numdeseq);
789         printseqstr(seqstr,0,numdeseq);
790         printf("\n");
791
792         mostrarjogosstr(baralhosstr,jogosaimprimir);
793         //contseq=0;
794
795     }
796
797     else if(contseq==2)
798     {
799
800         numdeseq=seqcomseqinicial(baralhosstr,seqstr,num,seqinicial);
801         ordenarsequencias(seqstr,numdeseq);
802         numdeseq=tirartracosinvertidos(seqstr,numdeseq);
803         numdeseq=separarseqinvertidas(seqstr,numdeseq);
804         tam=retiraseqinitrepetida(seqstr,numdeseq,seqinicial);
805         ordenarsequencias(seqstr,numdeseq);
806         printseqstr(seqstr,0,numdeseq-tam);
807
808         mostrarjogosstr(baralhosstr,jogosaimprimir);
809         //contseq=0;
810     }
811     else
812     {
813
814         mostrarjogosstr(baralhosstr,jogosaimprimir);
815
816     }
817
818     k=0;
819     printf("Qual o padrao que pretende substituir:\n");
820     scanf("%s",padrao);
821
822     //fazemos uma copia do padrao
823     strcpy(padraoaux,padrao);
824     //reinicia os contadores e variavies
825     esvaziaseqstr(seqinicialpartida,LINSEQ);
826     esvaziaseqstr(baralhosstrauxcominv,LINSEQ);
827     cont=0;
828
829     //copia para array auxiliar
830     for(i=0; strcmp(baralhosstr[i],"9|9")!=0; i++)
831     {
832
833         strcpy(baralhosstrauxcominv[i],baralhosstr[i]);
834
835     }
836

```

```

837          //copia invertidas para array auxiliar
838          for(j=0; strcmp(baralhosstr[j],"9|9")!=0; j++)
839          {
840              if(baralhosstr[j][0]!=baralhosstr[j][2])
841              {
842
843
844          strcpy(baralhosstrauxcominv[i++],inverterstr(baralhosstr[j],auxinv));
845              }
846          }
847      }
848
849      //se tivermos a usar uma sequencia inicial ao procurar uma subsequencia
temos de também poder procurar pelas peças usadas na sequencia inicial
850
851      if(strcmp(seqinicial,"9|9")!=0)
852      {
853
854          strtok(seqinicialpartida,seqinicial,"-");
855
856          for(j=0; strcmp(seqinicialpartida[j],"9|9")!=0; j++)
857          {
858
859              strcpy(baralhosstrauxcominv[i++],seqinicialpartida[j]);
860          }
861
862          for(j=0; strcmp(seqinicialpartida[j],"9|9")!=0; j++)
863          {
864
865              if(seqinicialpartida[j][0]!=seqinicialpartida[j][2])
866              {
867
868
869
870          strcpy(baralhosstrauxcominv[i++],inverterstr(seqinicialpartida[j],auxinv));
871              }
872          }
873      }
874
875      }
876
877
878
879      //guardo o array de baralhos, baralhos invertidos e sequencia inicial
se existir num array auxiliar
880
881      for(i=0; strcmp(baralhosstrauxcominv[i],"9|9")!=0; i++)
882      {
883
884          strcpy(baralhosstrauxcominvnovo[i],baralhosstrauxcominv[i]);
885      }
886
887
888      for(i=0; strcmp(seqstr[i],"9|9")!=0; i++)
889      {
890
891          esvaziaseqstr(seqpadraoaux,LINSEQ);
892          strtok(seqpadraoaux,seqstr[i],"-");
893
894          for(j=0; strcmp(seqpadraoaux[j],"9|9")!=0; j++)
895          {
896
897              for(k=0; strcmp(baralhosstrauxcominv[k],"9|9")!=0; k++)
898              {
899
900                  if(strcmp(seqpadraoaux[j],baralhosstrauxcominv[k])==0)
901                  {
902
903                      strcpy(baralhosstrauxcominv[k],"-");

```

```

904
905         }
906
907     }
908
909     }
910
911     esvaziaseqstr(seqpadroaux,LINSEQ);
912
913     }
914
915     //retiro no array auxiliar as pecas que nao sairam nas sequencias e
essas nao podem estar contidas no padrao a substituir
916
917     for(i=0; strcmp(baralhosstrauxcominv[i],"9|9")!=0; i++)
918     {
919
920
921         if(strcmp(baralhosstrauxcominvnovo[i],baralhosstrauxcominv[i])==0)
922         {
923             strcpy(baralhosstrauxcominvnovo[k],"-");
924
925         }
926
927     }
928
929     //retiro as pecas que nao utilizei nas sequencias
930
931     for(j=0; strcmp(baralhosstrauxcominvnovo[j],"9|9")!=0 ; j++)
932     {
933
934         if(strcmp(baralhosstrauxcominv[j],"-")!=0)
935         {
936
937             strcpy(baralhosstrauxcominvnovo[j],"-");
938
939         }
940
941     }
942
943     //verifico quais as pecas que podem ter troca de padrao e guardo no
array com as suas respectivas invertidas
944     cont=0;
945     for(i=0; strcmp(baralhosstrauxcominv[i],"9|9")!=0; i++)
946     {
947
948         if(strcmp(baralhosstrauxcominv[i],"-")!=0)
949         {
950
951             for(j=0; strcmp(baralhosstrauxcominvnovo[j],"9|9")!=0 ;
j++)
952             {
953
954
955                 if(strcmp(inverterstr(baralhosstrauxcominv[i],auxinv),baralhosstrauxcominvnovo[j])==0)
956                 {
957                     for(k=0;
strcmp(baralhosstrauxcominvnovo[k],"9|9")!=0 ; k++)
958                     {
959
960                         if(strcmp(baralhosstrauxcominvnovo[k],"-")==0
&& cont==0)
961                         {
962
963                             strcpy(baralhosstrauxcominvnovo[k],baralhosstrauxcominv[i]);
964                             cont++;
965
966                         }

```

```

967
968
969         }
970         cont=0;
971     }
972
973     }
974
975     }
976
977     }
978
979
980
981     //se for 1 é porque nao é válida o padrao
982     if(verificasequencia(padrao)==1)
983     {
984
985         k=1;
986
987     }
988
989     //faz uma copia das pecas que podem ser usadas no padrao a substituir
990
991
992     for(i=0; strcmp(baralhosstrauxcominvnovo[i],"9|9")!=0; i++)
993     {
994
995
996     strcpy(baralhosstrauxcominvnovopadraonovo[i],baralhosstrauxcominvnovo[i]);
997
998     }
999
1000     //verifica se a peca do padrao a substituir já está a ser usada para
1001     nao haver repetidas
1002
1003     tamsubseq=strtoque(seqpadraoaux,padraoaux,'-');
1004     cont=0;
1005
1006     for(i=0; strcmp(seqpadraoaux[i],"9|9")!=0; i++)
1007     {
1008
1009         for(j=0;
1010         strcmp(baralhosstrauxcominvnovopadraonovo[j],"9|9")!=0; j++)
1011         {
1012             //printf("%s ==
1013             %s\n",seqpadraoaux[i],baralhosstrauxcominvnovopadraonovo[j]);
1014
1015             if(strcmp(seqpadraoaux[i],baralhosstrauxcominvnovopadraonovo[j])==0)
1016             {
1017
1018                 strcpy(baralhosstrauxcominvnovopadraonovo[j],"-");
1019                 cont++;
1020
1021                 for(r=0;
1022                 strcmp(baralhosstrauxcominvnovopadraonovo[r],"9|9")!=0; r++)
1023                 {
1024
1025                     if(strcmp(inverterstr(seqpadraoaux[i],pecasstrinv),baralhosstrauxcominvnovopadraonovo[r]
1026                     )==0)
1027                     {
1028
1029                         strcpy(baralhosstrauxcominvnovopadraonovo[r],"-");
1030
1031                     }
1032
1033                 }
1034
1035             }
1036
1037         }
1038     }

```



```

1029
1030
1031         }
1032
1033
1034     }
1035
1036
1037 }
1038
1039 if(k==1 || tamsubseq!=cont)
1040 {
1041
1042     printf("O padrao nao e valido!!\n\n");
1043
1044     esvaziaseqstr(seqinicialpartida,LINSEQ);
1045     esvaziaseqstr(baralhosstrauxcominv,LINSEQ);
1046     esvaziaseqstr(baralhosstrauxcominvnovo,LINSEQ);
1047     esvaziaseqstr(baralhosstrauxcominvnovopadraonovo,LINSEQ);
1048     cont=0;
1049     k=0;
1050     op2='2';
1051
1052 }
1053
1054 else
1055 {
1056
1057     //Verifico o padrao novo
1058
1059     esvaziaseqstr(seqfpadrao,LINSEQ);
1060     esvaziaseqstr(baralhosstrauxcominvnovopadraonovo,LINSEQ);
1061     esvaziaseqstr(baralhosstrauxcominv,LINSEQ);
1062     //faz uma copia das pecas que NAO podem ser usadas no padrao a
substituir
1063
1064     for(i=0; strcmp(baralhosstrauxcominvnovo[i],"9|9")!=0; i++)
1065     {
1066
1067
strcpy(baralhosstrauxcominvnovopadraonovo[i],baralhosstrauxcominvnovo[i]);
1068
1069     }
1070
1071     //pecas iniciais e invertidas
1072     for(i=0; i<LIN; i++)
1073     {
1074
1075         strcpy(baralhosstrauxcominv[i],pecasstr[i]);
1076
1077     }
1078
1079     for(j=0; j<LIN; j++)
1080     {
1081
1082         if(pecasstr[j][0]!=pecasstr[j][2])
1083         {
1084
1085
strcpy(baralhosstrauxcominv[i++],inverterstr(pecasstr[j],auxinv));
1086
1087         }
1088
1089     }
1090
1091     for(i=0; strcmp(baralhosstrauxcominv[i],"9|9")!=0; i++)
1092     {
1093
1094         for(j=0;
strcpy(baralhosstrauxcominvnovopadraonovo[j],"9|9")!=0; j++)
1095     {

```

```

1096
1097
1098 if(strcmp(baralhosstrauxcominv[i],baralhosstrauxcominvnovopadraonovo[j])==0)
1099 {
1100     strcpy(baralhosstrauxcominv[i],"-");
1101 }
1102 }
1103 }
1104 }
1105 }
1106 }
1107 }
1108
1109 printf("\n\n");
1110
1111
1112 printf("Qual o padrao novo:\n");
1113 scanf("%s",padraonovo);
1114
1115
1116 tamsubseq=strtoque(seqpadraoaux,padraonovo,'-');
1117 cont=0;
1118
1119
1120 if(verificasequencia(padraonovo)==1)
1121 {
1122
1123     k=1;
1124 }
1125 }
1126
1127 //verifico de o padrao novo possui pecas repetidas ou nao
1128
1129 for(i=0; strcmp(seqpadraoaux[i],"9|9")!=0; i++)
1130 {
1131
1132     for(j=0; strcmp(baralhosstrauxcominv[j],"9|9")!=0; j++)
1133     {
1134         //printf("%s ==
1135 %s\n",seqpadraoaux[i],baralhosstrauxcominvnovopadraonovo[j]);
1136         if(strcmp(seqpadraoaux[i],baralhosstrauxcominv[j])==0)
1137         {
1138             strcpy(baralhosstrauxcominv[j],"-");
1139             cont++;
1140
1141             for(r=0; strcmp(baralhosstrauxcominv[r],"9|9")!=0;
1142 r++)
1143 {
1144
1145 if(strcmp(inverterstr(seqpadraoaux[i],pecasstrinv),baralhosstrauxcominv[r])==0)
1146 {
1147     strcpy(baralhosstrauxcominv[r],"-");
1148 }
1149 }
1150 }
1151 }
1152 }
1153 }
1154 }
1155 }
1156 }
1157 }
1158 }
1159 }
1160 }
1161 }
1162

```

```

1163
1164         if(k==1 || cont!=tamsubseq)
1165         {
1166
1167             printf("O padrao novo nao e valido!!\n\n");
1168             esvaziaseqstr(seqpadraoaux,LINSEQ);
1169             esvaziaseqstr(baralhosstrauxcominv,LINSEQ);
1170             esvaziaseqstr(baralhosstrauxcominvnovo,LINSEQ);
1171             esvaziaseqstr(baralhosstrauxcominvnovopadraonovo,LINSEQ);
1172
1173             cont=0;
1174             k=0;
1175             op2='2';
1176
1177         }
1178     else //se for valido substitui pelo padrao novo
1179     {
1180
1181         inverterstr(padrao,padraoinv);
1182         inverterstr(padraonovo,padraonovoinv);
1183
1184         //SE QUISERMOS SUBSTITUIR A STRING TODA FAZEMOS ESTE CICLO no
1185         fim acrescentamos ao array de seq de padroes
1186         for(i=0; strcmp(seqstr[i],"9|9")!=0; i++)
1187         {
1188
1189             if(strcmp(seqstr[i],padrao)==0 ||
1190 strcmp(padraoinv,seqstr[i])==0)
1191             {
1192                 strcpy(padraototal,padraonovo);
1193             }
1194         }
1195     }
1196
1197     //envia o padrao e o paadrao novo normal e invertido
1198
1199
1200 trocapadrao(seqstr,numdeseq,padrao,padraonovo,seqfpadrao,&sizeseqfpadrao);
1201 trocapadrao(seqstr,numdeseq,padrao,padraonovoinv,seqfpadrao,&sizeseqfpadrao);
1202 trocapadrao(seqstr,numdeseq,padraoinv,padraonovo,seqfpadrao,&sizeseqfpadrao);
1203 trocapadrao(seqstr,numdeseq,padraoinv,padraonovoinv,seqfpadrao,&sizeseqfpadrao);
1204
1205
1206
1207     strcpy(seqfpadrao[sizeseqfpadrao],padraototal);
1208
1209
1210     cont=0;
1211     for(i=0; strcmp(seqfpadrao[i],"9|9")!=0; i++)
1212     {
1213
1214         for(j=0; strcmp(seqfpadrao[j],"9|9")!=0; j++)
1215         {
1216
1217             if(i!=j)
1218             {
1219                 if(strcmp(seqfpadrao[i],seqfpadrao[j])==0)
1220                 {
1221                     strcpy(seqfpadrao[i],"-");
1222                 }
1223             }
1224         }
1225     }
1226
1227

```

```

1228
1229         }
1230
1231         cont++;
1232     }
1233
1234     ordenarsequencias(seqfpadiao,cont);
1235     cont=tirartracosinvertidos(seqfpadiao,cont);
1236
1237
1238     printseqstr(seqfpadiao,0,cont);
1239
1240
1241     esvaziaseqstr(seqfpadiao,LINSEQ);
1242     sizeseqfpadiao=0;
1243
1244     }
1245
1246     }
1247
1248     break;
1249
1250     case '3':
1251
1252         save_txt_jogo(p,filename);
1253
1254     break;
1255
1256
1257     case '0':
1258
1259         mostrarjogosstr(baralhosstr,jogosaimprimir);
1260         op='2';
1261
1262         break;
1263
1264     default:
1265
1266         if(contseq==1)
1267         {
1268
1269             //numdeseq=seq(baralhosstr,seqstr,num);
1270             ordenarsequencias(seqstr,numdeseq);
1271             numdeseq=tirartracosinvertidos(seqstr,numdeseq);
1272             numdeseq=separarseqinvertidas(seqstr,numdeseq);
1273             printseqstr(seqstr,0,numdeseq);
1274             printf("\n");
1275
1276             mostrarjogosstr(baralhosstr,jogosaimprimir);
1277             //contseq=0;
1278
1279         }
1280
1281         else if(contseq==2)
1282         {
1283
1284             numdeseq=seqcomseqinicial(baralhosstr,seqstr,num,seqinicial);
1285             ordenarsequencias(seqstr,numdeseq);
1286             numdeseq=tirartracosinvertidos(seqstr,numdeseq);
1287             numdeseq=separarseqinvertidas(seqstr,numdeseq);
1288             tam=retiraseqinitrepetida(seqstr,numdeseq,seqinicial);
1289             ordenarsequencias(seqstr,numdeseq);
1290             printseqstr(seqstr,0,numdeseq-tam);
1291
1292             mostrarjogosstr(baralhosstr,jogosaimprimir);
1293             //contseq=0;
1294         }
1295         else
1296         {
1297
1298             mostrarjogosstr(baralhosstr,jogosaimprimir);

```

```

1299
1300         }
1301
1302         break;
1303
1304     }
1305
1306     break;
1307
1308     case '4':
1309
1310         printf("\n\n--MENU--\n\n");
1311         printf("1- Começar com uma sequencia\n");
1312         printf("2- Começar sem sequencia\n");
1313         printf("0- Voltar ao menu principal\n\n");
1314
1315         scanf(" %c",&op4);
1316         system("CLS");
1317
1318         switch(op4)
1319         {
1320
1321
1322         case '1':
1323             k=0;
1324             mostrarjogosstr(baralhosstr,jogosaimprimir);
1325             esvaziaseqstr(seqinicialpartida,LINSEQ);
1326             cont=0;
1327             printf("Insira a sequencia inicial:\n");
1328             scanf("%s",seqinicial);
1329
1330             if(verificasequencia(seqinicial)==1)
1331             {
1332
1333                 k=1;
1334
1335             }
1336
1337             //copia as pecas iniciais para o baralho vazio
1338
1339             for(i=0; i<LIN; i++)
1340             {
1341
1342                 strcpy(baralhosstrauxcominv[i],pecasstr[i]);
1343
1344             }
1345
1346             //copio as respectivas invertidas para o baralho auxiliar
1347
1348             for(j=0; j<LIN; j++)
1349             {
1350
1351                 if(pecasstr[j][0]!=pecasstr[j][2])
1352                 {
1353
1354                     strcpy(baralhosstrauxcominv[i++],inverterstr(pecasstr[j],auxinv));
1355
1356                 }
1357
1358             }
1359
1360
1361             for(i=0; strcmp(baralhosstrauxcominv[i],"9|9")!=0; i++)
1362             {
1363
1364                 for(j=0; strcmp(baralhosstr[j],"9|9")!=0; j++)
1365                 {
1366
1367                     if(strcmp(baralhosstrauxcominv[i],baralhosstr[j])==0 ||
1368                        strcmp(baralhosstrauxcominv[i],inverterstr(baralhosstr[j],pecasstrinv))==0)

```

```

1368         {
1369
1370             strcpy(baralhosstrauxcominv[i],"-");
1371
1372         }
1373
1374
1375     }
1376
1377 }
1378
1379
1380 //partimos a sequencia inicial partida e guardamos no array auxiliar
1381 strcpy(seqinicialaux,seqinicial);
1382 tamsubseq=strtoque(seqinicialpartida,seqinicialaux,'-');
1383
1384 //verifico de o padrao novo possui pecas repetidas ou nao
1385
1386 for(i=0; strcmp(seqinicialpartida[i],"9|9")!=0; i++)
1387 {
1388
1389     for(j=0; strcmp(baralhosstrauxcominv[j],"9|9")!=0; j++)
1390     {
1391         //printf("%s ==
1392 %s\n",seqpadraoaux[i],baralhosstrauxcominvnovopadraonovo[j]);
1393
1394         if(strcmp(seqinicialpartida[i],baralhosstrauxcominv[j])==0)
1395         {
1396             strcpy(baralhosstrauxcominv[j],"-");
1397             cont++;
1398
1399             for(r=0; strcmp(baralhosstrauxcominv[r],"9|9")!=0; r++)
1400             {
1401
1402                 if(strcmp(inverterstr(seqinicialpartida[i],pecasstrinv),baralhosstrauxcominv[r])==0)
1403                 {
1404                     strcpy(baralhosstrauxcominv[r],"-");
1405
1406                 }
1407
1408             }
1409
1410         }
1411
1412     }
1413
1414 }
1415
1416 }
1417
1418 }
1419
1420
1421 if(cont!=tamsubseq || k==1)
1422 {
1423     printf("A sequequencia nao e valida!!\n\n");
1424
1425     //volta a carregar o pecasstraux com as pecas todas e esvazia o
1426 array da sequencia inicial partida
1427     for(i=0; i<LIN; i++)
1428     {
1429
1430         strcpy(pecasstraux[i],pecasstr[i]);
1431
1432     }
1433     k=0;
1434     op2='2';

```

```

1435
1436
1437     }
1438     else
1439     {
1440
1441         for(i=0; i<LIN; i++)
1442         {
1443
1444             strcpy(pecasstraux[i], pecasstr[i]);
1445
1446         }
1447
1448         numdeseq=jogoadois(baralhosstr, seqstr, num, seqinicial);
1449         ordernarsequencias(seqstr, numdeseq);
1450         numdeseq=tirartracosinvertidos(seqstr, numdeseq);
1451         numdeseq=separarseqinvertidas(seqstr, numdeseq);
1452         printseqstr(seqstr, 0, numdeseq);
1453
1454         mostrarjogosstr(baralhosstr, jogosaimprimir);
1455         contseq=2;
1456         op='3';
1457
1458     }
1459
1460     break;
1461
1462     case '2':
1463
1464         strcpy(seqinicial, "9|9");
1465
1466         numdeseq=jogoadois(baralhosstr, seqstr, num, seqinicial);
1467         ordernarsequencias(seqstr, numdeseq);
1468         numdeseq=tirartracosinvertidos(seqstr, numdeseq);
1469         numdeseq=separarseqinvertidas(seqstr, numdeseq);
1470         printseqstr(seqstr, 0, numdeseq);
1471
1472         mostrarjogosstr(baralhosstr, jogosaimprimir);
1473         contseq=3;
1474         op='3';
1475
1476         break;
1477
1478     case '0':
1479
1480         op='4';
1481
1482         break;
1483
1484     default:
1485
1486         printf("OPCAO INVALIDA\n");
1487
1488         break;
1489
1490     }
1491
1492     break;
1493
1494
1495     }
1496
1497     }
1498
1499     return 0;
1500
1501 }

```

void mostrarjogosint (int *baralhos*[][COL], int *njogos*)

Função mostrarjogosint.

Funcao que recebe o numero de jogos a imprimir em inteiros e os jogos dos jogadores e imprime separadamente os jogos dos jogadores

Parâmetros:

<i>int</i>	baralhosi[][COL] array de inteiros que guarda os jogos dos jogadores
<i>int</i>	njogos numero de jogos a imprimir

```

3194 {
3200     switch(njogos)
3201     {
3202
3203         case 7:
3204             printf("JOGO 1:\n");
3205             printpecasint(baralhosi,7,2,0);
3206             printf("\n");
3207
3208             break;
3209
3210         case 14:
3211             printf("JOGO 1:\n");
3212             printpecasint(baralhosi,7,2,0);
3213             printf("JOGO 2:\n");
3214             printpecasint(baralhosi,14,2,7);
3215             printf("\n");
3216
3217             break;
3218
3219         case 21:
3220             printf("JOGO 1:\n");
3221             printpecasint(baralhosi,7,2,0);
3222             printf("JOGO 2:\n");
3223             printpecasint(baralhosi,14,2,7);
3224             printf("JOGO 3:\n");
3225             printpecasint(baralhosi,21,2,14);
3226             printf("\n");
3227
3228             break;
3229
3230         case 28:
3231             printf("JOGO 1:\n");
3232             printpecasint(baralhosi,7,2,0);
3233             printf("JOGO 2:\n");
3234             printpecasint(baralhosi,14,2,7);
3235             printf("JOGO 3:\n");
3236             printpecasint(baralhosi,21,2,14);
3237             printf("JOGO 4:\n");
3238             printpecasint(baralhosi,28,2,21);
3239             printf("\n");
3240
3241             break;
3242
3243     }
3244 }
```

void mostrarjogosintstruct (PECASINIT p, int njogos)

Função mostrarjogosstr.

Funcao que recebe o numero de jogos a imprimir em string e os jogos dos jogadores e imprime separadamente os jogos dos jogadores

Parâmetros:

<i>PECASINIT</i>	p estrutura deste tipo
<i>int</i>	njogos numero de jogos a imprimir

```

3138 {
3144     switch(njogos)
3145     {
3146
3147         case 7:
3148             printf("JOGO 1:\n");
3149             printpecasintstruct(p,0,7);
3150             printf("\n");
3151
3152             break;
3153
3154         case 14:
3155             printf("JOGO 1:\n");
3156             printpecasintstruct(p,0,7);
3157             printf("JOGO 2:\n");
3158             printpecasintstruct(p,7,14);
3159             printf("\n");
3160
3161             break;
3162
3163         case 21:
3164             printf("JOGO 1:\n");
3165             printpecasintstruct(p,0,7);
3166             printf("JOGO 2:\n");
3167             printpecasintstruct(p,7,14);
3168             printf("JOGO 3:\n");
3169             printpecasintstruct(p,14,21);
3170             printf("\n");
3171
3172             break;
3173
3174         case 28:
3175             printf("JOGO 1:\n");
3176             printpecasintstruct(p,0,7);
3177             printf("JOGO 2:\n");
3178             printpecasintstruct(p,7,14);
3179             printf("JOGO 3:\n");
3180             printpecasintstruct(p,14,21);
3181             printf("JOGO 4:\n");
3182             printpecasintstruct(p,21,28);
3183             printf("\n");
3184
3185             break;
3186
3187     }
3188
3189 }

```

void mostrarjogosstr (char *baralhoss*[][COLSTR], int *njogos*)

Funcao que recebe o numero de jogos a imprimir em string e os jogos dos jogadores e imprime separadamente os jogos dos jogadores

Parâmetros:

<i>char</i>	baralhoss[][COLSTR] array de strings que guarda os jogos dos jogadores
<i>int</i>	njogos numero de jogos a imprimir

```

3079 {
3080
3087     switch(njogos)
3088     {
3089
3090         case 7:
3091             printf("JOGO 1:\n");

```

```

3092     printpecasstr(baralhoss,0,7);
3093     printf("\n");
3094
3095     break;
3096
3097     case 14:
3098         printf("JOGO 1:\n");
3099         printpecasstr(baralhoss,0,7);
3100         printf("JOGO 2:\n");
3101         printpecasstr(baralhoss,7,14);
3102         printf("\n");
3103
3104     break;
3105
3106     case 21:
3107         printf("JOGO 1:\n");
3108         printpecasstr(baralhoss,0,7);
3109         printf("JOGO 2:\n");
3110         printpecasstr(baralhoss,7,14);
3111         printf("JOGO 3:\n");
3112         printpecasstr(baralhoss,14,21);
3113         printf("\n");
3114
3115     break;
3116
3117     case 28:
3118         printf("JOGO 1:\n");
3119         printpecasstr(baralhoss,0,7);
3120         printf("JOGO 2:\n");
3121         printpecasstr(baralhoss,7,14);
3122         printf("JOGO 3:\n");
3123         printpecasstr(baralhoss,14,21);
3124         printf("JOGO 4:\n");
3125         printpecasstr(baralhoss,21,28);
3126         printf("\n");
3127
3128     break;
3129
3130 }
3131
3132 }

```

void mostrarjogosstrstruct (PECASINIT *p*, int *njogos*)

Função mostrarjogosstrstruct.

Funcao que recebe o numero de jogos a imprimir em string e os jogos dos jogadores e imprime separadamente os jogos dos jogadores

Parâmetros:

<i>PECASINIT</i>	p estrutura deste tipo
<i>int</i>	njogos numero de jogos a imprimir

```

3025 {
3031     switch(njogos)
3032     {
3033
3034     case 7:
3035         printf("JOGO 1:\n");
3036         printpecasstrstruct(p,0,14);
3037         printf("\n");
3038
3039     break;
3040
3041     case 14:
3042         printf("JOGO 1:\n");
3043         printpecasstrstruct(p,0,7);

```

```

3044     printf("JOGO 2:\n");
3045     printpecasstrstruct(p,7,14);
3046     printf("\n");
3047
3048     break;
3049
3050     case 21:
3051         printf("JOGO 1:\n");
3052         printpecasstrstruct(p,0,7);
3053         printf("JOGO 2:\n");
3054         printpecasstrstruct(p,7,14);
3055         printf("JOGO 3:\n");
3056         printpecasstrstruct(p,14,21);
3057         printf("\n");
3058
3059         break;
3060
3061     case 28:
3062         printf("JOGO 1:\n");
3063         printpecasstrstruct(p,0,7);
3064         printf("JOGO 2:\n");
3065         printpecasstrstruct(p,7,14);
3066         printf("JOGO 3:\n");
3067         printpecasstrstruct(p,14,21);
3068         printf("JOGO 4:\n");
3069         printpecasstrstruct(p,21,28);
3070         printf("\n");
3071
3072         break;
3073
3074     }
3075
3076 }

```

void ordernarmatrizinteiros (int m[][2], int size)

Função ordernarmatrizinteiros.

esta função recebe uma matriz e ordena crescente ou decrescente

Parâmetros:

<i>int</i>	m[][2] matriz a ordenar
<i>int</i>	size tamanho da matriz

```

4387 {
4388
4394     //FAÇO A ORDENAÇÃO DE UMA MATRIZ
4395     int aux=0;
4396     int aux2=0;
4397     int j=0;
4398     int i=0;
4399
4400     for(j=0; j<size; j++)
4401     {
4402
4403         for(i=0; i<size; i++)
4404         {
4405             //TROCANDO O SINAL FAÇO ORDEM CRESCENTE OU DECRESCENTE
4406             if(m[j][0]<m[i][0])
4407             {
4408
4409                 aux=m[j][0];
4410                 aux2=m[j][1];
4411
4412                 m[j][0]=m[i][0];
4413                 m[j][1]=m[i][1];
4414

```

```

4415             m[i][0]=aux;
4416             m[i][1]=aux2;
4417
4418         }
4419
4420     }
4421
4422 }
4423
4424 }

```

void ordenarsequencias (char seqf[][COLSEQ], int size)

Função ordenarsequencias.

esta função ordena as sequenciass por ordem decrescente

Parâmetros:

<i>seqf</i> [][COLSEQ]	array de strings das sequencias finais a ordenar
<i>int</i>	size numero de sequencias finais

```

4431 {
4432     //esta função ordena as sequenciass por ordem decrescente
4433     char seqss[LINSEQ][COLSEQ];
4434     int tamanhos[LINSEQ][2];
4435     int i=0;
4436     int j=0;
4437     int aux=0;
4438     int aux2=0;
4439
4440     //esvazio a sequencia de strings
4441     esvaziaseqstr(seqss,size);
4442
4443     //guardo o tamanho da seq na primeira posição de uma matriz e na segunda posição
4444     guardo qual a posição da sequencia do array de sequencias
4445     for(i=0; i<size; i++)
4446     {
4447         tamanhos[i][0]=strlen(seqf[i]);
4448         tamanhos[i][1]=i;
4449     }
4450
4451     //faço o selection sort para ordenar decrescentemente
4452     for(j=0; j<size; j++)
4453     {
4454         for(i=0; i<size; i++)
4455         {
4456             if(tamanhos[j][0]>tamanhos[i][0])
4457             {
4458                 aux=tamanhos[j][0];
4459                 aux2=tamanhos[j][1];
4460
4461                 tamanhos[j][0]=tamanhos[i][0];
4462                 tamanhos[j][1]=tamanhos[i][1];
4463
4464                 tamanhos[i][0]=aux;
4465                 tamanhos[i][1]=aux2;
4466             }
4467         }
4468     }
4469 }

```

