```
/*
  E-fólio A, Programação, Universidade Aberta, 2015/16
  Tema: Dominó (https://pt.wikipedia.org/wiki/Domin%C3%B3)

  Alínea A (1 valor): peça de dominó aleatória (de 0 a 6)
  Alínea B (1 valor): conjunto de 28 peças de dominó
  Alínea C (1 valor): sequência de peças
  Alínea D.1): partida com jogadores humanos (0,5 valores)
  Alínea D.2) Jogador artificial (0,5 valores)
    - entre as jogadas considerar por ordem de prioridade:
      - fica 1 só número, e possui peças para esse número
      - se tem hipótese de jogar doble, livrar-se dele
      - ficam 2 números, e possui peças para ambos os números
      - ficam 2 números, e possui peças para um dos números
      - fica 1 só número, e não possui peças para esse número
      - ficam 2 números e não possui peças para nenhum dos números
    - em caso de igualdade, jogar a peça com mais pontos
    - deixe o primeiro jogador humano, e os restantes artificiais
    - deixe visivel as peças de todos os jogadores, para que seja possível confirmar as jogadas

  Nota: vetores de dimensão variável iniciam-se com um número com a sua dimensão,
        de modo a reduzir parâmetros.
        o vetor em si segue de 1 a N, pelo que deve-se alocar sempre N+1 posições
*/

#include <time.h>

////////////////////////////////////
// função base para resposta à alínea A
void PrintPeca(int esquerda, int direita) {
  printf("[%d:%d]",esquerda,direita);
```

```
}

//////////////////////////////////////
// funções base para resposta à alínea B

/* reutilizado de AFs */
void Baralhar(int v[], int n)
{
  int i,j,aux;
  /* processar todos os elementos */
  for(i=0;i<n-1;i++) {
    /* gerar um valor aleatório para sortear o elemento do
       vector a ficar na posição i (entre i e n-1). */
    j=i+rand()%(n-i);
    aux=v[i];
    v[i]=v[j];
    v[j]=aux;
  }
}

// nota: orientação é necessário apenas para alínea C
void PrintPecas(int *v, int pecas[28][2], int *orientacao)
{
  int i, n=v[0];
  for(i=1;i<=n;i++)
    if(orientacao==NULL || orientacao[i]==0)
      PrintPeca(pecas[v[i]][0],pecas[v[i]][1]);
    else
      PrintPeca(pecas[v[i]][1],pecas[v[i]][0]);
}
```

```
//////////////////////////////////////
// funções base para resposta à alínea C

// calcula os números à esquerda e à direita na mesa
void LadosMesa(int mesa[2][28], int pecas[28][2], int *esquerda, int *direita)
{
  int nMesa=mesa[0][0];

  if(nMesa==0) {
    // todos os lances são permitidos
    *esquerda=*direita=-1;
  } else {
    if(mesa[1][1]==0)
      *esquerda=pecas[mesa[0][1]][0];
    else
      *esquerda=pecas[mesa[0][1]][1];

    if(mesa[1][nMesa]==0)
      *direita=pecas[mesa[0][nMesa]][1];
    else
      *direita=pecas[mesa[0][nMesa]][0];
  }
}

// obter informação sobre a mesa:
// inserir - introduz a peça à esquerda (1), direita (2) ou tanto faz (3), ou não introduz (0)
// retorna se a peça pode ser introduzida à esquerda (1), direita (2), ambas (3) ou não pode ser (0)
int Mesa(int mesa[2][28], int peca, int pecas[28][2], int inserir)
{
  int nMesa=mesa[0][0];
```

```
int i, resultado=0, esquerda, direita;


LadosMesa(mesa,pecas,&esquerda,&direita);
// ver se a peça pode ser colocada à esquerda e/ou à direita
if(peca>=0 && peca<28) {
 if(esquerda==-1 || pecas[peca][0]==esquerda || pecas[peca][1]==esquerda) {
  resultado+=1;
  if(inserir==1 || inserir==3) {
   nMesa++;
   inserir=0; // inserir apenas uma vez
   // inserir à esquerda
   for(i=nMesa;i>1;i--) {
    mesa[0][i]=mesa[0][i-1];
    mesa[1][i]=mesa[1][i-1];
   }
   mesa[0][1]=peca;
   // orientação
   if(pecas[peca][1]==esquerda)
    mesa[1][1]=0;
   else
    mesa[1][1]=1;
   mesa[0][0]=nMesa;
   mesa[1][0]=nMesa;
  }
 }
 if(direita==-1 || pecas[peca][0]==direita || pecas[peca][1]==direita) {
  resultado+=2;
  if(inserir==2 || inserir==3) {
   nMesa++;
   mesa[0][nMesa]=peca;
   if(pecas[peca][0]==direita)
```

```
