UNIVERSIDAD PONTIFICIA COMILLAS
Escuela Técnica Superior de Ingeniería (ICAI)
Ingeniero en Informática
Programación Básica

25 de junio de 2004 Alumno/a:_____Grupo: | A | B |

Apartado 1. (**1,5 puntos**). Indicar mediante un razonamiento breve qué realizan las siguientes funciones:

```
// ¿Qué hace esta función al recibir dos cadenas de caracteres?
void funcion1(char *c1, char *c2)
{
   char i;

   *c2 = *c1;
   i = *c2;
   while (i != '\0')
   {
      c1++;
      c2++;
      *c2 = *c1;
      i = *c2;
   }
}
```

Copia la cadena a la que apunta c1 en la cadena a la que apunta c2 carácter a carácter. Copia el primer carácter y avanza los dos punteros 1 byte (tamaño de un char) para copiar el siguiente, y así sucesivamente hasta el '\0'. Es similar a strcpy cambiando el fuente y el destino.

```
// ¿Qué hace esta función al recibir una cadena de caracteres?
int funcion2(char *c)
{
   char *p;
   p = c;

   while (*p != '\0')
      p++;
   return p - c;
}
```

Devuelve el número de caracteres de la cadena. Es una función similar a strlen. Calcula el número de bytes que hay entre el último y primer carácter de la cadena.

```
// ¿Qué hace esta función al recibir dos cadenas de caracteres?
int funcion3(char *c1, char *c2)
{
   int valor;

   while (*c1 == *c2)
   {
      c1++;
      c2++;
   }
   if (*c1 == '\0')
      valor = 0;
   else
      valor = *c1 - *c2;

   return valor;
}
```

Compara las dos cadenas de caracteres. Es una función similar a strcmp con variantes. Si las cadenas son iguales hasta el '\0' de la primera devuelve 0, es decir, cuando la primera cadena está contenida en la segunda. En cualquier otro caso, nos devuelve un valor menor que 0 si el primer carácter que difiere es anterior al de la segunda cadena y si no devuelve un valor mayor que 0.

Apartado 2. (**2 puntos**). Codificar una función `Validar` que reciba como datos un código de producto (*EAN 13*) en tres variables enteras (una con la cifra de la izquierda `a` y las otras dos `b` y `c`, con los valores de los dos bloques de seis cifras) y, nos devuelva el valor `1` si el código es correcto y `0` si no lo es. El prototipo de la función será:

```
int Validar (int a, int b, int c);
```

¿Qué es el código *EAN 13*?. Para la codificación de productos se utiliza, a nivel internacional, un código numérico (acompañado de una traducción a código de barras) denominado *Universal Product Code* (**UPC**) que tiene su versión europea bajo la denominación *European Article Numbering* - **EAN**.

Existen varias versiones de estos códigos siendo el **EAN 13** el más utilizado. Su estructura es:

Prefijo	Código de empresa	Código de producto	Dígito de control
8 4	**1 2 3 4 5 6 7**	**8 9 1**	**2**

El código **EAN 13** se construye según el siguiente esquema:

PREFIJO (de 1 a 3 cifras): El prefijo asignado por **EAN** *internacional* a **AECOC** es el 84. Todas las empresas que forman parte del sistema **EAN** a través de **AECOC** codifican sus artículos con el prefijo 84.

CÓDIGO DE EMPRESA (de 5 a 8 cifras): AECOC asigna a las empresas registradas un número entre 5 y 8 dígitos.

CÓDIGO DE PRODUCTO (12 – cifras (prefijo + cod. empresa)): Una empresa registrada dispone de una serie de cifras (hasta un total de 12) para identificar cada uno de sus productos.

El código **EAN 13** de un producto se obtendrá añadiendo a las 12 cifras anteriores, mediante cálculo, la correspondiente cifra o dígito de control que figurará siempre en último lugar.

