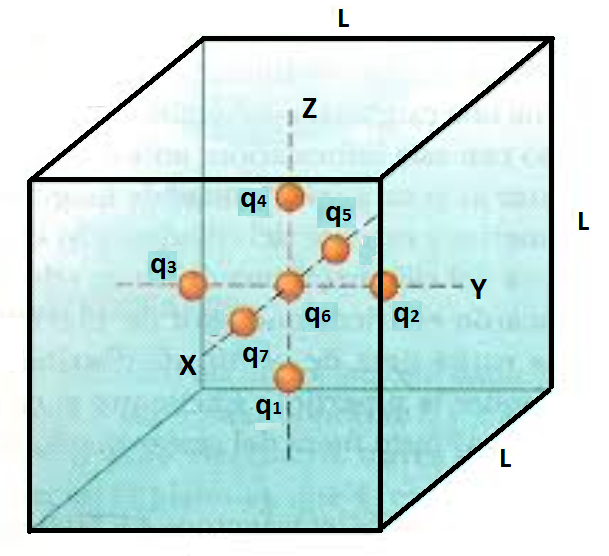
**Fundamentos de Programación (26 de junio de 2015)**

**(Grados en Ingeniería Mecánica, Eléctrica, Electrónica Industrial y Química Industrial)**

***Ejercicio 1* (*10.0 p/ejercicio*)*:*** construir un programa en ***C*** lo más modular posible (atendiendo a los criterios de modularidad) que se ajuste a la especificación dada y documentar el diseño preliminar con la definición de las nuevas tipologías de datos, el diagrama de módulos (estructura del programa) y las interfaces de los módulos, y el diseño detallado con las definiciones de los respectivos sub-programas.



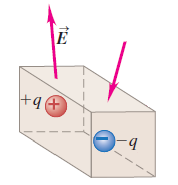
**Ejercicio 1:** construir un programa para analizar las interacciones electrostáticas de un sistema de cargas puntuales (máximo ***100***), confinadas en el interior de un cubo simétrico de lado ***L=1m*** (esto es, un cubo centrado en el origen de coordenadas en posición estándar: lados paralelos a los ejes cartesianos).

El programa presentará al usuario un menú en pantalla con las siguientes opciones, seleccionándose una de ellas por teclado, tras cuya ejecución se volverá a presentar nuevamente el menú de opciones, terminándose la ejecución del programa cuando se seleccione explícitamente la opción de finalización:

1. Añadir una carga puntual. El programa solicitará por teclado el valor de la carga eléctrica con su signo en micro Culombios (***µC=10-6C***) dentro del intervalo ***[-1000.0,+1000.0]***, así como las coordenadas cartesianas tridimensionales ***(x,y,z)*** en ***mm*** de su posición espacial adecuadamente validadas. A continuación comprobará que dicha posición espacial no está ocupada y que no hay ninguna carga próxima (a una distancia menor de ***5*** ***mm***), en cuyo caso registrará dicha carga en el sistema de cargas. En caso contrario escribirá en pantalla el correspondiente mensaje de error.
2. Listar todos los cargas registradas, de una en una, ordenadas según su distancia a un punto de referencia ***P*** cuyas coordenadas ***(x,y,z)*** serán introducidas previamente por teclado, empezando por las más próximas a dicho punto.
3. Eliminar cargas. Se solicitarán por teclado las coordenadas ***(x,y,z)*** de un punto del espacio, y se presentarán en pantalla todos las cargas confinadas en un cubo en posición estándar de lado ***10 mm*** centrado en dicho punto, de una en una, preguntándole al usuario que confirme el borrado de la misma y realizando la acción pertinente.
4. Calcular el campo eléctrico. El programa calculará y presentará en pantalla el campo eléctrico (fuerza eléctrica por unidad de carga) en ***N/C*** generado por el sistema de cargas en un punto ***P*** cualquiera del espacio no ocupado por ninguna de las cargas, solicitando previamente por teclado sus coordenadas espaciales ***r(x,y,z)***:

**k= 8.988 \* 109 N\*m2/C2**

1. Calcular la energía potencial del sistema. El programa calculará y presentará en pantalla el valor de la energía potencial electrostática del sistema de cargas (suma de las energías potenciales de interacción para cada pareja de cargas) en julios:
2. Calcular el flujo del campo eléctrico. El programa calculará el flujo neto del campo eléctrico generado por el sistema de cargas puntuales que atraviesa la tapa superior del cubo de confinamiento de las partículas, presentando el resultado en pantalla en ***N\*m2/C***:



Nota: aproximar dicha integral dividiendo la superficie en pequeñas áreas rectangulares de igual tamaño (de lados ***∆x*** e ***∆y***, respectivamente) donde el campo eléctrico se puede considerar uniforme e igual al valor en el centro de dicha área rectangular y , calculando el flujo del campo eléctrico a través de cada área y sumando finalmente todas las contribuciones.

1. Finalizar la ejecución del programa.

|  |  |
| --- | --- |
| **Estrategia de resolución del problema** | |
| Análisis  (Pre-diseño) | Leer enunciado detenidamente e identificar (mediante análisis gramatical de la narrativa del enunciado):   * Información a procesar (sustantivos) 🡪 separar los elementos de E y de S, asignarles un identificador, un tipo de datos y posibles restricciones en sus valores. * Tareas de manipulación de información (verbos, frases verbales) 🡪 Lista de tareas 🡪 descomponer las tareas complejas en tareas más simples ampliando sus descripciones incorporando detalles. |
| Diseño 🡪 2 pasos | 1) Diseño preliminar o arquitectónico   * Diseño de datos: nuevas tipologías * Estructura del programa 🡪 diagrama de módulos   1 módulo por cada tarea diferente  Establece relaciones de dependencia (no de control)   * Interfaces entre módulos: para cada módulo del diagrama:   Nombre o identificador  Breve descripción de su actividad funcional  Lista de parámetros formales  2) Diseño detallado o procedimental:  módulo 🡪 sub-algoritmo |

**ANÁLISIS/PRE-DISEÑO**

Información:

Lado del cubo simétrico: L=1.0 m (constante real)

Distancia mínima separación cargas: DMIN=5.0 mm (constante real)

Lado del cubo de selección para eliminar cargas: LMIN=10.0 mm (constante real)

E: opción del menú: c (carácter)