```

4481     }
4482 }
4483
4484 /*for (i=0;i<size;i++){
4485     for (j=0;j<2;j++){
4486         printf("%d|",tamanhos[i][j]);
4487     }
4488     printf("\n");
4489 }
4490 */
4491
4492     for(i=0; i<size; i++)
4493     {
4494         strcpy(seqss[i],seqf[tamanhos[i][1]]);
4495     }
4496     esvaziaseqstr(seqf,size);
4497
4498     for(i=0; i<size; i++)
4499     {
4500         strcpy(seqf[i],seqss[i]);
4501     }
4502     //printseqstr(seqf,0,size);
4503 }
4504 }
4505 }
4506 }
4507 }
4508 }
4509 }
4510 }
4511 }
4512 }
4513 }
4514 }
4515 }
4516 }
4517 }

```

void ordenarsequenciasstruct (PECASINIT * p)

Função ordenarsequencias struct.

esta função ordena as sequenciass por ordem decrescente

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo struct
------------------	-----------------------------

```

4521 {
4522     //esta função ordena as sequenciass por ordem decrescente
4523     int i=0;
4524     int j=0;
4525     int aux=0;
4526     int aux2=0;
4527     int nseq=0;
4528
4529     //esvazio a sequencia de strings
4530
4531     nseq=p->nseqss;
4532
4533     for(i=0;i<nseq;i++)
4534     {
4535         remove seqss(p);
4536     }
4537 }
4538 }
4539 }
4540 }
4541 }
4542 }
4543 }
4544 }
4545 }

```

```

4546 //guardo o tamanho da seq na primeira posição de uma matriz e na segunda posição
guardo qual a posição da sequencia do array de sequencias
4547 for(i=0; i<p->nseqf; i++)
4548 {
4549     inserir_ordenaseq(p, strlen((p->seqf+i)->seqstr), i);
4551 }
4552
4553
4554
4555 //faço o selection sort para ordenar decrescentemente
4556 for(i=0; i<p->nseqf; i++)
4557 {
4558     for(j=0; j<p->nseqf; j++)
4559     {
4560         if((p->ordenaseq+j)->tamanho < (p->ordenaseq+i)->tamanho)
4561         {
4562             aux=(p->ordenaseq+j)->tamanho;
4563             aux2=(p->ordenaseq+j)->indice;
4564             (p->ordenaseq+j)->tamanho=(p->ordenaseq+i)->tamanho;
4565             (p->ordenaseq+j)->indice=(p->ordenaseq+i)->indice;
4566             (p->ordenaseq+i)->tamanho=aux;
4567             (p->ordenaseq+i)->indice=aux2;
4568         }
4569     }
4570 }
4571
4572
4573
4574
4575
4576
4577
4578
4579
4580 for(i=0; i<p->nseqf; i++)
4581 {
4582     inserir_seqss(p, (p->seqf+(p->ordenaseq+i)->indice)->seqstr);
4584 }
4585
4586
4587 nseq=p->nseqf;
4588
4589 for(i=0; i<nseq; i++)
4590 {
4591     remove_seqf(p);
4592 }
4593
4594
4595
4596 for(i=0; i<p->nseqss; i++)
4597 {
4598     inserir_seqf(p, (p->seqss+i)->seqstr);
4599 }
4600
4601
4602
4603 }

```

int preenchebaralhosstruct (char pecass[][COLSTR], PECASINIT * p, int n)

função para inserir os baralhos dos jogadores manualmente

Parâmetros:

<i>int</i>	pecass[][COLSTR] array que possui as pecas todas do jogo
<i>PECASINIT</i>	*p estrutura do tipo struct
<i>int</i>	n numero de jogos a preencher manualmente

```

2203 {
2204
2211     //função para inserir os baralhos dos jogadores
2212     int i=0;
2213     int j=0;
2214     int jogos=0;
2215     char aux[4];
2216     int existe=0;
2217     int elimina=0;
2218     int tam=0;
2219     int cont=0;
2220     int begin=0;
2221     char pecasaux[LIN][COLSTR];
2222
2223     tam=LIN;
2224     jogos=n*7;
2225     //guardo as pecas num array auxiliar
2226     for(i=0; i<tam; i++)
2227     {
2228
2229         strcpy(pecasaux[i],pecass[i]);
2230
2231     }
2232
2233     printf("SELECIONE %d JOGOS DE (7 PECAS) DA LISTA: (ex: 5|2 )\n\n",n);
2234     //de 0 até ao numero de jogos a inserir., inserem-se peças peças
2235     while(j<n)
2236     {
2237
2238         for(i=begin; i<jogos; i++)
2239         {
2240
2241             printpecasstr(pecasaux,0,tam);
2242             scanf("%s",aux);
2243             system("CLS");
2244
2245             //Verifica as pecas todas
2246             for(existe=0; existe<tam; existe++)
2247             {
2248
2249                 //Se a peca existe nas pecas a escolher então copia-se para o baralho
da pessoa
2250                 if(strcmp(pecasaux[existe],aux)==0)
2251                 {
2252
2253
2254                     inserir_pecas(p,pecasaux[existe]);
2255
2256                     //strcpy(baralhoss[i],pecasaux[existe]);
2257
2258                     //Coloco todas as peças para cima
2259                     for(elimina=existe; elimina<(tam-1); elimina++)
2260                     {
2261
2262                         strcpy(pecasaux[elimina],pecasaux[elimina+1]);
2263
2264                     }
2265
2266                     cont++;
2267
2268                 }
2269             }
2270             //se guarda uma peca no baralho da proxima vez já não pode escolher essa
peça
2271             if(cont==1)
2272             {
2273                 tam--;
2274                 cont=0;
2275
2276             }
2277             else

```

```

2278         {
2279             //Se a peça não existe escreve a mensagem que não é válida
2280             printf("NAO E VALIDO!\n");
2281             printf("SELECIONE DA LISTA: (ex: 5|2 )\n\n");
2282             i--;
2283         }
2284     }
2285     begin=begin+7;
2286     jogos=jogos+7;
2287     j++;
2288 }
2289
2290 return n*7;
2291
2292 }

```

void printpecasint (int *pecas*[][COL], int *l*, int *c*, int *inicio*)

Função printpecasint.

função que imprime as pecas inteiras

Parâmetros:

<i>int</i>	pecasi[][COL] pecas total inteiras
<i>int</i>	l linhas da matriz
<i>int</i>	c colunas da matriz
<i>int</i>	inicio variavel que contem o numero a partir do qual queremos imprimir

```

2850 {
2851
2860     int i=0;
2861     int j=0;
2862
2863     for(i=inicio; i<l; i++)
2864     {
2865         for(j=0; j<c; j++)
2866         {
2867
2868             printf("%d",pecasi[i][j]);
2869
2870             if(j==0)
2871             {
2872
2873                 printf("|");
2874
2875             }
2876         }
2877         printf("\n");
2878     }
2879
2880 }

```

void printpecasintstruct (PECASINIT *p*, int *inicio*, int *fim*)

Função printpecasintstruct.

função para imprimir peças inteiras

Parâmetros:

<i>int</i>	inicio desde quando é que queremos imprimir
<i>int</i>	fim até onde queremos imprimir
<i>PECASINIT</i>	p estrutura deste tipo


```

2885 {
2886
2894     int i=0;
2895
2896     PECA *paux=NULL;
2897     paux=p.pfirst;
2898
2899     while(paux!=NULL && i<fim)
2900     {
2901
2902         if(i>=inicio)
2903         {
2904
2905             printf("%s\n",paux->str);
2906
2907         }
2908
2909         paux=paux->pnext;
2910         i++;
2911
2912     }
2913
2914 }

```

void printpecasstr (char pecass[][COLSTR], int inicio, int fim)

Função printpecasstr.

função para imprimir peças string

Parâmetros:

<i>int</i>	inicio desde quando é que queremos imprimir
<i>int</i>	fim até onde queremos imprimir

```

2953 {
2959     int i=0;
2960
2961     for(i=inicio; i<fim; i++)
2962     {
2963
2964         printf("%s\n",pecass[i]);
2965
2966     }
2967
2968 }

```

void printpecasstrstruct (PECASINIT p, int inicio, int fim)

Função printpecasintstruct.

função para imprimir peças string

Parâmetros:

<i>int</i>	inicio desde quando é que queremos imprimir
<i>int</i>	fim até onde queremos imprimir
<i>PECASINIT</i>	p estrutura deste tipo

```

2919 {
2920
2928     int i=0;
2929
2930     PECA *paux=NULL;
2931     paux=p.pfirst;

```

```

2932
2933     while(paux!=NULL && i<fim)
2934     {
2935
2936         if(i>=inicio)
2937         {
2938
2939             printf("%s\n",paux->str);
2940
2941         }
2942
2943         paux=paux->pnext;
2944         i++;
2945     }
2946 }
2947
2948 }

```

void printseqstr (char seqss[][COLSEQ], int inicio, int fim)

Função printseqstr.

função para imprimir sequencias

Parâmetros:

<i>int</i>	inicio desde quando é que queremos imprimir
<i>int</i>	fim até onde queremos imprimir

```

2974 {
2981     //funcao para imprimir sequencias de strings
2982     int i=0;
2983
2984     for(i=inicio; i<fim; i++)
2985     {
2986
2987         printf("i=[%d] %s\n",i,seqss[i]);
2988
2989     }
2990
2991 }

```

void printseqstrstruct (PECASINIT p)

Função printpecasstruct.

função para imprimir sequencias

Parâmetros:

<i>int</i>	inicio desde quando é que queremos imprimir
<i>int</i>	fim até onde queremos imprimir
<i>PECASINIT</i>	p estrutura deste tipo

```

2996 {
3004     //funcao para imprimir sequencias de strings
3005     int i=0;
3006
3007     for(i=0; i<p.nseqf; i++)
3008     {
3009         if((p.seqf+i)->seqstr!=NULL)
3010         {
3011
3012             printf("i=[%d] %s\n",i,(p.seqf+i)->seqstr);
3013

```

```

3014     }
3015
3016
3017     }
3018
3019 }

```

int procsubseq (char seqf[][COLSEQ], int size, char subs[])

Função procsubseq.

função para procurar uma substring numa string

Parâmetros:

<i>char</i>	seqf[][COLSEQ] array final de strings
<i>int</i>	size numero da linha do array de strings final que vamos querer procurar uma substring
<i>char</i>	subs[] sub string a procurar

Retorna:

retorna a posicao em que encontrou a substring

```

4272 {
4281     //esta função procura sequencias de peças noutras sequencias, ou peças em
sequencias
4282     int i=0;
4283     int k=0;
4284     int cont=0;
4285     int pos=0;
4286     int x=0;
4287     int v[LINSEQ];
4288     int tamstr1=0;
4289     int tamstr2=0;
4290     char arrayseq[LINSEQ][COLSEQ];
4291     char arraysub[LINSEQ][COLSEQ];
4292
4293     //o size neste caso é a linha do array de strings
4294
4295     //faço o strtok da sequencias de peças
4296     tamstr1=strtoque(arrayseq,seqf[size],'-');
4297     //faço o strtok da sequencia a procurar
4298     tamstr2=strtoque(arraysub,subs,'-');
4299
4300     for (i=0; i<tamstr1; i++)
4301     {
4302         //enquanto não estiver no fim da sequencia e houver igualdades entre a sequencia
e a sub sequencia vou comparando a proxima peça da subsequencia com a peça da sequencia
4303         if(strcmp(arrayseq[i],arraysub[k])==0)
4304         {
4305             if(k==0)
4306             {
4307                 //guardo a posição em que encontrei a primeira igualdade entre a
subsequencia e a sequencia
4308                 pos=i;
4309             }
4310             cont++;
4311             k++;
4312
4313
4314             //se a sub sequencia e a sequencia são iguais então guardo a posição num
array de inteiros e procuro outra igualdade
4315             //se é que existe entre a subsequencia e a sequencia de peças
4316
4317             if(cont==k && cont==tamstr2)
4318             {
4319                 v[x]=pos;
4320                 x++;

```

```

4321             k=0;
4322             cont=0;
4323             pos=0;
4324         }
4325     }
4326     }
4327     else
4328     {
4329
4330         k=0;
4331         cont=0;
4332         pos=0;
4333
4334     }
4335 }
4336
4337 return x;
4338 //se quisermos imprimir as varias ocorrencias
4339
4340 //imprime as posicoes nas quais encontrou o inicio da igualdade entre as
subsequencias e as sequencias
4341 for(i=0; i<x; i++)
4342 {
4343
4344     printf("%d\n",v[i]);
4345
4346 }
4347
4348 }

```

int procsubseq_ausar (char seqf[][COLSEQ], int size, char subs[])

Função procsubseq_ausar.

função para procurar uma substring numa string

Parâmetros:

<i>char</i>	seqf[][COLSEQ] array final de strings
<i>int</i>	size numero da linha do array de strings final que vamos querer procurar uma substring
<i>char</i>	subs[] sub string a procurar

Retorna:

retorna a posicao em que encontrou a substring

```

4178 {
4179
4187 //esta função procura sequencias de peças noutras sequencias, ou peças em
sequencias
4188 int i=0;
4189 int k=0;
4190 int cont=0;
4191 int pos=0;
4192 int x=0;
4193 int v[LINSEQ];
4194 int tamstr1=0;
4195 int tamstr2=0;
4196 char arrayseq[LINSEQ][COLSEQ];
4197 char arraysub[LINSEQ][COLSEQ];
4198
4199 //o size neste caso é a linha do array de strings
4200
4201 //faço o strtok da sequencias de peças
4202 tamstr1=strtoque(arrayseq,seqf[size],'-');
4203 //faço o strtok da sequencia a procurar
4204 tamstr2=strtoque(arraysub,subs,'-');
4205

```

```

4206     for (i=0; i<tamstr1; i++)
4207     {
4208         //enquanto não estiver no fim da sequencia e houver igualdades entre a sequencia
e a sub sequencia vou comparando a proxima peça da subsequencia com a peça da sequencia
4209         if(strcmp(arrayseq[i],arraysub[k])==0)
4210         {
4211             if(k==0)
4212             {
4213                 //guardo a posição em que encontrei a primeira igualdade entre a
subsequencia e a sequencia
4214                 pos=i;
4215             }
4216             cont++;
4217             k++;
4218
4219
4220             //se a sub sequencia e a sequencia são iguais então guardo a posição num
array de inteiros e procuro outra igualdade
4221             //se é que existe entre a subsequencia e a sequencia de peças
4222
4223             if(cont==k && cont==tamstr2)
4224             {
4225                 v[x]=pos;
4226                 x++;
4227                 k=0;
4228                 cont=0;
4229                 pos=0;
4230
4231             }
4232         }
4233         else
4234         {
4235
4236             k=0;
4237             cont=0;
4238             pos=0;
4239
4240         }
4241     }
4242     if(x==0)
4243     {
4244
4245         return -1;
4246     }
4247 }
4248 else
4249 {
4250
4251     return v[0];
4252 }
4253 }
4254
4255 //se quisermos imprimir as varias ocorrencias
4256
4257 //imprime as posições nas quais encontrou o inicio da igualdade entre as
subsequencias e as sequencias
4258 /*for(i=0;i<x;i++)
4259 {
4260
4261     printf("%d\n",v[i]);
4262 }*/
4263 */
4264
4265 }

```

int procsubseq_trocapadiao (char seqf[][COLSEQ], int size, char subs[LIN], int l)

Função procsubseq_trocapadiao.

função para procurar uma substring numa string (utilizei esta variante na torca de padrao)

Parâmetros:

<i>char</i>	seqf[][COLSEQ] array final de strings
<i>int</i>	size numero da linha do array de strings final que vamos querer procurar uma substring
<i>char</i>	subs[] sub string a procurar
<i>int</i>	l numero da string que vamos procurar uma substring

Retorna:

retorna a posicao em que encontrou a substring

```
4608 {
4609     char auxsub[LINSEQ];
4610     char auxseq[LINSEQ];
4611     char arrayseq[LINSEQ][COLSEQ];
4612     char arraysub[LINSEQ][COLSEQ];
4613     char *sub;
4614     char *seq;
4615     int contsub=0;
4616     int contseq=0;
4617     int i=0;
4618     int j=0;
4619     int k=0;
4620     int cont=0;
4621     int posicao=0;
4622     int sequencia=0;
4623     int imprime=0;
4624     int volta=0;
4625
4626     //todas as sequencias do array seqf
4627     for(sequencia=0; sequencia<size; sequencia++)
4628     {
4629
4630         esvaziaseqstr(arrayseq,LINSEQ);
4631         esvaziaseqstr(arraysub,LINSEQ);
4632
4633         strcpy(auxsub,subs);
4634         sub = strtok (auxsub,"-");
4635
4636         while (sub != NULL)
4637         {
4638
4639             strcpy(arraysub[contsub],sub);
4640             contsub++;
4641             sub = strtok (NULL, "-");
4642
4643         }
4644
4645         strcpy(auxseq,seqf[sequencia]);
4646         seq = strtok (auxseq,"-");
4647         i=0;
4648
4649         while (seq != NULL)
4650         {
4651
4652             strcpy(arrayseq[contseq],seq);
4653             contseq++;
4654             seq = strtok (NULL, "-");
4655
4656         }
4657
4658         /*printseqstr(arrayseq,0,contseq);
4659         printf("\n\n");
4660         printseqstr(arraysub,0,contsub);*/
4661
4662     }
```

```

4673         for(i=0; i<contsub; i++)
4674         {
4675             for(j=1; j<contseq; j++)
4676             {
4677                 //comparo ate encontrar a primeira peca da sub sequencia
4678                 if(strcmp(arraysub[k],arrayseq[j])==0)
4679                 {
4680                     if(cont==0)
4681                     {
4682                         posicao=j;
4683                         cont++;
4684                     }
4685                     //se o cont for igual às pecas da subseq e se estiverem seguidas
4686                     a volta tem de ser 0 para estarem seguidas
4687                     if(cont==contsub&&volta==0)
4688                     {
4689                         imprime=1;
4690                     }
4691                     k++;
4692                 }
4693                 if(cont>0)
4694                 {
4695                     cont++;
4696                 }
4697             }
4698             volta++;
4699         }
4700         if (imprime==1)
4701         {
4702             //retorna a posição na qual encontrou a subsequencia para permitir na
4703             função trocar padrao fazer a troca por um padrao diferente
4704             return posicao;
4705             //printf("%s --> %d\n",seqf[sequencia],posicao);
4706         }
4707         else
4708         {
4709             return -1;
4710         }
4711         posicao=0;
4712         k=0;
4713         cont=0;
4714         contsub=0;
4715         contseq=0;
4716         imprime=0;
4717         volta=0;
4718     }
4719     return 0;
4720 }

```

void remove_pecas (PECASINIT * p, char remove[])

Remover pecas dos jogos.

função para remover pecas nas listas ligadas

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>int</i>	n numero de jogos a preencher manualmente

```

3598 {
3599
3606     PECA *paux=NULL;
3607     PECA *pant=NULL;
3608
3609     paux=p->pfirst;
3610
3611     //cabeca
3612     if (strcmp (paux->str, remove)==0)
3613     {
3614
3615         p->pfirst=paux->pnext;
3616         p->npecas--;
3617         return;
3618     }
3619
3620     //cauda
3621     paux=p->pfirst;
3622     while (paux->pnext!=NULL)
3623     {
3624
3625
3626         pant=paux;
3627         paux=paux->pnext;
3628     }
3629
3630     if (strcmp (paux->str, remove)==0)
3631     {
3632
3633         pant->pnext=NULL;
3634         p->npecas--;
3635         return;
3636     }
3637
3638     //meio
3639     paux=p->pfirst;
3640     while (paux->pnext!=NULL)
3641     {
3642
3643
3644         pant=paux;
3645         paux=paux->pnext;
3646
3647         if (strcmp (paux->str, remove)==0)
3648         {
3649
3650             pant->pnext=paux->pnext;
3651             p->npecas--;
3652             return;
3653         }
3654     }
3655 }
3656
3657 }
```

void remove_seqf (PECASINIT * p)

```

3557 {
```



```

3558
3559     SEQ *paux=NULL;
3560     paux=p->seqf;
3561
3562     //cabeca
3563     p->seqf=paux+1;
3564     p->nseqf--;
3565     return;
3566
3567 }

```

void remove_seqss (PECASINIT * p)

```

3570 {
3571
3572     SEQ *paux=NULL;
3573     paux=p->seqss;
3574
3575     //cabeca
3576     p->seqss=paux+1;
3577     p->nseqss--;
3578     return;
3579
3580 }

```

void remove_seqssaux (PECASINIT * p)

```

3583 {
3584
3585     SEQ *paux=NULL;
3586     paux=p->seqssaux;
3587
3588     //cabeca
3589     p->seqssaux=paux+1;
3590     p->nseqssaux--;
3591     return;
3592
3593 }

```

int remover (PECASINIT * p, char pecass[][COLSTR], int num)

Função remover.

esta funcao remove as pecas inseridas

Parâmetros:

<i>char</i>	baralhoss[][COLSTR] baralhos do jogador em string
<i>char</i>	pecass[][COLSTR] pecas todas do jogo em strings
<i>int</i>	num numero de jogos a converter

Retorna:

retorna o numero de jogos a remover

```

3436 {
3444     int i=0;
3445     int n=num;
3446     char aux[4];
3447     char remove[4];
3448     int pecasaremove=0;
3449     n=n*7;
3450
3451     PECA *pfind=NULL;
3452
3453
3454     do
3455     {
3456

```

```

3457     mostrarjogosstrstruct(*p,n);
3458     printf("\nQUANTAS PECAS PRETENDE REMOVER?\n");
3459     scanf("%d",&pecasaremove);
3460     system("CLS");
3461
3462     if(pecasaremove>n)
3463     {
3464
3465         printf("NAO PODE REMOVER ESSE NUMERO DE PECAS!!!\n\n");
3466         mostrarjogosstrstruct(*p,n);
3467
3468     }
3469
3470 }
3471 while(pecasaremove>n);
3472
3473 system("CLS");
3474
3475 for(i=0; i<pecasaremove; i++)
3476 {
3477
3478     mostrarjogosstrstruct(*p,n);
3479     printf("QUAL A PECA A REMOVER:\n");
3480     scanf("%s",aux);
3481     system("CLS");
3482
3483     pfind=find_pecas_baralho(p,aux);
3484
3485     if(pfind==NULL)
3486     {
3487
3488         printf("A PECA QUE ESCOLHEU NAO E VALIDA!\n");
3489
3490         if(i==0)
3491         {
3492
3493             i=0;
3494
3495         }
3496
3497         else
3498         {
3499
3500             i--;
3501
3502         }
3503
3504     }
3505     else
3506     {
3507
3508         strcpy(remove,pfind->str);
3509         remove_pecas(p,remove);
3510
3511     }
3512
3513 }
3514
3515 printf("INSIRA AS %d PECAS NOVAS\n",pecasaremove);
3516 for(i=0; i<pecasaremove; i++)
3517 {
3518     mostrarjogosstrstruct(*p,n);
3519     scanf("%s",aux);
3520     pfind=find_pecas_baralho(p,aux);
3521
3522     if(pfind!=NULL)
3523     {
3524
3525         printf("A PECA QUE ESCOLHEU NAO E VALIDA!\n");
3526
3527         if(i==0)

```

```

3528         {
3529             i=0;
3530         }
3531     }
3532     else
3533     {
3534         i--;
3535     }
3536 }else
3537 {
3538     inserir_peca(p,aux);
3539 }
3540 }
3541 return n;
3542 }
3543 }

```

int retiraseqinitrepetida (char seqf[][COLSEQ], int size, char seqinit[])

Função retiraseqinitrepetida.

Nesta funcao verifico se encontro a sequencia inicial ou a inicial invertida repetida na mesma sequencia para poder eliminá-la

Parâmetros:

char	seqf[][COLSEQ] array final onde vou guardar as sequencias todas possiveis
int	size numero de jogos a retirar sequencias com sequencias iniciais repetidas
char	seqinit[] sequencia inicial

Retorna:

retorna o numero de sequencias que encontrou

```

5435 {
5436     int i=0;
5437     int cont=0;
5438     char seqinitinv[LINSEQ];
5439     for(i=0; seqinit[i]!='\0'; i++)
5440     {
5441         seqinitinv[strlen(seqinit)-1-i]=seqinit[i];
5442     }
5443     //aqui verifico se encontro a sequencia inicial ou a inicial invertida repetida
5444     //na mesma sequencia para poder eliminá-la
5445     //visto que considero a sequencia inicial ou a sequencia invertida como se fosse
5446     //uma peça tomando apenas em consideração
5447     //as extremidades
5448     for(i=0; i<size; i++)
5449     {
5450         if(probsubseq(seqf,i,seqinit)>1)

```

```

5464     {
5465
5466         strcpy(seqf[i],"9|9");
5467         cont++;
5468     }
5469     if (procsubseq(seqf,i,seqinitinv)>1)
5470     {
5471
5472         strcpy(seqf[i],"9|9");
5473         cont++;
5474     }
5475 }
5476 }
5477 }
5478 }
5479 }
5480 }
5481 return cont;
5482 }
5483 }

```

void save_jogo_bin (PECASINIT p, char fname[])

Função save_jogo_bin.

função para guardar jogos

Parâmetros:

<i>char</i>	fname[] nome do ficheiro a guardar
<i>PECASINIT</i>	p estrutura deste tipo

```

1506 {
1507
1514     PECA *paux=NULL;
1515     SEQ *paux2=NULL;
1516
1517     paux=p.pfirst;
1518     paux2=p.seqf;
1519
1520     FILE *fp=NULL;
1521
1522     int size=0;
1523     int i=0;
1524
1525     if ((fp=fopen(fname,"wb")) !=NULL)
1526     {
1527
1528         fwrite(&p.npecas,sizeof(int),1,fp);
1529         //gravar o numero de pecas
1530
1531         while (paux!=NULL)
1532         {
1533
1534             size=strlen((p.pfirst->str))+1;
1535             fwrite(&size,sizeof(int),1,fp);
1536             fwrite(paux->str,sizeof(char),size,fp);
1537             paux=paux->pnext;
1538
1539         }
1540
1541
1542
1543         for (i=0;i<p.nseqf;i++)
1544         {
1545
1546             size=strlen((p.seqf+i)->seqstr)+1;
1547

```

```

1548         fwrite(&size, sizeof(int), 1, fp);
1549         fwrite((paux2+i)->seqstr, sizeof(char), size, fp);
1550
1551     }
1552     fclose(fp);
1553 }
1554
1555
1556 }

```

void save_txt_jogo (PECASINIT p, char fname[])

Função save_jogo_txt.

função para guardar jogos txt

Parâmetros:

<i>char</i>	fname[] nome do ficheiro a guardar
<i>PECASINIT</i>	p estrutura deste tipo

```

1659 {
1660
1661     FILE *fp=NULL;
1662     int i=0;
1663
1664     if ((fp = fopen(fname, "w")) == NULL)
1665     {
1666         printf("save txt jogo(): Erro abrir ficheiro %s\n", fname);
1667         return;
1668     }
1669
1670     int j=0;
1671     int fim=7;
1672
1673     PECA *paux=NULL;
1674     paux=p.pfirst;
1675
1676     //guarda os jogos no ficheiro
1677     fprintf(fp, "JOGOS\n\n");
1678
1679     for(i=0; i<p.npecas/7; i++)
1680     {
1681         fprintf(fp, "JOGO %d\n", i+1);
1682
1683         while(paux!=NULL && j<fim)
1684         {
1685             fprintf(fp, "%s\n", paux->str);
1686             paux=paux->pnext;
1687             j++;
1688         }
1689         j=j+7;
1690         fim=fim+7;
1691     }
1692
1693     //guarda as sequencias
1694     fprintf(fp, "\n");
1695
1696     for(i=0; i<p.nseqf; i++)
1697     {
1698         if ((p.seqf+i)->seqstr!=NULL)

```

```

1713     {
1714
1715         fprintf(fp,"i=[%d]  %s\n",i,(p.seqf+i)->seqstr);
1716     }
1717
1718
1719
1720 }
1721
1722
1723 fclose(fp);
1724
1725 }

```

int separarseqinvertidas (char seqf[][COLSEQ], int numdeseq)

Função separarseqinvertidas.

função que distingue as sequencias iguais lidas da esquerda para a direita às lidas da direita para a esquerda e chama a funcao elimina rep para retirar as que sao iguais escritas ao contrário

Parâmetros:

<i>char</i>	seqf[][COLSEQ] array de sequencias
<i>int</i>	numdeseq guarda o numero de sequencias

Retorna:

retorna o numero de sequencias

dou indices às sequencias invertidas e às sequencias normais para dps eliminar os duplicados

```

2557 {
2558
2559     int i=0;
2560     int j=0;
2561     int v[LINSEQ];
2562     char auxseqinvertidas[LINSEQ][COLSEQ];
2563     char seqaux[LINSEQ][COLSEQ];
2564     int *vseqs=NULL;
2565
2566
2567     //inverte as sequencias
2568     esvaziaseqstr(auxseqinvertidas,LINSEQ);
2569     esvaziaseqstr(seqaux,LINSEQ);
2570
2571     for(i=0; i<numdeseq; i++)
2572     {
2573         while(seqf[i][j]!='\0')
2574         {
2575             auxseqinvertidas[i][strlen(seqf[i])-1-j]=seqf[i][j];
2576             j++;
2577         }
2578         auxseqinvertidas[i][strlen(seqf[i])]='\0'; // mete um '\0' no fim de string
2579         j=0;
2580         v[i]=-1;
2581     }
2582
2583     for(i=0; i<numdeseq; i++)
2584     {
2585         for(j=0; j<numdeseq; j++)
2586         {
2587             if(strcmp(seqf[i],seqf[j])==0 && v[j] == -1)

```

```

2603         {
2604             v[j]=i;
2605         }
2606     }
2607
2608     for(j=0; j<numdeseq; j++)
2609     {
2610         if(strcmp(auxseqinvertidas[i],seqf[j])==0 && v[j] == -1)
2611         {
2612             v[j]=i;
2613         }
2614     }
2615
2616     //eliminar repetidos
2617     //vseqs=eliminarrep(v, &numdeseq);
2618
2619     for(i=0; i<numdeseq; i++)
2620     {
2621         strcpy(seqaux[i], seqf[* (vseqs+i)]);
2622     }
2623     free(vseqs);
2624     esvaziaseqstr(seqf, LINSEQ);
2625
2626     for(i=0; i<numdeseq; i++)
2627     {
2628         strcpy(seqf[i], seqaux[i]);
2629     }
2630
2631     return numdeseq;
2632 }

```

int separarseqinvertidasstruct (PECASINIT * p)

Função separarseqinvertidas struct.

função que distingue as sequencias iguais lidas da esquerda para a direita às lidas da direita para a esquerda e chama a funcao elimina rep para retirar as que sao iguais escritas ao contrário

Parâmetros:

<i>PECASINIT</i>	*p estrutura deste tipo
------------------	-------------------------

Retorna:

retorna o numero de sequencias

dou indices às sequencias invertidas e às sequencias normais para dps eliminar os duplicados

