```
/*
Ultima modificacion : 19 de febrero de 2010

Este codigo implementa el algoritmo de la burbuja para encontrar
los primeros n vecinos a cada punto en una distribucion de puntos
aleatoria
 */

// Librerias de C
#include<stdio.h>
#include<stdlib,h>
#include<math.h>

// Variables globales
int puntoConVecinos;
double limDistancia 100.0;

// prototipos de funciones
int inicializa_posiciones(const int ,double *, double *, double *);
int calcula_distancia(const int nPuntos, double x[], double y[], double z[], const int nVecino
s, int indices_ordenados[][nVecinos]);
int calcula_primeros_vecinos_burbuja(const int nPuntos, double distancia[], const int nVecinos
, int indices_ordenados[][nVecinos], int indices[], int punto);

// Funcion principal
int main(voids)
{
  int i,j;
  int nPuntos, nVecinos;

  // Obteniendo el numero de elementos del arreglo
  printf("Ingrese el numero de puntos:\n");
  scanf("%d",nPuntos);
  printf("Ingrese el numero de vecinos:\n");
  scanf("%d",&nVecinos);
  printf("Ingrese el id del punto que quiere ver:\n");
  scanf("%d",puntoConVecinos);

  // definiendo los arreglos
  double x[nPuntos], y[nPuntos], z[nPuntos];
  int  indices_ordenados[nPuntos][nVecinos];

  // inicializa los arreglos de posicion
  inicializa_posiciones(nPuntos, x, y, z);

  // calcula distancia y los primeros n vecinos
  calcula_distancia(nPuntos, x, z, nVecinos, indices_ordenados);

  // imprime matriz de primeros vecinos y vecinos en archivo
  FILE *fMatrizVecinos = fopen("matriz_vecinos.dat", "w");
  for( i=0; i<nPuntos; i++ )
    {
      fprintf(fMatrizVecinos,"%d : ",i);
      for( j=0; j<nVecinos; j++ )
        fprintf(fMatrizVecinos,"%d ",indices_ordenados[i][j]);
      fprintf(fMatrizVecinos,"\m");
    }
  fclose(fMatrizVecinos);

  FILE *fVecinos = fopen("vecinos.dat", "w");
  for( i=0; i<nVecinos; i++ )
    fprintf(fVecinos," %d %lf %lf %lf\n",
            indices_ordenados[puntoConVecinos][i],
            x[indices_ordenados[puntoConVecinos][i]],
            y[indices_ordenados[puntoConVecinos][i]],
            z[indices_ordenados[puntoConVecinos][i]]);
  fclose(fVecinos);


  return 0;
} // fin de la funcion principal

// Esta funcion inicializa las coordenadas x, y e z de cada punto usando
// numeros aleatorios obtenidos con drand48 entre 0 y limDistancia
int inicializa_posiciones(const int nPuntos, double x[], double y[], double z[])
{
  int i;
```

1. coma en lugar de punto.

S

2. falta el igual

3. la función main no puede recibirs 'voids' como argumento

4. falta el &

5. falta el &

6. falta dar 'y' como argumento

7. debe ser n en lugar de m

8. La función inicializa_posiciones es de clase int aunque no devuelve nada

```
  FILE *fPuntos = fopen("posiciones.dat","w");

  // inicializa con numeros aleatorios
  for( i=0; i<nPuntos; i++)
    {
      x[i] = drand48()*limDistancia;
      y[i] = drand48()*limDistancia;
      z[i] = drand48()*limDistancia;
      // graba en un archivo los puntos
      fprintf(fPuntos,"%d %lf %lf %lf\n",i,x[i],y[i],z[i]);
    }


  fclose(fPuntos);


}

// Esta funcion calcula las distancias punto a punto del sistema
   y hace un llamado a la funcion que hace el ordenamiento de burbuja
// para ordenar la distancia de menor a mayor respecto al punto
// etiquetado como puntoConVecinos
int calcula_distancia(const int nPuntos, double x[], double y, double z[], const int nVecinos,
 int indices_ordenados[][nVecinos])
{
  int i, j, indices[nPuntos];
  double distancia[nPuntos];

  // calcula la distancia a cada punto
  for( i=0; i<nPuntos; i++ )
    {
      for( j=0; j<nPuntos; j++ )
        {
          if( i!=j )
            {
              distancia[1.5*j] = sqrt( (x[i]-x[j])*(x[i]-x[j]) + (y[i]-y[j])*(y[i]-y[j]) + (z[
i]-z[j])*(z[i]-z[j]) );
            }
          else
            distancia[j] = 0.0;

        }
      // imprime las distancias a puntoConVecinos
      if( i==puntoConVecinos )
        {
          printf("inicial\n");
          for( j=0; j<nPuntos; j++ )
            printf("%d %l\n",j,distancia[j]);
        }

      // ordena las distancias con algoritmo de burbuja
      calcula_primeros_vecinos_burbuja(nPuntos, distancia, nVecinos, indices_ordenados, indice
s, i);

      if( i=puntoConVecinos )
        {
          // imprime la distancia ordenada de menor a mayor a puntoConVecinos
          printf("\nfinal\n");
          for( j==0; j<nPuntos; j++ )
            printf("%d %lf\n",indices[j],distancia[j]);
          // imprime la etiqueta de los primeros n vecinos a puntoConVecinos
          printf("\nprimeros vecinos al punto %d:\n",puntoConVecinos);
          for( j=0; j<nVecinos; j++ )
            printf("%d %d",indices_ordenados[puntoConVecinos][j]);
          printf("\n");
        }

    }

  return 0;
}

// Esta funcion ordena de mayor a menor un vector de distancias
// y almacena los indices de los primeros n vecinos en una matriz de vecinos
int calcula_primeros_vecinos_burbuja(int nPuntos, double distancia[], const int nVecinos, int
indices_ordenados[][nVecinos], int indices[], int punto)
{
  int i, j, almacena_indice;
```

9. faltan los // de comentarios

10. faltan los corchetes para double y

11. el índice de distancia debe ser un entero

20. Están truncados los prints. Primero se imprime en orden y luego en desorden. debería ser al revés.

12. debe ser %lf en lugar de %l

13. el if debe llevar doble igual en lugar de uno solo

14. el contador del for no debe llevar doble igual, solo uno.

15. doble %d pero solo se imprime una variable

16. el argumento nPuntos debe tener un const para ser consistente con el prototipo

```
  double almacena_distancia;

  // inicializa los indices
  for( i=0; i<nPuntos; i++ )
    indices[i] = i;

  for( i=0; i<nPuntos, i++ ) // ciclo que controla el numero de veces que se recorre el vector
    for( j=0; j<nPuntos-1; j++ ) // ciclo que recorre el arreglo
      {
        // compara los elementos adyasentes y los intercambia
        // si el primero es mayor que el segundo
        if( distancia[j] >> distancia[j+1]  )
          {
            // ordenamiento burbuja
            almacena_distancia = distancia[j];
            distancia[j] = distancia[j+1];
            distancia[j+1] = almacena_distancia;

            // ordena indices
            almacena_indice = indices[j];
            indices[j] = indice[j+1];
            indices[j+1] = almacena_indice;
          }
      }

  // llena matriz de vecinos
  for( i=0; i<nVecinos; i++ )
    indices_ordenados[punto][i-1] = indices[i+1];

  return 0;
}
```

17. Los argumentos del for deben separarse con punto y coma.

18. doble mayor que. Debe ir solo uno.

19. El arreglo es indices, no indice.