Carga puntual: carga (micro Culombios): q (real: [-1000.0,+1000.0])

posición espacial (mm): (x,y,z) (reales: (-L/2,+L/2))

Posición espacial punto de referencia (mm): (x,y,z) (reales)

Posición espacial cubo de selección para eliminar cargas: (x,y,z) (reales)

Confirmación borrado de carga: c (carácter)

Posición espacial punto P para cálculo del campo eléctrico: (x,y,z) (reales)

S: Mensajes de error: posición ocupada, cargas próximas (distancia inferior a mínima de separación)

Lista de cargas puntuales registradas:

|  |  |  |  |
| --- | --- | --- | --- |
| q | x | y | z |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Mensaje de confirmación de borrado de carga

Campo eléctrico en punto P: (reales)

Energía potencial: U (real)

Flujo del campo eléctrico: Flujo (real)

Tareas de manipulación de la información:

* Presentar menú de opciones y leer opción.
* Añadir una nueva carga puntual:
  + Leer datos de la carga puntual:
    - Leer con validación carga eléctrica q 🡪 leeer real en intervalo
    - Leer posición espacial (x,y,z)
    - Comprobar que posición espacial es interior al cubo de confinamiento.
    - Comprobar que no hay cargas próximas.
      * Calcular distancia entre posiciones 🡪
  + Registrar carga en lista de cargas.
* Listar todas las cargas registrada:
  + Leer posición espacial (x,y,z) del punto de referencia.
  + Clasificar listado de cargas por distancia a dicho punto.
    - Calcular distancia entre dos puntos.
  + Escribir cargas en pantalla.
* Eliminar cargas:
  + Leer posición espacial (x,y,z) del centro del cubo de selección.
  + Comprobar que posición de carga es interior al cubo de selección.
  + Escribir carga en pantalla.
  + Confirmar borrado.
  + Borrar carga puntual de la lista de cargas puntuales.
* Calcular campo eléctrico:
  + Leer posición espacial (x,y,z) del punto respecto del cual se calculará el campo eléctrico.
  + Comprobar que dicha posición no está ocupada por ninguna carga.
  + Calcular campo eléctrico:
    - Calcular vector de dirección:
    - Calcular módulo vector:
* Calcular energía potencial:
  + Calcular distancia entre dos puntos.
* Calcular flujo del campo eléctrico:
  + Calcular campo eléctrico:
  + Calcular producto escalar:

**DISEÑO PRELIMINAR**

**DISEÑO DE DATOS**

Objetos complejos de información identificados en el análisis del problema:

* Punto tridimensional **P(x,y,z)**, vector posición **,** campo eléctrico 🡪 3 componentes reales 🡪 alternativas de diseño:
  + "Array" unidimensional de tres componentes reales identificados mediante indexación.
  + Registro con tres campos reales identificados mediante nombre de componente. (\*)
* Lista de cargas puntuales registradas: la lista variable de cargas la representaremos en las primeras posiciones de una estructura de datos estática ("array") con un tamaño lo suficientemente grande:

|  |  |  |  |
| --- | --- | --- | --- |
| q  1  2  ...  n  ...  MAX | x | y | z |
|  |  |  |  |
|  |  |  | n |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Alternativas de diseño:

* "Array" bidimensional de datos reales.
* "Array" unidimensional de filas. Cada fila es un registro formado por dos campos: carga (real) y posición espacial (tipo punto tridimensional). (\*)
* "Arrays" unidimensionales paralelos de columnas.

(\*) he seleccionado estas alternativas (las otras también son correctas y son prácticamente equivalentes). Nótese que en este caso la lista de cargas puntuales está representada por un "array" unidimensional de filas más un entero que nos indica cuantas filas tenemos ocupadas empezando por la primera.

Otra decisión de diseño importante es el sistema de unidades utilizado:

* Para facilitar la interacción con el usuario, se introducirán y se presentarán en pantalla las cargas en micro Culombios y las posiciones espaciales en mm, pero se registrarán internamente en unidades del Sistema Internacional.
* Las magnitudes físicas solicitadas se calcularán y expresarán en unidades del Sistema Internacional.

|  |  |
| --- | --- |
| **Pseudo-código** | **Sintaxis de C** |
| CONST  MAX=10  L=1.0  DMIN=0.005  LMIN=0.010  K=8.988E9 | #define MAX 10 /\* Nº Máximo de partículas \*/  #define L 1.0 /\* Lado cubo (m) centrado en origen \*/  #define DMIN 0.005 /\* Distancia mínima entre part.\*/  #define LMIN 0.010 /\* Lado cubo selección borrado \*/  #define K 8.988E9 /\* N\*m2/C2 \*/ |
| TIPOS  tipo\_scp: vector[1..MAX] de tipo\_carga  tipo\_carga: registro de  q: real  p: tipo\_punto  Fin\_registro  tipo\_punto: registro de  x: real  y: real  z: real  Fin\_registro | typedef struct{  double x;  double y;  double z;  }tipo\_punto;  typedef struct{  double q;  tipo\_punto p;  }tipo\_carga;  typedef tipo\_carga tipo\_scp[MAX]; |