```

2436 {
2437     int i=0;

```

```

2445     int j=0;
2446     int v[LINSEQ];
2447     int *vseqs=NULL;
2448     int nseq=0;
2449
2450     //esvaziar auxiliar
2451     nseq=p->nseqss;
2452
2453     for(i=0;i<nseq;i++)
2454     {
2455
2456         remove_seqss(p);
2457     }
2458
2459     nseq=p->nseqssaux;
2460
2461     for(i=0;i<nseq;i++)
2462     {
2463
2464         remove_seqssaux(p);
2465     }
2466
2467     //inverte as sequencias
2468
2469     for(i=0;i<p->nseqf;i++)
2470     {
2471
2472         inserir_seqss(p, (p->seqf+i)->seqstr);
2473         while((p->seqf+i)->seqstr[j]!='\0')
2474         {
2475
2476             (p->seqss+i)->seqstr[strlen((p->seqf+i)->seqstr)-1-j]=(p->seqf+i)->seqstr[j];
2477             j++;
2478         }
2479         (p->seqss+i)->seqstr[strlen((p->seqf+i)->seqstr)]='\0';
2480         j=0;
2481         v[i]=-1;
2482     }
2483
2484     for(i=0; i<p->nseqf; i++)
2485     {
2486
2487         for(j=0; j<p->nseqf; j++)
2488         {
2489
2490             if(strcmp((p->seqf+i)->seqstr, (p->seqf+j)->seqstr)==0 && v[j] == -1)
2491             {
2492
2493                 v[j]=i;
2494             }
2495         }
2496
2497         for(j=0; j<p->nseqf; j++)
2498         {
2499
2500             if(strcmp((p->seqss+i)->seqstr, (p->seqf+j)->seqstr)==0 && v[j] == -1)
2501             {
2502
2503                 v[j]=i;
2504             }
2505         }
2506     }
2507
2508     }
2509
2510     }
2511
2512     }
2513
2514     }
2515
2516

```



```

2517     }
2518
2519     }
2520
2521     //eliminar repetidos
2522     vseqs=eliminarrep(v,p);
2523
2524     for(i=0; i<p->nseqss; i++)
2525     {
2526
2527         inserir_seqssaux(p, (p->seqf+(* (vseqs+i)))->seqstr);
2528     }
2529     free(vseqs);
2530
2531     nseq=p->nseqf;
2532
2533     for(i=0;i<nseq;i++)
2534     {
2535         remove_seqf(p);
2536     }
2537
2538     for(i=0; i<p->nseqss; i++)
2539     {
2540         inserir_seqf(p, (p->seqssaux+i)->seqstr);
2541     }
2542
2543     return 0;
2544
2545 }
2546
2547 }

```

int seq (PECASINIT * p, int num)

Função seq.

função para criar as sequencias de peças existentes

Parâmetros:

<i>PECASINIT</i>	*p estrutura deste tipo
<i>int</i>	num numero de jogos a calcular sequencias

Retorna:

retorna o numero de sequencias que encontrou

```

3847 {
3848
3849     //função para criar as sequencias de peças existentes
3850     /*char seqss[LINSEQ][COLSEQ];
3851     char baralhoaux[LINSEQ][COLSTR];
3852     char seqssaux[LINSEQ][COLSEQ];
3853     int l=0;
3854     int i=0;
3855     int j=0;
3856     int k=0;
3857
3858     int cont=0;
3859     int invertidos=0;
3860
3861     char aux[2000];
3862     char *s=NULL;
3863     int contadorfinal=0;

```

```

3871     char auxdir='0';*/
3872     int nseq=0;
3873     int fimdastring=0;
3874     int iguais=0;
3875     char *s=NULL;
3876     char aux[2000];
3877     char auxseqf[2000];
3878     int n=0;
3879     int i=0;
3880     int j=0;
3881     int k=0;
3882     int cont=0;
3883     int invertidos=0;
3884     PECA *paux=NULL;
3885     char auxdir='0';
3886     paux=p->pfirst;
3887
3888     n=num*7;
3889     while (paux!=NULL)
3890     {
3891
3892         inserir_seqss(p,paux->str);
3893         paux=paux->pnext;
3894     }
3895
3896
3897
3898
3899     for(i=0;i<p->nseqss;i++)
3900     {
3901
3902         if ((p->seqss+i)->seqstr!=NULL)
3903         {
3904
3905             invertidos++;
3906
3907         }
3908     }
3909
3910
3911     p->nseqss=invertidos;
3912     n=invertidos;
3913
3914     while(j<n)
3915     {
3916
3917         if ((p->seqss+j)->seqstr[0]==(p->seqss+j)->seqstr[2])
3918         {
3919
3920             j++;
3921
3922         }
3923         else
3924         {
3925
3926             inserir_seqss(p, ((p->seqss+j)->seqstr));
3927             auxdir=((p->seqss+j)->seqstr)[0];
3928             ((p->seqss+invertidos)->seqstr)[0]=((p->seqss+invertidos)->seqstr)[2];
3929             ((p->seqss+invertidos)->seqstr)[2]=auxdir;
3930             invertidos++;
3931             j++;
3932
3933         }
3934     }
3935
3936
3937     for(i=0;i<invertidos;i++)
3938     {
3939
3940         inserir_baralhoaux(p, (p->seqss+i)->seqstr);
3941

```

```

3942     }
3943
3944     p->nbaralhoaux=invertidos;
3945
3946
3947     n=invertidos;
3948
3949     do{
3950         cont=0;
3951
3952         for(i=0; i<invertidos; i++)
3953         {
3954
3955             for(j=0; j<p->nseqss; j++)
3956             {
3957
3958                 if(k!=0)
3959                 {
3960
3961                     // strcpy(aux,seqss[j]);
3962                     strcpy(aux,(p->seqss+j)->seqstr);
3963
3964                     s = strtok (aux,"-");
3965
3966                     while (s!= NULL)
3967                     {
3968
3969                         if(strcmp((p->baralhoaux+i)->seqstr,s)==0 ||
3970 (s[2]==((p->baralhoaux+i)->seqstr[0]) && s[0]==((p->baralhoaux+i)->seqstr[2])))
3971                         {
3972
3973                             iguais++;
3974
3975                         }
3976
3977                         s = strtok (NULL, "-");
3978
3979                     }
3980
3981                     strcpy(aux,"");
3982                 }
3983
3984                 //se for uma peça diferente das que são usadas nas sequencias já
3985 inseridas
3986                 if(iguais==0)
3987                 {
3988
3989                     //verifico o tamanho da sequencia já existente
3990                     fimdastring=strlen((p->seqss+j)->seqstr)-1;
3991
3992                     //se houver encaixe entre a peça ou sequencias de peças e a peça
3993 for diferente da que já está guardada
3994                     if(((p->seqss+j)->seqstr)[fimdastring]==((p->baralhoaux+i)->seqstr)[0] &&
3995 strcmp(((p->seqss+j)->seqstr),((p->baralhoaux+i)->seqstr))!=0)
3996                     {
3997
3998                         //se já nao vou montar sequencias pela primeira vez e se há
3999 encaixe possivel normal ou invertidamente
4000                         if(k==0 &&
4001 ((p->seqss+j)->seqstr)[fimdastring]==((p->baralhoaux+i)->seqstr)[0] &&
4002 ((p->seqss+j)->seqstr)[0]==((p->baralhoaux+i)->seqstr)[2])
4003                         {
4004
4005                             cont++;
4006                             strcpy(auxseqf,((p->seqss+j)->seqstr));

```

```

4005             strcat(auxseqf, "-");
4006             strcat(auxseqf, ((p->baralhoaux+i)->seqstr));
4007             inserir_seqf(p, auxseqf);
4008             //guardo a sequencia que fiz num array final e incremento
a variavel que vou testar no final do ciclo todo para ver se houve posivbilidade montar
sequencias ou não
4009         }
4010     }
4011 }
4012 }
4013 }
4014 }
4015 }
4016     iguais=0;
4017 }
4018 }
4019 }
4020 }
4021 }
4022 //faço as mesmas verificações acima para aumentar o array de sequencias
auxiliar para as ultimas sequencias feitas
4023 //para tentar da proxima vez fazer sequencias novas com as sequencias já obtidas
4024 for(i=0; i<invertidos; i++)
4025 {
4026     for(j=0; j<p->nseqss; j++)
4027     {
4028         if(k!=0)
4029         {
4030             strcpy(aux, (p->seqss+j)->seqstr);
4031             s = strtok (aux, "-");
4032             while (s!= NULL)
4033             {
4034                 if(strcmp((p->baralhoaux+i)->seqstr,s)==0 ||
(s[2]==((p->baralhoaux+i)->seqstr[0]) && s[0]==((p->baralhoaux+i)->seqstr[2])))
4035                 {
4036                     iguais++;
4037                 }
4038             }
4039             s = strtok (NULL, "-");
4040             strcpy(aux, "");
4041             if(iguais==0)
4042             {
4043                 fimdastring=strlen((p->seqss+j)->seqstr)-1;
4044                 if(((p->seqss+j)->seqstr)[fimdastring]==((p->baralhoaux+i)->seqstr)[0] &&
strcmp(((p->seqss+j)->seqstr), ((p->baralhoaux+i)->seqstr))!=0)
4045                 {
4046                     strcpy(auxseqf, ((p->seqss+j)->seqstr));
4047                     strcat(auxseqf, "-");
4048                     strcat(auxseqf, ((p->baralhoaux+i)->seqstr));
4049                     inserir_seqssaux(p, auxseqf);
4050                 }
4051             }
4052         }
4053     }
4054 }

```

```

4070
4071         }
4072
4073         else
4074         {
4075
4076             fimdastring=strlen((p->seqss+j)->seqstr)-1;
4077
4078
4079 if(((p->seqss+j)->seqstr)[fimdastring])==((p->baralhoaux+i)->seqstr)[0] &&
4080 strcmp(((p->seqss+j)->seqstr),((p->baralhoaux+i)->seqstr))!=0)
4081         {
4082
4083 if(((p->seqss+j)->seqstr)[fimdastring])==((p->baralhoaux+i)->seqstr)[0] &&
4084 ((p->seqss+j)->seqstr)[0]==((p->baralhoaux+i)->seqstr)[2])
4085         {
4086
4087             }
4088         else
4089         {
4090
4091             strcpy(auxseqf,((p->seqss+j)->seqstr));
4092             strcat(auxseqf,"-");
4093             strcat(auxseqf,((p->baralhoaux+i)->seqstr));
4094             inserir_seqssaux(p,auxseqf);
4095
4096         }
4097
4098     }
4099
4100     }
4101     iguais=0;
4102 }
4103
4104
4105
4106 //por fim colocamos o array de seqss auxiliar vazio e preencho-o com as novas
4107 sequencias feitas
4108 nseq=p->nseqss;
4109
4110 for(i=0;i<nseq;i++)
4111 {
4112
4113     remove_seqss(p);
4114
4115 }
4116
4117
4118 for(i=0;i<p->nseqssaux;i++)
4119 {
4120
4121     inserir_seqss(p,(p->seqssaux+i)->seqstr);
4122
4123 }
4124
4125 nseq=p->nseqssaux;
4126
4127 for(i=0;i<nseq;i++)
4128 {
4129
4130     remove_seqssaux(p);
4131
4132 }
4133
4134 k++;
4135

```

```

4136     }
4137     //faço tudo isto até não haver possibilidades de fazer combinações de sequencias,
4138     //porque quando não houver sequencias a fazer é sinal que o ciclo tem de acabar
4139     while(cont>0);
4140
4141     //retorno o numero de sequencias feitas
4142     return 0;
4143
4144 }

```

int seqcomseqinicial (char *baralhoss*[][COLSTR], char *seqf*[][COLSEQ], int *num*, char *seqinit*[])

Função seqcomseqinicial.

esta funcao é igual à função das sequencias, só que nestas começo inicialmente com uma sequencia

Parâmetros:

<i>char</i>	baralhoss[][COLSTR] baralhos do jogador em string
<i>char</i>	seqf[][COLSEQ] array final onde vou guardar as sequencias todas possiveis
<i>int</i>	num numero de jogos a converter
<i>char</i>	seqinit[] sequencia inicial

Retorna:

retorna o numero de sequencias que encontrou

VERIFICA QUANTIDADE DE CORRESPONDENCIAS

JUNTA AS CORRESPONDENCIAS

```

5025 {
5034     //esta funcao é igual à função das sequencias, só que nestas começo inicialmente
com uma sequencia
5035     char seqss[LINSEQ][COLSEQ];
5036     char baralhoaux[LINSEQ][COLSEQ];
5037     char invertidas[LINSEQ][COLSEQ];
5038     char vinvertidas[LINSEQ][COLSEQ];
5039     char auxinv[COLSTR];
5040     char invfinal[COLSTR];
5041     int i=0;
5042     int j=0;
5043     int k=0;
5044     int tam=LINSEQ;
5045     int n=0;
5046     int cont=0;
5047     int fimdastring=0;
5048     int decont=LINSEQ-1;
5049     int iguais=0;
5050     char aux[2000];
5051     char *s;
5052     char *inv;
5053     char *invseq;
5054     int invertidos=0;
5055     int contadorfinal=0;
5056     char auxdir='0';
5057     int contapecasvinvertidas=0;
5058     int seqinitexiste=0;
5059     /*a variavel ok acrescentada para apenas passar para o array secundário o que foi
passado para o final sem concatenar
5060     peças mal*/
5061     int ok=0;
5062
5063
5064     //n=num*7;
5065     n=num*7;

```

```

5066     esvaziaseqstr(seqss,LINSEQ);
5067     esvaziaseqstr(baralhoaux,LINSEQ);
5069     for(i=0; i<n; i++)
5070     {
5071         strcpy(seqss[i],baralhoss[i]);
5072     }
5073     invertidos=n;
5074     //acrescentar pecas ao contrario
5075     while(j<n)
5076     {
5077         if(seqss[j][0]==seqss[j][2])
5078         {
5079             j++;
5080             cont++;
5081         }
5082         else
5083         {
5084             strcpy(seqss[invertidos],seqss[j]);
5085             auxdir=seqss[invertidos][0];
5086             seqss[invertidos][0]=seqss[invertidos][2];
5087             seqss[invertidos][2]=auxdir;
5088             strcpy(invertidas[k],seqss[invertidos]);
5089             invertidos++;
5090             j++;
5091         }
5092     }
5093     /*printf("\nINVERTIDAS\n");
5094     for(i=0;i<k;i++){
5095         printf("%s\n",invertidas[i]);
5096     }
5097     printf("\n");*/
5098     for(i=0; i<(invertidos); i++)
5099     {
5100         strcpy(seqf[i],seqss[i]);
5101     }
5102     for(i=0; i<(invertidos); i++)
5103     {
5104         strcpy(baralhoaux[i],seqss[i]);
5105     }
5106     strcpy(baralhoaux[invertidos],seqinit);
5107     for(i=0; i<strlen(seqinit); i++)
5108     {
5109         baralhoaux[invertidos+1][i]=seqinit[strlen(seqinit)-1-i];
5110     }

```

```

5137 //sequencia normal e invertidas acrescento ao array
5138 invertidos=invertidos+2;
5139
5140 for(i=0; i<invertidos-2; i++)
5141 {
5142     if(strcmp(seqinit,baralhoaux[i])==0)
5143     {
5144         seqinitexiste++;
5145     }
5146 }
5147
5148 //copiar sequencia inicial para dentro dos arrays de string, tanto o final como
5149 o aux
5150
5151 strcpy(seqss[invertidos-2],baralhoaux[invertidos-2]);
5152 strcpy(seqss[invertidos-1],baralhoaux[invertidos-1]);
5153
5154 strcpy(seqf[invertidos-2],baralhoaux[invertidos-2]);
5155 strcpy(seqf[invertidos-1],baralhoaux[invertidos-1]);
5156
5157 n=invertidos;
5158 k=0;
5159
5160 do
5161 {
5162     cont=0;
5163     for(i=0; i<invertidos; i++)
5164     {
5165         for(j=0; j<tam; j++)
5166         {
5167             if(k!=0)
5168             {
5169                 strcpy(aux,seqss[j]);
5170                 s = strtok (aux,"-");
5171                 while (s!= NULL)
5172                 {
5173                     if(strcmp(baralhoaux[i],s)==0 || (s[2]==baralhoaux[i][0] &&
5174 s[0]==baralhoaux[i][2]))
5175                     {
5176                         iguais++;
5177                     }
5178                     s = strtok (NULL, "-");
5179                 }
5180                 strcpy(aux,"");
5181             }
5182             if(iguais==0)
5183             {
5184                 /*Ve o tamanho da string*/
5185                 fimdastring=strlen(seqss[j])-1;
5186                 /*compara as strings iniciasi e invertidas com aquelas que vamos
5187 aumentar*/
5188                 if(seqss[j][fimdastring]==baralhoaux[i][0] && i!=j &&
5189 strcmp(seqss[i], "9|9")!=0)

```



```

5206         {
5207
5208             cont++;
5209             strcpy(seqf[contadorfinal],seqss[j]);
5210             strcat(seqf[contadorfinal],"-");
5211             strcat(seqf[contadorfinal],baralhoaux[i]);
5212
5213             contadorfinal++;
5214             ok++;
5215             //este array é o array final
5216         }
5217
5218     }
5219     iguais=0;
5220
5221 }
5222
5223 }
5224
5225
5228     for(i=0; i<invertidos; i++)
5229     {
5230
5231         for(j=0; j<ok; j++)
5232         {
5233
5234             if(k!=0)
5235             {
5236
5237                 strcpy(aux,seqss[j]);
5238
5239                 s = strtok (aux,"-");
5240                 //parte para ver se a sequencia já montada contem a peca que vamos
5241                 tentar inserir
5242                 while (s!= NULL)
5243                 {
5244                     if(strcmp(baralhoaux[i],s)==0 || (s[2]==baralhoaux[i][0] &&
5245 s[0]==baralhoaux[i][2]))
5246                     {
5247                         iguais++;
5248                     }
5249
5250                     s = strtok (NULL, "-");
5251                 }
5252
5253                 strcpy(aux,"");
5254
5255                 if(iguais==0)
5256                 {
5257                     fimdastring=strlen(seqss[j])-1;
5258
5259                     if(seqss[j][fimdastring]==baralhoaux[i][0] && i!=j &&
5260 strcmp(seqss[i],"9|9")!=0)
5261                     {
5262                         strcpy(seqss[decont],seqss[j]);
5263                         strcat(seqss[decont],"-");
5264                         strcat(seqss[decont],baralhoaux[i]);
5265                         decont--;
5266                     }
5267                 }
5268             }
5269         }
5270     }
5271 }
5272
5273 }
5274
5275     else

```

```

5276         {
5277
5278             for(fimdastring=0; seqss[j][fimdastring]!='\0'; fimdastring++);
5279             fimdastring--;
5280
5281             if(seqss[j][fimdastring]==baralhoaux[i][0] && i!=j &&
5282             strcmp(seqss[i],"9|9")!=0)
5283             {
5284                 strcpy(seqss[decont],seqss[j]);
5285                 strcat(seqss[decont],"-");
5286                 strcat(seqss[decont],baralhoaux[i]);
5287                 //printf("%s == %s | JUNTA -->
5288                 %s\n\n",seqss[j],baralhoaux[i],seqss[decont]);
5289                 decont--;
5290             }
5291
5292         }
5293
5294         iguais=0;
5295
5296     }
5297
5298 }
5299
5300 //Agora limpa o array seqss em cima e assa de baixo para cima para continuar
5301 a juntar
5302 i=0;
5303 while (strcmp(seqss[i],"9|9")!=0)
5304 {
5305     strcpy(seqss[i],"9|9");
5306     i++;
5307 }
5308
5309 decont=LINSEQ-1;
5310 i=0;
5311 //apaga array em baixo
5312 while (strcmp(seqss[decont],"9|9")!=0)
5313 {
5314     strcpy(seqss[i],seqss[decont]);
5315     strcpy(seqss[decont],"9|9");
5316     decont--;
5317     i++;
5318 }
5319 tam=(LINSEQ-decont-1);
5320 decont=LINSEQ-1;
5321 k++;
5322 }
5323 while(cont>0);
5324 //verifica se contem invertidas e normais e retira-as
5325 cont=0;
5326 i=0;
5327 while(strcmp(seqf[cont],"9|9")!=0)
5328 {
5329     strcpy(seqss[i],seqf[cont]);
5330     cont++;
5331     i++;
5332 }

```

```

5344     }
5345
5346     esvaziaseqstr(invertidas,LINSEQ);
5347
5348     for(i=0; i<cont; i++)
5349     {
5350
5351         strcpy(invertidas[i],seqss[i]);
5352
5353     }
5354
5355     j=0;
5356     for(i=0; i<cont; i++)
5357     {
5358
5359         invseq = strtok (invertidas[i],"-");
5360
5361         while (invseq != NULL)
5362         {
5363
5364             strcpy(vinvertidas[j],invseq);
5365             j++;
5366             invseq = strtok (NULL, "-");
5367
5368         }
5369         contapecasvinvertidas=j;
5370         j=0;
5371         inv = strtok (seqss[i],"-");
5372
5373         while (inv != NULL)
5374         {
5375
5376             strcpy(auxinv,inv);
5377
5378             invfinal[2]=auxinv[0];
5379             invfinal[1]=auxinv[1];
5380             invfinal[0]=auxinv[2];
5381
5382             if(invfinal[2]!=invfinal[0])
5383             {
5384
5385                 for(k=0; k<contapecasvinvertidas; k++)
5386                 {
5387
5388                     if(strcmp(invfinal,vinvertidas[k])==0)
5389                     {
5390
5391                         strcpy(seqf[i],"-");
5392
5393                     }
5394
5395                 }
5396
5397             }
5398
5399             inv = strtok (NULL, "-");
5400
5401         }
5402         esvaziaseqstr(vinvertidas,LINSEQ);
5403         j=0;
5404     }
5405
5406     contadorfinal=0;
5407     i=0;
5408
5409     while(i<LINSEQ)
5410     {
5411
5412         if(strcmp(seqf[i],"9|9")!=0)
5413         {
5414

```

```

5415         contadorfinal++;
5416     }
5417 }
5418     i++;
5419 }
5420 }
5421
5422     if(seqinitexiste>0)
5423     {
5424         contadorfinal=-1;
5425     }
5426     return contadorfinal;
5427 }
5428 }
5429 }
5430 }

```

int strtouq (char *stra*[][COLSEQ], char *str*[], char *car*)

Função strtouq.

esta função recebe um array de strings, uma string e o carater pelo qual vai partir e faz o strtok()

Parâmetros:

<i>char</i>	<i>stra</i> [][COLSEQ] array de strings na qual vamos guardar as strings partidas
<i>char</i>	<i>str</i> [] string que queremos partir
<i>char</i>	<i>car</i> carater pelo qual partimos

Retorna:

retorna o numero de pecas partidas

```

4354 {
4355     //esta função recebe um array de strings, uma string e o carater pelo qual vai partir
e faz o strtok()
4364     int k=0,i=0,j=0;
4365     for (i=0; str[i]!='\0'; i++)
4366     {
4367         if (str[i]!=car)
4368         {
4369             stra[k][j]=str[i];
4370             j++;
4371         }
4372         else
4373         {
4374             stra[k][j]='\0';
4375             k++;
4376             j=0;
4377         }
4378     }
4379     return k+1;
4380 }
4381 }

```

int tirartracosinvertidos (char *seqf*[][COLSEQ], int *numdeseq*)

Função tirartracosinvertidos.

esta função serve para eliminar as sequencias que ão repetidas mas que estão escritas da direita para a esquerda, nas quais coloquei um "-" e agora elimino-as e puxo as outras para ci

Parâmetros:

<i>char</i>	seqf[][COLSEQ] array final de strings
<i>int</i>	numdeseq numero de sequencias

Retorna:

retorna o numero de sequencias menos as que estavam escritas da esquerda para a direita

```

4988 {
4989
4996     //esta função serve para eliminar as sequencias que ão repetidas mas que estão
escritas da direita para a esquerda
4997     //mas quais coloquei um "-" e agora elimino-as e puxo as outras para cima
4998
4999     int i=0;
5000     int cont=0;
5001
5002     while(strcmp(seqf[i], "9|9")!=0)
5003     {
5004
5005         if(strcmp(seqf[i], "-")==0)
5006         {
5007
5008             cont++;
5009
5010         }
5011
5012         i++;
5013     }
5014
5015     }
5016
5017     return numdeseq-cont;
5018
5019
5020 }
```

void trocapadiao (char seqf[][COLSEQ], int size, char padiao[], char padraon[], char seqfpadiao[][COLSEQ], int * sizeseqfpadiao)

Função trocapadiao.

função para procurar uma substring numa string (utilizei esta variante na torca de padiao)

Parâmetros:

<i>char</i>	seqf[][COLSEQ] array final de strings
<i>int</i>	size numero da linha do array de strings final que vamos querer procurar uma substring
<i>char</i>	padiao[] string a substituir
<i>char</i>	padraon[] string substituta
<i>char</i>	seqfpadiao[][COLSEQ]
<i>int</i>	*sizeseqfpadiao indice da sequencia padiao na qual vou trocar o padiao

CICLO DAS OCURENCIAS DO PADIAO por indices da string

```

4743 {
4744
4755     int tampadiao=0;
4756     int tampadraon=0;
4757     char *s;
4758     char *p;
4759     char auxseq[LINSEQ][COLSEQ];
4760     char auxpad[COLSEQ];
4761     int contseq=0;
4762     int encontrasub=0;
4763     int i=0;
```

```

4764     int k=0;
4765     int sequencia=0;
4766     int posini=0;
4767     int contpad=0;
4768     int vetorpadrao[COLSEQ];
4769     int vposition[COLSEQ];
4770     char padraoasubstaux[LINSEQ];
4771     char auxpadraon[LINSEQ];
4772     int confim=0;
4773     int troca=0;
4774     int r=0;
4775     /*printf("%s\n\n",seqf[0]);
4776     printf("%s\n\n",padrao);
4777     printf("%s\n\n",padraon);*/
4778     strcpy(auxpadraon,padraon);
4779     for(sequencia=0; sequencia<size; sequencia++)
4780     {
4781
4782         esvaziaseqstr(auxseq,size);
4783
4784
4785         //fazemos o strtok() da sequencia, do padrão a trocar e do padrão novo
4786         for(i=0; i<size; i++)
4787         {
4788
4789             strcpy(auxseq[i],seqf[i]);
4790
4791         }
4792
4793         s = strtok (auxseq[sequencia],"-");
4794
4795         while (s != NULL)
4796         {
4797
4798             contseq++;
4799
4800             s = strtok (NULL, "-");
4801
4802         }
4803
4804         //partir o padrao
4805
4806         strcpy(auxpad,padrao);
4807
4808         p = strtok (auxpad,"-");
4809
4810         while (p != NULL)
4811         {
4812
4813             contpad++;
4814
4815             p = strtok (NULL, "-");
4816
4817         }
4818
4819         for(i=0; i<contseq; i++)
4820         {
4821             //chamo a função procurar sub string e mando o endereço porque eu envio
4822             // o size é o numero de sequencias a fazer trocas
4823             encontraSub=proSubseq_trocPadrao(&seqf[sequencia],size,padrao,i);
4824
4825             if(encontraSub!=-1)
4826             {
4827
4828                 if(i==0)
4829                 {
4830
4831                     vetorpadrao[k]=encontraSub;
4832                     k++;
4833

```

```

4834
4835     }
4836     else
4837     {
4838         if(vetorpadrao[k-1]!=encontrasub)
4839         {
4840
4841             vetorpadrao[k]=encontrasub;
4842             k++;
4843
4844         }
4845     }
4846 }
4847
4848 }
4849
4850 }
4851
4852
4853 //k sao os padroes encontrados que guardo num array
4854 if(k==0)
4855 {
4856
4857     //printf("O padrao nao foi encontrado, logo nao pode substituir! ");
4858
4859 }
4860 else
4861 {
4862
4863     for(i=0; i<k; i++)
4864     {
4865
4866         tampadrao = strlen(padrao);
4867         tampadrao--;
4868         tampadraon = strlen(padraon);
4869         tampadraon--;
4870
4871         if(vetorpadrao[i]==0)
4872         {
4873             posini=0;
4874
4875         }
4876         else
4877         {
4878             if(i==0)
4879             {
4880
4881                 posini=vetorpadrao[i]*4;
4882
4883             }
4884             else
4885             {
4886                 //depois de tirar pecas as posicoes alteram e isto prevê essas
4887                 trocas
4888                 //multiplicar por quatro dame o carater tipo 16 carater 4*4
4889                 contpad=contpad*4;
4890                 posini=vetorpadrao[i]*4;
4891
4892             }
4893         }
4894     }
4895
4896     vposition[i]=posini;
4897
4898 }
4899
4900 }
4901
4902 for(i=(k-1); i>=0; i--)
4903 {
4904
4905

```

```

4906         //se for uma substituição do padrao no fim
4907         if (seqf[sequencia][vposition[i]+tampadrao+1]=='\0')
4908         {
4909
4910
4911         if (padraon[0]==seqf[sequencia][strlen(seqf[sequencia])-1-tampadrao-2])
4912         {
4913
4914             //printf("i-> %d   pos-> %d\n",i,vposition[i]);
4915             seqf[sequencia][vposition[i]-1]='\0';
4916             strcat(seqf[sequencia],"-");
4917             strcat(seqf[sequencia],padraon);
4918             contfim++;
4919             troca++;
4920         }
4921     }
4922 }
4923
4924
4925 //se for uma substituição do padrao no inicio
4926 if (vposition[i]==0)
4927 {
4928
4929     if (padraon[strlen(padraon)-1]==seqf[sequencia][tampadrao+2])
4930     {
4931
4932         //printf("i-> %d   pos-> %d\n",i,vposition[i]);
4933
4934         strcpy(seqf[sequencia],&seqf[sequencia][vposition[i]+tampadrao+2]);
4935         strcat(padraon,"-");
4936         strcat(padraon,seqf[sequencia]);
4937         strcpy(seqf[sequencia],padraon);
4938         troca++;
4939     }
4940 }
4941
4942 }
4943
4944
4945 //se for uma substituição do padrao no meio
4946 if (seqf[sequencia][vposition[i]+tampadrao+1]!='\0' && vposition[i]!=0
&& contfim==0)
4947 {
4948
4949
4950     if (seqf[sequencia][vposition[i]+tampadrao+2]==padraon[strlen(padraon)-1] &&
seqf[sequencia][vposition[i]-2]==padraon[0])
4951     {
4952
4953         //printf("i-> %d   pos-> %d\n",i,vposition[i]);
4954
4955         strcpy(padraoasubstaux,&seqf[sequencia][vposition[i]+tampadrao+1]);
4956         seqf[sequencia][vposition[i]]='\0';
4957         strcat(seqf[sequencia],padraon);
4958         strcat(seqf[sequencia],padraoasubstaux);
4959         troca++;
4960     }
4961 }
4962 contfim=0;
4963
4964 }
4965
4966 r=*sizeseqfpadrao;
4967 if (troca==1)
4968 {
4969
4970     strcpy(seqfpadrao[r],seqf[sequencia]);

```



```

4971         //printf("[%d] ---> %s\n",sequencia,seqfpadrao[*sizeeqfpadrao]);
4972         r++;
4973     }
4974     *sizeeqfpadrao=r;
4975     troca=0;
4976     k=0;
4977     contseq=0;
4978     contpad=0;
4979     strcpy(padraon,auxpadraon);
4980
4981 }
4982
4983 }

```

int verificasequencia (char seq[])

Função verificasequencia.

verifica se a sequência é possível juntar ex:2|3-3|4 estas pecas encaixam e esta funcao faz esta verificação

Parâmetros:

<i>char</i>	seq[] recebe a sequência de peças
-------------	-----------------------------------

Retorna:

Retorna 0 se for possível e 1 se não for

