UNIVERSIDAD PONTIFICIA ICAI ICADE COMILLAS MADRID ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA	**Fundamentos de Informática** 1º IIND **Examen: ABRIL 2009**	24 - Abril - 2009

Nombre: **Apellidos:**

Grupo: □ A □ B □ C □ D □ E □ F

Advertencias:

1. Duración del examen 1 horas y 30 minutos
2. No desgrape el cuadernillo del examen.
3. Puede utilizar lápiz o bolígrafo indistintamente.
4. No puede utilizar `exit`, `continue`, ni `break` (salvo en la instrucción `switch`()).
5. Las funciones sólo pueden tener un `return`

Calificación:

Código 1 **(4 puntos)**	**Código 2** **(1.5 puntos)**	**Código 3** **(1.5 puntos)**	**Código 4** **(3 puntos)**	**Total**

Código 1 (4 puntos)

Un instituto de meteorología quiere obtener las temperaturas máxima, mínima y media anual de todas las ciudades del mundo. Para ello, cuenta con un vector de estructuras de tipo `T_TEMP` con los registros de temperaturas de todas las ciudades. Nos piden escribir una función para crear un vector de estructuras de tipo `T_TEMP_MMM`. La dificultad viene por el hecho de que algunas ciudades registran la temperatura diariamente (365 datos), otras lo hacen semanalmente (53 datos), y otras mensualmente (12 datos), por lo tanto es necesario basarse en el campo `n_datos` para saber realmente cuántos datos de temperatura hay para cada ciudad.

```
typedef struct {
   char ciudad[N];
   double datos[365];
   int n_datos;
} T_TEMP;

typedef struct {
   char ciudad[N];
   double max;
   double min;
   double med;
} T_TEMP_MMM;

void GeneraMMM(T_TEMP vec[],int n, T_TEMP_MMM mmm[]);
```

La función anterior debe llamar a la función `Calcula_mmm()` que recibe un vector de temperaturas, calcula y calcula la temperatura máxima, la mínima y la media:

```
void Calcula_mmm(double t[], int n, double *pmax, double *pmin, double *pmed);
```

Escribir el código de las funciones `GeneraMMM()` y `Calcula_mmm()`

```
void GeneraMMM(T_temp vec1[],int n,T_temp_mmm mmm[])
{
        int i;
        double ma;
        double mi;
        double me;
        for(i=0;i<n;++i)
        {
                strcpy(mmm[i].ciudad,vec1[i].ciudad);
                Calcula_mmm(vec1[i].datos, vec1[i].n_datos, &ma,&mi,&me);
                mmm[i].max=ma;
                mmm[i].min=mi;
                mmm[i].med=me;
        }
        return;
}
```

```
void Calcula_mmm(double t[],int n, double *pmax,double *pmin,double *pmed)
{
        int i;
        double suma;
        *pmax=t[0];
        *pmin=t[0];
        suma=0;

        for(i=0;i<n;++i)
        {
                if(t[i]>*pmax)
                {
                        *pmax=t[i];
                }

                if(t[i]<*pmin)
                {
                        *pmin=t[i];
                }

                suma+=t[i];
        }
        *pmed=suma/n;
        return;
}
```

Código 2 (1,5 puntos)

Un vector de estructuras almacena los datos de los libros de una biblioteca. La información que se almacena de cada libro es (se utilizará una estructura de datos llamada `T_LIBRO`):

- Título del libro (una cadena de caracteres)
- Autor (una cadena de caracteres)
- ISBN (suponer que es un número entero)

Escribir una función (`BuscarLibros()`) para buscar cuántos libros existen en la biblioteca que pertenezcan a un autor (el nombre del autor es un dato a enviar a la función)

Escribir el prototipo y el código de la función `BuscarLibros`

Prototipo:

```
int BuscarLibros(T_LIBRO v[], int dim, char *autor);
```

Código de la función:

```
int BuscarLibros(T_LIBRO v[], int dim, char *autor)
{
     int i;
     int contador;

     contador = 0;
     for (i = 0; i < dim; i++){
          if (strcmp(autor, v[i].autor) == 0){
               contador++;
          }
     }

     return contador;
}
```