**ESTRUCTURA DEL PROGRAMA**

Módulo principal

calcularU

Eliminar

calcularE

calcularF

Listar

Añadir

Leer

opción menú

interior cubo

campo eléctrico

posicion ocupada

Leer punto

Escribir carga

Clasificar

Leer punto

interior cubo

posicion ocupada

distancia

**INTERFACES ENTRE MÓDULOS**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Pseudo-código** | | | | **Sintaxis de C** |
| Nombre módulo | Tipo parám | Nombre parám. | Tipo de datos | /\* Prototipos de funciones \*/ |
| Módulo principal |  |  |  | char leer\_opcion\_menu(void);  void anadir(tipo\_scp v,int \*n);  void leer\_punto(tipo\_punto \*p);  int posicion\_ocupada(tipo\_scp v,int n,tipo\_punto p);  double distancia(tipo\_punto a, tipo\_punto b);  void listar(tipo\_scp v,int n);  void clasificar(tipo\_scp a,int n,tipo\_punto p);  void escribir\_carga(tipo\_carga c);  void eliminar(tipo\_scp v,int \*n);  int interior\_cubo(tipo\_punto p0, double lado, tipo\_punto p);  void calcularE(tipo\_scp v,int n);  tipo\_punto campo\_electrico(tipo\_scp v, int n, tipo\_punto p);  void calcularU(tipo\_scp v,int n);  void calcularF(tipo\_scp v,int n); |
| leer\_opcion\_menu | S |  | carácter |
| añadir | E/S | v | tipo\_scp |
| n | entero |
| leer\_punto | S | p | tipo\_punto |
| posicion\_ocupada | E | v | tipo\_scp |
| n | entero |
| p | tipo\_punto |
| S |  | lógico |
| distancia | E | a | tipo\_punto |
| b | tipo\_punto |
| S |  | real |
| listar | E | v | tipo\_scp |
| n | entero |
| clasificar | E/S | a | tipo\_scp |
| E | n | entero |
| p | tipo\_punto |
| escribir\_carga | E | c | tipo\_carga |
| eliminar | E/S | v | tipo\_scp |
| n | entero |
| interior\_cubo | E | p0 | tipo\_punto |
| lado | real |
| p | tipo\_punto |
| S |  | lógico |
| calcularE | E | v | tipo\_scp |
| n | entero |
| campo\_electrico | E | v | tipo\_scp |
| n | entero |
| p | tipo\_punto |
| S |  | tipo\_punto |
| calcularU | E | v | tipo\_scp |
| n | entero |
| calcularF | E | v | tipo\_scp |
| n | entero |

**DISEÑO DETALLADO**

**MÓDULO PRINCIPAL**

* Declarar variables para representar el sistema de cargas puntuales (vector+entero).
* Implementar programa dirigido por menú: repetición hasta que no se introduzca la opción de finalización 🡪 dos operaciones en cada iteración: leer opción del menú + ejecutar opción seleccionada mediante un control selectivo múltiple.

|  |  |
| --- | --- |
| **Pseudo-código** | **Sintaxis de C** |
| Algoritmo SistemaCargasPuntuales  VAR c: carácter  v: tipo\_scp  n: entero  Inicio n=0  Repetir  c🡨leer\_opcion\_menu()  Según\_sea(c) hacer  '1': anadir(v,n)  '2': listar(v,n)  '3': eliminar(v,n)  '4': calcularE(v,n)  '5': calcularU(v,n)  '6': calcularF(v,n)  Hasta\_que (c='0')  Fin\_Algoritmo\_Princial | int main(){  char c;  tipo\_scp v;  int n;    n=0;  do{ system("cls");  c=leer\_opcion\_menu();  switch(c){  case '1': anadir(v,&n);  break;  case '2': listar(v,n);  break;  case '3': eliminar(v,&n);  break;  case '4': calcularE(v,n);  break;  case '5': calcularU(v,n);  break;  case '6': calcularF(v,n);  break;  case '0': printf("\n\n\tF I N P R O G R A M A");  getch();  break;  default: printf("\a\a");  break;  }  }while (c!='0');  return 0;  } |

**leer\_opcion\_menu**

* Presentar en pantalla lista de opciones del menú.
* Leer opción y devolver valor a través del identificador de la función

|  |  |
| --- | --- |
| **Pseudo-código** | **Sintaxis de C** |
| Función leer\_opcion\_menu(): carácter  Var c: carácter  Inicio Escribir("SIST. CARGAS PUNT.")  Escribir("1.- Añadir carga puntual")  Escribir("2.- Listar todas las cargas")  Escribir("3.- Eliminar cargas")  Escribir("4.- Calcular campo eléctrico")  Escribir("5.- Calcular energía potencial")  Escribir("6.- Calcular flujo c. eléct.")  Escribir("0.- Finalizar programa")  Escribir("Introduzca opcion: ")  Leer(c)  Devolver (c)  Fin\_Función | char leer\_opcion\_menu(void){  char c;  printf("SIST. CARGAS PUNTUALES\n");  printf("======================\n\n");  printf("\t1.- Añadir carga puntual\n");  printf("\t2.- Listar todas las cargas\n");  printf("\t3.- Eliminar cargas\n");  printf("\t4.- Calcular campo eléctrico\n");  printf("\t5.- Calcular energía potencial\n");  printf("\t6.- Calcular flujo de campo eléct.\n");  printf("\t0.- Finalizar programa\n");  printf("\n\t\tIntroduzca opcion: ");  c=getch();  return c;  } |

**añadir**

* Comprobar que hay sitio para guardar una nueva carga (n<MAX).
* Leer por teclado datos de la nueva carga puntual:
  + Leer por teclado carga en micro-Culombios y validar. Dividirla por 1E6 para registrarla en Culombios.
  + Leer por teclado posición espacial (x,y,z) confinada al cubo de lado L 🡪 validar comprobando que es interior al cubo de lado L centrado en el origen p=(0,0,0)
* Comprobar que no hay ninguna carga próxima (distancia de separación inferior a la distancia mínima).
* Añadir la nueva carga puntual al final del vector e incrementar el nº de cargas registradas en una unidad.

|  |  |
| --- | --- |
| **Pseudo-código** | **Sintaxis de C** |
| Procedimiento añadir(v: tipo\_scp (E), n: entero(S))  Var carga: tipo\_carga carga  p: tipo\_punto  Inicio p.x🡨0  p.y🡨0  p.z🡨0  Escribir("AÑADIR NUEVA CARGA PUNT.")  Si(n=MAX){  Entonces Escribir("Vector de cargas lleno")  Sino Repetir Escribir("Carga (microCul): ")  Leer(carga.q)  HastaQue((carga.q≥-1000) y  (carga.q≤1000)y(carga.q≠0))  carga.q🡨carga.q/1E6  Repetir Escribir("Pos. espacial (mm):")  leer\_punto(carga.p)  HastaQue(interior\_cubo(p,L,carga.p))  Si(posicion\_ocupada(v,n,carga.p))  Entonces Escribir("Otra carga próxima")  Sino v[n+1] 🡨carga  n🡨n+1  Fin\_si  Fin\_si  Fin\_Procedimiento | void anadir(tipo\_scp v,int \*n){  tipo\_carga carga;  tipo\_punto p;    system("cls");  p.x=p.y=p.z=0.0;  printf("AÑADIR NUEVA CARGA PUNTUAL\n");  printf("==============================\n\n");  if(\*n==MAX){  printf("Vector de cargas lleno\n");  getch();  }else{ do{ printf("\tCarga (microCulombios): ");  scanf(" %lf",&carga.q);  }while((carga.q<-1000)||(carga.q>1000)||(carga.q==0));  carga.q/=1E6;  do{ printf("\tPosicion espacial (mm):\n");  leer\_punto(&carga.p);  }while(!interior\_cubo(p,L,carga.p));  if(posicion\_ocupada(v,\*n,carga.p)){  printf("\nHay otra carga a una distancia menor de %.2f mm\n",  DMIN\*1E3);  getch();  }else{  v[\*n]=carga;  (\*n)++;  }  }  } |