```

2303 {
2304
2305
2312     int cont=0;
2313     int i=0;
2314
2315     for(i=0; i<strlen(seq); i++)
2316     {
2317
2318         if(seq[i]=='-')
2319         {
2320
2321             if(seq[i+1]!=seq[i-1])
2322             {
2323
2324                 cont++;
2325
2326             }
2327
2328         }
2329
2330     }
2331
2332     if(cont==0)
2333     {
2334
2335         return 0;
2336
2337     }
2338     else
2339     {
2340
2341         return 1;
2342
2343     }
2344
2345 }

```

Referência ao ficheiro projeto.h

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

Estruturas de Dados

- struct **pecaint**
- struct **peca**
- struct **seq**
- struct **ordena**
- struct **pecasinit**

Macros

- #define **LIN** 28
- #define **COL** 2
- #define **COLSTR** 4
- #define **LINSEQ** 800
- #define **COLSEQ** 200

Definições de tipos

- typedef struct **pecaint** **PECAINT**
- typedef struct **peca** **PECA**
- typedef struct **seq** **SEQ**
- typedef struct **ordena** **ORDENA**
- typedef struct **pecasinit** **PECASINIT**

Funções

- void **load_jogo_bin** (**PECASINIT** *, char [])
Load do ficheiro binario.
- void **save_jogo_bin** (**PECASINIT**, char [])
Função save_jogo_bin.
- int **separarseqinvertidasstruct** (**PECASINIT** *)
Função separarseqinvertidas struct.
- void **load_txt_jogo** (**PECASINIT** *p, char [])
Load do ficheiro txt.
- void **save_txt_jogo** (**PECASINIT**, char [])
Função save_jogo_txt.
- void **remove_seqf** (**PECASINIT** *)
- void **ordernarsequenciasstruct** (**PECASINIT** *)
Função ordernarsequencias struct.
- void **remove_seqss** (**PECASINIT** *)
- void **remove_seqssaux** (**PECASINIT** *)
- void **criapecasint** (int pecasi[][**COL**], int)
Função criapecasint.
- void **inserir_seqf** (**PECASINIT** *, char [])
Inserir sequencias no array dinamico das seqs.

- void **inserir_baralhoaux** (PECASINIT *, char [])
Inserir sequencias de pecas no array dinamico.
- void **inserir_seqssaux** (PECASINIT *, char [])
Inserir sequencias no array dinamico das seqs auxiliares 2.
- void **create_array_baralhoaux** (PECASINIT *, int)
Criar array dinamico de sequencias.
- void **inserir_seqss** (PECASINIT *, char [])
Inserir sequencias no array dinamico das seqs auxiliares.
- void **create_array_seqf** (PECASINIT *, int)
Criar array dinamico de sequencias.
- void **create_array_seqss** (PECASINIT *, int)
Criar array dinamico de sequencias.
- void **create_array_seqssaux** (PECASINIT *, int)
Criar array dinamico de sequencias.
- void **remove_pecas** (PECASINIT *p, char remove[])
Remover pecas dos jogos.
- void **criapecasstr** (char pecass[][COLSTR], int)
Função criapecasstr.
- void **printpecasint** (int pecasi[][COL], int, int, int)
Função printpecasint.
- void **printpecasstr** (char pecass[][COLSTR], int, int)
Função printpecasstr.
- void **printseqstr** (char seqss[][COLSEQ], int, int)
Função printseqstr.
- int **entregarbaralhos** (char pecass[][COLSTR], PECASINIT *p, int n)
Função entregarbaralhos.
- int **preenchebaralhos** (char pecass[][COLSTR], char baralhoss[][COLSTR], int)
- int **preenchebaralhosstruct** (char pecass[][COLSTR], PECASINIT *, int)
- void **inserir_pecas** (PECASINIT *, char pecanova[COLSTR])
Inserir pecas na lista ligada do baralho.
- void **inserir_pecaint** (PECASINIT *, int, int)
Inserir pecas inteiras na lista ligada do baralho.
- void **mostrarjogosstr** (char baralhoss[][COLSTR], int)
- void **mostrarjogosstrstruct** (PECASINIT, int)
Função mostrarjogosstrstruct.
- void **mostrarjogosintstruct** (PECASINIT, int)
Função mostrarjogosstr.
- void **mostrarjogosint** (int baralhosint[][COL], int)
Função mostrarjogosint.
- void **esvaziabaralhoint** (int baralhosi[][COL], int, int)
Função esvaziabaralhoint.
- void **esvaziabaralhostr** (char baralhoss[][COLSTR], int)
Função esvaziabaralhostr.
- void **esvaziabaralhostrstruct** (PECASINIT *)
Função esvaziabaralhostr.
- void **esvaziaseqstr** (char seqss[][COLSEQ], int)

- *Função esvaziaseqstr.*
- int **convertestrtoint** (PECASINIT *, int)
Função convertestrtoint.
- int **converteinttostr** (PECASINIT *, int)
Função converteinttostr.
- int **remover** (PECASINIT *, char pecass[][COLSTR], int)
Função remover.
- PECA * **find_peca_baralho** (PECASINIT *, char[])
Procurar peca no baralho.
- int **modificar** (PECASINIT *, char pecass[][COLSTR], int)
- int **seq** (PECASINIT *, int)
Função seq.
- int **prosubseq** (char seqf[][COLSEQ], int size, char subs[])
Função prosubseq.
- int **prosubseq_trocapadiao** (char seqf[][COLSEQ], int size, char subs[], int)
- void **ordenarsequencias** (char seqf[][COLSEQ], int)
Função ordenarsequencias.
- void **ordenarmatrizinteiros** (int m[][2], int)
Função ordenarmatrizinteiros.
- void **trocapadiao** (char seqf[][COLSEQ], int size, char padiao[], char padraon[], char seqfpadiao[][COLSEQ], int *sizeseqfpadiao)
Função trocapadiao.
- int **tirartracosinvertidos** (char seqf[][COLSEQ], int numdeseq)
Função tirartracosinvertidos.
- int **seqcomseqinicial** (char baralhoss[][COLSTR], char seqf[][COLSEQ], int, char seqinit[])
Função seqcomseqinicial.
- int **separarseqinvertidas** (char seqf[][COLSEQ], int)
Função separarseqinvertidas.
- int **strtoque** (char stra[][COLSEQ], char str[], char)
Função strtoque.
- int **retiraseqinitrepetida** (char seqf[][COLSEQ], int size, char seqinit[])
Função retiraseqinitrepetida.
- int **jogoadois** (char baralhoss[][COLSTR], char seqf[][COLSEQ], int, char seqinit[])
Função jogoadois.
- int * **eliminarrep** (int *, PECASINIT *p)
Função eliminarrep.
- char * **inverterstr** (char str1[], char str2[])
Função inverterstr.
- int **baralhoausar** (char baralhoss[][COLSTR], char baralhoaux[][COLSEQ], int)
Função baralhoausar.
- int **verificasequencia** (char seq[])
Função verificasequencia.
- int **prosubseq_ausar** (char seqf[][COLSEQ], int, char subs[])
Função prosubseq_ausar.
- int **main_domino** (int argc, char *argv[])
Função main.

- void **printseqstrstruct** (**PECASINIT**)
Função printpecasstruct.

Documentação das macros

#define COL 2

#define COLSEQ 200

#define COLSTR 4

#define LIN 28

#define LINSEQ 800

Documentação dos tipos

typedef struct ordena ORDENA

typedef struct peca PECA

typedef struct pecaint PECAINT

typedef struct pecasinit PECASINIT

typedef struct seq SEQ

Documentação das funções

int baralhoausar (char *baralhoss*[][COLSTR], char *baralhoaux*[][COLSEQ], int)

Função baralhoausar.

esta funcao recebe as pecas de um baralho e acrescenta as invertidas

Parâmetros:

<i>char</i>	baralhoss[][COLSTR] baralhos do jogador em string
<i>char</i>	baralhoaux[][COLSEQ] baralhos do jogador em string auxiliar na qual guardo também as peças invertidas
<i>int</i>	numero contém o numero do baralho ao qual vamos acrescentar as pecas invertidas também

Retorna:

retorna o numero de pecas do baralho do jogador com as pecas invertidas incluídas e retirando as repetidas

```
5858 {
5859
```

```

5868     int i=0;
5869     char auxdir=' ';
5870     int invertidos=0;
5871     int k=0;
5872     int init=0;
5873
5874     esvaziaseqstr(baralhoaux,LINSEQ);
5875
5876     for(i=1; i<numero; i++)
5877     {
5878
5879         init=init+7;
5880
5881     }
5882
5883     for(i=init; i<(numero*7); i++)
5884     {
5885
5886         strcpy(baralhoaux[k],baralhoss[i]);
5887         k++;
5888
5889     }
5890
5891     invertidos=7;
5892
5893     //acrescentar pecas ao contrario
5894     i=0;
5895     while(i<7)
5896     {
5897
5898         if(baralhoaux[i][0]==baralhoaux[i][2])
5899         {
5900
5901             i++;
5902
5903         }
5904         else
5905         {
5906
5907             strcpy(baralhoaux[invertidos],baralhoaux[i]);
5908             auxdir=baralhoaux[invertidos][0];
5909             baralhoaux[invertidos][0]=baralhoaux[invertidos][2];
5910             baralhoaux[invertidos][2]=auxdir;
5911             invertidos++;
5912             i++;
5913
5914         }
5915     }
5916
5917     return invertidos;
5918
5919 }

```

int converteinttostr (PECASINIT * , int)

Função converteinttostr.

esta funcao converte as peças inteiras em string

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>int</i>	num numero de jogos a converter

Retorna:

retorna o numero de jogos a imprimir

```
3399 {
```

```

3406 //converte as peças inteiras em string
3407
3408 char aux[4];
3409 int i=0;
3410 num=num*7;
3411
3412 PECAINT *paux=NULL;
3413
3414 paux=p->pfirstint;
3415
3416 while(i<(p->npecasint))
3417 {
3418     aux[0]='0'+paux->direito;
3419     aux[1]='|';
3420     aux[2]='0'+paux->esquerdo;
3421
3422     inserir_pecas(p,aux);
3423
3424     paux=paux->pnext;
3425
3426     i++;
3427 }
3428
3429 return num;
3430
3431 }

```

int convertestrtoint (PECASINIT *, int)

Função convertestrtoint.

esta funcao converte as pecas do jogador de string para inteiros

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>int</i>	num numero de jogos a converter

Retorna:

retorna o numero de jogos a imprimir

```

3355 {
3356
3364 //converte as peças dos jogadores de strings para inteiros
3365
3366 char aux1[4];
3367 char aux2[4];
3368
3369 num=num*7;
3370
3371 PECA *paux=NULL;
3372
3373 paux=p->pfirst;
3374 while(paux!=NULL)
3375 {
3376
3377     strcpy(aux1,paux->str);
3378     strcpy(aux2,paux->str);
3379     aux1[1]='\0';
3380     aux2[0]=aux2[2];
3381     aux2[1]='\0';
3382     inserir_pecaint(p,atoi(aux1),atoi(aux2));
3383     paux=paux->pnext;
3384
3385 }
3386
3387
3388

```

```

3389     return num;
3390
3391
3392 }

```

void create_array_baralhoaux (PECASINIT *, int)

Criar array dinamico de sequencias.

função para colocar sequencias no array dinâmico

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>int</i>	n contem a qtde a alocar

```

1733 {
1734
1741     SEQ *pnew=NULL;
1742     SEQ *paux=NULL;
1743
1744     int i=0;
1745     int j=0;
1746
1747     if (p->baralhoaux==NULL || p->nbaralhoaux==0)
1748     {
1749
1750         pnew=(SEQ*) malloc (sizeof (SEQ) *n) ;
1751         p->nbaralhoaux=n;
1752
1753         for (i=0; i<n; i++)
1754         {
1755             (pnew+i)->seqstr=NULL;
1756
1757         }
1758
1759         p->baralhoaux=pnew;
1760
1761     }
1762     else
1763     {
1764
1765         p->nbaralhoaux=n;
1766         pnew=(SEQ*) malloc (sizeof (SEQ) *n) ;
1767         paux=p->baralhoaux;
1768
1769         for (i=0; i<(p->nbaralhoaux-2); i++)
1770         {
1771
1772             (pnew+i)->seqstr=(paux+i)->seqstr;
1773
1774
1775         }
1776
1777         for (j=i; j<(p->nbaralhoaux); j++)
1778         {
1779
1780             (pnew+j)->seqstr=NULL;
1781
1782         }
1783
1784         p->baralhoaux=pnew;
1785
1786     }
1787
1788 }
1789 }

```


void create_array_seqf (PECASINIT * , int)

Criar array dinamico de sequencias.

função para colocar sequencias no array dinâmico

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>int</i>	n contem a qtde a alocar

```

1983 {
1984
1991     SEQ *pnew=NULL;
1992     SEQ *paux=NULL;
1993
1994     int i=0;
1995     int j=0;
1996
1997     if (p->seqf==NULL || p->nseqf==0)
1998     {
1999
2000         pnew=(SEQ*) malloc (sizeof (SEQ) *n) ;
2001         p->nseqf=n;
2002
2003         for (i=0; i<n; i++)
2004         {
2005
2006             (pnew+i)->seqstr=NULL;
2007
2008         }
2009
2010         p->seqf=pnew;
2011
2012     }
2013     else
2014     {
2015
2016         p->nseqf=n;
2017         pnew=(SEQ*) malloc (sizeof (SEQ) *n) ;
2018         paux=p->seqf;
2019
2020         for (i=0; i<(p->nseqf-2); i++)
2021         {
2022
2023             (pnew+i)->seqstr=(paux+i)->seqstr;
2024
2025
2026         }
2027
2028         for (j=i; j<(p->nseqf); j++)
2029         {
2030
2031             (pnew+j)->seqstr=NULL;
2032
2033         }
2034
2035         p->seqf=pnew;
2036
2037     }
2038
2039 }
```

void create_array_seqss (PECASINIT * , int)

Criar array dinamico de sequencias.

função para colocar sequencias no array dinâmico

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>int</i>	n contem a qtde a alocar

```

1795 {
1796
1803     SEQ *pnew=NULL;
1804     SEQ *paux=NULL;
1805
1806     int i=0;
1807     int j=0;
1808
1809     if (p->seqss==NULL || p->nseqss==0)
1810     {
1811
1812         pnew=(SEQ*)malloc(sizeof(SEQ)*n);
1813         p->nseqss=n;
1814
1815         for(i=0; i<n; i++)
1816         {
1817
1818             (pnew+i)->seqstr=NULL;
1819
1820         }
1821         p->seqss=pnew;
1822
1823     }
1824     else
1825     {
1826
1827         p->nseqss=n;
1828         pnew=(SEQ*)malloc(sizeof(SEQ)*n);
1829         paux=p->seqss;
1830
1831         for(i=0; i<(p->nseqss-2); i++)
1832         {
1833
1834             (pnew+i)->seqstr=(paux+i)->seqstr;
1835
1836
1837
1838         }
1839
1840         for(j=i; j<(p->nseqss); j++)
1841         {
1842
1843             (pnew+j)->seqstr=NULL;
1844
1845         }
1846         p->seqss=pnew;
1847
1848     }
1849
1850
1851 }
```

void create_array_seqssaux (PECASINIT * , int)

Criar array dinamico de sequencias.

função para colocar sequencias no array dinâmico

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>int</i>	n contam a qtde a alocar

```

1922 {
1923
1930     SEQ *pnew=NULL;
1931     SEQ *paux=NULL;
1932
1933     int i=0;
1934     int j=0;
1935
1936     if (p->seqssaux==NULL || p->nseqssaux==0)
1937     {
1938
1939         pnew=(SEQ*) malloc (sizeof (SEQ) *n) ;
1940         p->nseqssaux=n;
1941
1942         for (i=0; i<n; i++)
1943         {
1944             (pnew+i)->seqstr=NULL;
1945
1946         }
1947
1948         p->seqssaux=pnew;
1949
1950     }
1951     else
1952     {
1953
1954         p->nseqssaux=n;
1955         pnew=(SEQ*) malloc (sizeof (SEQ) *n) ;
1956         paux=p->seqssaux;
1957
1958         for (i=0; i<(p->nseqssaux-2); i++)
1959         {
1960
1961             (pnew+i)->seqstr=(paux+i)->seqstr;
1962
1963
1964         }
1965
1966         for (j=i; j<(p->nseqssaux); j++)
1967         {
1968
1969             (pnew+j)->seqstr=NULL;
1970
1971         }
1972
1973         p->seqssaux=pnew;
1974
1975     }
1976 }
1977
1978 }

```

void criapecasint (int *pecas*[][COL], int)

Função criapecasint.

função que cria as pecas em inteiros

Parâmetros:

<i>char</i>	pecass[][COLSTR] array que guarda as pecas em int
<i>int</i>	size contém o numero de pecas a ser guardadas

```

2662 {

```

```

2663
2671     int dominomax=6;
2672     int cont=0;
2673     int linha=0;
2674     int coluna=0;
2675
2676
2677     //percorre 28 vezes para fornecer as 28 peças
2678     for(linha=0; linha<size; linha++)
2679     {
2680
2681         for(coluna=0; coluna<COL; coluna++)
2682         {
2683
2684             if (coluna==0)
2685             {
2686                 //Comeca com o maior lado esquerdo na peça e vai decrementando
2687                 pecasi[linha][coluna]=dominomax;
2688                 //printf("[%d][%d] = %d | ",linha,coluna,pecasi[linha][coluna]);
2689
2690             }
2691             if (coluna==1)
2692             {
2693
2694                 //comeca com o menor lado direito e vai incrementando
2695                 pecasi[linha][coluna]=cont;
2696                 //printf("%d = [%d][%d]\n",pecasi[linha][coluna],linha,coluna);
2697                 cont++;
2698
2699             }
2700             if (cont>dominomax)
2701             {
2702
2703                 dominomax--;
2704                 cont=0;
2705
2706             }
2707
2708         }
2709
2710     }
2711
2712 }

```

void criapecasstr (char *pecass*[][COLSTR], int)

Função criapecasstr.

função que cria as pecas em strings

Parâmetros:

<i>char</i>	<i>pecass</i> [][COLSTR] array que guarda as pecas em string
<i>int</i>	s contém o numero de pecas a ser guardadas

```

2717 {
2718
2725     //defino as pecas 28 em string estáticamente
2726     strcpy(pecass[0],"6|0");
2727     strcpy(pecass[1],"6|1");
2728     strcpy(pecass[2],"6|2");
2729     strcpy(pecass[3],"6|3");
2730     strcpy(pecass[4],"6|4");
2731     strcpy(pecass[5],"6|5");
2732     strcpy(pecass[6],"6|6");
2733     strcpy(pecass[7],"5|0");
2734     strcpy(pecass[8],"5|1");
2735     strcpy(pecass[9],"5|2");

```

```

2736     strcpy(pecass[10],"5|3");
2737     strcpy(pecass[11],"5|4");
2738     strcpy(pecass[12],"5|5");
2739     strcpy(pecass[13],"4|0");
2740     strcpy(pecass[14],"4|1");
2741     strcpy(pecass[15],"4|2");
2742     strcpy(pecass[16],"4|3");
2743     strcpy(pecass[17],"4|4");
2744     strcpy(pecass[18],"3|0");
2745     strcpy(pecass[19],"3|1");
2746     strcpy(pecass[20],"3|2");
2747     strcpy(pecass[21],"3|3");
2748     strcpy(pecass[22],"2|0");
2749     strcpy(pecass[23],"2|1");
2750     strcpy(pecass[24],"2|2");
2751     strcpy(pecass[25],"1|0");
2752     strcpy(pecass[26],"1|1");
2753     strcpy(pecass[27],"0|0");
2754
2755 }

```

int* eliminarep (int * , PECASINIT * p)

Função eliminarep.

função eliminarep, serve para eliminar as sequencias repetidas, recebe os indices das sequencias e a quantidade de sequencias e remove as repetidas que são as que contém -1 e aloca espaço para as que não são repetidas e retorna essa quantidade para imprimir só as que não são repetidas

Parâmetros:

<i>int</i>	*pv apontador para um array de inteiros que contém os indices das sequencias
<i>int</i>	PECASINIT *p estrutura do tipo PECASINIT *p

Retorna:

retorna o novo tamanho do array de inteiros para depois apenas imprimir as sequências que possuem os indices com sequencias que não são repetidas

```

2355 {
2356
2363     int i=0;
2364     int j=0;
2365     int cont=0;
2366     int k=0;
2367     int *vaux=NULL;
2368
2369     for(i=0; i<(p->nseqss); i++)
2370     {
2371         if (*(pv+i) != -1)
2372         {
2373
2374             for(j=0; j<(p->nseqss); j++)
2375             {
2376
2377                 if (*(pv+i) == *(pv+j))
2378                 {
2379
2380                     cont++;
2381
2382                 }
2383
2384                 if (*(pv+i) == *(pv+j) && cont>1)
2385                 {
2386
2387                     *(pv+j) = -1;
2388

```

```

2389         }
2390
2391     }
2392
2393     cont=0;
2394
2395 }
2396
2397 }
2398
2399 for(i=0; i<(p->nseqss); i++)
2400 {
2401
2402     if (*(pv+i)==-1)
2403     {
2404
2405         cont++;
2406
2407     }
2408
2409 }
2410
2411 vaux = (int*)malloc(sizeof(int)*(p->nseqss)-cont);
2412
2413 for(i=0; i<(p->nseqss); i++)
2414 {
2415
2416     //printf("%d ", *(pv+i));
2417     if (*(pv+i) != -1)
2418     {
2419
2420         *(vaux+k)=*(pv+i);
2421         k++;
2422     }
2423
2424 }
2425
2426 p->nseqss=(p->nseqss-cont);
2427
2428 return vaux;
2429
2430 }

```

int entregarbaralhos (char *pecass*[][COLSTR], PECASINIT * *p*, int *n*)

Função entregarbaralhos.

função que entrega os baralhos aos jogadores aleatoriamente sem pecas repetidas para cada jogador (baralhos de 7 pecas e máximo de 4 jogadores)

Parâmetros:

<i>pecass</i> [][COLSTR] <i>]</i>	array de string que contém as pecas todas
<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>int</i>	n numero de jogos entregues

```

2763 {
2764
2772     int pecas=0;
2773     //int verifica=0;
2774     int aux=0;
2775     int verify=0;
2776     PECA *paux=NULL;
2777     paux=p->pfirst;
2778
2779     srand( (unsigned)time(NULL) );

```

```

2780
2781 //Entrega 4 baralhos
2782 if (n>4)
2783 {
2784
2785     printf("Nao e possivel entregar mais de 4 baralhos\n");
2786
2787 }
2788 else
2789 {
2790     n=n*7;
2791
2792     for(pecas=0; pecas<n; pecas++)
2793     {
2794         //guardo um valor aleatório entre 0 e 28
2795         aux=rand() % 28 + 0;
2796
2797         if(pecas!=0)
2798         {
2799             //ciclo para verificar se já existe, senao volta a ter outro numero
aleatorio para verificar
2800             paux=p->pfirst;
2801
2802             while(paux!=NULL)
2803             {
2804
2805                 if(strcmp(paux->str,pecass[aux])==0)
2806                 {
2807
2808                     pecas--;
2809                     verify--;
2810
2811                 }
2812
2813                 paux=paux->pnext;
2814
2815             }
2816
2817         }
2818         else
2819         {
2820             //na primeira vez insere a peca sem problema
2821             inserir_pecas(p,pecass[aux]);
2822
2823         }
2824
2825         //verifica se não é a primeira peca, se não está repetida.
2826         if(verify==0 && pecas!=0)
2827         {
2828             //se a peca a inserir é diferente de todas das outras já inseridas,
insiro
2829             inserir_pecas(p,pecass[aux]);
2830
2831         }
2832         verify=0;
2833
2834     }
2835
2836 }
2837
2838 return n;
2839
2840
2841 }

```

void esvaziabaralhoint (int *baralhos*[][COL], int , int)

Função esvaziabaralhoint.

esta funcao coloca o baralho de inteiros vazio

Parâmetros:

<i>int</i>	baralhosi[][COL] baralhos de pecas dos jogadores em inteiros
<i>int</i>	lin linhas do baralho a esvaziar
<i>int</i>	col colunas do baralho a esvaziar

```
3250 {
3251
3259     //esta funcao coloca o baralho de inteiros vazio
3260     int i=0;
3261     int j=0;
3262
3263     for(i=0; i<lin; i++)
3264     {
3265         for(j=0; j<col; j++)
3266         {
3267             baralhosi[i][j]=9;
3268         }
3269     }
3270 }
3271
3272
3273
3274
3275
3276 }
```

void esvaziabaralhostr (char *baralhoss*[][COLSTR], int)

Função esvaziabaralhostr.

esta funcao coloca o baralho de strings vazio

Parâmetros:

<i>char</i>	baralhoss[][COLSTR] baralhos de pecas dos jogadores em strings
<i>int</i>	size tamanho do baralho

```
3308 {
3309
3316     //esta funcao coloca o baralho de strings vazio
3317     int i=0;
3318
3319     for(i=0; i<size; i++)
3320     {
3321         strcpy(baralhoss[i], "9|9");
3322     }
3323 }
3324
3325
3326 }
```

void esvaziabaralhostrstruct (PECASINIT *)

Função esvaziabaralhostr.

esta funcao coloca o baralho de strings vazio

Parâmetros:

<i>char</i>	baralhoss[][COLSTR] baralhos de pecas dos jogadores em strings
-------------	--

<i>int</i>	size tamanho do baralho
------------	-------------------------

```

3280 {
3281
3288     //esta funcao coloca o baralho de strings vazio
3289     PECA *paux=NULL;
3290
3291     paux=p->pfirst;
3292
3293     while (paux!=NULL)
3294     {
3295
3296         paux->str=NULL;
3297         p->npecas--;
3298         paux=paux->pnext;
3299     }
3300 }
3301
3302 }
```

void esvaziaseqstr (char seqss[][COLSEQ], int)

Função esvaziaseqstr.

esta funcao coloca o array das sequencias vazias

Parâmetros:

<i>char</i>	seqss[][COLSEQ] array de strings das sequencias
<i>int</i>	size numero de sequencias inseridas

```

3332 {
3333
3340     //esvazia as sequencias de peças
3341     int i=0;
3342
3343     for(i=0; i<size; i++)
3344     {
3345
3346         strcpy(seqss[i], "9|9");
3347     }
3348 }
3349
3350 }
```

PECA* find_peca_baralho (PECASINIT * , char [])

Procurar peca no baralho.

função para procurar e retornar a peca se encontrar e null se nao encontrar

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>char</i>	aux[] peca a procurar

```

3662 {
3663
3670     PECA *paux=NULL;
3671     paux=p->pfirst;
3672
3673
3674     while (paux!=NULL)
3675     {
3676
3677         if (strcmp(paux->str, aux) ==0)
```

```

3678     {
3679
3680         return paux;
3681     }
3682
3683     paux=paux->pnext;
3684
3685 }
3686
3687 return NULL;
3688
3689 }
3690

```

void inserir_baralhoaux (PECASINIT * , char [])

Inserir sequencias de pecas no array dinamico.

função para inserir pecas no array dinâmico

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>char</i>	sequencia[] contam a sequencia a inserir

```

3809 {
3810
3817     SEQ *paux=NULL;
3818     paux=p->baralhoaux;
3819
3820     while(paux!=NULL && paux->seqstr!=NULL && (paux - (p->baralhoaux)) <
p->nbaralhoaux)
3821     {
3822
3823         paux++;
3824
3825     }
3826
3827     if ((paux - (p->baralhoaux)) == p->nbaralhoaux)
3828     {
3829
3830         create array baralhoaux(p,p->nbaralhoaux+2);
3831         paux = p->baralhoaux+p->nbaralhoaux-2;
3832         paux->seqstr=create_dyn_string(sequencia);
3833
3834     }else
3835     {
3836
3837         paux->seqstr=create_dyn_string(sequencia);
3838
3839     }
3840
3841 }

```

void inserir_pecas (PECASINIT * , char pecanova[COLSTR])

Inserir pecas na lista ligada do baralho.

função para inserir pecas nas listas ligadas

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>char</i>	pecanova[COLSTR] que contam a peca nova a inserir

```

2157 {

```

```

2158
2165     PECA *pnew = (PECA*)malloc(sizeof(PECA));
2166     PECA *paux = NULL;
2167
2168     pnew->str = create_dyn_string(pecanova);
2169     pnew->pnext=NULL;
2170
2171     paux=p->pfirst;
2172
2173     if(p->pfirst==NULL)
2174     {
2175
2176         p->pfirst=pnew;
2177         p->npecas++;
2178         //printf("PRIMEIRA %s ----   INSERIDAS: %d \n",pnew->str,p->npecas);
2179
2180     }
2181     else
2182     {
2183
2184         //cauda
2185         while(paux->pnext!=NULL)
2186         {
2187
2188             paux=paux->pnext;
2189
2190         }
2191
2192         paux->pnext=pnew;
2193         p->npecas++;
2194         //printf("CAUDA %s ----   INSERIDAS: %d \n",pnew->str,p->npecas);
2195         return;
2196
2197     }
2198
2199 }

```

void inserir_pecaint (PECASINIT *, int , int)

Inserir pecas inteiras na lista ligada do baralho.

função para inserir pecas inteiras nas listas ligadas

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>int</i>	dir que contem lado direito da peca nova a inserir
<i>int</i>	dir que contem lado esquerdo da peca nova a inserir

```

2067 {
2068
2076     PECAINT *pnew = (PECAINT*)malloc(sizeof(PECAINT));
2077     PECAINT *paux = NULL;
2078
2079     pnew->direito=dir;
2080     pnew->esquerdo=esq;
2081     pnew->pnext=NULL;
2082
2083     paux=p->pfirstint;
2084
2085     if(p->pfirstint==NULL)
2086     {
2087
2088         p->pfirstint=pnew;
2089         p->npecasint++;
2090
2091     }
2092     else

```

```

2093     {
2094
2095         //cauda
2096         while (paux->pnext!=NULL)
2097         {
2098
2099             paux=paux->pnext;
2100
2101         }
2102
2103         paux->pnext=pnew;
2104         p->npecasint++;
2105
2106         return;
2107
2108     }
2109
2110 }

```

void inserir_seqf (PECASINIT * , char [])

Inserir sequencias no array dinamico das seqs.

função para inserir sequencias no array dinamico

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>char</i>	sequencia[] contem sequencia a inserir

```

3772 {
3773
3780     SEQ *paux=NULL;
3781     paux=p->seqf;
3782
3783     while (paux!=NULL && paux->seqstr!=NULL && (paux - (p->seqf)) < p->nseqf)
3784     {
3785
3786         paux++;
3787
3788     }
3789
3790     if ((paux - (p->seqf)) == p->nseqf)
3791     {
3792
3793         create_array_seqf(p,p->nseqf+2);
3794         paux = p->seqf+p->nseqf-2;
3795         paux->seqstr=create_dyn_string(sequencia);
3796
3797     }else
3798     {
3799
3800         paux->seqstr=create_dyn_string(sequencia);
3801
3802     }
3803
3804 }

```

void inserir_seqss (PECASINIT * , char [])

Inserir sequencias no array dinamico das seqs auxiliares.

função para inserir sequencias no array dinamico

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>char</i>	sequencia[] contem sequencia a inserir

