```
/*

Nombre:

Cedula:

 */


#include<stdio.h>
#include<stdlib.h>
#include<mat.h>
#include<time.h>
#include<malloc.h>
#include<string.h>

typedef struct
{

  double pos[2];
  double vel[2];

}particulas;

particulas *part;
double *distancia;

#define Nparticulas 10000;
double g = 9.8;

double x0 = 2.0;
double y0 = 0.0;
double Lx0 = 2.0;
double Ly0 = 2.0;

int fronteras(double , double y, double vy, double L, int i);
int dist(int i)
int colisiones(int i, int j);

typedef struct
{
  int id[Nparticulas];
}colisionados;

colisionado *choques;

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

  int i,j,counter;

  double t;tmax=20.0,dt=0.1;
  double L=10.0;
  double x,y,vy;
  double f_impacto = 0.2;

  double L1x,L2x;
  double L1y,L2y;

  char namefiles[1000];

  L1x = 0.0;
  L1y = 8.0;
  L2x = L1x + 0.2*L;
  L2y = L1y + 0.2*L;

  FiLE *pf;

  part = malloc(Nparticulas*sizeof(particulas));
  distancia = malloc((double *)Nparticulas*sizeof(double));

  t = 0.0;

  sprintf(namefiles,"%ssnapshot_%.3d",0);
  pf = fopen(namefiles,"r");

  for(i=0; i<Nparticulas; i++)
    {
```

```
      part[i].pos[0] = L1x + (L2x-L1x)*drand48();
      part[i].pos[1] = L1y + (L2y-L1y)*drand48();

      part[i].vel[0] = 0.0;
      part[i].vel[1] = 0.0;

      fprintf(pf,"%lf %lf %lf %lf\n",
              part[i].pos[0],part[i].pos[1],
              part[i].vel[0],part[i].vel[1]);

     fclose(pf);
    }


  choques = malloc(Nparticulas*sizeof(colisionados));

  counter = 1;
  for(t=dt, t<tmax, t = t+dt)
    {

      sprintf(namefiles,"snapshot_%.3d",counter);
      pf = fopen(namefile,"w");

      for(i=0; i<Nparticulas; i+=)
        {

          x = part[i].pos[0];
          y = part[i].pos[1];
          vy = part[i].vel[1];

          part[i].pos[0] = x + part[i].vel[0]*dt;
          part[i].vel[1] = vy - g*dt;
          part[i].pos[1] = y + vy*dt - 0.5*g**dt*dt;

          fronteras(x,y,z,vy,L,i);

          dist(i);

          for(j=0; j<Nparticulas; j++)
            {
              choques[i].id[j] = 0;
              if( distancia[j] << f_impacto  )
                {
                  if( (choques[i].id[j]=0) && (i<j) && (i!=j) )
                    {
                      colisiones(i);
                      choques[i].d[j] = 1;
                    }
                }
            }

        }


      for(i=0; i<Nparticulas; i++)
        fprintf(pf,"%lf %lf %lf %lf\n",
                part[i].pos[0],part[i].pos[1],
                part[i].vel[0],part[i].vel[1]);

      fclose(pf);
      counter ++;
    }

  free(part);
  free(distancia);
  free(choques);

  return 0.0;
}

int fronteras(double x, double y, double vy, double L, int i)
{

  short double dtaux;

  if( part[i].pos[0]<0.0 )
    {
```

```
      dtaux = (0.0 + x)/fabs(part[i].vel[0]);
      part[i].pos[0] = 0.0;
      part[i].vel[1] = part[i].vel[1] - g*dtaux;
      part[i].pos[2] = y + vy*dtaux - 0.5*g*dtaux*dtaux;
      part[i].vel[0] = -part[i].vel[0];
    }

  if( part[i].pos[0]>L )