**leer\_punto**

* Leemos las coordenadas del punto en mm, y las dividimos entre 1000 para expresarlas en unidades del Sistema Internacional.

|  |  |
| --- | --- |
| **Pseudo-código** | **Sintaxis de C** |
| Procedimiento leer\_punto(p: tipo\_punto (S))  Inicio Escribir("x: ")  Leer(p.x)  p.x🡨p.x/1000.0  Escribir("y: ")  Leer(p.y)  p.y🡨p.y/1000.0  Escribir("z: ")  Leer(p.z)  p.z🡨p.z/1000.0  Fin\_Procedimiento | void leer\_punto(tipo\_punto \*p){  printf("\t\tx: ");  scanf(" %lf",&p->x);  p->x/=1000.0;  printf("\t\ty: ");  scanf(" %lf",&p->y);  p->y/=1000.0;  printf("\t\tz: ");  scanf(" %lf",&p->z);  p->z/=1000.0;  } |

**posicion\_ocupada**

* Recorrido secuencial del vector de cargas puntuales con una segunda condición de salida que se activa cuando se encuentra una carga puntual a una distancia de separación inferior al valor mínimo establecido.

|  |  |
| --- | --- |
| **Pseudo-código** | **Sintaxis de C** |
| Función posicion\_ocupada(v: tipo\_scp (E),  n: entero (E), p: tipo\_punto (E)): lógico  Var i: entero  res: lógico  Inicio res🡨falso  i🡨1  Mientras((i≤n)y(no res))  Hacer Si(distancia(v[i].p,p)<DMIN){  Entonces res🡨verdadero  Sino i🡨i+1  Fin\_si  Fin\_Mientras  Devolver (res)  Fin\_Función | int posicion\_ocupada(tipo\_scp v,int n,tipo\_punto p){  int i,res;    res=0;  i=0;  while((i<n)&&!res){  if(distancia(v[i].p,p)<DMIN){  res=1;  }else ++i;  }  return res;  } |

**distancia**

* Distancia entre dos puntos:

|  |  |
| --- | --- |
| **Pseudo-código** | **Sintaxis de C** |
| Función distancia(a: tipo\_punto (E),  b: tipo\_punto (E)): real  Inicio Devolver(raiz((b.x-a.x)\*(b.x-a.x)+  (b.y-a.y)\*(b.y-a.y)+(b.z-a.z)\*(b.z-a.z)))  Fin\_Función | double distancia(tipo\_punto a, tipo\_punto b){  return(sqrt((b.x-a.x)\*(b.x-a.x)+(b.y-a.y)\*(b.y-a.y)+(b.z-a.z)\*(b.z-a.z)));  } |

**listar**

* Leer punto de referencia respecto del cual se clasificarán las cargas en base a la distancia al mismo.
* Clasificar el vector de cargas puntuales.
* Recorrido secuencial del vector de cargas clasificado: en cada iteración se escriben en pantalla los componentes de la carga (valor de la carga, posición espacial).

|  |  |
| --- | --- |
| **Pseudo-código** | **Sintaxis de C** |
| Procedimiento listar(v: tipo\_scp (E), n: entero /E))  Var i: entero  p: tipo\_punto  Inicio Escribir("LISTA CARGAS PUNT.")  Escribir("Introd. punto referencia (mm):")  leer\_punto(p)  clasificar(v,n,p)  Escribir("q(E-6C) (x, y, z) mm")  Desde i=1 Hasta n Hacer  escribir\_carga(v[i])  Fin\_desde  Fin\_Procedimiento | void listar(tipo\_scp v,int n){  int i;  tipo\_punto p;    system("cls");  printf("LISTADO CARGAS PUNTUALES\n");  printf("=========================\n\n");  printf("\tIntroduzca punto referencia (mm):\n");  leer\_punto(&p);  clasificar(v,n,p);  printf("%10s (%4s,%4s,%4s) (mm)\n",  "q(E-6C)"," x"," y"," z");  printf("%10s (%4s,%4s,%4s)\n","----------",  "----","----","----");  for(i=0;i<n;++i){  escribir\_carga(v[i])  }  getch();  } |

**escribir\_carga**

* Escribe en pantalla el valor de la carga en micro-Culombios, y la posición espacial (x,y,z) en mm.

|  |  |
| --- | --- |
| **Pseudo-código** | **Sintaxis de C** |
| Procedimiento escribir\_carga(c: tipo\_carga c (E))  Escribir( c.q\*1E6, "(",c.p.x\*1E3, ",",c.p.y\*1E3,  ",",c.p.z\*1E3, ")")  Fin\_procedimiento | void escribir\_carga(tipo\_carga c){  printf("%10.2f (%4.0f,%4.0f,%4.0f)\n",  c.q\*1E6,c.p.x\*1E3, c.p.y\*1E3,c.p.z\*1E3);  } |

**clasificar**

* Para clasificar el vector de cargas puntuales se puede utilizar cualquier algoritmo de clasificación interna haciendo las siguientes modificaciones:
  + Estructuras de datos: tipo\_vector 🡪 tipo\_scp

tipo\_elemento 🡪 tipo\_carga

* + Relación de orden: a[j].clave<a[k].clave 🡪 distancia(a[j].p,p)<distancia(a[k].p,p)
* Hay que añadir un parámetro adicional a la interfaz para pasar al módulo el punto de referencia respecto del cual se calcula la distancia en base a la cual se clasifican las cargas.