```

3697 {
3698
3705     SEQ *paux=NULL;
3706     paux=p->seqss;
3707
3708     while(paux!=NULL && paux->seqstr!=NULL && (paux - (p->seqss)) < p->nseqss)
3709     {
3710
3711         paux++;
3712
3713     }
3714
3715     if((paux-(p->seqss))==p->nseqss)
3716     {
3717
3718         create_array_seqss(p,p->nseqss+2);
3719         paux = p->seqss+p->nseqss-2;
3720         paux->seqstr=create_dyn_string(sequencia);
3721
3722     }else
3723     {
3724
3725         paux->seqstr=create_dyn_string(sequencia);
3726
3727     }
3728
3729 }

```

void inserir_seqssaux (PECASINIT *, char [])

Inserir sequencias no array dinamico das seqs auxiliares 2.

função para inserir sequencias no array dinamico

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>char</i>	sequencia[] contem sequencia a inserir

```

3735 {
3736
3743     SEQ *paux=NULL;
3744     paux=p->seqssaux;
3745
3746     while(paux!=NULL && paux->seqstr!=NULL && (paux - (p->seqssaux)) < p->nseqssaux)
3747     {
3748
3749         paux++;
3750
3751     }
3752
3753     if((paux-(p->seqssaux))==p->nseqssaux)
3754     {
3755
3756         create_array_seqssaux(p,p->nseqssaux+2);
3757         paux = p->seqssaux+p->nseqssaux-2;
3758         paux->seqstr=create_dyn_string(sequencia);
3759
3760     }else
3761     {
3762
3763         paux->seqstr=create_dyn_string(sequencia);
3764
3765     }

```

```
3766
3767 }
```

char* inverterstr (char str1[], char str2[])

Função inverterstr.

função para inverter uma sequencia

Parâmetros:

char	str1[] string a inverter
char	str2[] string invertida a retornar

Retorna:

retorna a string invertida

```
4150 {
4151
4159 //funcao que retorna uma sequencia de peças invertida
4160 int invertidas=0;
4161
4162 for(invertidas=0; str1[invertidas]!='\0'; invertidas++)
4163 {
4164
4165     str2[invertidas]=str1[strlen(str1)-1-invertidas];
4166
4167 }
4168
4169 return str2;
4170
4171 }
```

int jogadois (char baralhoss[][COLSTR], char seqf[][COLSEQ], int , char seqinit[])

Função jogadois.

esta funcao é igual à função das sequencias, só que nestas posso começar inicialmente com uma sequencia on nao e os jogadores jogam à vez de cada baralho

Parâmetros:

char	baralhoss[][COLSTR] baralhos do jogador em string
char	seqf[][COLSEQ] array final onde vou guardar as sequencias todas possiveis
int	num numero de jogos a calcular sequencias
char	seqinit[] sequencia inicial

Retorna:

retorna o numero de sequencias que encontrou

VERIFICA QUANTIDADE DE CORRESPONDENCIAS

JUNTA AS CORRESPONDENCIAS

```
5491 {
5492
5501 //esta função é igual à função de sequencias com a particularidade que os jogadores
jogam alternadamente para fazerem sequencias
5502
5503 char seqss[LINSEQ][COLSEQ];
5504 char baralhoaux[LINSEQ][COLSEQ];
5505 char invertidas[LINSEQ][COLSEQ];
5506 char vinvertidas[LINSEQ][COLSEQ];
5507 char auxinv[COLSTR];
```

```

5508     char invfinal[COLSTR];
5509     int i=0;
5510     int j=0;
5511     int k=0;
5512     int tam=LINSEQ;
5513     int cont=0;
5514     int fimdastring=0;
5515     int decont=LINSEQ-1;
5516     int iguais=0;
5517     char aux[2000];
5518     char seqinitinvertida[200];
5519     char *s;
5520     char *inv;
5521     char *invseq=NULL;
5522     int contadorfinal=0;
5523     int contapecasvinvertidas=0;
5524     int numero=1;
5525     int tambaralhosaux=0;
5526     /*a variavel ok acrescentada para apenas passar para o array secundário o que foi
passado para o final sem concatenar
5527     peças mal*/
5528     int ok=0;
5529
5530     esvaziaseqstr(seqss,LINSEQ);
5531     esvaziaseqstr(baralhoaux,LINSEQ);
5532
5533     //se nao tiver uma sequencia inicial, comecao com o primeiro jogo
5534
5535     tambaralhosaux=baralhoausar(baralhoss,baralhoaux,numero);
5536
5537     if(strcmp(seqinit,"9|9")==0)
5538     {
5539
5540         for(i=0; i<tambaralhosaux; i++)
5541         {
5542
5543             strcpy(seqss[i],baralhoaux[i]);
5544
5545
5546         }
5547
5548         tambaralhosaux=baralhoausar(baralhoss,baralhoaux,++numero);
5549
5550     }
5551     else
5552     {
5553
5554         strcpy(seqss[1],seqinit);
5555         for(i=0; seqinit[i]!='\0'; i++)
5556         {
5557
5558             seqinitinvertida[i]=seqinit[strlen(seqinit)-1-i];
5559
5560         }
5561         strcpy(seqss[0],seqinitinvertida);
5562
5563     }
5564
5565     do
5566     {
5567
5568         cont=0;
5569         for(i=0; i<tambaralhosaux; i++)
5570         {
5571
5572             for(j=0; j<tam; j++)
5573             {
5574
5575                 if(k!=0)
5576                 {

```

```

5580
5581         strcpy(aux,seqss[j]);
5582
5583         s = strtok (aux,"-");
5584
5585         while (s!= NULL)
5586         {
5587
5588             if(strcmp(baralhoaux[i],s)==0 || (s[2]==baralhoaux[i][0] &&
s[0]==baralhoaux[i][2]))
5589             {
5590
5591                 iguais++;
5592
5593             }
5594
5595             s = strtok (NULL, "-");
5596
5597         }
5598
5599         strcpy(aux,"");
5600     }
5601
5602     if(iguais==0)
5603     {
5604
5605         /*Ve o tamanho da string*/
5606
5607         for(fimdastring=0; seqss[j][fimdastring]!='\0'; fimdastring++);
5608         fimdastring--;
5609
5610         /*compara as strings iniciasi e invertidas com aquelas que vamos
aumentar*/
5611
5612
5613         if(seqss[j][fimdastring]==baralhoaux[i][0] && i!=j)
5614         {
5615
5616
5617             cont++;
5618             strcpy(seqf[contadorfinal],seqss[j]);
5619             strcat(seqf[contadorfinal],"-");
5620             strcat(seqf[contadorfinal],baralhoaux[i]);
5621             contadorfinal++;
5622             ok++;
5623
5624         }
5625
5626     }
5627     iguais=0;
5628
5629 }
5630
5631 }
5632
5633
5634 for(i=0; i<tambaralhosaux; i++)
5635 {
5636
5637     for(j=0; j<ok; j++)
5638     {
5639
5640         if(k!=0)
5641         {
5642
5643             strcpy(aux,seqss[j]);
5644
5645             s = strtok (aux,"-");
5646
5647             while (s!= NULL)
5648             {

```



```

5651
5652             if(strcmp(baralhoaux[i],s)==0 || (s[2]==baralhoaux[i][0] &&
s[0]==baralhoaux[i][2]))
5653             {
5654
5655                 iguais++;
5656
5657             }
5658
5659             s = strtok (NULL, "-");
5660
5661         }
5662
5663         strcpy(aux,"");
5664
5665         if(iguais==0)
5666         {
5667
5668             for(fimdastring=0; seqss[j][fimdastring]!='\0';
fimdastring++);
5669             fimdastring--;
5670
5671
5672             if(seqss[j][fimdastring]==baralhoaux[i][0] && i!=j)
5673             {
5674
5675                 strcpy(seqss[decont],seqss[j]);
5676                 strcat(seqss[decont],"-");
5677                 strcat(seqss[decont],baralhoaux[i]);
5678                 decont--;
5679
5680             }
5681         }
5682
5683     }
5684
5685     else
5686     {
5687
5688         for(fimdastring=0; seqss[j][fimdastring]!='\0'; fimdastring++);
5689         fimdastring--;
5690
5691         if(seqss[j][fimdastring]==baralhoaux[i][0] && i!=j)
5692         {
5693
5694             strcpy(seqss[decont],seqss[j]);
5695             strcat(seqss[decont],"-");
5696             strcat(seqss[decont],baralhoaux[i]);
5697             decont--;
5698
5699         }
5700
5701     }
5702
5703     iguais=0;
5704
5705 }
5706
5707 }
5708
5709 //Agora limpa o array seqss em cima e assa de baixo para cima para continuar
a juntar
5710     i=0;
5711
5712     while (strcmp(seqss[i],"9|9")!=0)
5713     {
5714
5715         strcpy(seqss[i],"9|9");
5716         i++;
5717
5718     }

```

```

5719
5720     decont=LINSEQ-1;
5721     i=0;
5722
5723     //apaga array em baixo
5724
5725     while (strcmp(seqss[decont],"9|9")!=0)
5726     {
5727
5728         strcpy(seqss[i],seqss[decont]);
5729         strcpy(seqss[decont],"9|9");
5730         decont--;
5731         i++;
5732     }
5733
5734     tam=(LINSEQ-decont-1);
5735     decont=LINSEQ-1;
5736     k++;
5737
5738
5739     //aqui faço com que o baralho auxiliar a usar seja usado alternadamente ou seja,
na primeira vez
5740     //comparo o baralhoaux do jogador 1 mas na segunda vez comparo com o baralho
auxiliar do jogador dois de modo
5741     // a fazer sequencias com os baralhos dos jogadores alternadamente
5742
5743     if(numero<num)
5744     {
5745
5746         tambaralhosaux=baralhoausar(baralhoss,baralhoaux,++numero);
5747
5748     }
5749     else
5750     {
5751
5752         numero=1;
5753         tambaralhosaux=baralhoausar(baralhoss,baralhoaux,numero);
5754
5755     }
5756
5757 }
5758
5759 while(cont>0);
5760
5761 //verifica se contem invertidas e normais e retira-as
5762 cont=0;
5763 i=0;
5764
5765 while(strcmp(seqf[cont],"9|9")!=0)
5766 {
5767     strcpy(seqss[i],seqf[cont]);
5768     cont++;
5769     i++;
5770 }
5771
5772
5773 esvaziaseqstr(invertidas,LINSEQ);
5774
5775 for(i=0; i<cont; i++)
5776 {
5777
5778     strcpy(invertidas[i],seqss[i]);
5779 }
5780
5781 j=0;
5782 for(i=0; i<cont; i++)
5783 {
5784
5785     invseq = strtok (invertidas[i],"-");
5786
5787

```

```

5788     while (invseq != NULL)
5789     {
5790
5791         strcpy(vinvertidas[j],invseq);
5792         j++;
5793         invseq = strtok (NULL, "-");
5794
5795     }
5796     contapecasvinvertidas=j;
5797     j=0;
5798     inv = strtok (seqss[i],"-");
5799
5800     while (inv != NULL)
5801     {
5802
5803         strcpy(auxinv,inv);
5804
5805         invfinal[2]=auxinv[0];
5806         invfinal[1]=auxinv[1];
5807         invfinal[0]=auxinv[2];
5808
5809         if(invfinal[2]!=invfinal[0])
5810         {
5811
5812             for(k=0; k<contapecasvinvertidas; k++)
5813             {
5814
5815                 if(strcmp(invfinal,vinvertidas[k])==0)
5816                 {
5817
5818                     strcpy(seqf[i],"-");
5819
5820                 }
5821
5822             }
5823
5824         }
5825
5826         inv = strtok (NULL, "-");
5827
5828     }
5829     esvaziaseqstr(vinvertidas,LINSEQ);
5830     j=0;
5831 }
5832
5833 contadorfinal=0;
5834 i=0;
5835
5836 while(i<LINSEQ)
5837 {
5838
5839     if(strcmp(seqf[i],"9|9")!=0)
5840     {
5841
5842         contadorfinal++;
5843
5844     }
5845
5846     i++;
5847 }
5848
5849 return contadorfinal;
5850
5851 }

```

void load_jogo_bin (PECASINIT * , char [])

Load do ficheiro binario.

função para carregar um ficheiro binario

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>char</i>	fname[] nome do ficheiro a carregar

```

1561 {
1562     //SE QUISERMOS GUARDAR TODA A ESTRUTURA NUM FICHEIRO
1563     //fwrite(&t,sizeof(t),1,fp);
1564     FILE *fp=NULL;
1565
1566     char nome[50];
1567     int size=0;
1568     int i=0;
1569     int n=0;
1570
1571     if((fp=fopen(fname,"rb"))!=NULL)
1572     {
1573         fread(&(p->npecas),sizeof(int),1,fp);
1574         n=p->npecas;
1575         p->npecas=0;
1576
1577         for(i=0;i<n;i++)
1578         {
1579             fread(&size,sizeof(int),1,fp);
1580             fread(nome,sizeof(char),size,fp);
1581             inserir_pecas(p,nome);
1582         }
1583         fclose(fp);
1584     }
1585 }
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600 }
```

void load_txt_jogo (PECASINIT * p, char [])

Load do ficheiro txt.

função para carregar um ficheiro txt

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
<i>char</i>	fname[] nome do ficheiro a carregar

```

1605 {
1606
1607     FILE *fp=NULL;
1608     char nome[50];
1609     char ignora[50];
1610     char njogos[1];
1611     int i=0;
1612     int n=0;
1613     /*%[^:] -> ler até ':'
1614     %*[:] -> ignorar ':'
1615     %[^,] -> ler até ','
1616     %*[,] -> ignorar ','
1617     %[^:] -> ler até ':'
1618     %*[:] -> ignorar ':'
1619 }
```

```

1625     %d    -> ler inteiro*/
1626
1627     if((fp=fopen(fname,"r"))==NULL)
1628     {
1629
1630         printf("... ERRO ...");
1631         return;
1632     }
1633
1634     //gravo o numero de jogos
1635     fscanf(fp,"%[^ ]",njogos);
1636     n=atoi(njogos);
1637
1638     //nao gravo a palavra jogos
1639     fscanf(fp,"%[^-] %*[-] %*[\n]",ignora);
1640     //pecas
1641
1642
1643
1644     for(i=0;i<n*7;i++)
1645     {
1646
1647         fscanf(fp,"%[^\\n] %*[ ]",nome);
1648         inserir_peca(p,nome);
1649     }
1650
1651
1652
1653 }

```

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

Função main.

função que contém o menu do programa e as respectivas chamadas as funcoes, neste momento estou também a fazer algumas verificações para verificar se as subsequências a procurar são válidas, se os padrões a substituir são válidos. Estas verificações passarão posteriormente a ser realizadas dentro de funções

Parâmetros:

<i>int</i>	argc não está a ser usado mas contém o numero de posicoes do array de strings argv[] usadas,
<i>char</i>	*argv[] não está a ser usado, mas contém o numero de posições do array de strings

Retorna:

0

MENU

```

66 {
67     PECASINIT p= {NULL,NULL,NULL,0,0,0};
68
69     int num=1;
70     int jogosaaimprimir=0;
71     int numdeseq=0;
72     int pecasint[LIN][COL];
73     int baralhosint[LIN][COL];
74     char pecasstr[LIN][COLSTR];
75     char baralhosstr[LIN][COLSTR];
76     char seqstr[LINSEQ][COLSEQ];
77     char subseq[LINSEQ];
78     char padrao[LINSEQ]="4|3";
79     char padraonovo[LINSEQ]="0|9-9|9-9|3";
80     char seqinicial[LINSEQ]="9|9";
81     int tam=0;

```

```

89
90
91     //var menu
92     char seqinicialaux[LINSEQ];
93     char seqinicialpartida[LINSEQ][COLSEQ];
94     char pecasstrinv[COLSTR];
95     char pecasstraux[LIN][COLSTR];
96     char op='0';
97     char op1='0';
98     char op2='0';
99     char op3='0';
100    char op4='0';
101    int i=0;
102    int j=0;
103    int r=0;
104    int cont=0;
105    char subseqinv[COLSEQ];
106    int contseq=0;
107    int matrizprocurasub[LINSEQ][2];
108    int k=0;
109    char padraonovoinv[LINSEQ];
110    char padraoinv[COLSEQ];
111    char seqfpadiao[LINSEQ][COLSEQ];
112    int sizeseqfpadiao=0;
113    char subseqaux[COLSEQ];
114    char auxinv[COLSEQ];
115    int tamsubseq=0;
116    char seqpadraoaux[LINSEQ][COLSEQ];
117    char padraoaux[COLSEQ];
118    char baralhosstrauxcominv[LINSEQ][COLSEQ];
119    char baralhosstrauxcominvnovo[LINSEQ][COLSEQ];
120    char baralhosstrauxcominvnovopadraonovo[LINSEQ][COLSEQ];
121    char pecastrauxinvpadroes[LINSEQ][COLSTR];
122    char padraototal[COLSEQ];
123
124    //cria pecas todas
125    criapecasint(pecasint,LIN);
126    criapecasstr(pecasstr,LIN);
127    criapecasstr(pecasstraux,LIN);
128
129    //limpa baralhos
130    esvaziabaralho(int)(baralhosint,LIN,COL);
131    esvaziabaralhostr(baralhosstr,LIN);
132    esvaziaseqstr(seqstr,LINSEQ);
133    esvaziaseqstr(seqfpadiao,LINSEQ);
134    esvaziaseqstr(seqinicialpartida,LINSEQ);
135    esvaziaseqstr(baralhosstrauxcominv,LINSEQ);
136    esvaziaseqstr(baralhosstrauxcominvnovo,LINSEQ);
137    esvaziabaralhostr(pecastrauxinvpadroes,LINSEQ);
138    esvaziaseqstr(baralhosstrauxcominvnovopadraonovo,LINSEQ);
139
140    create_array_seqf(&p,2);
141    create_array_seqss(&p,2);
142    create_array_seqssaux(&p,2);
143    create_array_baralhoaux(&p,2);
144
145    char filename[]="./lp1.txt";
146
147
148    while(1)
149    {
150
151        switch(op)
152        {
153
154            case '0':
155
156                printf("--DOMINO--\n\n");
157                printf("1- Ler Jogos do ficheiro\n");
158                printf("2- Inserir manualmente\n");
159                printf("3- Ler Jogos de um ficheiro binario\n");

```

```

162
163
164         scanf("%c",&op1);
165         system("CLS");
166
167         switch(op1)
168         {
169
170             case '0':
171
172                 op='0';
173
174                 break;
175
176             case '1':
177
178                 load txt jogo(&p,filename);
179                 mostrarjogosstrstruct(p,7);
180                 op='2';
181                 scanf("%c",&op2);
182
183                 break;
184
185             break;
186
187             case '2':
188
189                 printf("Quantos jogos pretende:\n");
190                 scanf("%d",&num);
191                 system("CLS");
192
193                 if (num<1||num>4)
194                 {
195
196                     printf("O NUMERO DE JOGOS TEM DE SER ENTRE 1 e 4!!\n");
197                     op='0';
198
199                 }
200                 else
201                 {
202
203                     op='1';
204
205                 }
206
207                 break;
208
209             case '3':
210
211                 load jogo bin(&p,"DOMINO.BIN");
212                 mostrarjogosstrstruct(p,7);
213                 op='3';
214                 scanf("%c",&op2);
215
216                 break;
217
218             break;
219
220             default:
221
222                 printf("OPCAO INVALIDA\n");
223                 op='0';
224
225                 break;
226
227         }
228
229
230     break;
231
232

```

```

233     case '1':
234
235         printf("--MENU--\n\n");
236         printf("1- Criar jogo(s) aleatoriamente\n");
237         printf("2- Preencher jogo(s)\n");
238         printf("0- Voltar atras\n");
239         scanf(" %c",&op2);
240         system("CLS");
241
242
243         switch(op2)
244         {
245
246             case '1':
247
248                 p.npecas=0;
249                 p.pfirst=NULL;
250                 jogosaimprimir=entregarbaralhos(pecasstr,&p,num);
251
252                 /*inserir_pecas(&p,"2|6");
253                 inserir_pecas(&p,"6|6");
254                 inserir_pecas(&p,"6|4");
255                 inserir_pecas(&p,"4|5");
256                 inserir_pecas(&p,"5|0");
257                 inserir_pecas(&p,"0|3");
258                 inserir_pecas(&p,"3|1");*/
259                 mostrarjogosstrstruct(p,7);
260                 //mostrarjogosstrstruct(p,jogosaimprimir);
261                 /*
262                 strcpy(baralhosstr[0],"6|4");
263                 strcpy(baralhosstr[1],"4|1");
264                 strcpy(baralhosstr[2],"2|2");
265                 strcpy(baralhosstr[3],"6|6");
266                 strcpy(baralhosstr[4],"5|1");
267                 strcpy(baralhosstr[5],"4|0");
268                 strcpy(baralhosstr[6],"3|2");
269                 */
270                 op='2';
271
272                 break;
273
274             case '2':
275
276                 p.npecas=0;
277                 p.pfirst=NULL;
278                 jogosaimprimir=preenchebaralhosstruct(pecasstr,&p,num);
279                 mostrarjogosstrstruct(p,jogosaimprimir);
280                 op='2';
281
282                 break;
283
284             case '0':
285
286                 op='0';
287
288                 break;
289
290             default:
291
292                 printf("OPCAO INVALIDA\n");
293                 op='1';
294
295                 break;
296
297         }
298
299
300
301         break;
302
303     case '2':

```



```

304
305     printf("--MENU--\n\n");
306     printf("1- Mostrar sequencias possiveis, (decrecente)\n");
307     printf("2- Mostrar sequencias possiveis, (decrecente), com sequencia
inicial\n");
308     printf("3- Mostrar sequencias possiveis, (decrecente) com jogadores a
jogar alternadamente\n");
309     printf("4- Alterar Pecas\n");
310     printf("5- Converter String para Inteiro\n");
311     printf("0- Voltar ao Menu anterior\n\n");
312
313     scanf(" %c",&op2);
314     system("CLS");
315
316     switch(op2)
317     {
318
319     case '1':
320
321         numdeseq=seq(&p,num);
322         ordenarsequenciasstruct(&p);
323         separarsequinvertidasstruct(&p);
324         printseqstrstruct(p);
325         /*numdeseq=seq(baralhosstr,seqstr,num);
326         ordenarsequencias(seqstr,numdeseq);
327         numdeseq=tirartracosinvertidos(seqstr,numdeseq);
328         numdeseq=separarsequinvertidas(seqstr,numdeseq);
329         printseqstr(seqstr,0,numdeseq);
330         printf("\n");
331
332         mostrarjogosstr(baralhosstr,jogosaimprimir);
333         contseq=1;
334         */
335         op='3';
336
337         break;
338
339     case '2':
340
341         mostrarjogosstr(baralhosstr,jogosaimprimir);
342         esvaziaseqstr(seqinicialpartida,LINSEQ);
343         esvaziaseqstr(baralhosstrauxcominv,LINSEQ);
344         cont=0;
345         printf("Insira a sequencia inicial: (SO PODE ESCOLHER PECAS DAS
SEGUINTEs)\n\n");
346         //copia as pecas iniciais para o baralho vazio
347
348         for(i=0; i<LIN; i++)
349         {
350
351             strcpy(baralhosstrauxcominv[i],pecasstr[i]);
352
353         }
354
355         //copio as respectivas invertidas para o baralho auxiliar
356
357         for(j=0; j<LIN; j++)
358         {
359
360             if(pecasstr[j][0]!=pecasstr[j][2])
361             {
362
363                 strcpy(baralhosstrauxcominv[i++],inverterstr(pecasstr[j],auxinv));
364
365             }
366
367         }
368
369         for(i=0; strcmp(baralhosstrauxcominv[i],"9|9")!=0; i++)

```

```

371         {
372             for(j=0; strcmp(baralhosstr[j],"9|9")!=0; j++)
373             {
374                 if(strcmp(baralhosstrauxcominv[i],baralhosstr[j])==0 ||
375                 strcmp(baralhosstrauxcominv[i],inverterstr(baralhosstr[j],pecasstrinv))==0)
376                 {
377                     strcpy(baralhosstrauxcominv[i],"-");
378                 }
379             }
380         }
381     }
382 }
383
384
385
386
387
388
389     scanf("%s",seqinicial);
390
391     if(verificasequencia(seqinicial)==1)
392     {
393         k=1;
394     }
395
396
397     //partimos a sequencia inicial partida e guardamos no array auxiliar
398     strcpy(seqinicialaux,seqinicial);
399     tamsubseq=strtoque(seqinicialpartida,seqinicialaux,'-');
400
401     //verifico de o padrao novo possui pecas repetidas ou nao
402     for(i=0; strcmp(seqinicialpartida[i],"9|9")!=0; i++)
403     {
404         for(j=0; strcmp(baralhosstrauxcominv[j],"9|9")!=0; j++)
405         {
406             //printf("%s ==
407             %s\n",seqpadraoaux[i],baralhosstrauxcominvnovopadraonovo[j]);
408         }
409         if(strcmp(seqinicialpartida[i],baralhosstrauxcominv[j])==0)
410         {
411             strcpy(baralhosstrauxcominv[j],"-");
412             cont++;
413             for(r=0; strcmp(baralhosstrauxcominv[r],"9|9")!=0; r++)
414             {
415                 if(strcmp(inverterstr(seqinicialpartida[i],pecasstrinv),baralhosstrauxcominv[r])==0)
416                 {
417                     strcpy(baralhosstrauxcominv[r],"-");
418                 }
419             }
420         }
421     }
422 }
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437

```

```

438
439         if(cont!=tamsubseq || k==1)
440         {
441
442             printf("A sequequencia nao e valida!!\n\n");
443
444             //volta a carregar o pecasstraux com as pecas todas e esvazia o
array da sequencia inicial partida
445             for(i=0; i<LIN; i++)
446             {
447
448                 strcpy(pecasstraux[i],pecasstr[i]);
449
450             }
451             k=0;
452             op2='2';
453
454
455         }
456     else
457     {
458
459         numdeseq=seqcomseqinicial(baralhosstr,seqstr,num,seqinicial);
460         ordenarsequencias(seqstr,numdeseq);
461         numdeseq=tirartracosinvertidos(seqstr,numdeseq);
462         numdeseq=separarseqinvertidas(seqstr,numdeseq);
463         tam=retiraseqinitrepetida(seqstr,numdeseq,seqinicial);
464         ordenarsequencias(seqstr,numdeseq);
465         numdeseq=numdeseq-tam;
466         printseqstr(seqstr,0,numdeseq);
467
468         mostrarjogosstr(baralhosstr,jogosaimprimir);
469         contseq=2;
470         op='3';
471
472     }
473
474     break;
475
476     case '0':
477
478         op='1';
479
480         break;
481
482     case '3':
483
484         if(num<2)
485         {
486
487             printf("O numero de baralhos para esta opcao tem de ser no minimo
(2)");
488             op='0';
489
490         }
491     else
492     {
493
494         op='4';
495
496     }
497
498     break;
499
500
501     case '4':
502
503         jogosaimprimir=remover(&p,pecasstr,num);
504         mostrarjogosstrstruct(p,jogosaimprimir);
505
506         break;

```

```

507
508         case '5':
509
510             printf("String para inteiro || Esvazia o baralho (string) || Inteiro
para string\n");
511             jogosaimprimir=convertestrtoint(&p,num);
512             mostrarjogosintstruct(p,jogosaimprimir);
513
514             esvaziabaralhostruct(&p);
515             mostrarjogosstruct(p,jogosaimprimir);
516
517             jogosaimprimir=converteinttostr(&p,num);
518             mostrarjogosstruct(p,jogosaimprimir);
519
520             break;
521
522         default:
523
524             printf("OPCAO INVALIDA\n");
525             mostrarjogosstr(baralhosstr,jogosaimprimir);
526
527             break;
528
529     }
530
531
532
533     break;
534
535     case '3':
536
537         printf("\n\n--MENU--\n\n");
538         printf("1- Procurar subsequencias\n");
539         printf("2- Substituir padroes nas sequencias\n");
540         printf("3- Gravar para ficheiro txt\n");
541         printf("4- Gravar para ficheiro binario\n");
542
543
544         printf("0- Voltar ao menu principal\n\n");
545
546         scanf(" %c",&op3);
547         system("CLS");
548
549         switch(op3)
550         {
551
552             case '4':
553
554                 save_jogo_bin(p,"DOMINO.BIN");
555
556             break;
557
558             case '1':
559
560                 if(contseq==1)
561                 {
562
563                     //numdeseq=seq(baralhosstr,seqstr,num);
564                     ordenarsequencias(seqstr,numdeseq);
565                     numdeseq=tirartracosinvertidos(seqstr,numdeseq);
566                     numdeseq=separarseqinvertidas(seqstr,numdeseq);
567                     printseqstr(seqstr,0,numdeseq);
568                     printf("\n");
569
570                     mostrarjogosstr(baralhosstr,jogosaimprimir);
571
572                 }
573
574                 if(contseq==2)
575                 {
576

```

```

577         numdeseq=seqcomseqinicial(baralhosstr,seqstr,num,seqinicial);
578         ordenarsequencias(seqstr,numdeseq);
579         numdeseq=tirartracosinvertidos(seqstr,numdeseq);
580         numdeseq=separarseqinvertidas(seqstr,numdeseq);
581         tam=retiraseqinitrepetida(seqstr,numdeseq,seqinicial);
582         ordenarsequencias(seqstr,numdeseq);
583         numdeseq=numdeseq-tam;
584         printseqstr(seqstr,0,numdeseq);
585
586         mostrarjogosstr(baralhosstr,jogosaimprimir);
587
588     }
589
590
591     k=0;
592     printf("Qual a sub sequencia a procurar:\n");
593     scanf("%s",subseq);
594
595     //fazemos uma copia da subseq
596     strcpy(subseqaux,subseq);
597     //reinicia os contadores e variavies
598     esvaziaseqstr(seqinicialpartida,LINSEQ);
599     esvaziaseqstr(baralhosstrauxcominv,LINSEQ);
600     cont=0;
601
602
603     //copia para array auxiliar
604     for(i=0; strcmp(baralhosstr[i],"9|9")!=0; i++)
605     {
606
607         strcpy(baralhosstrauxcominv[i],baralhosstr[i]);
608
609     }
610
611     //copia invertidas para array auxiliar
612     for(j=0; strcmp(baralhosstr[j],"9|9")!=0; j++)
613     {
614         if(baralhosstr[j][0]!=baralhosstr[j][2])
615         {
616
617
618     strcpy(baralhosstrauxcominv[i++],inverterstr(baralhosstr[j],auxinv));
619         }
620
621
622     }
623
624     //se tivermos a usar uma sequencia inicial ao procurar uma subsequencia
625     temos de também poder procurar pelas peças usadas na sequencia inicial
626     if(strcmp(seqinicial,"9|9")!=0)
627     {
628
629         strtok(seqinicialpartida,seqinicial,"-");
630
631         for(j=0; strcmp(seqinicialpartida[j],"9|9")!=0; j++)
632         {
633
634             strcpy(baralhosstrauxcominv[i++],seqinicialpartida[j]);
635
636         }
637
638         for(j=0; strcmp(seqinicialpartida[j],"9|9")!=0; j++)
639         {
640
641             if(seqinicialpartida[j][0]!=seqinicialpartida[j][2])
642             {
643
644
645     strcpy(baralhosstrauxcominv[i++],inverterstr(baralhosstrauxcominv[j],auxinv));

```