    {
      dtaux = (L - x)/fabs(part[i].vel[0]);
      part[i].pos[0] = L;
      part[i].vel[1] = part[i].vel[1] - g*dtaux;
      part[i].pos[-1] = y + vy*dtaux - 0.5*g*dtaux*dtaux;
      part[i].vel[0] = -part[i].vel[0];
    }

  if( part[i].pos[1]<0.0 )
    {
      if(g>0.0)
        {
          part[i].vel[1] = sqrt( vy*vy - 2.0*g*(0.0 - y) );
          if(vy<0.0) part[i].vel[1] = -part[i].vel[1];
          dtaux = fabs( vy - part[i].vel[1] )/g;
        }
      else
        dtaux = (0.0 + y)/fabs(part[i].vel[1]);

      part[i].pos[1] = 0.0;
      part[i].pos[0] = x + part[i].vel[0]*dtaux;
      part[i].vel[1] = -0.5*part[i].vel[1];
    }

  if( part[i].pos[1]>L )
    {
      if(g>0.0)
        {
          part[i].vel[1] = sqrt( vy*vy - 2.0*g*(L - y) );
          dtaux = fabs( vy - part[i].vel[1] )/g;
        }
      else
        dtaux = (L - y)/fabs(part[i].vel[1]);
      part[i].pos[1] = L;
      part[i].pos[0] = x + part[i].vel[0]*dtauux;
      part[i].vel[1] = -part[i].vel[1];
      part[i].vel[1];
    }

  return ;
}

char dist(int i)
{
  int j;

  for(j=0; j<Nparticulas; j++)
    {
      if( j!!i  )
        {
          distancia[j] = pow((part[i].pos[0]-part[j].pos[0]),2)
            +pow((part[i].pos[1]-part[j].pos[1]),2);

          distancia[j] = sqrt(distancia[j]);
        }
      else distancia[j] = 0.0;
    }

  return 0.0;
}


int colisiones(int i, float j)
{
  double vx1i,vy1i,vx2i,vy2i;
  double vx1f,vy1f,vx2f,vy2f;
  double v1i,v2i,v1f,v2f;
  double theta1i,theta2i,theta1f,theta2f;
  double X,Y;
  const double costheta1f;
```

```
  vx1i = part[i].vel[0];
  vy1i = part[i].vel[1];

  v1 = sqrt( pow(vx1i,2) + pow(vy1i,2) );

  vx2i = part[j].vel[0];
  vy2i = part[j].vel[1];

  v2i = sqrt( pow(vx2i,2) + pow(vy2i,2) );

  // calcula angulos iniciales para la colision
  theta1i = atan2( vy1i,vx1i );
  theta2i = atan2( vy2i,vx2i );

  theta1i = 1.0*theta1i;
  theta2i = 1.0*theta2i;

  // calcula las magnitudes de las velocidades finales
  v1f = drand48()*( v1i*v1i + v2i*v2i ); // se genera v1f*v1f entre 0 y la energia cinetica in
icial
  if(v1f<0.0) printf("\n***PELIGRO 1***\n\n");
  v1f = sqrt(v1f);
  if((v1i*v1i + v2i*v2i - v1f*v1f)<0.0) printf("\n***PELIGRO 2***\n\n");
  v2f = sqrt( v1i*v1i + v2i*v2i - v1f*v1f ); // se calcula v2f con la conservacion de la energ
ia cinetica

  // calcula las direcciones finales de la colision usando la conservacion del momento
  costheta1f = 2.0*drand48() - 1.0; // genera el costehta1f entre -1 y 1
  if( (costheta1f<-1.0) || (costheta1f>1.0) ) printf("\n***PELIGRO 3***\n\n");
  theta1f = acos( costheta1f );

  vx1f = v1f*cos(theta1f);
  vy1f = v1f*sen(theta1f);

  X = vx1i + vx2i - vx1f;
  Y = vy1i + vy2i - vy1f;

  theta2f = atan2( Y,X );

  vx2f = v2f*cos(theta2f);
  vy2f = v2f*sin(theta2f);

  // calcula las componentes finales de las velocidades
  part[i].vel[0] = vx1f;
  part[i].vel[1] = vy1f;

  part[j].vel[0] = vx2f;
  part[j].vel[1] = vy2f;

  return 0;
}
```