        mesa[1][nMesa]=0;
       else
        mesa[1][nMesa]=1;
       mesa[0][0]=nMesa;
       mesa[1][0]=nMesa;
     }
    }
  }

  return resultado;
}

////////////////////////////////////////
// funções base para resposta à alínea D1

// distribuir as peças segundo a ordem (após baralhar)
void InicioJogo(int ordem[28], int jogador[4][28],int compra[28],int nJogadores)
{
  int i, j, id=0;
  // cada jogador fica com 6 peças
  for(i=0;i<nJogadores;i++) {
    for(j=0;j<6;j++)
      jogador[i][j+1]=ordem[id++];
    jogador[i][0]=6;
  }
  // restantes peças ficam para compra
  for(i=0;id<28;i++)
    compra[i+1]=ordem[id++];
  compra[0]=i;
}
```

```
void MostraEstado(int mesa[2][28], int *jogador, int nJogador, int pecas[28][2])
{
  printf("\nMesa:");
  PrintPecas(mesa[0],pecas,mesa[1]);
  printf("\nJogador %c: ",'A'+nJogador);
  PrintPecas(jogador, pecas, NULL);
}


// fazer a jogada com a informação atual, e se tiver de passar,
// retornar 0, c.c. retornar 1
int Jogada(int mesa[2][28], int *jogador,
  int nJogador, int pecas[28][2], int *compra)
{
  char str[80];
  int naoPassou=1, jogada;
  int i,resultado;

  // mostrar o estado ao utilizador
  MostraEstado(mesa, jogador, nJogador, pecas);

  if(nJogador==0) {  // alínea D1: condicional sempre ativo (todos humanos)
    // jogador humano
    printf("\nLance (-%d a %d):",jogador[0],jogador[0]);
    gets(str);
    jogada=atoi(str);
  } else { // alínea D2
    // jogador artificial
    jogada=JogadaArtificial(mesa,jogador,nJogador,pecas);
    printf("\nLance artificial: %d", jogada);
  }
```

```
 if(jogada>0 && jogada<=jogador[0] ||
   jogada<0 && -jogada<=jogador[0]) { // lance válido

  if(jogada<0) { // joga à esquerda
    jogada=-jogada;
    resultado=Mesa(mesa, jogador[jogada], pecas, 1);
  } else // joga à direita
    resultado=Mesa(mesa, jogador[jogada], pecas, 2);

  if(resultado==0) { // não inseriu, pelo que passou o lance
    naoPassou=0;
  } else { // peça jogada, pelo que deve-se retirar do jogador
    for(i=jogada;i<jogador[0];i++)
      jogador[i]=jogador[i+1];
    jogador[0]--;
  }
 } else
  naoPassou=0;

 if(naoPassou==0) { // não executou uma jogada válida, carregar uma peça se possível
  if(compra[0]>0) {
    jogador[++(jogador[0])]=compra[compra[0]--];
    printf("\nPeca adquirida");
    return Jogada(mesa,jogador,nJogador,pecas,compra);
  }
 }

 return naoPassou;
}
```

```
// soma os pontos de um conjunto de peças
int Pontos(int *v, int pecas[28][2])
{
  int i, total=0, n=v[0];
  for(i=1;i<=n;i++)
    total+=pecas[v[i]][0]+pecas[v[i]][1];
  return total;
}

////////////////////////////////////////
// funções base para resposta à alínea D2

// identicar a Jogada, mas com o jogador artificial
int JogadaArtificial(int mesa[2][28], int *jogador,
  int nJogador, int pecas[28][2])
{
  int i, j, resultado, melhorJogada=0, nivel=-1, ladoMelhor=-1, esquerda=0, direita=0;
  int nPecas[7], nMesa=mesa[0][0], n1,n2;
  int tPontos[28];

  // número de peças de cada número
  for(i=0;i<7;i++)
    nPecas[i]=0;
  for(i=1;i<=jogador[0];i++) {
    nPecas[pecas[jogador[i]][0]]++;
    if(pecas[jogador[i]][0]!=pecas[jogador[i]][1])
      nPecas[pecas[jogador[i]][1]]++;
  }

  LadosMesa(mesa,pecas,&esquerda,&direita);
  if(esquerda<0)
```

```
 esquerda=direita=0;

// calcular pontos de cada alternativa
for(i=1;i<=jogador[0];i++)
 tPontos[i]=pecas[jogador[i]][0]+pecas[jogador[i]][1];

// ver as alternativas (mais pontos primeiro)

while(1) {
 // calcular a peça com mais pontos
 for(i=1, j=2;j<=jogador[0];j++)
  if(tPontos[j]>tPontos[i])
   i=j;
 // tudo processado
 if(tPontos[i]<0)
  break;
 else
  tPontos[i]=-1; // marcar como processado

 resultado=Mesa(mesa, jogador[i], pecas, 0);
 // doble, nível 5
 if(resultado>0 && pecas[jogador[i]][0]==pecas[jogador[i]][1] && nivel<5) {
  nivel=5;
  melhorJogada=i;
  ladoMelhor=resultado;
 }

