Segue a listagem de um programa incompleto para lidar com um jogo de dominó.

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

#define pecas     28
#define maoInicial  7
#define jogadores   4

typedef enum _TValor_ {zero, um, dois, tres, quatro, cinco, seis} TValor;

typedef struct _TPecaDeDomino_ {
  int codigo;
  TValor face1, face2;
} TPecaDeDomino;

typedef struct _TJogoDeDomino_ {
  TPecaDeDomino pecasDoJogo[pecas];
  TPecaDeDomino pecasDosJogadores[jogadores][maoInicial];
} TJogoDeDomino;

void iniciaPecasDoJogo (TJogoDeDomino*);
void imprimePeca(TPecaDeDomino);
void imprimePecasDoJogo(TJogoDeDomino);
void imprimePecasDosJogadores(TJogoDeDomino);
void imprimePecasComCodigosGerados(TJogoDeDomino*);
void distribuiPecas(TJogoDeDomino*); //FALTA PROGRAMAR – MATRIZES E TIPOS
void arquivaPecasDeJogador(TJogoDeDomino, int, FILE**); // FALTA PROGRAMAR - ARQUIVOS
void arquivoTxtParaBin(FILE**, FILE**, int); //FALTA PROGRAMAR - ARQUIVOS
int  min(TPecaDeDomino);
int  max(TPecaDeDomino);
int  geraCodigo(TPecaDeDomino); //FALTA PROGRAMAR – RACIOCÍNIO LÓGICO

void iniciaPecasDoJogo (TJogoDeDomino *jogo) {
  int i, j, k = 0;
  for (i = 0; i < maoInicial; i++)
    for (j = i; j < maoInicial; j++) {
      (*jogo).pecasDoJogo[k].codigo = k;
      (*jogo).pecasDoJogo[k].face1  = (TValor)i;
      (*jogo).pecasDoJogo[k].face2  = (TValor)j;
      k = k+1;
    }

  for (i = 0; i < jogadores; i++)
    for (j = 0; j < maoInicial; j++) {
      (*jogo).pecasDosJogadores[i][j].codigo = (TValor)-1;
      (*jogo).pecasDosJogadores[i][j].face1  = (TValor)-1;
      (*jogo).pecasDosJogadores[i][j].face2  = (TValor)-1;
    }
}

void imprimePeca(TPecaDeDomino peca) {  printf("%2d ---> %d | %d\n", peca.codigo, peca.face1, peca.face2); }
```

```
void imprimePecasDoJogo(TJogoDeDomino jogo) {
  int i;
  for (i = 0; i < pecas; i++)
    if ((jogo).pecasDoJogo[i].codigo != -1)
      imprimePeca((jogo).pecasDoJogo[i]);
  printf("\n");
}

void imprimePecasDosJogadores(TJogoDeDomino jogo) {
  int i, j;
  for (i = 0; i < jogadores; i++) {
    for (j = 0; j < maoInicial; j++)
      if ((jogo).pecasDosJogadores[i][j].codigo != -1)
        imprimePeca((jogo).pecasDosJogadores[i][j]);
    printf("\n");
  }
}

void imprimePecasComCodigosGerados(TJogoDeDomino *jogo) {
  int i;
  for(i = 0; i < pecas; i = i+1) {
    //(*jogo).pecasDoJogo[i].codigo = geraCodigo((*jogo).pecasDoJogo[i]);
    imprimePeca((*jogo).pecasDoJogo[i]);
  }
  printf("\n");
}

int min(TPecaDeDomino peca) { return peca.face1 < peca.face2 ? peca.face1 : peca.face2; }

int max(TPecaDeDomino peca) { return peca.face1 > peca.face2 ? peca.face1 : peca.face2; }

main() {
  TJogoDeDomino jogo;
  FILE * arqTxt, *arqBin;
  int i;

  iniciaPecasDoJogo(&jogo);
  //imprimePecasComCodigosGerados(&jogo);
  imprimePecasDoJogo(jogo);

  //distribuiPecas(&jogo);
  imprimePecasDosJogadores(jogo);

  arqTxt = fopen("jogador1.txt", "w+");
  //arquivaPecasDeJogador(jogo, 0, &arqTxt);

  arqBin = fopen("jogador1.dat", "wb+");
  //arquivoTxtParaBin(&arqTxt, &arqBin, sizeof(TPecaDeDomino));

  fclose(arqTxt);
  fclose(arqBin);
}
```

Você encontrará esse código dividido em 3 arquivos:

- definicoes.h
- implementacoesIniciais.h
- main.c

Implemente as funções que faltam ser programadas. ALTERE APENAS O ARQUIVO main.c. Responda às questões na ordem em que elas aparecem e teste cada função durante a programação, uma a uma.

1) Função para distribuir peças **(3, 0 pontos)**

É preciso entender que a mesa do jogo encontra-se com as 28 peças do jogo (ver o tipo jogo). Essa função deve sortear peças aleatórias aos 4 jogadores. À medida que uma peça é sorteada, ela deve seguir para a mão de um jogador. O lugar que ela ocupava na mesa do jogo deve ser preenchido pelo valor de uma peça vazia. Veja as linhas de código das funções de impressão para bolar valores apropriados para uma peça vazia.

2) Função para imprimir a mão de um jogador em um arquivo texto **(3, 0 pontos)**

Varre a mão de determinado jogador do jogo de dominó e armazena as peças encontradas em um arquivo texto com uma peça em cada linha do arquivo. Atenção! A mão de um jogador é implementada com um vetor. Se uma posição do vetor estiver ocupada por uma peça vazia, você não deve ignorar esta posição do vetor e partir para a próxima posição.

3) Função para copiar de arquivo texto para arquivo binário de peças de dominó **(3, 0 pontos)**

Varre um arquivo texto com a mão de um jogador de dominó, desde seu início até seu fim. A cada linha lida, o que corresponde a uma peça, insere um novo registro de peça de dominó em um arquivo binário.

4) Função para gerar um código a uma peça **(1, 0 ponto)**

Imagine a seguinte situação. Seria possível escrever apenas as faces de cada pedra no arquivo texto de peças de dominó, ignorando seu código. Se a implementação tivesse sido feita dessa maneira, no momento de copiar uma peça do arquivo de texto para o arquivo de registros, seria necessário descobrir o código de uma peça. Existe um padrão na determinação dos códigos das peças de dominó. Descubra a função matemática que define um código de uma peça e implemente essa função. Segue a listagem com os códigos das peças.

Código	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Face 1	0	0	0	0	0	0	0	1	1	1	1	1	1	2
Face 2	0	1	2	3	4	5	6	1	2	3	4	5	6	2
Código	14	15	16	17	18	19	20	21	22	23	24	25	26	27
Face 1	2	2	2	2	3	3	3	3	4	4	4	5	5	6
Face 2	3	4	5	6	3	4	5	6	4	5	6	5	6	6