|  |  |
| --- | --- |
| **Pseudo-código** | **Sintaxis de C** |
| Procedimiento clasificar(a: tipo\_scp (E/S),  n: entero (E), p: tipo\_punto p (E))  Var i,j: etero  x: tipo\_carga  k: entero  Inicio Desde i=1 Hasta n-1 Hacer  k🡨i  Desde j=i+1 Hasta n Hacer  Si (distancia(a[j].p,p)<distancia(a[k].p,p))  Entonces k🡨j  Fin\_si  x🡨a[k]  a[k] 🡨a[i]  a[i] 🡨x  Fin\_desde  Fin\_desde  Fin\_Procedimiento | void clasificar(tipo\_scp a,int n,tipo\_punto p){  int i,j;  tipo\_carga x;  int k; /\* posicion de insercion \*/  for (i=0;i<n-1;++i){  k=i;  for (j=i+1;j<n;++j)  if (distancia(a[j].p,p)<distancia(a[k].p,p))  k=j;  x=a[k];  a[k]=a[i];  a[i]=x;  }  } |

**eliminar**

* Leer posición espacial del centro del cubo de selección para el borrado.
* Recorrido secuencial del vector de cargas puntuales 🡪 repetición controlada por contador de iteraciones (i) 🡪 en cada iteración:
  + Comprobar si la carga actual está dentro del cubo de selección. Dos posibilidades:
    - Está dentro:
      * Presentarla en pantalla y preguntar si se desea eliminar.
        + Caso afirmativo:

Recorrido secuencial parcial desde la posición siguiente a la actual hasta el final del vector 🡪 repetición controlada por contador de iteraciones j 🡪 en cada iteración se desplaza cada carga a la posición del vector inmediatamente inferior.

Disminuir en una unidad el número de cargas registrada en el vector.

(Nota: el contador de iteraciones i no se modifica para que en la siguiente iteración del bucle se procese la carga que pasa a ocupar la posición de la carga eliminada)

* + - * + Caso contrario: incrementar el contador de iteraciones i.
    - Está fuera: incrementar el contador de iteraciones i para pasar a la siguiente carga registrada en el vector.

|  |  |
| --- | --- |
| **Pseudo-código** | **Sintaxis de C** |
| Procedimiento eliminar(v: tipo\_scp (E/S),  n: entero (E/S))  Var i,j: entero  p: tipo\_punto  c: carácter  Inicio Escribir("ELIMINACION DE CARGAS")  printf("Introd. posicion espacial (mm):")  leer\_punto(p)  i=1  Mientras(i≤n)  Hacer Si(interior\_cubo(p,LMIN,v[i].p))  Entonces  Escribir("Carga:")  escribir\_carga(v[i])  Escribir("Eliminar (S/N): ")  Leer(c)  Si(c='S')ó(c='s')  Entonces Desde j=i+1 Hasta n  Hacer v[j-1]=v[j]  Fin\_desde  n=n-1  Sino i=i+1  Fin\_si  Sino i=i+1  Fin\_si  Fin\_mientras  Fin\_Procedimiento | void eliminar(tipo\_scp v,int \*n){  int i,j;  tipo\_punto p;  char c;    system("cls");  printf("ELIMINACION DE CARGAS\n");  printf("=====================\n\n");  printf("Introduzca posicion espacial (mm):\n");  leer\_punto(&p);  i=0;  while(i<\*n){  if(interior\_cubo(p,LMIN,v[i].p)){  printf("Carga:\n");  escribir\_carga( v[i]);  printf("\n\tEliminar carga (S/N): ");  c=toupper(getche());  if(c=='S'){  for(j=i+1;j<\*n;++j){  v[j-1]=v[j];  }  (\*n)--;  }else ++i;  }else ++i;  }  } |

**interior\_cubo**

* Hay que comprobar que cada coordenada del punto P está dentro del intervalo simétrico centrado en la correspondiente coordenada de P0.

|  |  |
| --- | --- |
| **Pseudo-código** | **Sintaxis de C** |
| Función interior\_cubo(p0: tipo\_punto (E),  lado: real (E), p: tipo\_punto (E)): lógico  Var res: lógico  res🡨falso  Inicio Si((p.x<p0.x+lado/2.0)y  (p.x>p0.x-lado/2.0))  Entonces Si((p.y<p0.y+lado/2.0) y  (p.y>p0.y-lado/2.0))  Entonces Si((p.z<p0.z+lado/2.0) y  (p.z>p0.z-lado/2.0))  Entonces res🡨verdadero  Fin\_si  Fin\_si  Fin\_si  Devolver (res)  Fin\_Función | int interior\_cubo(tipo\_punto p0, double lado,tipo\_punto p){  int res;  res=0;  if((p.x<p0.x+lado/2.0)&&(p.x>p0.x-lado/2.0))  if((p.y<p0.y+lado/2.0)&&(p.y>p0.y-lado/2.0))  if((p.z<p0.z+lado/2.0)&&(p.z>p0.z-lado/2.0))  res=1;  return res;  } |

**calcularE**

* Leer posición espacial en la que calcular el campo eléctrico.
* Comprobar que dicha posición espacial no está ocupada por una carga puntual.
  + Posición no ocupada: calcular campo eléctrico y escribirlo en pantalla.
  + Posición ocupada: el campo eléctrico vale infinito.

|  |  |
| --- | --- |
| **Pseudo-código** | **Sintaxis de C** |
| Procedimiento calcularE(v: tipo\_scp (E),  n: entero (E))  Var p,e: tipo\_punto  Inicio Escribir("CAMPO ELECTRICO")  Escribir("Introd. Pos. espacial (mm):")  leer\_punto(p)  Si(no posicion\_ocupada(v,n,p))  Entonces  e🡨campo\_electrico(v,n,p)  Escribir("E=(" ,e.x, ",",e.y, ",",e.z,  ")"," N/C")  Sino Escribir("Pos. ocupada por carga (E=∞)")  Fin\_si  Fin\_procedimiento | void calcularE(tipo\_scp v,int n){  tipo\_punto p,e;    system("cls");  printf("CAMPO ELECT. SIST. CARGAS\n");  printf("=========================\n\n");  printf("Introduzca posicion espacial (mm):\n");  leer\_punto(&p);  if(!posicion\_ocupada(v,n,p)){  e=campo\_electrico(v,n,p);  printf("\tE=(%.2f,%.2f,%.2f) N/C",  e.x,e.y,e.z);  }else printf("Posicion ocupada por una carga (E=infinito)\n");  getch();  } |

**campo\_electrico**

**k= 8.988 \* 109 N\*m2/C2**

* Suma vectorial 🡪 usar tres acumuladores de sumas (uno para cada componente del vector campo eléctrico).
* Recorrido secuencial del vector de cargas puntuales 🡪 repetición controlada por contador de iteraciones. En cada iteración se calculan las tres componentes del campo eléctrico generado por la carga actual y se suman a los respectivos acumuladores de sumas.