Procedimiento de Cálculo del dígito de control:

El proceso de cálculo es muy sencillo, basta con seguir tres pasos:

- Se numeran las cifras del código de derecha a izquierda (sin contar el dígito de control), y se multiplica cada una de ellas por 1 si ocupan posición par y por 3 si ocupan posición impar.
- Se suman los valores de todos los productos obtenidos en el paso anterior (serán un total de 12 productos).
- Se busca la decena o múltiplo de 10 superior o igual al resultado obtenido de la suma anterior y se restan estos dos valores (decena superior o igual – suma de productos). El resultado obtenido es el dígito de control.

Ejemplo práctico:

12	11	10	9	8	7	6	5	4	3	2	1	DC	Posición
8	**4**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**1**		**Cifra código EAN 13**
*	*	*	*	*	*	*	*	*	*	*	*		**multiplicada por**
1	3	1	3	1	3	1	3	1	3	1	3		**peso**
⇓	⇓	⇓	⇓	⇓	⇓	⇓	⇓	⇓	⇓	⇓	⇓		**igual a**
8	12	1	6	3	12	5	18	7	24	9	3		**suma parcial**
8 + 12 + 1 + 6 + 3 + 12 + 5 + 18 + 7 + 24 + 9 + 3 = **108**													**suma de productos**
8	**4**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**1**	**2**	**Código EAN 13 final**

110 – 108 = 2 Resultado (código completo): **8 412345 678912**

Nota: La función deberá recibir el valor **8** en el primer parámetro **a**, el valor **412345** en el segundo **b** y **678912** en el tercero **c** y devolver, en este caso, como el código es correcto, el valor **1**.

```
    /*
        ficha de la función
    */
    int Validar (int a, int b, int c)
    {
        // Variables locales
        int digito_control;
        int d12, d11, d10, d9, d8, d7, d6, d5, d4, d3, d2, d1;
        int suma, valor, dc;

        // Cálculo de cada una de las cifras del código
        digito_control = c%10;
        d1 = (c/10)%10;
        d2 = (c/100)%10;
        d3 = (c/1000)%10;
        d4 = (c/10000)%10;
        d5 = (c/100000)%10;
        d6 = b%10;
        d7 = (b/10)%10;
        d8 = (b/100)%10;
        d9 = (b/1000)%10;
        d10 = (b/10000)%10;
        d11 = (b/100000)%10;
        d12 = a;

        // Cálculo de la suma de productos
        suma = d1*3+d2+d3*3+d4+d5*3+d6+d7*3+d8+d9*3+d10+d11*3+d12;

        // Cálculo del dígito de control
        dc = 10-suma%10;
        if (dc == 10)
            dc = 0;

        // Cálculo del valor: se puede poner valor = dc == digito_control;
        if (dc == digito_control)
            valor = 1;
        else
            valor = 0;

        return valor;
    }
```

Apartado 3. (**1,5 puntos**) Escribir una función **Cifrar** que reciba una frase y retorne dicha frase en la que cada letra se convierta en su opuesta en el abecedario, es decir, en las minúsculas una **'a'** se transformará en una **'z'**, una **'b'** se transformará en una **'y'** y así sucesivamente hasta la **'m'** que se transformará en una **'n'**. Haremos lo mismo con las letras mayúsculas, es decir, una **'A'** se transformará en una **'Z'**, una **'B'** se transformará en una **'Y'** y así sucesivamente hasta la **'M'** se transformará en una **'N'**, el resto de caracteres y la ñ no se modificarán. El prototipo de la función será: **char * Cifrar (char *);**

Notas: La función asignará memoria dinámica para la cadena cifrada y supondremos que será el main quien se encargue de liberarla. Se sabe que los códigos de las letras son: 'a' = 97, ..., 'z' = 122, 'A' = 65, ..., 'Z' = 90.

```
/*
        ficha de la función
*/
char * Cifrar (char * cadena)
{
    // Variables locales
    char * cadena_cifrada;
    int longitud, i;

    // Calculo de la longitud de la cadena
    longitud = strlen(cadena);

    // Petición de memoria dinámica
    cadena_cifrada = (char *) calloc (longitud + 1, sizeof(char));
    if (cadena_cifrada == NULL)
        printf("\nError en la asignacion de memoria...");
    else
    {
        // Codificación de la cadena
        for (i=0; i<=longitud; i++)
        {
            if (cadena[i]>=97 && cadena[i]<=122)
                cadena_cifrada[i] = 97 + (122-cadena[i]);
            else
                if (cadena[i]>=65 && cadena[i]<=90)
                    cadena_cifrada[i] = 65 + (90-cadena[i]);
                else
                    cadena_cifrada[i] = cadena[i];
        }
    }

    return cadena_cifrada;
}
```

Apartado 4. (**2 puntos**) Problema de archivos.