```

645
646         }
647
648     }
649
650 }
651
652 //esvaziamos o array de strings da sequencia inicial partida para
podermos partir a subsequencia
653
654     esvaziaseqstr(seqinicialpartida,LINSEQ);
655
656     tamsubseq=strtoque(seqinicialpartida,subseqaux,'-');
657
658
659     //verifico de o padrao novo possui pecas repetidas ou nao
660
661     for(i=0; strcmp(seqinicialpartida[i],"9|9")!=0; i++)
662     {
663
664         for(j=0; strcmp(baralhosstrauxcominv[j],"9|9")!=0; j++)
665         {
666             //printf("%s ==
667 %s\n",seqpadraoaux[i],baralhosstrauxcominvnovopadraonovo[j]);
668
669         if(strcmp(seqinicialpartida[i],baralhosstrauxcominv[j])==0)
670         {
671             strcpy(baralhosstrauxcominv[j],"-");
672             cont++;
673
674             for(r=0; strcmp(baralhosstrauxcominv[r],"9|9")!=0; r++)
675             {
676
677         if(strcmp(inverterstr(seqinicialpartida[i],pecasstrinv),baralhosstrauxcominv[r])==0)
678         {
679             strcpy(baralhosstrauxcominv[r],"-");
680
681         }
682
683     }
684
685     }
686
687     }
688
689     }
690
691     }
692
693     }
694
695
696     if(k==1 || tamsubseq!=cont)
697     {
698
699         printf("A sequequencia nao e valida!!\n\n");
700
701         //volta a carregar o pecasstraux com as pecas todas e esvazia o
array da sequencia inicial partida
702         for(i=0; i<LIN; i++)
703         {
704
705             strcpy(pecasstraux[i],pecasstr[i]);
706
707         }
708
709         esvaziaseqstr(seqinicialpartida,LINSEQ);
710         esvaziaseqstr(baralhosstrauxcominv,LINSEQ);

```

```

711             cont=0;
712             k=0;
713             op2='2';
714
715
716         }
717
718
719         if(verificasequencia(subseq)==0)
720         {
721
722             //procura sub-sequencia normal na sequencia e se encontrar guarda
723             //e na segunda posição da matriz guarda a primeira peça em que fez
724             o match
725             for(i=0; i<numdeseq; i++)
726             {
727                 if(probsubseq_ausar(seqstr,i,subseq)!=-1)
728                 {
729
730                     matrizprocurasub[k][0]=i;
731                     matrizprocurasub[k][1]=probsubseq_ausar(seqstr,i,subseq);
732                     k++;
733                 }
734             }
735
736             //procura sub-sequencia invertida na sequencia e se encontrar
737             //guarda indicena primeira posicao de uma matriz
738             //e na segunda posição da matriz guarda a primeira peça em que fez
739             o match
740             inverterstr(subseq,subseqinv);
741             for(i=0; i<numdeseq; i++)
742             {
743                 if(probsubseq_ausar(seqstr,i,subseqinv)!=-1)
744                 {
745                     matrizprocurasub[k][0]=i;
746                     matrizprocurasub[k][1]=probsubseq_ausar(seqstr,i,subseqinv);
747                     k++;
748                 }
749             }
750
751             ordenarmatrizinteiros(matrizprocurasub,k);
752             for(i=0; i<k; i++)
753             {
754                 printf("[%d] %s --->
755                 %d\n",matrizprocurasub[i][0],seqstr[matrizprocurasub[i][0]],matrizprocurasub[i][1]);
756             }
757
758             //esvaziaseqstr(seqfpadiao,LINSEQ);
759
760         }
761         else
762         {
763             op3='1';

```

```

775
776         }
777
778         break;
779
780     case '2':
781
782         if (contseq==1)
783         {
784
785             //numdeseq=seq(baralhosstr, seqstr, num);
786             ordenarsequencias (seqstr, numdeseq);
787             numdeseq=tirartracosinvertidos (seqstr, numdeseq);
788             numdeseq=separarseqinvertidas (seqstr, numdeseq);
789             printseqstr (seqstr, 0, numdeseq);
790             printf("\n");
791
792             mostrarjogosstr (baralhosstr, jogosaimprimir);
793             //contseq=0;
794
795         }
796
797     else if (contseq==2)
798     {
799
800         numdeseq=seqcomseqinicial (baralhosstr, seqstr, num, seqinicial);
801         ordenarsequencias (seqstr, numdeseq);
802         numdeseq=tirartracosinvertidos (seqstr, numdeseq);
803         numdeseq=separarseqinvertidas (seqstr, numdeseq);
804         tam=retiraseqinitrepetida (seqstr, numdeseq, seqinicial);
805         ordenarsequencias (seqstr, numdeseq);
806         printseqstr (seqstr, 0, numdeseq-tam);
807
808         mostrarjogosstr (baralhosstr, jogosaimprimir);
809         //contseq=0;
810     }
811     else
812     {
813
814         mostrarjogosstr (baralhosstr, jogosaimprimir);
815
816     }
817
818     k=0;
819     printf("Qual o padrao que pretende substituir:\n");
820     scanf("%s", padrao);
821
822     //fazemos uma copia do padrao
823     strcpy(padraoaux, padrao);
824     //reinicia os contadores e variavies
825     esvaziaseqstr (seqinicialpartida, LINSEQ);
826     esvaziaseqstr (baralhosstrauxcominv, LINSEQ);
827     cont=0;
828
829     //copia para array auxiliar
830     for (i=0; strcmp(baralhosstr[i], "9|9")!=0; i++)
831     {
832
833         strcpy(baralhosstrauxcominv[i], baralhosstr[i]);
834
835     }
836
837     //copia invertidas para array auxiliar
838     for (j=0; strcmp(baralhosstr[j], "9|9")!=0; j++)
839     {
840         if (baralhosstr[j][0]!=baralhosstr[j][2])
841         {
842
843             strcpy(baralhosstrauxcominv[i++], inverterstr (baralhosstr[j], auxinv));
844

```



```

845         }
846     }
847 }
848
849 //se tivermos a usar uma sequencia inicial ao procurar uma subsequencia
temos de também poder procurar pelas peças usadas na sequencia inicial
850
851 if(strcmp(seqinicial,"9|9")!=0)
852 {
853     strtouq(seqinicialpartida,seqinicial,'-');
854     for(j=0; strcmp(seqinicialpartida[j],"9|9")!=0; j++)
855     {
856         strcpy(baralhosstrauxcominv[i++],seqinicialpartida[j]);
857     }
858     for(j=0; strcmp(seqinicialpartida[j],"9|9")!=0; j++)
859     {
860         if(seqinicialpartida[j][0]!=seqinicialpartida[j][2])
861         {
862             strcpy(baralhosstrauxcominv[i++],inverterstr(seqinicialpartida[j],auxinv));
863         }
864     }
865 }
866
867 //guardo o array de baralhos, baralhos invertidos e sequencia inicial
se existir num array auxiliar
868
869 for(i=0; strcmp(baralhosstrauxcominv[i],"9|9")!=0; i++)
870 {
871     strcpy(baralhosstrauxcominvnovo[i],baralhosstrauxcominv[i]);
872 }
873
874 for(i=0; strcmp(seqstr[i],"9|9")!=0; i++)
875 {
876     esvaziaseqstr(seqpadraoaux,LINSEQ);
877     strtouq(seqpadraoaux,seqstr[i],'-');
878     for(j=0; strcmp(seqpadraoaux[j],"9|9")!=0; j++)
879     {
880         for(k=0; strcmp(baralhosstrauxcominv[k],"9|9")!=0; k++)
881         {
882             if(strcmp(seqpadraoaux[j],baralhosstrauxcominv[k])==0)
883             {
884                 strcpy(baralhosstrauxcominv[k],"-");
885             }
886         }
887     }
888     esvaziaseqstr(seqpadraoaux,LINSEQ);
889 }
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912

```

```

913     }
914
915     //retiro no array auxiliar as pecas que nao sairam nas sequencias e
essas nao podem estar contidas no padrao a substituir
916
917     for(i=0; strcmp(baralhosstrauxcominv[i],"9|9")!=0; i++)
918     {
919
920
921     if(strcmp(baralhosstrauxcominvnovo[i],baralhosstrauxcominv[i])==0)
922     {
923
924         strcpy(baralhosstrauxcominvnovo[k],"-");
925
926     }
927     }
928
929     //retiro as pecas que nao utilizei nas sequencias
930
931     for(j=0; strcmp(baralhosstrauxcominvnovo[j],"9|9")!=0 ; j++)
932     {
933
934         if(strcmp(baralhosstrauxcominv[j],"-")!=0)
935         {
936
937             strcpy(baralhosstrauxcominvnovo[j],"-");
938
939         }
940
941     }
942
943     //verifico quais as pecas que podem ter troca de padrao e guardo no
array com as suas respectivas invertidas
944     cont=0;
945     for(i=0; strcmp(baralhosstrauxcominv[i],"9|9")!=0; i++)
946     {
947
948         if(strcmp(baralhosstrauxcominv[i],"-")!=0)
949         {
950
951             for(j=0; strcmp(baralhosstrauxcominvnovo[j],"9|9")!=0 ;
j++)
952             {
953
954
955             if(strcmp(inverterstr(baralhosstrauxcominv[i],auxinv),baralhosstrauxcominvnovo[j])==0)
956             {
957
958                 for(k=0;
strcmp(baralhosstrauxcominvnovo[k],"9|9")!=0 ; k++)
959                 {
960
961                     if(strcmp(baralhosstrauxcominvnovo[k],"-")==0
&& cont==0)
962                     {
963
964                         strcpy(baralhosstrauxcominvnovo[k],baralhosstrauxcominv[i]);
965                         cont++;
966
967                     }
968                 }
969                 cont=0;
970
971             }
972
973         }
974
975     }

```

```

976
977     }
978
979
980
981     //se for 1 é porque nao é válida o padrao
982     if(verificasequencia(padrao)==1)
983     {
984
985         k=1;
986
987     }
988
989     //faz uma copia das pecas que podem ser usadas no padrao a substituir
990
991
992     for(i=0; strcmp(baralhosstrauxcominvnovo[i],"9|9")!=0; i++)
993     {
994
995
996     strcpy(baralhosstrauxcominvnovopadraonovo[i],baralhosstrauxcominvnovo[i]);
997     }
998
999     //verifica se a peca do padrao a substituir já está a ser usada para
nao haver repetidas
1000
1001     tamsubseq=strtoque(seqpadraoaux,padraoaux,'-');
1002     cont=0;
1003
1004
1005     for(i=0; strcmp(seqpadraoaux[i],"9|9")!=0; i++)
1006     {
1007
1008         for(j=0;
1009     strcmp(baralhosstrauxcominvnovopadraonovo[j],"9|9")!=0; j++)
1010     {
1011         //printf("%s ==
1012     %s\n",seqpadraoaux[i],baralhosstrauxcominvnovopadraonovo[j]);
1013
1014     if(strcmp(seqpadraoaux[i],baralhosstrauxcominvnovopadraonovo[j])==0)
1015     {
1016
1017         strcpy(baralhosstrauxcominvnovopadraonovo[j],"-");
1018         cont++;
1019
1020         for(r=0;
1021     strcmp(baralhosstrauxcominvnovopadraonovo[r],"9|9")!=0; r++)
1022     {
1023
1024     if(strcmp(inverterstr(seqpadraoaux[i],pecasstrinv),baralhosstrauxcominvnovopadraonovo[r]
1025     )==0)
1026     {
1027
1028     strcpy(baralhosstrauxcominvnovopadraonovo[r],"-");
1029
1030
1031     }
1032
1033     }
1034
1035     }
1036
1037     }

```

```

1038
1039         if(k==1 || tamsubseq!=cont)
1040         {
1041
1042             printf("O padrao nao e valido!!\n\n");
1043
1044             esvaziaseqstr(seqinicialpartida,LINSEQ);
1045             esvaziaseqstr(baralhosstrauxcominv,LINSEQ);
1046             esvaziaseqstr(baralhosstrauxcominvnovo,LINSEQ);
1047             esvaziaseqstr(baralhosstrauxcominvnovopadraonovo,LINSEQ);
1048             cont=0;
1049             k=0;
1050             op2='2';
1051
1052
1053         }
1054     else
1055     {
1056
1057         //Verifico o padrao novo
1058
1059         esvaziaseqstr(seqfpadrao,LINSEQ);
1060         esvaziaseqstr(baralhosstrauxcominvnovopadraonovo,LINSEQ);
1061         esvaziaseqstr(baralhosstrauxcominv,LINSEQ);
1062         //faz uma copia das pecas que NAO podem ser usadas no padrao a
substituir
1063
1064         for(i=0; strcmp(baralhosstrauxcominvnovo[i],"9|9")!=0; i++)
1065         {
1066
1067             strcpy(baralhosstrauxcominvnovopadraonovo[i],baralhosstrauxcominvnovo[i]);
1068
1069         }
1070
1071         //pecas iniciais e invertidas
1072         for(i=0; i<LIN; i++)
1073         {
1074
1075             strcpy(baralhosstrauxcominv[i],pecasstr[i]);
1076
1077         }
1078
1079         for(j=0; j<LIN; j++)
1080         {
1081
1082             if(pecasstr[j][0]!=pecasstr[j][2])
1083             {
1084
1085                 strcpy(baralhosstrauxcominv[i++],inverterstr(pecasstr[j],auxinv));
1086
1087             }
1088
1089         }
1090
1091         for(i=0; strcmp(baralhosstrauxcominv[i],"9|9")!=0; i++)
1092         {
1093
1094             for(j=0;
strcpy(baralhosstrauxcominvnovopadraonovo[j],"9|9")!=0; j++)
1095             {
1096
1097                 if(strcmp(baralhosstrauxcominv[i],baralhosstrauxcominvnovopadraonovo[j])==0)
1098                 {
1099
1100                     strcpy(baralhosstrauxcominv[i],"-");
1101
1102                 }
1103

```

```

1104
1105         }
1106
1107     }
1108
1109     printf("\n\n");
1110
1111
1112     printf("Qual o padrao novo:\n");
1113     scanf("%s",padraonovo);
1114
1115
1116     tamsubseq=strtoque(seqpadraoaux,padraonovo,'-');
1117     cont=0;
1118
1119
1120     if(verificasequencia(padraonovo)==1)
1121     {
1122
1123         k=1;
1124
1125     }
1126
1127     //verifico de o padrao novo possui pecas repetidas ou nao
1128
1129     for(i=0; strcmp(seqpadraoaux[i],"9|9")!=0; i++)
1130     {
1131
1132         for(j=0; strcmp(baralhosstrauxcominv[j],"9|9")!=0; j++)
1133         {
1134             //printf("%s ==
1135             %s\n",seqpadraoaux[i],baralhosstrauxcominvnovopadraonovo[j]);
1136             if(strcmp(seqpadraoaux[i],baralhosstrauxcominv[j])==0)
1137             {
1138                 strcpy(baralhosstrauxcominv[j],"-");
1139                 cont++;
1140
1141                 for(r=0; strcmp(baralhosstrauxcominv[r],"9|9")!=0;
1142 r++)
1143                 {
1144
1145 if(strcmp(inverterstr(seqpadraoaux[i],pecasstrinv),baralhosstrauxcominv[r])==0)
1146                 {
1147                     strcpy(baralhosstrauxcominv[r],"-");
1148
1149                 }
1150
1151             }
1152
1153         }
1154
1155     }
1156
1157 }
1158
1159
1160
1161 }
1162
1163
1164 if(k==1 || cont!=tamsubseq)
1165 {
1166
1167     printf("O padrao novo nao e valido!!\n\n");
1168     esvaziaseqstr(seqpadraoaux,LINSEQ);
1169     esvaziaseqstr(baralhosstrauxcominv,LINSEQ);
1170     esvaziaseqstr(baralhosstrauxcominvnovo,LINSEQ);
1171     esvaziaseqstr(baralhosstrauxcominvnovopadraonovo,LINSEQ);

```

```

1172
1173         cont=0;
1174         k=0;
1175         op2='2';
1176
1177     }
1178     else //se for valido substitui pelo padrao novo
1179     {
1180
1181         inverterstr(padrao,padraoinv);
1182         inverterstr(padraonovo,padraonovoinv);
1183
1184         //SE QUIERMOS SUBSTITUIR A STRING TODA FAZEMOS ESTE CICLO no
1185         fim acrescentamos ao array de seq de padroes
1186         for(i=0; strcmp(seqstr[i],"9|9")!=0; i++)
1187         {
1188             if(strcmp(seqstr[i],padrao)==0 ||
1189 strcmp(padraoinv,seqstr[i])==0)
1190             {
1191                 strcpy(padraototal,padraonovo);
1192             }
1193         }
1194         //envia o padrao e o paadrao novo normal e invertido
1195
1196 trocapadrao(seqstr,numdeseq,padrao,padraonovo,seqfpadrao,&sizeseqfpadrao);
1197 trocapadrao(seqstr,numdeseq,padrao,padraonovoinv,seqfpadrao,&sizeseqfpadrao);
1198 trocapadrao(seqstr,numdeseq,padraoinv,padraonovo,seqfpadrao,&sizeseqfpadrao);
1199 trocapadrao(seqstr,numdeseq,padraoinv,padraonovoinv,seqfpadrao,&sizeseqfpadrao);
1200
1201         strcpy(seqfpadrao[sizeseqfpadrao],padraototal);
1202
1203         cont=0;
1204         for(i=0; strcmp(seqfpadrao[i],"9|9")!=0; i++)
1205         {
1206             for(j=0; strcmp(seqfpadrao[j],"9|9")!=0; j++)
1207             {
1208                 if(i!=j)
1209                 {
1210                     if(strcmp(seqfpadrao[i],seqfpadrao[j])==0)
1211                     {
1212                         strcpy(seqfpadrao[i],"-");
1213                     }
1214                 }
1215             }
1216             cont++;
1217         }
1218         ordernarsequencias(seqfpadrao,cont);
1219         cont=tirartracosinvertidos(seqfpadrao,cont);
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236

```

```

1237
1238             printseqstr(seqfpadrao,0,cont);
1239
1240
1241             esvaziaseqstr(seqfpadrao,LINSEQ);
1242             sizeseqfpadrao=0;
1243
1244             }
1245
1246         }
1247
1248         break;
1249
1250     case '3':
1251
1252         save txt jogo(p,filename);
1253
1254     break;
1255
1256
1257     case '0':
1258
1259         mostrarjogosstr(baralhosstr,jogosaimprimir);
1260         op='2';
1261
1262         break;
1263
1264     default:
1265
1266         if(contseq==1)
1267         {
1268
1269             //numdeseq=seq(baralhosstr,seqstr,num);
1270             ordenarsequencias(seqstr,numdeseq);
1271             numdeseq=tirartracosinvertidos(seqstr,numdeseq);
1272             numdeseq=separarseqinvertidas(seqstr,numdeseq);
1273             printseqstr(seqstr,0,numdeseq);
1274             printf("\n");
1275
1276             mostrarjogosstr(baralhosstr,jogosaimprimir);
1277             //contseq=0;
1278
1279         }
1280
1281         else if(contseq==2)
1282         {
1283
1284             numdeseq=seqcomseqinicial(baralhosstr,seqstr,num,seqinicial);
1285             ordenarsequencias(seqstr,numdeseq);
1286             numdeseq=tirartracosinvertidos(seqstr,numdeseq);
1287             numdeseq=separarseqinvertidas(seqstr,numdeseq);
1288             tam=retiraseqinitrepetida(seqstr,numdeseq,seqinicial);
1289             ordenarsequencias(seqstr,numdeseq);
1290             printseqstr(seqstr,0,numdeseq-tam);
1291
1292             mostrarjogosstr(baralhosstr,jogosaimprimir);
1293             //contseq=0;
1294         }
1295         else
1296         {
1297
1298             mostrarjogosstr(baralhosstr,jogosaimprimir);
1299
1300         }
1301
1302         break;
1303
1304     }
1305
1306     break;
1307

```

```

1308         case '4':
1309
1310             printf("\n\n--MENU--\n\n");
1311             printf("1- Começar com uma sequencia\n");
1312             printf("2- Começar sem sequencia\n");
1313             printf("0- Voltar ao menu principal\n\n");
1314
1315             scanf(" %c",&op4);
1316             system("CLS");
1317
1318             switch(op4)
1319             {
1320
1321             case '1':
1322                 k=0;
1323                 mostrarjogosstr(baralhosstr,jogosaimprimir);
1324                 esvaziaseqstr(seqinicialpartida,LINSEQ);
1325                 cont=0;
1326                 printf("Insira a sequencia inicial:\n");
1327                 scanf("%s",seqinicial);
1328
1329                 if(verificasequencia(seqinicial)==1)
1330                 {
1331                     k=1;
1332                 }
1333
1334                 //copia as pecas iniciais para o baralho vazio
1335
1336                 for(i=0; i<LIN; i++)
1337                 {
1338                     strcpy(baralhosstrauxcominv[i],pecasstr[i]);
1339                 }
1340
1341                 //copio as respectivas invertidas para o baralho auxiliar
1342
1343                 for(j=0; j<LIN; j++)
1344                 {
1345                     if(pecasstr[j][0]!=pecasstr[j][2])
1346                     {
1347                         strcpy(baralhosstrauxcominv[i++],inverterstr(pecasstr[j],auxinv));
1348                     }
1349                 }
1350
1351                 for(i=0; strcmp(baralhosstrauxcominv[i],"9|9")!=0; i++)
1352                 {
1353                     for(j=0; strcmp(baralhosstr[j],"9|9")!=0; j++)
1354                     {
1355                         if(strcmp(baralhosstrauxcominv[i],baralhosstr[j])==0 ||
1356                            strcmp(baralhosstrauxcominv[i],inverterstr(baralhosstr[j],pecasstrinv))==0)
1357                         {
1358                             strcpy(baralhosstrauxcominv[i],"-");
1359                         }
1360                     }
1361                 }
1362             }
1363
1364             }
1365
1366             }
1367
1368             }
1369
1370             }
1371
1372             }
1373
1374             }
1375
1376             }

```



```

1377     }
1378
1379
1380     //partimos a sequencia inicial partida e guardamos no array auxiliar
1381     strcpy(seqinicialaux,seqinicial);
1382     tamsubseq=strtoque(seqinicialpartida,seqinicialaux,'-');
1383
1384     //verifico de o padrao novo possui pecas repetidas ou nao
1385
1386     for(i=0; strcmp(seqinicialpartida[i],"9|9")!=0; i++)
1387     {
1388         for(j=0; strcmp(baralhosstrauxcominv[j],"9|9")!=0; j++)
1389         {
1390             //printf("%s ==
1391             %s\n",seqpadraoaux[i],baralhosstrauxcominvnovopadraonovo[j]);
1392
1393             if(strcmp(seqinicialpartida[i],baralhosstrauxcominv[j])==0)
1394             {
1395                 strcpy(baralhosstrauxcominv[j],"-");
1396                 cont++;
1397
1398                 for(r=0; strcmp(baralhosstrauxcominv[r],"9|9")!=0; r++)
1399                 {
1400
1401
1402                 if(strcmp(inverterstr(seqinicialpartida[i],pecasstrinv),baralhosstrauxcominv[r])==0)
1403                 {
1404                     strcpy(baralhosstrauxcominv[r],"-");
1405
1406                 }
1407
1408             }
1409
1410         }
1411
1412     }
1413
1414 }
1415
1416 }
1417
1418 }
1419
1420
1421     if(cont!=tamsubseq || k==1)
1422     {
1423
1424         printf("A sequequencia nao e valida!!\n\n");
1425
1426         //volta a carregar o pecasstraux com as pecas todas e esvazia o
1427         array da sequencia inicial partida
1428         for(i=0; i<LIN; i++)
1429         {
1430             strcpy(pecasstraux[i],pecasstr[i]);
1431
1432         }
1433         k=0;
1434         op2='2';
1435
1436     }
1437     else
1438     {
1439
1440         for(i=0; i<LIN; i++)
1441         {
1442
1443

```

```

1444             strcpy(pecasstraux[i], pecasstr[i]);
1445
1446         }
1447
1448         numdeseq=jogoadois(baralhosstr, seqstr, num, seqinicial);
1449         ordenarsequencias(seqstr, numdeseq);
1450         numdeseq=tirartracosinvertidos(seqstr, numdeseq);
1451         numdeseq=separarseqinvertidas(seqstr, numdeseq);
1452         printseqstr(seqstr, 0, numdeseq);
1453
1454         mostrarjogosstr(baralhosstr, jogosaimprimir);
1455         contseq=2;
1456         op='3';
1457
1458     }
1459
1460     break;
1461
1462     case '2':
1463
1464         strcpy(seqinicial, "9|9");
1465
1466         numdeseq=jogoadois(baralhosstr, seqstr, num, seqinicial);
1467         ordenarsequencias(seqstr, numdeseq);
1468         numdeseq=tirartracosinvertidos(seqstr, numdeseq);
1469         numdeseq=separarseqinvertidas(seqstr, numdeseq);
1470         printseqstr(seqstr, 0, numdeseq);
1471
1472         mostrarjogosstr(baralhosstr, jogosaimprimir);
1473         contseq=3;
1474         op='3';
1475
1476         break;
1477
1478     case '0':
1479
1480         op='4';
1481
1482         break;
1483
1484     default:
1485
1486         printf("OPCAO INVALIDA\n");
1487
1488         break;
1489
1490     }
1491
1492     break;
1493
1494 }
1495
1496 }
1497
1498 }
1499     return 0;
1500
1501 }

```

int modificar (PECASINIT * , char pecass[][COLSTR], int)

void mostrarjogosint (int baralhosint[][COL], int)

Função mostrarjogosint.

Funcao que recebe o numero de jogos a imprimir em inteiros e os jogos dos jogadores e imprime separadamente os jogos dos jogadores

Parâmetros:

<i>int</i>	baralhosi[][COL] array de inteiros que guarda os jogos dos jogadores
<i>int</i>	njogos numero de jogos a imprimir

```

3194 {
3200     switch(njogos)
3201     {
3202
3203         case 7:
3204             printf("JOGO 1:\n");
3205             printpecasint(baralhosi,7,2,0);
3206             printf("\n");
3207
3208             break;
3209
3210         case 14:
3211             printf("JOGO 1:\n");
3212             printpecasint(baralhosi,7,2,0);
3213             printf("JOGO 2:\n");
3214             printpecasint(baralhosi,14,2,7);
3215             printf("\n");
3216
3217             break;
3218
3219         case 21:
3220             printf("JOGO 1:\n");
3221             printpecasint(baralhosi,7,2,0);
3222             printf("JOGO 2:\n");
3223             printpecasint(baralhosi,14,2,7);
3224             printf("JOGO 3:\n");
3225             printpecasint(baralhosi,21,2,14);
3226             printf("\n");
3227
3228             break;
3229
3230         case 28:
3231             printf("JOGO 1:\n");
3232             printpecasint(baralhosi,7,2,0);
3233             printf("JOGO 2:\n");
3234             printpecasint(baralhosi,14,2,7);
3235             printf("JOGO 3:\n");
3236             printpecasint(baralhosi,21,2,14);
3237             printf("JOGO 4:\n");
3238             printpecasint(baralhosi,28,2,21);
3239             printf("\n");
3240
3241             break;
3242     }
3243 }
3244 }
```

void mostrarjogosintstruct (PECASINIT , int)

Função mostrarjogosstr.

Funcao que recebe o numero de jogos a imprimir em string e os jogos dos jogadores e imprime separadamente os jogos dos jogadores

Parâmetros:

<i>PECASINIT</i>	p estrutura deste tipo
<i>int</i>	njogos numero de jogos a imprimir

```

3138 {
```

```

3144     switch(njogos)
3145     {
3146
3147         case 7:
3148             printf("JOGO 1:\n");
3149             printpecasintstruct(p,0,7);
3150             printf("\n");
3151
3152             break;
3153
3154         case 14:
3155             printf("JOGO 1:\n");
3156             printpecasintstruct(p,0,7);
3157             printf("JOGO 2:\n");
3158             printpecasintstruct(p,7,14);
3159             printf("\n");
3160
3161             break;
3162
3163         case 21:
3164             printf("JOGO 1:\n");
3165             printpecasintstruct(p,0,7);
3166             printf("JOGO 2:\n");
3167             printpecasintstruct(p,7,14);
3168             printf("JOGO 3:\n");
3169             printpecasintstruct(p,14,21);
3170             printf("\n");
3171
3172             break;
3173
3174         case 28:
3175             printf("JOGO 1:\n");
3176             printpecasintstruct(p,0,7);
3177             printf("JOGO 2:\n");
3178             printpecasintstruct(p,7,14);
3179             printf("JOGO 3:\n");
3180             printpecasintstruct(p,14,21);
3181             printf("JOGO 4:\n");
3182             printpecasintstruct(p,21,28);
3183             printf("\n");
3184
3185             break;
3186     }
3187 }
3188
3189 }

```

void mostrarjogosstr (char *baralhoss*[][COLSTR], int)

Funcao que recebe o numero de jogos a imprimir em string e os jogos dos jogadores e imprime separadamente os jogos dos jogadores

Parâmetros:

<i>char</i>	baralhoss[][COLSTR] array de strings que guarda os jogos dos jogadores
<i>int</i>	njogos numero de jogos a imprimir

```

3079 {
3080
3087     switch(njogos)
3088     {
3089
3090         case 7:
3091             printf("JOGO 1:\n");
3092             printpecasstr(baralhoss,0,7);
3093             printf("\n");
3094
3095             break;
3096
3097         case 14:

```

```

3098     printf("JOGO 1:\n");
3099     printpecasstr(baralhoss,0,7);
3100     printf("JOGO 2:\n");
3101     printpecasstr(baralhoss,7,14);
3102     printf("\n");
3103
3104     break;
3105
3106     case 21:
3107         printf("JOGO 1:\n");
3108         printpecasstr(baralhoss,0,7);
3109         printf("JOGO 2:\n");
3110         printpecasstr(baralhoss,7,14);
3111         printf("JOGO 3:\n");
3112         printpecasstr(baralhoss,14,21);
3113         printf("\n");
3114
3115         break;
3116
3117     case 28:
3118         printf("JOGO 1:\n");
3119         printpecasstr(baralhoss,0,7);
3120         printf("JOGO 2:\n");
3121         printpecasstr(baralhoss,7,14);
3122         printf("JOGO 3:\n");
3123         printpecasstr(baralhoss,14,21);
3124         printf("JOGO 4:\n");
3125         printpecasstr(baralhoss,21,28);
3126         printf("\n");
3127
3128         break;
3129
3130     }
3131
3132 }

```

void mostrarjogosstrstruct (PECASINIT , int)

Função mostrarjogosstrstruct.

Funcao que recebe o numero de jogos a imprimir em string e os jogos dos jogadores e imprime separadamente os jogos dos jogadores

Parâmetros:

<i>PECASINIT</i>	p estrutura deste tipo
<i>int</i>	njogos numero de jogos a imprimir

```

3025 {
3031     switch(njogos)
3032     {
3033
3034         case 7:
3035             printf("JOGO 1:\n");
3036             printpecasstrstruct(p,0,14);
3037             printf("\n");
3038
3039             break;
3040
3041         case 14:
3042             printf("JOGO 1:\n");
3043             printpecasstrstruct(p,0,7);
3044             printf("JOGO 2:\n");
3045             printpecasstrstruct(p,7,14);
3046             printf("\n");
3047
3048             break;
3049

```