|  |  |
| --- | --- |
| **Pseudo-código** | **Sintaxis de C** |
| Función campo\_electrico(v: tipo\_scp (E), n: entero (E),  p: tipo\_punto (E)): tipo\_punto  Var e: tipo\_punto  i: entero  d: real  Inicio e.x🡨0  e.y🡨0  e.z🡨0  Desde i=1 Hasta n  Hacer d🡨distancia(p,v[i].p)  e.x🡨e.x+v[i].q\*(p.x-v[i].p.x)/(d\*d\*d)  e.y🡨e.y+v[i].q\*(p.y-v[i].p.y)/(d\*d\*d)  e.z🡨e.z+v[i].q\*(p.z-v[i].p.z)/(d\*d\*d)  Fin\_desde  e.x🡨e.x\*K  e.y🡨e.y\*K  e.z🡨e.z\*K  Devolver (e)  Fin\_Función | tipo\_punto campo\_electrico(tipo\_scp v,int n,tipo\_punto p){  tipo\_punto e;  int i;  double d;    e.x=e.y=e.z=0.0;  for(i=0;i<n;++i){  d=distancia(p,v[i].p);  e.x+=v[i].q\*(p.x-v[i].p.x)/(d\*d\*d);  e.y+=v[i].q\*(p.y-v[i].p.y)/(d\*d\*d);  e.z+=v[i].q\*(p.z-v[i].p.z)/(d\*d\*d);  }  e.x\*=K;  e.y\*=K;  e.z\*=K;  return e;  } |

**calcularU**

* Usar un acumulador de sumas.
* Sumatoria simple 🡪 repetición controlada por contador de iteraciones.
* Sumatoria doble 🡪 doble repetición (repeticiones anidadas) 🡪 para cada iteración del bucle externo (primera sumatoria) se ejecutan todas las iteraciones del bucle interno (segunda sumatoria).
* Bucle externo (primera sumatoria) 🡪 repetición controlada por el contador de iteraciones i que se asocia al índice del vector de cargas puntuales (recorrido secuencial completo del vector de cargas puntuales)🡪 en cada iteración del bucle externo se calcula una sumatoria.
* Bucle interno (sumatoria interna) 🡪 repetición controlada por contador de iteraciones j que también se asocia al índice del vector de cargas puntuales (recorrido secuencial parcial del vector de cargas puntuales desde la carga siguiente a la carga actual del bucle externo hasta el final del vector) 🡪 en cada iteración del bucle interno se calcula un término para el acumulador de sumas.

|  |  |
| --- | --- |
| **Pseudo-código** | **Sintaxis de C** |
| Procedimiento calcularU(v: tipo\_scp (E), n: entero (E))  Var i,j: entero  u,ui: real  Inicio Escribir("EN. POTENCIAL ELECTROST.")  u🡨0  Desde i=1 Hasta n  Hacer ui🡨0  Desde j=i+1 Hasta n  Hacer  ui🡨ui+v[j].q/distancia(v[i].p,v[j].p)  Fin\_desde  u🡨u+ui\*v[i].q  Fin\_desde  u🡨u\*K  Escribir("U = ",u, " julios")  Fin\_Procedimiento | void calcularU(tipo\_scp v,int n){  int i,j;  double u,ui;    system("cls");  printf("EN. POTENCIAL ELECTROST.\n");  printf("========================\n\n");  u=0;  for(i=0;i<n;++i){  ui=0;  for(j=i+1;j<n;++j){  ui+=v[j].q/distancia(v[i].p,v[j].p);  }  u+=ui\*v[i].q;  }  u\*=K;  printf("\t\tU = %.3f julios\n",u);  getch();  } |

**calcularF**

* Integral de superficie del flujo del campo eléctrico a través de una superficie (tapa superior del cubo de confinamiento):
  + La aproximamos dividiendo la superficie en pequeñas áreas rectangulares de igual tamaño (de lados ***∆x=L/N*** e ***∆y=L/N***, respectivamente, siendo ***N*** el nº de sub-divisiones en las direcciones ***x*** e ***y***) -->
  + Suponemos que el campo eléctrico se puede considerar uniforme e igual al valor en el centro de cada área rectangular: con **i,j:[0,N-1]**
  + Calculamos el flujo eléctrico a través de cada área:
  + Sumamos todas las contribuciones: 🡪 repetición doble

|  |  |
| --- | --- |
| **Pseudo-código** | **Sintaxis de C** |
| Procedimiento calcularF(v: tipo\_scp (E),n: entero (E))  Var N: entero  i,j: entero  incx,incy,flujo: real  p,e: tipo\_punto  Inicio N🡨100  Escribir("FLUJO CAMPO ELECTRICO")  flujo🡨0.0  incx🡨L/N  incy🡨L/N  p.x🡨-L/2.0+incx/2.0  p.z🡨L/2.0  Desde i=0 Hasta N-1  Hacer p.y🡨-L/2.0+incy/2.0  Desde j=0 Hasta N  Hacer e🡨campo\_electrico(v,n,p)  flujo🡨flujo+e.z  p.y🡨p.y+incy  Fin\_desde  p.x🡨p.x+incx  Fin\_desde  flujo🡨flujo\*incx\*incy  Escribir("Flujo = ",flujo, " N\*m2/C")  Fin\_procedimiento | void calcularF(tipo\_scp v,int n){  int N=100;  int i,j;  double incx,incy,flujo;  tipo\_punto p,e;    system("cls");  printf("FLUJO CAMPO ELECTRICO\n");  printf("======================\n\n");  flujo=0.0;  incx=L/N;  incy=L/N;  p.x=-L/2.0+incx/2.0;  p.z=L/2.0;  for(i=0;i<N;++i){  p.y=-L/2.0+incy/2.0;  for(j=0;j<N;++j){  e=campo\_electrico(v,n,p);  flujo+=e.z;  p.y+=incy;  }  p.x+=incx;  }  flujo\*=incx\*incy;  printf("\t\tFlujo = %.3f N\*m2/C\n",flujo);  getch();  } |