La base de datos de datos de una compañía aérea está formada por un archivo de aviones y otro de recorridos para hacer la planificación semanal de sus aviones. Ambos archivos son binarios de acceso directo y contienen los siguientes campos:

```
typedef struct avion
{
   int identificador;         /* ID único utilizado en el otro archivo       */
   char modelo[20];           /* Tipo de avión, por ejemplo "Boeing 747-400" */
   int plazas;                /* Número máximo de pasajeros                  */
   int velocidad;             /* Velocidad en km/h                           */
}T_AVION;

typedef struct recorrido
{
   int num_vuelo;             /* Número de vuelo                             */
   char origen[4];            /* Código de origen de tres letras y el '\0'   */
   char destino[4];           /* Código de destino de tres letras y el '\0'  */
   int avion_asignado;        /* Identificador del avión que ha sido asignado
                                 a este recorrido                            */
   int dia;                   /* Dia de la semana: 1=lunes, ...7=domingo     */
   int num_pasajeros;         /* Número de pasajeros en este vuelo           */
} T_RECORRIDO;
```

Nótese que en la estructura `T_RECORRIDO` hay un campo que hace referencia al avión que ha sido asignado para dicho vuelo esta semana.

Se pide: Escribir una función para listar toda la información de los vuelos previstos para esta semana, incluyendo la información sobre el avión asignado y el grado de ocupación del avión. No es necesario hacer ninguna ordenación, sino que los datos aparecerán en el orden en que estén almacenados en el archivo de recorridos. Por cada registro del archivo de recorridos será necesario hacer una búsqueda en el archivo de aviones para obtener los datos del avión asignado.