```

3050     case 21:
3051         printf("JOGO 1:\n");
3052         printpecasstrstruct(p,0,7);
3053         printf("JOGO 2:\n");
3054         printpecasstrstruct(p,7,14);
3055         printf("JOGO 3:\n");
3056         printpecasstrstruct(p,14,21);
3057         printf("\n");
3058
3059         break;
3060
3061     case 28:
3062         printf("JOGO 1:\n");
3063         printpecasstrstruct(p,0,7);
3064         printf("JOGO 2:\n");
3065         printpecasstrstruct(p,7,14);
3066         printf("JOGO 3:\n");
3067         printpecasstrstruct(p,14,21);
3068         printf("JOGO 4:\n");
3069         printpecasstrstruct(p,21,28);
3070         printf("\n");
3071
3072         break;
3073
3074     }
3075
3076 }

```

void ordernarmatrizinteiros (int m[][2], int)

Função ordernarmatrizinteiros.

esta função recebe uma matriz e ordena crescente ou decrescente

Parâmetros:

<i>int</i>	m[][2] matriz a ordenar
<i>int</i>	size tamanho da matriz

```

4387 {
4388
4394     //FAÇO A ORDENAÇÃO DE UMA MATRIZ
4395     int aux=0;
4396     int aux2=0;
4397     int j=0;
4398     int i=0;
4399
4400     for(j=0; j<size; j++)
4401     {
4402
4403         for(i=0; i<size; i++)
4404         {
4405             //TROCANDO O SINAL FAÇO ORDEM CRESCENTE OU DECRESCENTE
4406             if(m[j][0]<m[i][0])
4407             {
4408
4409                 aux=m[j][0];
4410                 aux2=m[j][1];
4411
4412                 m[j][0]=m[i][0];
4413                 m[j][1]=m[i][1];
4414
4415                 m[i][0]=aux;
4416                 m[i][1]=aux2;
4417
4418             }
4419
4420         }

```

```

4421
4422     }
4423
4424 }

```

void ordenarsequencias (char seqf[][COLSEQ], int)

Função ordenarsequencias.

esta função ordena as sequenciass por ordem decrescente

Parâmetros:

<i>seqf</i> [][COLSEQ]	array de strings das sequencias finais a ordenar
<i>int</i>	size numero de sequencias finais

```

4431 {
4432     //esta função ordena as sequenciass por ordem decrescente
4433     char seqss[LINSEQ][COLSEQ];
4434     int tamanhos[LINSEQ][2];
4435     int i=0;
4436     int j=0;
4437     int aux=0;
4438     int aux2=0;
4439
4440     //esvazio a sequencia de strings
4441     esvaziaseqstr(seqss,size);
4442
4443     //guardo o tamanho da seq na primeira posição de uma matriz e na segunda posição
4444     qual a posição da sequencia do array de sequencias
4445     for(i=0; i<size; i++)
4446     {
4447         tamanhos[i][0]=strlen(seqf[i]);
4448         tamanhos[i][1]=i;
4449     }
4450
4451     //faço o selection sort para ordenar decrescentemente
4452     for(j=0; j<size; j++)
4453     {
4454         for(i=0; i<size; i++)
4455         {
4456             if(tamanhos[j][0]>tamanhos[i][0])
4457             {
4458                 aux=tamanhos[j][0];
4459                 aux2=tamanhos[j][1];
4460
4461                 tamanhos[j][0]=tamanhos[i][0];
4462                 tamanhos[j][1]=tamanhos[i][1];
4463
4464                 tamanhos[i][0]=aux;
4465                 tamanhos[i][1]=aux2;
4466             }
4467         }
4468     }
4469
4470     /*for(i=0;i<size;i++){
4471         for(j=0;j<2;j++){

```

```

4487
4488         printf("%d|", tamanhos[i][j]);
4489     }
4490
4491     printf("\n");
4492
4493     */
4494 }*/
4495
4496 for(i=0; i<size; i++)
4497 {
4498     strcpy(seqss[i], seqf[tamanhos[i][1]]);
4499 }
4500
4501 esvaziaseqstr(seqf, size);
4502
4503 for(i=0; i<size; i++)
4504 {
4505     strcpy(seqf[i], seqss[i]);
4506 }
4507
4508 //printseqstr(seqf, 0, size);
4509
4510 }
4511 }

```

void ordenarsequenciasstruct (PECASINIT *)

Função ordenarsequencias struct.

esta função ordena as sequenciass por ordem decrescente

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo struct
------------------	-----------------------------

```

4521 {
4522     //esta função ordena as sequenciass por ordem decrescente
4523     int i=0;
4524     int j=0;
4525     int aux=0;
4526     int aux2=0;
4527     int nseq=0;
4528
4529     //esvazio a sequencia de strings
4530
4531     nseq=p->nseqss;
4532
4533     for(i=0; i<nseq; i++)
4534     {
4535         remove seqss(p);
4536     }
4537
4538     //guardo o tamanho da seq na primeira posição de uma matriz e na segunda posição
4539     guardo qual a posição da sequencia do array de sequencias
4540     for(i=0; i<p->nseqf; i++)
4541     {
4542         inserir_ordenaseq(p, strlen((p->seqf+i)->seqstr), i);
4543     }
4544 }

```



```

4552     }
4553
4554
4555     //faço o selection sort para ordenar decrescentemente
4556     for(i=0; i<p->nseqf; i++)
4557     {
4558
4559         for(j=0; j<p->nseqf; j++)
4560         {
4561             if ( (p->ordenaseq+j)->tamanho<(p->ordenaseq+i)->tamanho)
4562             {
4563
4564                 aux=(p->ordenaseq+j)->tamanho;
4565                 aux2=(p->ordenaseq+j)->indice;
4566
4567                 (p->ordenaseq+j)->tamanho=(p->ordenaseq+i)->tamanho;
4568                 (p->ordenaseq+j)->indice=(p->ordenaseq+i)->indice;
4569
4570                 (p->ordenaseq+i)->tamanho=aux;
4571                 (p->ordenaseq+i)->indice=aux2;
4572
4573             }
4574
4575         }
4576     }
4577
4578
4579     for(i=0;i<p->nseqf;i++)
4580     {
4581
4582         inserir_seqss(p, (p->seqf+((p->ordenaseq+i)->indice))->seqstr);
4583
4584     }
4585
4586     nseq=p->nseqf;
4587
4588     for(i=0;i<nseq;i++)
4589     {
4590
4591         remove_seqf(p);
4592
4593     }
4594
4595     for(i=0;i<p->nseqss;i++)
4596     {
4597
4598         inserir_seqf(p, (p->seqss+i)->seqstr);
4599
4600     }
4601
4602 }
4603 }

```

int preenchebaralhos (char pecass[][COLSTR], char baralhoss[][COLSTR], int)

int preenchebaralhosstruct (char pecass[][COLSTR], PECASINIT *, int)

função para inserir os baralhos dos jogadores manualmente

Parâmetros:

<i>int</i>	pecass[][COLSTR] array que possui as pecas todas do jogo
<i>PECASINIT</i>	*p estrutura do tipo struct
<i>int</i>	n numero de jogos a preencher manualmente

```

2203 {
2204
2211     //função para inserir os baralhos dos jogadores
2212     int i=0;

```

```

2213     int j=0;
2214     int jogos=0;
2215     char aux[4];
2216     int existe=0;
2217     int elimina=0;
2218     int tam=0;
2219     int cont=0;
2220     int begin=0;
2221     char pecasaux[LIN][COLSTR];
2222
2223     tam=LIN;
2224     jogos=n*7;
2225     //guardo as pecas num array auxiliar
2226     for(i=0; i<tam; i++)
2227     {
2228
2229         strcpy(pecasaux[i],pecass[i]);
2230     }
2231
2232
2233     printf("SELECIONE %d JOGOS DE (7 PECAS) DA LISTA: (ex: 5|2 )\n\n",n);
2234     //de 0 até ao numero de jogos a inserir., inserem-se peças peças
2235     while(j<n)
2236     {
2237
2238         for(i=begin; i<jogos; i++)
2239         {
2240
2241             printpecasstr(pecasaux,0,tam);
2242             scanf("%s",aux);
2243             system("CLS");
2244
2245             //Verifica as pecas todas
2246             for(existe=0; existe<tam; existe++)
2247             {
2248
2249                 //Se a peca existe nas pecas a escolher então copia-se para o baralho
da pessoa
2250                 if(strcmp(pecasaux[existe],aux)==0)
2251                 {
2252
2253
2254                     inserir_pecas(p,pecasaux[existe]);
2255
2256                     //strcpy(baralhoss[i],pecasaux[existe]);
2257
2258                     //Coloco todas as peças para cima
2259                     for(elimina=existe; elimina<(tam-1); elimina++)
2260                     {
2261
2262                         strcpy(pecasaux[elimina],pecasaux[elimina+1]);
2263
2264                     }
2265
2266                     cont++;
2267
2268                 }
2269             }
2270             //se guarda uma peca no baralho da proxima vez já não pode escolher essa
peça
2271             if(cont==1)
2272             {
2273                 tam--;
2274                 cont=0;
2275
2276             }
2277             else
2278             {
2279                 //Se a peça não existe escreve a mensagem que não é válida
2280                 printf("NAO E VALIDO!\n");
2281                 printf("SELECIONE DA LISTA: (ex: 5|2 )\n\n");

```

```

2282         i--;
2283     }
2284 }
2285     begin=begin+7;
2286     jogos=jogos+7;
2287     j++;
2288 }
2289
2290     return n*7;
2291
2292 }

```

void printpecasint (int *pecas*[][COL], int , int , int)

Função printpecasint.

função que imprime as pecas inteiras

Parâmetros:

<i>int</i>	pecasi[][COL] pecas total inteiras
<i>int</i>	l linhas da matriz
<i>int</i>	c colunas da matriz
<i>int</i>	inicio variavel que contem o numero a partir do qual queremos imprimir

```

2850 {
2851
2860     int i=0;
2861     int j=0;
2862
2863     for(i=inicio; i<l; i++)
2864     {
2865         for(j=0; j<c; j++)
2866         {
2867
2868             printf("%d",pecasi[i][j]);
2869
2870             if(j==0)
2871             {
2872
2873                 printf("|");
2874
2875             }
2876         }
2877         printf("\n");
2878     }
2879
2880 }

```

void printpecasstr (char *pecass*[][COLSTR], int , int)

Função printpecasstr.

função para imprimir peças string

Parâmetros:

<i>int</i>	inicio desde quando é que queremos imprimir
<i>int</i>	fim até onde queremos imprimir

```

2953 {
2959     int i=0;
2960
2961     for(i=inicio; i<fim; i++)
2962     {

```

```

2963
2964     printf("%s\n",pecass[i]);
2965
2966 }
2967
2968 }

```

void printseqstr (char seqss[][COLSEQ], int , int)

Função printseqstr.

função para imprimir sequencias

Parâmetros:

<i>int</i>	inicio desde quando é que queremos imprimir
<i>int</i>	fim até onde queremos imprimir

```

2974 {
2981     //funcao para imprimir sequencias de strings
2982     int i=0;
2983
2984     for(i=inicio; i<fim; i++)
2985     {
2986
2987         printf("i=[%d] %s\n",i,seqss[i]);
2988
2989     }
2990
2991 }

```

void printseqstrstruct (PECASINIT)

Função printpecasstruct.

função para imprimir sequencias

Parâmetros:

<i>int</i>	inicio desde quando é que queremos imprimir
<i>int</i>	fim até onde queremos imprimir
<i>PECASINIT</i>	p estrutura deste tipo

```

2996 {
3004     //funcao para imprimir sequencias de strings
3005     int i=0;
3006
3007     for(i=0; i<p.nseqf; i++)
3008     {
3009         if((p.seqf+i)->seqstr!=NULL)
3010         {
3011
3012             printf("i=[%d] %s\n",i,(p.seqf+i)->seqstr);
3013
3014         }
3015
3016     }
3017
3018
3019 }

```

int procsbseq (char seqf[][COLSEQ], int size, char subs[])

Função procsubseq.

função para procurar uma substring numa string

Parâmetros:

<i>char</i>	seqf[][COLSEQ] array final de strings
<i>int</i>	size numero da linha do array de strings final que vamos querer procurar uma substring
<i>char</i>	subs[] sub string a procurar

Retorna:

retorna a posicao em que encontrou a substring

```
4272 {
4281 //esta função procura sequencias de peças noutras sequencias, ou peças em
sequencias
4282 int i=0;
4283 int k=0;
4284 int cont=0;
4285 int pos=0;
4286 int x=0;
4287 int v[LINSEQ];
4288 int tamstr1=0;
4289 int tamstr2=0;
4290 char arrayseq[LINSEQ][COLSEQ];
4291 char arraysub[LINSEQ][COLSEQ];
4292
4293 //o size neste caso é a linha do array de strings
4294
4295 //faço o strtok da sequencias de peças
4296 tamstr1=strtoque(arrayseq,seqf[size],'-');
4297 //faço o strtok da sequencia a procurar
4298 tamstr2=strtoque(arraysub,subs,'-');
4299
4300 for (i=0; i<tamstr1; i++)
4301 {
4302 //enquanto não estiver no fim da sequencia e houver igualdades entre a sequencia
e a sub sequencia vou comparando a proxima peça da subsequencia com a peça da sequencia
4303 if(strcmp(arrayseq[i],arraysub[k])==0)
4304 {
4305 if(k==0)
4306 {
4307 //guardo a posição em que encontrei a primeira igualdade entre a
subsequencia e a sequencia
4308 pos=i;
4309 }
4310 cont++;
4311 k++;
4312
4313
4314 //se a sub sequencia e a sequencia são iguais então guardo a posição num
array de inteiros e procuro outra igualdade
4315 //se é que existe entre a subsequencia e a sequencia de peças
4316
4317 if(cont==k && cont==tamstr2)
4318 {
4319 v[x]=pos;
4320 x++;
4321 k=0;
4322 cont=0;
4323 pos=0;
4324
4325 }
4326 }
4327 else
4328 {
4329
4330 k=0;
```

```

4331         cont=0;
4332         pos=0;
4333
4334     }
4335 }
4336
4337 return x;
4338 //se quisermos imprimir as varias ocorrencias
4339
4340 //imprime as posições nas quais encontrou o inicio da igualdade entre as
subsequencias e as sequencias
4341 for(i=0; i<x; i++)
4342 {
4343
4344     printf("%d\n",v[i]);
4345
4346 }
4347
4348 }

```

int procsubseq_ausar (char seqf[][COLSEQ], int , char subs[])

Função procsubseq_ausar.

função para procurar uma substring numa string

Parâmetros:

<i>char</i>	seqf[][COLSEQ] array final de strings
<i>int</i>	size numero da linha do array de strings final que vamos querer procurar uma substring
<i>char</i>	subs[] sub string a procurar

Retorna:

retorna a posicao em que encontrou a substring

```

4178 {
4179
4187 //esta função procura sequencias de peças noutras sequencias, ou peças em
sequencias
4188 int i=0;
4189 int k=0;
4190 int cont=0;
4191 int pos=0;
4192 int x=0;
4193 int v[LINSEQ];
4194 int tamstr1=0;
4195 int tamstr2=0;
4196 char arrayseq[LINSEQ][COLSEQ];
4197 char arraysub[LINSEQ][COLSEQ];
4198
4199 //o size neste caso é a linha do array de strings
4200
4201 //faço o strtok da sequencias de peças
4202 tamstr1=strtoque(arrayseq,seqf[size],'-');
4203 //faço o strtok da sequencia a procurar
4204 tamstr2=strtoque(arraysub,subs,'-');
4205
4206 for (i=0; i<tamstr1; i++)
4207 {
4208     //enquanto não estiver no fim da sequencia e houver igualdades entre a sequencia
e a sub sequencia vou comparando a proxima peça da subsequencia com a peça da sequencia
4209     if(strcmp(arrayseq[i],arraysub[k])==0)
4210     {
4211         if(k==0)
4212         {
4213             //guardo a posição em que encontrei a primeira igualdade entre a
subsequencia e a sequencia

```

```

4214         pos=i;
4215     }
4216     cont++;
4217     k++;
4218
4219
4220     //se a sub sequencia e a sequencia são iguais então guardo a posição num
array de inteiros e procuro outra igualdade
4221     //se é que existe entre a subsequencia e a sequencia de peças
4222
4223     if(cont==k && cont==tamstr2)
4224     {
4225         v[x]=pos;
4226         x++;
4227         k=0;
4228         cont=0;
4229         pos=0;
4230
4231     }
4232 }
4233 else
4234 {
4235
4236     k=0;
4237     cont=0;
4238     pos=0;
4239
4240 }
4241 }
4242 if(x==0)
4243 {
4244
4245     return -1;
4246
4247 }
4248 else
4249 {
4250
4251     return v[0];
4252
4253 }
4254
4255 //se quisermos imprimir as varias ocorrencias
4256
4257 //imprime as posições nas quais encontrou o inicio da igualdade entre as
subsequencias e as sequencias
4258 /*for(i=0;i<x;i++)
4259 {
4260
4261     printf("%d\n",v[i]);
4262
4263 }*/
4264
4265 }

```

int procsubseq_trocapadiao (char seqf[][COLSEQ], int size, char subs[], int)

void remove_pecas (PECASINIT * p, char remove[])

Remover pecas dos jogos.

função para remover pecas nas listas ligadas

Parâmetros:

<i>PECASINIT</i>	*p estrutura do tipo PECASINIT
------------------	--------------------------------

<i>int</i>	n numero de jogos a preencher manualmente
3598 {	
3599	PECA *paux=NULL;
3606	PECA *pant=NULL;
3607	
3608	
3609	paux=p->pfirst;
3610	
3611	//cabeca
3612	if(strcmp(paux->str,remove)==0)
3613	{
3614	
3615	p->pfirst=paux->pnext;
3616	p->npecas--;
3617	return;
3618	
3619	}
3620	
3621	//cauda
3622	paux=p->pfirst;
3623	while(paux->pnext!=NULL)
3624	{
3625	
3626	pant=paux;
3627	paux=paux->pnext;
3628	
3629	}
3630	
3631	if(strcmp(paux->str,remove)==0)
3632	{
3633	
3634	pant->pnext=NULL;
3635	p->npecas--;
3636	return;
3637	
3638	}
3639	//meio
3640	paux=p->pfirst;
3641	while(paux->pnext!=NULL)
3642	{
3643	
3644	pant=paux;
3645	paux=paux->pnext;
3646	
3647	if(strcmp(paux->str,remove)==0)
3648	{
3649	
3650	pant->pnext=paux->pnext;
3651	p->npecas--;
3652	return;
3653	}
3654	
3655	}
3656	
3657 }	

void remove_seqf (PECASINIT *)

3557 {	
3558	
3559	SEQ *paux=NULL;
3560	paux=p->seqf;
3561	
3562	//cabeca
3563	p->seqf=paux+1;
3564	p->nseqf--;
3565	return;
3566	
3567 }	

void remove_seqss (PECASINIT *)

```

3570 {
3571
3572     SEQ *paux=NULL;
3573     paux=p->seqss;
3574
3575     //cabeca
3576     p->seqss=paux+1;
3577     p->nseqss--;
3578     return;
3579
3580 }
```

void remove_seqssaux (PECASINIT *)

```

3583 {
3584
3585     SEQ *paux=NULL;
3586     paux=p->seqssaux;
3587
3588     //cabeca
3589     p->seqssaux=paux+1;
3590     p->nseqssaux--;
3591     return;
3592
3593 }
```

int remover (PECASINIT *, char pecass[][COLSTR], int)

Função remover.

esta funcao remove as pecas inseridas

Parâmetros:

<i>char</i>	baralhoss[][COLSTR] baralhos do jogador em string
<i>char</i>	pecass[][COLSTR] pecas todas do jogo em strings
<i>int</i>	num numero de jogos a converter

Retorna:

retorna o numero de jogos a remover

```

3436 {
3444     int i=0;
3445     int n=num;
3446     char aux[4];
3447     char remove[4];
3448     int pecasaremove=0;
3449     n=n*7;
3450
3451     PECA *pfind=NULL;
3452
3453
3454     do
3455     {
3456
3457         mostrarjogosstrstruct(*p,n);
3458         printf("\nQUANTAS PECAS PRETENDE REMOVER?\n");
3459         scanf("%d",&pecasaremove);
3460         system("CLS");
3461
3462         if(pecasaremove>n)
3463         {
3464
3465             printf("NAO PODE REMOVER ESSE NUMERO DE PECAS!!!\n\n");
3466             mostrarjogosstrstruct(*p,n);
3467 }
```

```

3468     }
3469 }
3470 while(pecasaremove>n);
3471
3472 system("CLS");
3473
3474 for(i=0; i<pecasaremove; i++)
3475 {
3476     mostrarjogosstrstruct(*p,n);
3477     printf("QUAL A PECA A REMOVER:\n");
3478     scanf("%s",aux);
3479     system("CLS");
3480
3481     pfind=find_pecas_baralho(p,aux);
3482
3483     if(pfind==NULL)
3484     {
3485         printf("A PECA QUE ESCOLHEU NAO E VALIDA!\n");
3486
3487         if(i==0)
3488         {
3489             i=0;
3490         }
3491         else
3492         {
3493             i--;
3494         }
3495     }
3496     else
3497     {
3498         strcpy(remove,pfind->str);
3499         remove_pecas(p,remove);
3500     }
3501 }
3502
3503 printf("INSIRA AS %d PECAS NOVAS\n",pecasaremove);
3504 for(i=0; i<pecasaremove; i++)
3505 {
3506     mostrarjogosstrstruct(*p,n);
3507     scanf("%s",aux);
3508     pfind=find_pecas_baralho(p,aux);
3509
3510     if(pfind!=NULL)
3511     {
3512         printf("A PECA QUE ESCOLHEU NAO E VALIDA!\n");
3513
3514         if(i==0)
3515         {
3516             i=0;
3517         }
3518         else
3519         {
3520             i--;
3521         }
3522     }
3523 }
3524
3525 }
3526
3527 }
3528
3529 }
3530
3531 }
3532
3533 }
3534
3535 }
3536
3537 }
3538

```

```

3539         }
3540
3541         }else
3542         {
3543
3544             inserir_pecas(p,aux);
3545
3546         }
3547
3548
3549
3550     }
3551
3552     return n;
3553
3554 }

```

int retiraseqinitrepetida (char seqf[][COLSEQ], int size, char seqinit[])

Função retiraseqinitrepetida.

Nesta funcao verifico se encontro a sequencia inicial ou a inicial invertida repetida na mesma sequencia para poder eliminá-la

Parâmetros:

<i>char</i>	seqf[][COLSEQ] array final onde vou guardar as sequencias todas possiveis
<i>int</i>	size numero de jogos a retirar sequencias com sequencias iniciais repetidas
<i>char</i>	seqinit[] sequencia inicial

Retorna:

retorna o numero de sequencias que encontrou

```

5435 {
5436
5445     int i=0;
5446     int cont=0;
5447     char seqinitinv[LINSEQ];
5448
5449     for(i=0; seqinit[i]!='\0'; i++)
5450     {
5451
5452         seqinitinv[strlen(seqinit)-1-i]=seqinit[i];
5453
5454     }
5455
5456     //aqui verifico se encontro a sequencia inicial ou a inicial invertida repetida
na mesma sequencia para poder eliminá-la
5457     //visto que considero a sequencia inicial ou a sequencia invertida como se fosse
uma peça tomando apenas em consideração
5458     //as extremidades
5459
5460     for(i=0; i<size; i++)
5461     {
5462
5463         if(procsubseq(seqf,i,seqinit)>1)
5464         {
5465
5466             strcpy(seqf[i],"9|9");
5467             cont++;
5468
5469         }
5470         if(procsubseq(seqf,i,seqinitinv)>1)
5471         {
5472
5473             strcpy(seqf[i],"9|9");
5474             cont++;

```

```

5475
5476     }
5477
5478
5479     }
5480
5481     return cont;
5482
5483 }

```

void save_jogo_bin (PECASINIT , char [])

Função save_jogo_bin.

função para guardar jogos

Parâmetros:

<i>char</i>	fname[] nome do ficheiro a guardar
<i>PECASINIT</i>	p estrutura deste tipo

```

1506 {
1507
1514     PECA *paux=NULL;
1515     SEQ *paux2=NULL;
1516
1517     paux=p.pfirst;
1518     paux2=p.seqf;
1519
1520     FILE *fp=NULL;
1521
1522     int size=0;
1523     int i=0;
1524
1525     if ( (fp=fopen(fname,"wb")) !=NULL)
1526     {
1527
1528         fwrite(&p.npecas,sizeof(int),1,fp);
1529         //gravar o numero de pecas
1530
1531         while(paux!=NULL)
1532         {
1533
1534             size=strlen((p.pfirst->str))+1;
1535             fwrite(&size,sizeof(int),1,fp);
1536             fwrite(paux->str,sizeof(char),size,fp);
1537             paux=paux->pnext;
1538
1539         }
1540
1541
1542
1543
1544         for(i=0;i<p.nseqf;i++)
1545         {
1546
1547             size=strlen((p.seqf+i->seqstr))+1;
1548             fwrite(&size,sizeof(int),1,fp);
1549             fwrite((paux2+i->seqstr,sizeof(char),size,fp);
1550
1551         }
1552     fclose(fp);
1553     }
1554
1555
1556 }

```

void save_txt_jogo (PECASINIT , char [])

Função save_jogo_txt.

função para guardar jogos txt

Parâmetros:

<i>char</i>	fname[] nome do ficheiro a guardar
<i>PECASINIT</i>	p estrutura deste tipo

```
1659 {
1660
1668     FILE *fp=NULL;
1669     int i=0;
1670
1671     if ((fp = fopen(fname, "w")) == NULL)
1672     {
1673
1674         printf("save_txt_jogo(): Erro abrir ficheiro %s\n",fname);
1675         return;
1676     }
1677
1678     int j=0;
1680     int fim=7;
1681
1682     PECA *paux=NULL;
1683     paux=p.pfirst;
1684
1685     //guarda os jogos no ficheiro
1686     fprintf(fp,"JOGOS\n\n");
1688     for(i=0;i<p.npecas/7;i++)
1689     {
1690
1691         fprintf(fp,"JOGO %d\n",i+1);
1692
1693         while(paux!=NULL && j<fim)
1694         {
1695
1696             fprintf(fp,"%s\n",paux->str);
1697             paux=paux->pnext;
1698             j++;
1699
1700         }
1701         j=j+7;
1702         fim=fim+7;
1703     }
1704
1705     //guarda as sequencias
1706     fprintf(fp,"\n");
1707
1708     for(i=0; i<p.nseqf; i++)
1709     {
1710         if ((p.seqf+i)->seqstr!=NULL)
1711         {
1712
1713             fprintf(fp,"i=[%d] %s\n",i, (p.seqf+i)->seqstr);
1714
1715         }
1716     }
1717
1718
1719 }
1720
1721
1722
1723     fclose(fp);
```

```
1724
1725 }
```

int separarseqinvertidas (char seqf[][COLSEQ], int)

Função separarseqinvertidas.

função que distingue as sequencias iguais lidas da esquerda para a direita às lidas da direita para a esquerda e chama a funcao elimina rep para retirar as que sao iguais escritas ao contrário

Parâmetros:

<i>char</i>	seqf[][COLSEQ] array de sequencias
<i>int</i>	numdeseq guarda o numero de sequencias

Retorna:

retorna o numero de sequencias

dou indices às sequencias invertidas e às sequencias normais para dps eliminar os duplicados

```
2557 {
2558
2566     int i=0;
2567     int j=0;
2568     int v[LINSEQ];
2569     char auxseqinvertidas[LINSEQ][COLSEQ];
2570     char seqaux[LINSEQ][COLSEQ];
2571     int *vseqs=NULL;
2572
2573
2574     //inverte as sequencias
2575     esvaziaseqstr(auxseqinvertidas,LINSEQ);
2576     esvaziaseqstr(seqaux,LINSEQ);
2577
2578
2579     for(i=0; i<numdeseq; i++)
2580     {
2581
2582         while(seqf[i][j]!='\0')
2583         {
2584             auxseqinvertidas[i][strlen(seqf[i])-1-j]=seqf[i][j];
2585             j++;
2586         }
2587         auxseqinvertidas[i][strlen(seqf[i])]='\0'; // mete um '\0' no fim de string
2588         j=0;
2589         v[i]=-1;
2590
2591     }
2592
2593
2596     for(i=0; i<numdeseq; i++)
2597     {
2598
2599         for(j=0; j<numdeseq; j++)
2600         {
2601
2602             if(strcmp(seqf[i],seqf[j])==0 && v[j] == -1)
2603             {
2604
2605                 v[j]=i;
2606
2607             }
2608
2609         }
2610
2611
2612         for(j=0; j<numdeseq; j++)
2613         {
```

```

2614
2615         if(strcmp(auxseqinvertidas[i],seqf[j])==0 && v[j] == -1)
2616         {
2617             v[j]=i;
2618
2619         }
2620     }
2621 }
2622
2623
2624 }
2625
2626 //eliminar repetidos
2627
2628 //vseqs=eliminarrep(v,&numdeseq);
2629
2630
2631 for(i=0; i<numdeseq; i++)
2632 {
2633
2634     strcpy(seqaux[i],seqf[*vseqs+i]);
2635 }
2636
2637 free(vseqs);
2638
2639 esvaziaseqstr(seqf,LINSEQ);
2640
2641 for(i=0; i<numdeseq; i++)
2642 {
2643
2644     strcpy(seqf[i],seqaux[i]);
2645 }
2646 }
2647
2648 return numdeseq;
2649
2650 }

```

int separarseqinvertidasstruct (PECASINIT *)

Função separarseqinvertidas struct.

função que distingue as sequencias iguais lidas da esquerda para a direita às lidas da direita para a esquerda e chama a funcao elimina rep para retirar as que sao iguais escritas ao contrário

Parâmetros:

<i>PECASINIT</i>	*p estrutura deste tipo
------------------	-------------------------

Retorna:

retorna o numero de sequencias

dou indices às sequencias invertidas e às sequencias normais para dps eliminar os duplicados

```

2436 {
2437
2444     int i=0;
2445     int j=0;
2446     int v[LINSEQ];
2447     int *vseqs=NULL;
2448     int nseq=0;
2449
2450     //esvaziar auxiliar
2451     nseq=p->nseqss;
2452
2453     for(i=0; i<nseq; i++)
2454     {
2455

```

```

2456         remove_seqss(p);
2457     }
2458 }
2459 nseq=p->nseqssaux;
2460 for(i=0;i<nseq;i++)
2461 {
2462     remove_seqssaux(p);
2463 }
2464 //inverte as sequencias
2465 for(i=0;i<p->nseqf;i++)
2466 {
2467     inserir_seqss(p, ((p->seqf+i)->seqstr));
2468     while(((p->seqf+i)->seqstr[j])!='\0')
2469     {
2470         (p->seqss+i)->seqstr[strlen((p->seqf+i)->seqstr)-1-j]=(p->seqf+i)->seqstr[j];
2471         j++;
2472     }
2473     (p->seqss+i)->seqstr[strlen((p->seqf+i)->seqstr)]='\0';
2474     j=0;
2475     v[i]=-1;
2476 }
2477 for(i=0; i<p->nseqf; i++)
2478 {
2479     for(j=0; j<p->nseqf; j++)
2480     {
2481         if(strcmp((p->seqf+i)->seqstr, (p->seqf+j)->seqstr)==0 && v[j] == -1)
2482         {
2483             v[j]=i;
2484         }
2485     }
2486     for(j=0; j<p->nseqf; j++)
2487     {
2488         if(strcmp((p->seqss+i)->seqstr, (p->seqf+j)->seqstr)==0 && v[j] == -1)
2489         {
2490             v[j]=i;
2491         }
2492     }
2493 }
2494 //eliminar repetidos
2495 vseqs=eliminarrep(v,p);
2496 for(i=0; i<p->nseqss; i++)
2497 {

```