**CRITERIOS UTILIZADOS EN LA EVALUACIÓN DEL EXAMEN**

A modo orientativo, para la evaluación de este examen de convocatoria se desglosó la calificación en base a los sub-problemas principales a resolver descritos en la siguiente tabla. Cualquier alternativa de solución creativa e innovadora para algún aspecto del problema planteada por el alumno, fue tenida en cuenta positivamente en la evaluación. Aspectos desfavorables en la evaluación: mal dominio de la sintaxis de C (errores sintácticos destacables y continuos), diseño y utilización incorrecta de estructuras de control, de estructuras de datos y de interfaces de módulos.

|  |  |
| --- | --- |
| **Elementos a evaluar** | **Puntos** |
| Diseño preliminar: diseño de datos (nuevas tipologías) + estructura programa (diagrama de módulos) + interfaces entre módulos (prototipos de funciones)  Módulo principal: programa dirigido por menús + llamadas a funciones que realizan cada opción del mismo | 1 |
| Añadir nueva carga puntual + validación | 1 |
| Listar + clasificar | 1 |
| Eliminación de cargas | 2 |
| Calcular campo eléctrico | 1 |
| Calcular energía potencial | 2 |
| Calcular flujo del campo eléctrico | 2 |
| Total | 10 |

La calificación final en la asignatura consta de tres apartados cuya contribución es la siguiente:

* Examen final: 80%
* Trabajo en Equipo (Grupo de Trabajo): 10%
* Trabajo Individual: 10%