Ejemplo de salida:

```
Lunes:   vuelo:  0450 FRA --> LAX, 385 pasajeros. Avión: Boeing 747-400, 409 plazas, 927 km/h,  94.1%
Lunes:   vuelo:  2581 MAD --> FRA, 131 pasajeros. Avión: Airbus 319,     126 plazas, 853 km/h, 104.0%
...
Domingo: vuelo:  0349 MAD --> SEV,  63 pasajeros. Avión: Airbus 319,     126 plazas, 850 km/h,  50.0%
```

```
/*
Solucion examen Jun/2004
Problema de ficheros
*/
#include <stdio.h>

typedef struct avion {
   int identificador; /*ID único utilizado en el otro archivo */
   char modelo[20]; /*tipo de avión, por ejemplo "Boeing 747-400" */
   int plazas; /* Número máximo de pasajeros */
   int velocidad; /* Velocidad en km/h */
} T_AVION;

typedef struct recorrido {
   int num_vuelo;  /* número de vuelo */
   char origen[4];  /* código de origen de tres letras y \0 */
   char destino[4]; /* código de destino de tres letras y \0 */
   int avion_asignado; /* identificador del avión que ha sido asignado a este recorrido */
   int dia; /* dia de la semana: 1=lunes, 2=martes...7=domingo */
   int num_pasageros; /* número de pasajeros en este vuelo */
} T_RECORRIDO;

int BuscarAvion(FILE *fp_avion, int identificador, T_AVION *avion);
void MostrarInformacion(FILE *fp_recorrido, FILE *fp_avion);



void MostrarInformacion(FILE *fp_recorrido, FILE *fp_avion)
/*
Función para listar toda la información de vuelos
Argumentos:
   fp_recorrido archivo con información de recorridos
   fp_avion archivo con la información de cada avión
Valor retornado:
   ninguno
```

```
Advertencias:
   Los archivos deben estar abiertos y validados.  El main se encarga de abrirlos y cerrarlos.
*/
{
   int ctrl;    /* control de error de lectura */
   int ctrl_av; /* control de localización del avión */
   T_RECORRIDO rec;
   T_AVION av;
   double ocupacion; /* porcentaje de ocupación */

   /* bucle principal para mostrar información */
   rewind(fp_recorrido);  /* garantiza que empieza a leer desde el principio */
   ctrl=fread(&rec,sizeof(T_RECORRIDO),1,fp_recorrido);
   while (ctrl == 1)  {
      switch (rec.dia)  {
         case 1: printf("Lunes:    "); break;
         case 2: printf("Martes:   "); break;
         case 3: printf("Miércoles:"); break;
         case 4: printf("Jueves:   "); break;
         case 5: printf("Viernes:  "); break;
         case 6: printf("Sábado:   "); break;
         case 7: printf("Domingo:  "); break;
         default: printf("Error de día:%d\n",rec.dia);
      }
      printf("vuelo: %04d %3s --> %3s, %3d pasajeros.",
         rec.num_vuelo, rec.origen, rec.destino, rec.num_pasageros);

      /* Datos del avión */
      ctrl_av=BuscarAvion(fp_avion, rec.avion_asignado, &av);
      if (ctrl_av == 0)  {
         printf("Avión: %s, ",av.modelo);
         printf("%3d plazas, %3d km/h, ", av.plazas, av.velocidad);
         ocupacion = (double)rec.num_pasageros/av.plazas * 100.0;
         printf("%5.1f%%\n",ocupacion);
      } else  {
         printf("*** Error en el avión asignado: %d\n",rec.avion_asignado);
      }

      /* Sigo leyendo */
      ctrl=fread(&rec,sizeof(T_RECORRIDO),1,fp_recorrido);
   }
}



int BuscarAvion(FILE *fp_avion, int identificador, T_AVION *avion)
/*
Función para buscar un avión concreto en la tabla de aviones
Argumentos:
   fp_avion    archivo con la información de cada avión
   identificador:  clave de búsqueda del avión
   avion   estructura a rellenar con los datos del avión
Valor retornado:
   0: no hay error
   1: avión no encontrado o error de lectura
Advertencias:
   El archivo debe estar abierto y validado.
*/
{
   int ret;  /* valor a retornar */
   int ctrl; /* control de lectura */

   rewind(fp_avion);  /* voy al principio del archivo, independientemente de la posición
                         en la que estuviese */


   /* Bucle de lectura hasta encontrarlo o hasta llegar al final */
   do {
      ctrl=fread(avion,sizeof(T_AVION),1,fp_avion);
                  /* En este caso avion no lleva & porque avion
                     es puntero a T_AVIOM, es decir que este fread lee del
                     archivo y mete los dator directamente en la variable
                     av del main */
   } while (ctrl==1 && avion->identificador != identificador);


   /* Control de error */
   if (avion->identificador == identificador) ret=1;
   else ret=0;

   return ret;
}
```

Apartado 5. (**1,5 puntos)**. Bucles y cálculos.

Escribir un programa que pregunte tres términos de una serie y que identifique si la serie es aritmética o geométrica (admitiendo un error de $1e^{-3}$) y que calcule la razón.
Como resultado mostrar los 10 primeros términos de la serie, siempre que sean menores de 100.

Ejemplo 1:

```
Dame los tres primeros números de la serie: 4.0, 7.0, 10.0001
Es una serie aritmética de razón: 3.000
Los primeros términos son:        4.0  7.0  10.0  13.0  16.0  19.0  22.0  25.0  28.0  31.0
```

Ejemplo 2:

```
Dame los tres primeros números de la serie: 10, 12, 16
No es una serie aritmética ni geométrica.
```