Código 3 (1,5 puntos)

Una tienda utiliza estructuras tipo T_TODO para gestionar sus productos y almacenes y quieren separar la información en estructuras de tipo T_PROD para definir sus productos y estructuras tipo T_ALMACEN para gestionar su almacén. Nos piden una función (pensada para ser llamada desde el programa principal para hacer conversiones de datos) que copie la información de una variable de tipo T_TODO en una variable de tipo T_PROD y otra de tipo T_ALMACEN.

```
typedef struct {
   int id;
   char nombre[N];
   double precio;
   int unidades;
   char localizacion[M];
} T_TODO;

typedef struct {
   int id;
   char nombre[N];
   double precio;
} T_PROD;

typedef struct {
   int id;
   int unidades;
   char localizacion[M];
} T_ALMACEN;
```

Escribir el prototipo y el código de la función de transformación.

Prototipo:

```
void Convertir(T_TODO t, T_PROD *pp, T_ALMACEN *pa);
```

Código de la función:

```
void Convertir(T_TODO t, T_PROD *pp, T_ALMACEN *pa);
{
	//Datos de T_PROD
	pp->id = t.id;
	strcpy(pp->nombre, t.nombre);
	pp->precio = t.precio;

	//Datos de T_ALMACEN
	pa->id = t.id;
	pa->unidades = t.unidades;
	strcpy(pa->localizacion, t.localizacion);
}
```

Llamada desde main (no se pide):

```
Convertir(todo, &prod, &almacen);
```

Código 4 (3 puntos)

Realizar un programa completo que utilizando un vector estático de número reales `vec_real` (no mayor de 100 posiciones) construya los vectores dinámicos `menores` y `mayores` con los valores que son inferiores a la media y superiores a la media respectivamente.

El programa debe seguir los siguientes pasos:

- Solicitar la dimensión del vector, comprobando que sea válida.
- Rellenar el vector `vec_real` utilizando la función `RellenarVectorReal()`(que no es necesario programar).. Esta función, además de rellenar el vector, debe devolver la media de los valores introducidos.
- Calcular cuántos elementos del vector son mayores que la media, y cuántos son menores.
- Reservar memoria dinámica para los vectores `menores` y `mayores`
- Rellenar los vectores con los valores apropiados de `vec_real`.
- Mostrar el contenido de todos los vectores utilizando la función:

  `void Mostrar(double v[], int n)` (que no es necesario programar).

Escribir el programa principal completo (includes, prototipos…) pero no escribir el código de las funciones.

/* FICHA

*/

```
#include <stdio.h>
#include <stdlib.h>
#define N 100

double RellenarVectorReal(double v[], int n);
void Mostrar(double v[], int n);

int main(void)
{
	double vec_real[N];
	double *mayores;	//vector de mayores
	double *menores;	//vector de menores
	int n;
	double med;  //media
	int n_may;
	int n_men;
	int i;
	int j;
	int k;

	do{
		printf("Numero de elementos: ");
		scanf("%d", &n);
	}while(n<=0||n>N);

	med = RellenarVectorReal(vec_real, n);

	n_may=0;
	n_men=0;
	for (i=0; i<n; i++){
		if (vec_real[i]>med)
			n_may++;
		else
			n_men++;
	}
```

```
        mayores = (double *)calloc(n_may, sizeof(double));
        menores = (double *)calloc(n_men, sizeof(double));
        if(mayores==NULL ||menores==NULL){
                printf("Error al dar memoria dinamica");
                if(mayores!=NULL)
                        free(mayores);
                if(menores!=NULL)
                        free(menores);
        }else{
                j=0;
                k=0;
                for (i=0; i<n; i++){
                        if (vec_real[i]>med){
                                mayores[j] = vec_real[i];
                                j++;
                        }else{
                                menores[k] = vec_real[i];
                                k++;
                        }
                }
                Mostrar(vec_real, n);
                Mostrar(mayores, n_may);
                Mostrar(menores, n_men);
                free(mayores);
                free(menores);
        }
        return 0;
}
```