|  |
| --- |
| **Implementación completa del programa en lenguaje de programación C** |
| #include <stdlib.h>  #include <stdio.h>  #include <conio.h>  #include <ctype.h>  #include <math.h>  #define MAX 10 /\* Nº Máximo de partículas \*/  #define L 1.0 /\* Dimensiones del cubo (m)--> centrado en origen \*/  #define DMIN 0.005 /\* Distancia minima entre particulas (m) \*/  #define LMIN 0.010 /\* Lado del cubo de seleccion para eliminar (m) \*/  #define K 8.988E9 /\* N\*m2/C2 \*/  #define CARGAS "cargas.txt" /\* Archivo de texto \*/  #define BAK "cargas.bak" /\* Archivo con últimas modificaciones \*/  /\* Diseño Preliminar \*/  /\* Diseño de Datos \*/  /\* Nuevos tipos de datos \*/  typedef struct{  double x;  double y;  double z;  }tipo\_punto;  typedef struct{  double q;  tipo\_punto p;  }tipo\_carga;  typedef tipo\_carga tipo\_scp[MAX];  /\* Interfaces entre módulos \*/  /\* Prototipos de funciones \*/  char leer\_opcion\_menu(void);  void anadir(tipo\_scp v,int \*n);  void leer\_punto(tipo\_punto \*p);  int posicion\_ocupada(tipo\_scp v,int n,tipo\_punto p);  double distancia(tipo\_punto a, tipo\_punto b);  void clasificar(tipo\_scp a,int n,tipo\_punto p);  void listar(tipo\_scp v,int n);  void escribir\_carga(tipo\_carga c);  void eliminar(tipo\_scp v,int \*n);  int interior\_cubo(tipo\_punto p0,double lado,tipo\_punto p);  void calcularE(tipo\_scp v,int n);  tipo\_punto campo\_electrico(tipo\_scp v,int n,tipo\_punto p);  void calcularU(tipo\_scp v,int n);  void calcularF(tipo\_scp v,int n);  void inicializar\_de\_archivo(tipo\_scp v,int \*n);  void grabar\_en\_archivo(tipo\_scp v, int n);  /\* Diseño Detallado \*/  /\* Definiciones de funciones \*/  int main(){  char c;  tipo\_scp v;  int n;  //n=0;  inicializar\_de\_archivo(v,&n);  do{ system("cls");  c=leer\_opcion\_menu();  switch(c){  case '1': anadir(v,&n);  break;  case '2': listar(v,n);  break;  case '3': eliminar(v,&n);  break;  case '4': calcularE(v,n);  break;  case '5': calcularU(v,n);  break;  case '6': calcularF(v,n);  break;  case '0': printf("\n\n\tF I N D E P R O G R A M A");  getch();  break;  default: printf("\a\a");  break;  }  }while (c!='0');  grabar\_en\_archivo(v,n);  return 0;  }  char leer\_opcion\_menu(void){  char c;  printf("SISTEMA DE CARGAS PUNTUALES\n");  printf("===========================\n\n");  printf("\t1.- Añadir carga puntual\n");  printf("\t2.- Listar todas las cargas\n");  printf("\t3.- Eliminar cargas\n");  printf("\t4.- Calcular campo eléctrico\n");  printf("\t5.- Calcular energía potencial\n");  printf("\t6.- Calcular flujo de campo eléctrico\n");  printf("\t0.- Finalizar programa\n");  printf("\n\t\tIntroduzca opcion: ");  c=getch();  return c;  }  void anadir(tipo\_scp v,int \*n){  tipo\_carga carga;  tipo\_punto p;  system("cls");  p.x=p.y=p.z=0.0;  printf("AÑADIR UNA NUEVA CARGA PUNTUAL\n");  printf("==============================\n\n");  if(\*n==MAX){  printf("Vector de cargas lleno\n");  getch();  }else{  do{  printf("\tCarga (microCulombios): ");  scanf(" %lf",&carga.q);  }while((carga.q<-1000)||(carga.q>1000)||(carga.q==0));  carga.q/=1E6;  do{  printf("\tPosicion espacial (mm):\n");  leer\_punto(&carga.p);  }while(!interior\_cubo(p,L,carga.p));  if(posicion\_ocupada(v,\*n,carga.p)){  printf("\nHay otra carga a una distancia menor de %.2f mm\n",  DMIN\*1E3);  getch();  }else{  v[\*n]=carga;  (\*n)++;  }  }  }  void leer\_punto(tipo\_punto \*p){  printf("\t\tx: ");  scanf(" %lf",&p->x);  p->x/=1000.0;  printf("\t\ty: ");  scanf(" %lf",&p->y);  p->y/=1000.0;  printf("\t\tz: ");  scanf(" %lf",&p->z);  p->z/=1000.0;  }  int posicion\_ocupada(tipo\_scp v,int n,tipo\_punto p){  int i,res;  res=0;  i=0;  while((i<n)&&!res){  if(distancia(v[i].p,p)<DMIN){  res=1;  }else ++i;  }  return res;  }  double distancia(tipo\_punto a, tipo\_punto b){  return(sqrt((b.x-a.x)\*(b.x-a.x)+(b.y-a.y)\*(b.y-a.y)+(b.z-a.z)\*(b.z-a.z)));  }  void listar(tipo\_scp v,int n){  int i;  tipo\_punto p;  system("cls");  printf("LISTADO DE CARGAS PUNTUALES\n");  printf("===========================\n\n");  printf("\tIntroduzca punto de referencia (mm):\n");  leer\_punto(&p);  clasificar(v,n,p);  printf("%10s (%4s,%4s,%4s) (mm)\n","q(E-6C)"," x"," y"," z");  printf("%10s (%4s,%4s,%4s)\n","----------","----","----","----");  for(i=0;i<n;++i){  escribir\_carga(v[i]);  }  getch();  }  void clasificar(tipo\_scp a,int n,tipo\_punto p){  int i,j;  tipo\_carga x;  int k; /\* posicion de insercion \*/  for (i=0;i<n-1;++i){  k=i;  for (j=i+1;j<n;++j)  if (distancia(a[j].p,p)<distancia(a[k].p,p))  k=j;  x=a[k];  a[k]=a[i];  a[i]=x;  }  }  void escribir\_carga(tipo\_carga c){  printf("%10.2f (%4.0f,%4.0f,%4.0f)\n",c.q\*1E6,c.p.x\*1E3,  c.p.y\*1E3,c.p.z\*1E3);  }  void eliminar(tipo\_scp v,int \*n){  int i,j;  tipo\_punto p;  char c;  system("cls");  printf("ELIMINACION DE CARGAS\n");  printf("=====================\n\n");  printf("Introduzca posicion espacial (mm):\n");  leer\_punto(&p);  i=0;  while(i<\*n){  if(interior\_cubo(p,LMIN,v[i].p)){  printf("Carga:\n");  escribir\_carga(v[i]);  printf("\n\tEliminar carga (S/N): ");  c=toupper(getche());  if(c=='S'){  for(j=i+1;j<\*n;++j){  v[j-1]=v[j];  }  (\*n)--;  }else ++i;  }else ++i;  }  }  int interior\_cubo(tipo\_punto p0,double lado,tipo\_punto p){  int res;  res=0;  if((p.x<p0.x+lado/2.0)&&(p.x>p0.x-lado/2.0))  if((p.y<p0.y+lado/2.0)&&(p.y>p0.y-lado/2.0))  if((p.z<p0.z+lado/2.0)&&(p.z>p0.z-lado/2.0))  res=1;  return res;  }  void calcularE(tipo\_scp v,int n){  tipo\_punto p,e;  system("cls");  printf("CAMPO ELECTRICO DE UN SISTEMA DE CARGAS PUNTUALES\n");  printf("=================================================\n\n");  printf("Introduzca posicion espacial (mm):\n");  leer\_punto(&p);  if(!posicion\_ocupada(v,n,p)){  e=campo\_electrico(v,n,p);  printf("\tE=(%.2f,%.2f,%.2f) N/C",e.x,e.y,e.z);  }else printf("Posicion ocupada por una carga (E=infinito)\n");  getch();  }  tipo\_punto campo\_electrico(tipo\_scp v,int n,tipo\_punto p){  tipo\_punto e;  int i;  double d;  e.x=e.y=e.z=0.0;  for(i=0;i<n;++i){  d=distancia(p,v[i].p);  e.x+=v[i].q\*(p.x-v[i].p.x)/(d\*d\*d);  e.y+=v[i].q\*(p.y-v[i].p.y)/(d\*d\*d);  e.z+=v[i].q\*(p.z-v[i].p.z)/(d\*d\*d);  }  e.x\*=K;  e.y\*=K;  e.z\*=K;  return e;  }  void calcularU(tipo\_scp v,int n){  int i,j;  double u,ui;  system("cls");  printf("ENERGIA POTENCIAL ELECTROSTATICA DEL SISTEMA DE CARGAS\n");  printf("======================================================\n\n");  u=0;  for(i=0;i<n;++i){  ui=0;  for(j=i+1;j<n;++j){  ui+=v[j].q/distancia(v[i].p,v[j].p);  }  u+=ui\*v[i].q;  }  u\*=K;  printf("\t\tU = %.3f julios\n",u);  getch();  }  void calcularF(tipo\_scp v,int n){  int N=100;  int i,j;  double incx,incy,flujo;  tipo\_punto p,e;  system("cls");  printf("FLUJO DEL CAMPO ELECTRICO A TRAVES DE LA TAPA SUPERIOR\n");  printf("======================================================\n\n");  flujo=0.0;  incx=L/N;  incy=L/N;  p.x=-L/2.0+incx/2.0;  p.z=L/2.0;  for(i=0;i<N;++i){  p.y=-L/2.0+incy/2.0;  for(j=0;j<N;++j){  e=campo\_electrico(v,n,p);  flujo+=e.z;  p.y+=incy;  }  p.x+=incx;  }  flujo\*=incx\*incy;  printf("\t\tFlujo = %.3f N\*m2/C\n",flujo);  getch();  }  void inicializar\_de\_archivo(tipo\_scp v,int \*n){  FILE \*f;  int i;  float q,x,y,z;  f=fopen(CARGAS,"r");  if(f==NULL){  \*n=0;  }else{  i=0;  fscanf(f,"%f %f %f %f",&q,&x,&y,&z);  while((i<MAX)&&(!feof(f))){  v[i].q=q/1E6;  v[i].p.x=x/1E3;  v[i].p.y=y/1E3;  v[i].p.z=z/1E3;  ++i;  fscanf(f,"%f %f %f %f",&q,&x,&y,&z);  }  \*n=i;  fclose(f);  }  }  void grabar\_en\_archivo(tipo\_scp v, int n){  FILE \*f;  int i;  if((f=fopen(CARGAS,"r"))!=NULL){  fclose(f);  if((f=fopen(BAK,"r"))!=NULL){  fclose(f);  remove(BAK);  }  rename(CARGAS,BAK);  }  f=fopen(CARGAS,"w");  if(f==NULL){  printf("Error al crear el archivo: %s",CARGAS);  getch();  exit(0);  }else{  for(i=0;i<n;++i){  fprintf(f,"%10.2f %10.2f %10.2f %10.2f\n",v[i].q\*1E6,  v[i].p.x\*1E3,v[i].p.y\*1E3,v[i].p.z\*1E3);  }  fclose(f);  }  } |