```

2528         inserir_seqssaux(p, (p->seqf+*(vseqs+i)) ->seqstr);
2529     }
2530 }
2531 free(vseqs);
2532
2533 nseq=p->nseqf;
2534
2535 for(i=0;i<nseq;i++)
2536 {
2537     remove seqf(p);
2538 }
2539
2540 for(i=0; i<p->nseqss; i++)
2541 {
2542     inserir_seqf(p, (p->seqssaux+i) ->seqstr);
2543 }
2544
2545 return 0;
2546
2547 }
2548
2549 }
2550
2551 }

```

int seq (PECASINIT * , int)

Função seq.

função para criar as sequencias de peças existentes

Parâmetros:

<i>PECASINIT</i>	*p estrutura deste tipo
<i>int</i>	num numero de jogos a calcular sequencias

Retorna:

retorna o numero de sequencias que encontrou

```

3847 {
3848
3855     //função para criar as sequencias de peças existentes
3856     /*char seqss[LINSEQ][COLSEQ];
3857     char baralhoaux[LINSEQ][COLSTR];
3858     char seqssaux[LINSEQ][COLSEQ];
3859     int l=0;
3860     int i=0;
3861     int j=0;
3862     int k=0;
3863
3864     int cont=0;
3865     int invertidos=0;
3866
3867     char aux[2000];
3868     char *s=NULL;
3869     int contadorfinal=0;
3870     char auxdir='0';*/
3871     int nseq=0;
3872     int fimdastring=0;
3873     int iguais=0;
3874     char *s=NULL;
3875     char aux[2000];
3876     char auxseqf[2000];
3877     int n=0;
3878     int i=0;
3879     int j=0;
3880     int k=0;
3881

```

```

3882     int cont=0;
3883     int invertidos=0;
3884     PECA *paux=NULL;
3885     char auxdir='0';
3886     paux=p->pfirst;
3887
3888     n=num*7;
3889     while (paux!=NULL)
3890     {
3891
3892         inserir_seqss(p,paux->str);
3893         paux=paux->pnext;
3894     }
3895
3896
3897
3898
3899     for(i=0;i<p->nseqss;i++)
3900     {
3901
3902         if((p->seqss+i)->seqstr!=NULL)
3903         {
3904
3905             invertidos++;
3906
3907         }
3908     }
3909
3910
3911     p->nseqss=invertidos;
3912     n=invertidos;
3913
3914     while (j<n)
3915     {
3916
3917         if(((p->seqss+j)->seqstr)[0]==((p->seqss+j)->seqstr)[2])
3918         {
3919
3920             j++;
3921
3922         }
3923         else
3924         {
3925
3926             inserir_seqss(p,((p->seqss+j)->seqstr));
3927             auxdir=((p->seqss+j)->seqstr)[0];
3928             ((p->seqss+invertidos)->seqstr)[0]=((p->seqss+invertidos)->seqstr)[2];
3929             ((p->seqss+invertidos)->seqstr)[2]=auxdir;
3930             invertidos++;
3931             j++;
3932
3933         }
3934     }
3935
3936
3937     for(i=0;i<invertidos;i++)
3938     {
3939
3940         inserir_baralhoaux(p,(p->seqss+i)->seqstr);
3941
3942     }
3943
3944     p->nbaralhoaux=invertidos;
3945
3946
3947     n=invertidos;
3948
3949     do{
3950         cont=0;
3951
3952         for(i=0; i<invertidos; i++)

```

```

3953     {
3954
3955         for(j=0; j<p->nseqss; j++)
3956         {
3957
3958             if(k!=0)
3959             {
3960
3961                 // strcpy(aux,seqss[j]);
3962                 strcpy(aux,(p->seqss+j)->seqstr);
3963
3964                 s = strtok (aux,"-");
3965
3966                 while (s!= NULL)
3967                 {
3968
3969                     if(strcmp((p->baralhoaux+i)->seqstr,s)==0 ||
3970 (s[2]==(p->baralhoaux+i)->seqstr[0]) && s[0]==(p->baralhoaux+i)->seqstr[2]))
3971                     {
3972
3973                         iguais++;
3974
3975                     }
3976
3977                     s = strtok (NULL, "-");
3978
3979                 }
3980                 strcpy(aux,"");
3981             }
3982
3983
3984             //se for uma peça diferente das que são usadas nas sequencias já
3985 inseridas
3986             if(iguais==0)
3987             {
3988
3989                 //verifico o tamanho da sequencia já existente
3990                 fimdastring=strlen((p->seqss+j)->seqstr)-1;
3991
3992                 //se houver encaixe entre a peça ou sequencias de peças e a peça
3993 for diferente da que já está guardada
3994                 if(((p->seqss+j)->seqstr)[fimdastring]==((p->baralhoaux+i)->seqstr)[0] &&
3995 strcmp(((p->seqss+j)->seqstr),((p->baralhoaux+i)->seqstr))!=0)
3996                 {
3997
3998                     //se já nao vou montar sequencias pela primeira vez e se há
3999 encaixe possivel normal ou invertidamente
4000                     if(k==0 &&
4001 (((p->seqss+j)->seqstr)[fimdastring]==((p->baralhoaux+i)->seqstr)[0] &&
4002 ((p->seqss+j)->seqstr)[0]==((p->baralhoaux+i)->seqstr)[2])
4003                     {
4004
4005                         cont++;
4006                         strcpy(auxseqf,((p->seqss+j)->seqstr));
4007                         strcat(auxseqf,"-");
4008                         strcat(auxseqf,((p->baralhoaux+i)->seqstr));
4009                         inserir seqf(p,auxseqf);
4010                         //guardo a sequencia que fiz num array final e incremento
4011 a variavel que vou testar no final do ciclo todo para ver se houve posivbilidade montar
4012 sequencias ou não
4013
4014                     }
4015                 }
4016             }
4017         }
4018     }

```

```

4014
4015         }
4016         iguais=0;
4017
4018     }
4019
4020 }
4021
4022 //faço as mesmas verificações acima para aumentar o array de sequencias
auxiliar para as ultimas sequencias feitas
4023 //para tentar da proxima vez fazer sequencias novas com as sequencias já obtidas
4024 for(i=0; i<invertidos; i++)
4025 {
4026
4027     for(j=0; j<p->nseqss; j++)
4028     {
4029
4030         if(k!=0)
4031         {
4032
4033             strcpy(aux, (p->seqss+j)->seqstr);
4034
4035             s = strtok (aux, "-");
4036
4037             while (s!= NULL)
4038             {
4039
4040                 if(strcmp((p->baralhoaux+i)->seqstr,s)==0 ||
(s[2]==((p->baralhoaux+i)->seqstr[0]) && s[0]==((p->baralhoaux+i)->seqstr[2])))
4041                 {
4042
4043                     iguais++;
4044
4045                 }
4046
4047                 s = strtok (NULL, "-");
4048
4049             }
4050
4051             strcpy(aux, "");
4052
4053             if(iguais==0)
4054             {
4055
4056                 fimdastring=strlen((p->seqss+j)->seqstr)-1;
4057
4058
4059                 if(((p->seqss+j)->seqstr)[fimdastring]==((p->baralhoaux+i)->seqstr)[0] &&
strcmp((p->seqss+j)->seqstr), ((p->baralhoaux+i)->seqstr) !=0)
4060                 {
4061
4062                     strcpy(auxseqf, ((p->seqss+j)->seqstr));
4063                     strcat (auxseqf, "-");
4064                     strcat (auxseqf, ((p->baralhoaux+i)->seqstr));
4065                     inserir_seqssaux(p,auxseqf);
4066
4067                 }
4068
4069             }
4070
4071         }
4072
4073     else
4074     {
4075
4076         fimdastring=strlen((p->seqss+j)->seqstr)-1;
4077
4078
4079         if(((p->seqss+j)->seqstr)[fimdastring]==((p->baralhoaux+i)->seqstr)[0] &&
strcmp((p->seqss+j)->seqstr), ((p->baralhoaux+i)->seqstr) !=0)

```

```

4079         {
4080
4081
4082
4083 if(((p->seqss+j)->seqstr)[fimdastring])==((p->baralhoaux+i)->seqstr)[0] &&
4084 ((p->seqss+j)->seqstr)[0]==((p->baralhoaux+i)->seqstr)[2])
4085     {
4086
4087     }
4088     else
4089     {
4090
4091         strcpy(auxseqf, ((p->seqss+j)->seqstr));
4092         strcat(auxseqf, "-");
4093         strcat(auxseqf, ((p->baralhoaux+i)->seqstr));
4094         inserir seqssaux(p,auxseqf);
4095     }
4096
4097     }
4098
4099     }
4100     iguais=0;
4101 }
4102
4103 }
4104
4105
4106 //por fim colocamos o array de seqss auxiliar vazio e preencho-o com as novas
sequencias feitas
4107
4108     nseq=p->nseqss;
4109
4110     for(i=0;i<nseq;i++)
4111     {
4112
4113         remove seqss(p);
4114
4115     }
4116
4117
4118     for(i=0;i<p->nseqssaux;i++)
4119     {
4120
4121         inserir seqss(p, (p->seqssaux+i)->seqstr);
4122
4123     }
4124
4125     nseq=p->nseqssaux;
4126
4127     for(i=0;i<nseq;i++)
4128     {
4129
4130         remove seqssaux(p);
4131
4132     }
4133
4134     k++;
4135
4136 }
4137 //faço tudo isto até não haver possibilidades de fazer combinações de sequencias,
4138 //porque quando não houver sequencias a fazer é sinal que o ciclo tem de acabar
4139 while(cont>0);
4140
4141 //retorno o numero de sequencias feitas
4142 return 0;
4143
4144 }

```

int seqcomseqinicial (char *baralhoss*[][COLSTR], char *seqf*[][COLSEQ], int , char *seqinit*[])

Função seqcomseqinicial.

esta funcao é igual à função das sequencias, só que nestas começo inicialmente com uma sequencia

Parâmetros:

<i>char</i>	baralhoss[][COLSTR] baralhos do jogador em string
<i>char</i>	seqf[][COLSEQ] array final onde vou guardar as sequencias todas possiveis
<i>int</i>	num numero de jogos a converter
<i>char</i>	seqinit[] sequencia inicial

Retorna:

retorna o numero de sequencias que encontrou

VERIFICA QUANTIDADE DE CORRESPONDENCIAS

JUNTA AS CORRESPONDENCIAS

```

5025 {
5034     //esta funcao é igual à função das sequencias, só que nestas começo inicialmente
com uma sequencia
5035     char seqss[LINSEQ][COLSEQ];
5036     char baralhoaux[LINSEQ][COLSEQ];
5037     char invertidas[LINSEQ][COLSEQ];
5038     char vinvertidas[LINSEQ][COLSEQ];
5039     char auxinv[COLSTR];
5040     char invfinal[COLSTR];
5041     int i=0;
5042     int j=0;
5043     int k=0;
5044     int tam=LINSEQ;
5045     int n=0;
5046     int cont=0;
5047     int fimdastring=0;
5048     int decont=LINSEQ-1;
5049     int iguais=0;
5050     char aux[2000];
5051     char *s;
5052     char *inv;
5053     char *invseq;
5054     int invertidos=0;
5055     int contadorfinal=0;
5056     char auxdir='0';
5057     int contapecasvinvertidas=0;
5058     int seqinitexiste=0;
5059     /*a variavel ok acrescentada para apenas passar para o array secundário o que foi
passado para o final sem concatenar
5060     peças mal*/
5061     int ok=0;
5062
5063
5064     //n=num*7;
5065     n=num*7;
5066
5067     esvaziaseqstr(seqss,LINSEQ);
5068     esvaziaseqstr(baralhoaux,LINSEQ);
5069
5070     for(i=0; i<n; i++)
5071     {
5072
5073         strcpy(seqss[i],baralhoss[i]);
5074
5075     }

```

```

5076     invertidos=n;
5077
5078
5079
5080     //acresecentar pecas ao contrario
5081
5082     while(j<n)
5083     {
5084
5085         if(seqss[j][0]==seqss[j][2])
5086         {
5087
5088             j++;
5089             cont++;
5090
5091         }
5092         else
5093         {
5094
5095             strcpy(seqss[invertidos],seqss[j]);
5096             auxdir=seqss[invertidos][0];
5097             seqss[invertidos][0]=seqss[invertidos][2];
5098             seqss[invertidos][2]=auxdir;
5099             strcpy(invertidas[k],seqss[invertidos]);
5100             invertidos++;
5101             j++;
5102
5103         }
5104     }
5105
5106     /*printf("\nINVERTIDAS\n");
5107     for(i=0;i<k;i++){
5108
5109         printf("%s\n",invertidas[i]);
5110
5111     }
5112     printf("\n");*/
5113
5114
5115     for(i=0; i<(invertidos); i++)
5116     {
5117
5118         strcpy(seqf[i],seqss[i]);
5119
5120     }
5121
5122     for(i=0; i<(invertidos); i++)
5123     {
5124
5125         strcpy(baralhoaux[i],seqss[i]);
5126
5127     }
5128
5129     strcpy(baralhoaux[invertidos],seqinit);
5130
5131     for(i=0; i<strlen(seqinit); i++)
5132     {
5133
5134         baralhoaux[invertidos+1][i]=seqinit[strlen(seqinit)-1-i];
5135
5136     }
5137     //sequencia normal e invertidas acresceto ao array
5138     invertidos=invertidos+2;
5139
5140     for(i=0; i<invertidos-2; i++)
5141     {
5142
5143         if(strcmp(seqinit,baralhoaux[i])==0)
5144         {
5145
5146             seqinitexiste++;

```

```

5147     }
5148
5149     }
5150
5151     //copiar sequencia inicial para dentro dos arrays de string, tanto o final como
5152 o aux
5153     strcpy(seqss[invertidos-2],baralhoaux[invertidos-2]);
5154     strcpy(seqss[invertidos-1],baralhoaux[invertidos-1]);
5155
5156     strcpy(seqf[invertidos-2],baralhoaux[invertidos-2]);
5157     strcpy(seqf[invertidos-1],baralhoaux[invertidos-1]);
5158
5159
5160     n=invertidos;
5161     k=0;
5162
5163     do
5164     {
5165         cont=0;
5166         for(i=0; i<invertidos; i++)
5167         {
5168             for(j=0; j<tam; j++)
5169             {
5170                 if(k!=0)
5171                 {
5172                     strcpy(aux,seqss[j]);
5173
5174                     s = strtok (aux,"-");
5175
5176                     while (s!= NULL)
5177                     {
5178                         if(strcmp(baralhoaux[i],s)==0 || (s[2]==baralhoaux[i][0] &&
5179 s[0]==baralhoaux[i][2]))
5180                         {
5181                             iguais++;
5182                         }
5183
5184                         s = strtok (NULL, "-");
5185                     }
5186                     strcpy(aux,"");
5187                 }
5188             }
5189             if(iguais==0)
5190             {
5191                 /*Ve o tamanho da string*/
5192                 fimdastring=strlen(seqss[j])-1;
5193                 /*compara as strings iniciasi e invertidas com aquelas que vamos
5194 aumentar*/
5195                 if(seqss[j][fimdastring]==baralhoaux[i][0] && i!=j &&
5196 strcmp(seqss[i],"9|9")!=0)
5197                 {
5198                     cont++;
5199                     strcpy(seqf[contadorfinal],seqss[j]);
5200                     strcat(seqf[contadorfinal],"-");
5201                     strcat(seqf[contadorfinal],baralhoaux[i]);
5202
5203                     contadorfinal++;
5204                     ok++;
5205                     //este array é o array final

```



```

5216             }
5217
5218         }
5219         iguais=0;
5220
5221     }
5222
5223 }
5224
5225
5228     for(i=0; i<invertidos; i++)
5229     {
5230
5231         for(j=0; j<ok; j++)
5232         {
5233
5234             if(k!=0)
5235             {
5236
5237                 strcpy(aux,seqss[j]);
5238
5239                 s = strtok (aux,"-");
5240                 //parte para ver se a sequencia já montada contem a peca que vamos
tentar inserir
5241                 while (s!= NULL)
5242                 {
5243
5244                     if(strcmp(baralhoaux[i],s)==0 || (s[2]==baralhoaux[i][0] &&
s[0]==baralhoaux[i][2]))
5245                     {
5246
5247                         iguais++;
5248
5249                     }
5250
5251                     s = strtok (NULL, "-");
5252
5253                 }
5254
5255                 strcpy(aux,"");
5256
5257                 if(iguais==0)
5258                 {
5259
5260                     fimdastring=strlen(seqss[j])-1;
5261
5262                     if(seqss[j][fimdastring]==baralhoaux[i][0] && i!=j &&
strcmp(seqss[i],"9|9")!=0)
5263                     {
5264
5265                         strcpy(seqss[decont],seqss[j]);
5266                         strcat(seqss[decont],"-");
5267                         strcat(seqss[decont],baralhoaux[i]);
5268                         decont--;
5269
5270                     }
5271                 }
5272
5273             }
5274
5275         else
5276         {
5277
5278             for(fimdastring=0; seqss[j][fimdastring]!='\0'; fimdastring++);
5279             fimdastring--;
5280
5281             if(seqss[j][fimdastring]==baralhoaux[i][0] && i!=j &&
strcmp(seqss[i],"9|9")!=0)
5282             {
5283
5284                 strcpy(seqss[decont],seqss[j]);

```

```

5285             strcat(seqss[decont], "-");
5286             strcat(seqss[decont], baralhoaux[i]);
5287             //printf("%s == %s | JUNTA -->
%s\n\n", seqss[j], baralhoaux[i], seqss[decont]);
5288             decont--;
5289
5290             }
5291
5292             }
5293
5294             iguais=0;
5295
5296             }
5297
5298             }
5299
5300             //Agora limpa o array seqss em cima e assa de baixo para cima para continuar
a juntar
5301             i=0;
5302
5303             while (strcmp(seqss[i], "9|9") != 0)
5304             {
5305
5306                 strcpy(seqss[i], "9|9");
5307                 i++;
5308             }
5309
5310             decont=LINSEQ-1;
5311             i=0;
5312
5313             //apaga array em baixo
5314
5315             while (strcmp(seqss[decont], "9|9") != 0)
5316             {
5317
5318                 strcpy(seqss[i], seqss[decont]);
5319                 strcpy(seqss[decont], "9|9");
5320                 decont--;
5321                 i++;
5322             }
5323
5324             }
5325
5326             tam=(LINSEQ-decont-1);
5327             decont=LINSEQ-1;
5328             k++;
5329
5330             }
5331
5332             while(cont>0);
5333
5334             //verifica se contem invertidas e normais e retira-as
5335             cont=0;
5336             i=0;
5337
5338             while(strcmp(seqf[cont], "9|9") != 0)
5339             {
5340                 strcpy(seqss[i], seqf[cont]);
5341                 cont++;
5342                 i++;
5343             }
5344
5345             esvaziaseqstr(invertidas, LINSEQ);
5346
5347             for(i=0; i<cont; i++)
5348             {
5349
5350                 strcpy(invertidas[i], seqss[i]);
5351
5352             }
5353

```

```

5354
5355     j=0;
5356     for(i=0; i<cont; i++)
5357     {
5358
5359         invseq = strtok (invertidas[i],"-");
5360
5361         while (invseq != NULL)
5362         {
5363
5364             strcpy(vinvertidas[j],invseq);
5365             j++;
5366             invseq = strtok (NULL, "-");
5367
5368         }
5369         contapecasvinvertidas=j;
5370         j=0;
5371         inv = strtok (seqss[i],"-");
5372
5373         while (inv != NULL)
5374         {
5375
5376             strcpy(auxinv,inv);
5377
5378             invfinal[2]=auxinv[0];
5379             invfinal[1]=auxinv[1];
5380             invfinal[0]=auxinv[2];
5381
5382             if(invfinal[2]!=invfinal[0])
5383             {
5384
5385                 for(k=0; k<contapecasvinvertidas; k++)
5386                 {
5387
5388                     if(strcmp(invfinal,vinvertidas[k])==0)
5389                     {
5390
5391                         strcpy(seqf[i],"-");
5392
5393                     }
5394
5395                 }
5396
5397             }
5398
5399             inv = strtok (NULL, "-");
5400
5401         }
5402         esvaziaseqstr(vinvertidas,LINSEQ);
5403         j=0;
5404     }
5405
5406     contadorfinal=0;
5407     i=0;
5408
5409     while(i<LINSEQ)
5410     {
5411
5412         if(strcmp(seqf[i],"9|9")!=0)
5413         {
5414
5415             contadorfinal++;
5416
5417         }
5418
5419         i++;
5420     }
5421
5422     if(seqinitexiste>0)
5423     {
5424

```

```

5425     contadorfinal=-1;
5426
5427     }
5428     return contadorfinal;
5429
5430 }

```

int strtoque (char *stra*[][COLSEQ], char *str*[], char)

Função strtoque.

esta função recebe um array de strings, uma string e o carater pelo qual vai partir e faz o strtok()

Parâmetros:

<i>char</i>	<i>stra</i> [][COLSEQ] array de strings na qual vamos guardar as strings partidas
<i>char</i>	<i>str</i> [] string que queremos partir
<i>char</i>	car carater pelo qual partimos

Retorna:

retorna o numero de pecas partidas

```

4354 {
4355
4363     //esta função recebe um array de strings, uma string e o carater pelo qual vai partir
e faz o strtok()
4364     int k=0,i=0,j=0;
4365     for (i=0; str[i]!='\0'; i++)
4366     {
4367         if (str[i]!=car)
4368         {
4369             stra[k][j]=str[i];
4370             j++;
4371         }
4372         else
4373         {
4374             stra[k][j]='\0';
4375             k++;
4376             j=0;
4377         }
4378     }
4379     return k+1;
4380
4381 }

```

int tirartracosinvertidos (char *seqf*[][COLSEQ], int *numdeseq*)

Função tirartracosinvertidos.

esta função serve para eliminar as sequencias que ão repetidas mas que estão escritas da direita para a esquerda, nas quais coloquei um "-" e agora elimino-as e puxo as outras para ci

Parâmetros:

<i>char</i>	<i>seqf</i> [][COLSEQ] array final de strings
<i>int</i>	numdeseq numero de sequencias

Retorna:

retorna o numero de sequencias menos as que estavam escritas da esquerda para a direita

```

4988 {
4989
4996     //esta função serve para eliminar as sequencias que ão repetidas mas que estão
escritas da direita para a esquerda

```

```

4997 //mas quais coloquei um "-" e agora elimino-as e puxo as outras para cima
4998
4999 int i=0;
5000 int cont=0;
5001
5002 while(strcmp(seqf[i],"9|9")!=0)
5003 {
5004
5005     if(strcmp(seqf[i],"-")==0)
5006     {
5007
5008         cont++;
5009
5010     }
5011
5012
5013     i++;
5014
5015 }
5016
5017 return numdeseq-cont;
5018
5019
5020 }

```

void trocapadiao (char seqf[][COLSEQ], int size, char padrao[], char padraon[], char seqfpadiao[][COLSEQ], int * sizeseqfpadiao)

Função trocapadiao.

função para procurar uma substring numa string (utilizei esta variante na torca de padrao)

Parâmetros:

<i>char</i>	seqf[][COLSEQ] array final de strings
<i>int</i>	size numero da linha do array de strings final que vamos querer procurar uma substring
<i>char</i>	padrao[] string a substituir
<i>char</i>	padraon[] string substituta
<i>char</i>	seqfpadiao[][COLSEQ]
<i>int</i>	*sizeseqfpadiao indice da sequencia padrao na qual vou trocar o padrao

CICLO DAS OCURENCIAS DO PADRAO por indices da string

```

4743 {
4744
4755     int tampadiao=0;
4756     int tampadraon=0;
4757     char *s;
4758     char *p;
4759     char auxseq[LINSEQ][COLSEQ];
4760     char auxpad[COLSEQ];
4761     int contseq=0;
4762     int encontrasub=0;
4763     int i=0;
4764     int k=0;
4765     int sequencia=0;
4766     int posini=0;
4767     int contpad=0;
4768     int vetorpadiao[COLSEQ];
4769     int vposition[COLSEQ];
4770     char padraoasubstaux[LINSEQ];
4771     char auxpadraon[LINSEQ];
4772     int confim=0;
4773     int troca=0;
4774     int r=0;
4775     /*printf("%s\n\n", seqf[0]);

```

```

4776     printf("%s\n\n",padrao);
4777     printf("%s\n\n",padraon);*/
4778     strcpy(auxpadraon,padraon);
4779     for(sequencia=0; sequencia<size; sequencia++)
4780     {
4781         esvaziaseqstr(auxseq,size);
4782
4783         //fazemos o strtok() da sequencia, do padrão a trocar e do padrão novo
4784         for(i=0; i<size; i++)
4785         {
4786             strcpy(auxseq[i],seqf[i]);
4787         }
4788
4789         s = strtok (auxseq[sequencia],"-");
4790
4791         while (s != NULL)
4792         {
4793             contseq++;
4794
4795             s = strtok (NULL, "-");
4796         }
4797
4798         //partir o padrao
4799         strcpy(auxpad,padrao);
4800
4801         p = strtok (auxpad,"-");
4802
4803         while (p != NULL)
4804         {
4805             contpad++;
4806
4807             p = strtok (NULL, "-");
4808         }
4809
4810         for(i=0; i<contseq; i++)
4811         {
4812             //chamo a função procurar sub string e mando o endereço porque eu envio
4813             só aquela sequencia
4814             // o size é o numero de sequencias a fazer trocas
4815
4816             encontrasub=proctsubseq trocapadrao(&seqf[sequencia],size,padrao,i);
4817
4818             if(encontrasub!=-1)
4819             {
4820                 if(i==0)
4821                 {
4822                     vetorpadrao[k]=encontrasub;
4823                     k++;
4824                 }
4825                 else
4826                 {
4827                     if(vetorpadrao[k-1]!=encontrasub)
4828                     {
4829                         vetorpadrao[k]=encontrasub;
4830                         k++;
4831                     }
4832                 }
4833             }
4834         }
4835     }
4836 }
4837
4838
4839
4840
4841
4842
4843
4844
4845

```

```

4846         }
4847     }
4848 }
4849 }
4850 }
4851
4852
4853 //k sao os padroes encontrados que guardo num array
4854 if(k==0)
4855 {
4856
4857     //printf("O padrao nao foi encontrado, logo nao pode substituir! ");
4858 }
4859 else
4860 {
4861
4862     for(i=0; i<k; i++)
4863     {
4864
4865         tampadrao = strlen(padrao);
4866         tampadrao--;
4867         tampadraon = strlen(padraon);
4868         tampadraon--;
4869
4870         if(vetorpadrao[i]==0)
4871         {
4872             posini=0;
4873         }
4874         else
4875         {
4876             if(i==0)
4877             {
4878                 posini=vetorpadrao[i]*4;
4879             }
4880             else
4881             {
4882                 //depois de tirar pecas as posicoes alteram e isto prevê essas
4883                 //multiplicar por quatro dame o carater tipo 16 carater 4*4
4884                 contpad=contpad*4;
4885                 posini=vetorpadrao[i]*4;
4886             }
4887         }
4888     }
4889     vposition[i]=posini;
4890 }
4891 }
4892
4893 for(i=(k-1); i>=0; i--)
4894 {
4895
4896     //se for uma substituição do padrao no fim
4897     if(seqf[sequencia][vposition[i]+tampadrao+1]=='\0')
4898     {
4899
4900
4901
4902
4903     if(padraon[0]==seqf[sequencia][strlen(seqf[sequencia])-1-tampadrao-2])
4904     {
4905
4906         //printf("i-> %d pos-> %d\n",i,vposition[i]);
4907         seqf[sequencia][vposition[i]-1]='\0';
4908         strcat(seqf[sequencia],"-");
4909         strcat(seqf[sequencia],padraon);

```

```

4917             confim++;
4918             troca++;
4919
4920         }
4921
4922     }
4923
4924
4925     //se for uma substituição do padrao no inicio
4926     if(vposition[i]==0)
4927     {
4928
4929         if(padraon[strlen(padraon)-1]==seqf[sequencia][tampadrao+2])
4930         {
4931
4932             //printf("i-> %d   pos-> %d\n",i,vposition[i]);
4933
4934             strcpy(seqf[sequencia],&seqf[sequencia][vposition[i]+tampadrao+2]);
4935             strcat(padraon,"-");
4936             strcat(padraon,seqf[sequencia]);
4937             strcpy(seqf[sequencia],padraon);
4938             troca++;
4939
4940         }
4941
4942     }
4943
4944
4945     //se for uma substituição do padrao no meio
4946     if(seqf[sequencia][vposition[i]+tampadrao+1]!='\0' && vposition[i]!=0
&& confim==0)
4947     {
4948
4949         if(seqf[sequencia][vposition[i]+tampadrao+2]==padraon[strlen(padraon)-1] &&
seqf[sequencia][vposition[i]-2]==padraon[0])
4950         {
4951
4952             //printf("i-> %d   pos-> %d\n",i,vposition[i]);
4953
4954             strcpy(padraoasubstaux,&seqf[sequencia][vposition[i]+tampadrao+1]);
4955             seqf[sequencia][vposition[i]]='\0';
4956             strcat(seqf[sequencia],padraon);
4957             strcat(seqf[sequencia],padraoasubstaux);
4958             troca++;
4959
4960         }
4961
4962         confim=0;
4963
4964     }
4965
4966     r=*sizeseqfpadrao;
4967     if(troca==1)
4968     {
4969
4970         strcpy(seqfpadrao[r],seqf[sequencia]);
4971         //printf("[%d] ---> %s\n",sequencia,seqfpadrao[*sizeseqfpadrao]);
4972         r++;
4973     }
4974     *sizeseqfpadrao=r;
4975     troca=0;
4976     k=0;
4977     contseq=0;
4978     contpad=0;
4979     strcpy(padraon,auxpadraon);
4980
4981 }
4982

```


int verificasequencia (char seq[])

Função verificasequencia.

verifica se a sequência é possível juntar ex:2|3-3|4 estas pecas encaixam e esta funcao faz esta verificação

Parâmetros:

<i>char</i>	seq[] recebe a sequência de peças
-------------	-----------------------------------

Retorna:

Retorna 0 se for possível e 1 se não for

```

2303 {
2304
2305
2312     int cont=0;
2313     int i=0;
2314
2315     for(i=0; i<strlen(seq); i++)
2316     {
2317
2318         if(seq[i]=='-')
2319         {
2320
2321             if(seq[i+1]!=seq[i-1])
2322             {
2323
2324                 cont++;
2325
2326             }
2327
2328         }
2329
2330     }
2331
2332     if(cont==0)
2333     {
2334
2335         return 0;
2336
2337     }
2338     else
2339     {
2340
2341         return 1;
2342
2343     }
2344
2345 }
```

Índice

INDEX