Ejemplo 3:

```
Dame los tres primeros números de la serie: 10, 20, 40
Es una serie geométrica de razón: 2.000
Los primeros términos son:        10.0  20.0  40.0  80.0
```

```
/*
Examen Junio 2004
Apartado 5: Series aritméticas y geométricas: bucles y cálculos
Rafael Palacios
*/

#include <stdio.h>

#define TOLERANCIA 0.001

int main(void)
{
	double n1, n2, n3;	/* tres primeros términos de la serie */
	double razon1, razon2; /* razón de la serie */
	int i; /*contador */
	double x; /* término de la serie */
	int es_aritmetica, es_geometrica;  /* indica el tipo de serie */

	/*** Lectura de datos ***/
	printf("Dame los tres primeros números de la serie: ");
	scanf("%lf",&n1);
	scanf("%lf",&n2);
	scanf("%lf",&n3);

	/*** Serie Aritmética? ***/
	es_aritmetica=0;
	razon1=n2-n1;
	razon2=n3-n2;
	if (razon1<razon2+TOLERANCIA && razon1>razon2-TOLERANCIA)  {
		es_aritmetica=1;
		printf("Es una serie aritmética de razón: %.3f\n",razon1);
		printf("Los primeros términos son:        ");
		i=0;
		x=n1;
		while(i<10 && x<100)  {
			printf("%.1f ",x);
			x=x+razon1;
			i++;
		}
		printf("\n");
	}

	/*** Serie Geométrica ***/
	es_geometrica=0;
	if ((n1<1e-10 && n1>-1e-10) ||
		(n2<1e-10 && n2>-1e-10) ||
		(n3<1e-10 && n3>-1e-10)) {
		printf("No puede ser una serie geométrica porque algún término es cero.\n");
```

```
	} else  {
		razon1=n2/n1;
		razon2=n3/n2;
		if (razon1<razon2+TOLERANCIA && razon1>razon2-TOLERANCIA)  {
			es_geometrica=1;
			printf("Es una serie geométrica de razón: %.3f\n",razon1);
			printf("Los primeros términos son:        ");
			i=0;
			x=n1;
			while(i<10 && x<100)  {
				printf("%.1f ",x);
				x=x*razon1;
				i++;
			}
			printf("\n");
		}
	}

	if (es_aritmetica==0 && es_geometrica==0) {
		printf("No es una serie aritmética ni geométrica.\n");
	}
	printf("Fin del programa.\n");
	return 0;
}
```

Apartado 6. (**2 puntos)**. Listas.

Suponiendo una lista simplemente encadenada con información de personas en elementos de tipo **T_PERSONA** con la estructura **struct persona** compuesta por los miembros **char nombre[20];**, **char apellidos[30];** y **struct persona *siguiente;** indica si las siguientes *funciones recursivas* son correctas o fallan. Si fallan explica cuál es la razón, y si funcionan explica qué es lo que hacen.

```
void Sorpresa1(T_PERSONA *p)
{
   /* Condición de continuidad/salida de la recursividad */
   if (p != NULL)
   {
      /* Código */
      printf("Nombre: %s", p->nombre);
      printf(" Apellidos: %s\n",p->apellidos);
      Sorpresa1(p->siguiente);
      free(p);
   }
   return;
}
```

Funciona correctamente.
Esta función recorre la lista empezando por inicio. Imprime los datos del primer elemento y hace una llamada recursiva para imprimir el resto.
Cuando vuelve de la recursividad libera el elemento. Por lo tanto va liberando del final hacia el principio.

```
void Sorpresa2(T_PERSONA *p)
{
   /* Condición de continuidad/salida de la recursividad */
   if (p != NULL)
   {
      /* Código */
      printf("Nombre: %s", p->nombre);
      printf(" Apellidos: %s\n",p->apellidos);
      free(p);
      Sorpresa2(p->siguiente);
   }
   return;
}
```

Funciona MAL.
Esta función empieza a recorrer la lista desde el principio. Pero después de imprimir los datos del primer elemento, lo libera. Por lo tanto la llamada recursiva utilizando p->siguiente es incorrecta porque p ya está liberado.

```
void Sorpresa3(T_PERSONA *p)
{
   /* Condición de continuidad/salida de la recursividad */
   if (p != NULL)
   {
      /* Código */
      Sorpresa3(p->siguiente);
      printf("Nombre: %s", p->nombre);
      printf(" Apellidos: %s\n",p->apellidos);
      free(p);
   }
   return;
}
```

Funciona correctamente.
Esta función empieza a recorrer la lista desde el principio. Pero llama inmediatamente a la recursividad antes de hacer nada, por lo tanto recorre toda la lista hasta el final.
Conforme va volviendo de la recursividad, imprime los datos y libera el elemento. El resultado final es que imprime y libera desde el final hasta el principio.