 if(resultado==3) { // fica um só número (iguais a esquerda/direita)
  if(nPecas[esquerda]>1) { // joga à direita
   nivel=6;
   melhorJogada=i;
```

```
    ladoMelhor=2;
  } else if(nPecas[direita]>1) { // joga à esquerda
    nivel=6;
    melhorJogada=i;
    ladoMelhor=1;
  } else if(nivel<2) { // apenas se não existir melhor
    nivel=2;
    melhorJogada=i;
    ladoMelhor=1;
  }
 } else if(resultado>0) {
  if(resultado==1) { // fica o número da direita mais o outro da peça
    n1=direita;
    if(pecas[jogador[i]][0]==esquerda)
      n2=pecas[jogador[i]][1];
    else
      n2=pecas[jogador[i]][0];
  } else if(resultado==2) { // fica o número da esquerda mais o outro da peça
    n1=esquerda;
    if(pecas[jogador[i]][0]==direita)
      n2=pecas[jogador[i]][1];
    else
      n2=pecas[jogador[i]][0];
  }
  if(nivel<4 && nPecas[n1]>0 && nPecas[n2]>1) {
    // ficam 2 números, e possui peças para ambos os números (4)
    nivel=4;
    melhorJogada=i;
    ladoMelhor=resultado;
  } else if(nivel<3 && (nPecas[n1]>0 || nPecas[n2]>1)) {
    // ficam 2 números, e possui peças para um dos números (3)
```

```
        nivel=3;
        melhorJogada=i;
        ladoMelhor=resultado;
      } else if(nivel<1) {
        // ficam 2 números e não possui peças para nenhum dos números (1)
        nivel=1;
        melhorJogada=i;
        ladoMelhor=resultado;
      }
    }
  }
  if(nivel>0 && ladoMelhor==1)
    melhorJogada=-melhorJogada;
  return melhorJogada;
}


void main(int argc, char **argv)
{
  int pecas[28][2]; // para cada peça, o número da esquerda e direita
  int ordem[29]; // ordem pela qual as peças podem ser processadas
  int i,j,id, nJogadores, passou, pontos, melhor;

  int mesa[2][28]; // mesa, em que é preciso o id da peça, e orientação
  int jogador[4][28]; // 4 jogadores e para cada um o id das peças que têm
  int compra[28]; // ids das peças que podem ser compradas

  srand(time(NULL));

  ///////////
  // alínea A
```

```
//  PrintPeca(rand()%7,rand()%7);
//  return;

 ///////////
 // alínea B

 // iniciar as peças
 for(i=0,id=0;i<=6;i++)
   for(j=i;j<=6;j++) {
     pecas[id][0]=i;
     pecas[id][1]=j;
     id++;
   }

 // ordem das peças inicial
 for(i=0;i<28;i++)
   ordem[i]=i;

 Baralhar(ordem,28);

 // inserir o número de peças no início, para chamar PrintPecas
 // (estrutura necessária por causa das restantes alíneas, em que conjuntos de peças variam)
//  for(i=28;i>0;i--)
//    ordem[i]=ordem[i-1];
//  ordem[0]=28;

//  PrintPecas(ordem,pecas,NULL);
//  return;

 ///////////
```

```
 // alínea C

 // a mesa inicialmente não tem peças
 mesa[0][0]=mesa[1][0]=0;

 // inserir as peças por onde for possível (esquerda ou direita)
//  for(i=0;i<28;i++)
//    Mesa(mesa,ordem[i+1],pecas,3);
//  printf("\nMesa:");
//  PrintPecas(mesa[0],pecas,mesa[1]);
//  return;

 ////////////////////
 // alínea D (1 e 2)

 // determinar número de jogadores
 nJogadores=atoi(argv[1]);
 if(nJogadores<2)
   nJogadores=2;
 else if(nJogadores>4)
   nJogadores=4;

 InicioJogo(ordem,jogador,compra,nJogadores);

 // jogar até não ser possível mais
 i=0; // primeiro jogador
 passou=0;
 while(passou<nJogadores) {
   if(!Jogada(mesa, jogador[i], i, pecas, compra))
     passou++;
   else
```

```
    passou=0;
   // teste de vitória
   if(jogador[i][0]==0)
    break;
   i++;
   if(i>=nJogadores)
    i=0;
  }

  // determinar a vitória
  if(passou>=nJogadores) {
   // calcular vitória com base nos pontos
   printf("\nGanha jogador com menos pontos");
   for(j=0;j<nJogadores;j++) {
    printf("\nJogador %c: ",'A'+j);
    PrintPecas(jogador[j],pecas, NULL);
    printf(" %d pontos.",pontos=Pontos(jogador[j], pecas));
    if(j==0 || pontos<melhor) {
     melhor=pontos;
     i=j;
    }
   }
  }
  // colocar a mesa final
  printf("\nMesa:");
  PrintPecas(mesa[0],pecas,mesa[1]);
  printf("\nVitoria do jogador %c!",'A'+i);
}
```