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

12 de junio de 2006 Alumno/a:_____Grupo: 1ºA

Apartado 1. (**1 punto**). Indicar el resultado que producen sobre la pantalla los siguientes programas:

a)

```
/*
   ...
*/
#include <stdio.h>

int main(void)
{
    int i, j;
    float a, b;

    a = 1;
    b = 5;

    for (i=1; i<=4; i++)
        for (j=4; j>1; j--)
        {
            printf("\na = %.2f", ++a);
            a += --b;
        }

    return 0;
}
```

```
a = 2.00
a = 7.00
a = 11.00
a = 14.00
a = 16.00
a = 17.00
a = 17.00
a = 16.00
a = 14.00
a = 11.00
a = 7.00
a = 2.00
```

b)

```
/*
   ...
*/
#include<stdio.h>
#include<string.h>
#define N 30

void funcion(char *, int);

int main()
{
    char cadena[N];
    int n;

    strcpy(cadena,"En el 2006 ganamos el mundial");
    n = 0;

    funcion(cadena,n);

    return 0;
}

void funcion(char *cadena, int n)
{
    if(n < strlen(cadena))
    {
        funcion(cadena, n+1);
        printf("%c", cadena[n]);
    }
    return;
}
```

```
laidnum le somanag 6002 le nE
```

Apartado 2. (**2 puntos**). Codificar una función **Dias_imposición** que reciba como datos de entrada la fecha de imposición de un capital en un banco (**fecha_inicial**) y la fecha de retirada del mismo (**fecha_final**) y, calcule y devuelva el número total de días que dicho capital ha estado depositado en el banco.

```
int Dias_imposicion (T_FECHA fecha_inicial, T_FECHA fecha_final);
```

Ejemplo: Para una imposición el día **11/11/2004** con una retirada el día **12/06/2006**

Devolverá: **578** días.

Que son, 19+31 = 50 días de 2004
365 días de 2005 (año no bisiesto)
31+28+31+30+31+12 = 163 días de 2006
Total = 50 + 365 + 163 = **578 días**

```
typedef struct
{
    short dia;
    short mes;
    short anio;
}T_FECHA;
```

Nota: Se sabe que un año es bisiesto si es múltiplo de 4 excepto los años múltiplos de 4 que, además, son múltiplos de 100 pero no son múltiplos de 400. Siempre se supondrá que la fecha de imposición es anterior a la fecha de retirada.

```
/*
    ...
*/
int Dias_imposicion (T_FECHA fecha_inicial, T_FECHA fecha_final)
{
    // Variables locales
    int  total_dias, dias_fecha1, dias_fecha2, anio;
    // Cálculo de los días transcurridos desde el día 1 de enero
    // a cada una de las fechas recibidas
    dias_fecha1 = Dias_transcurridos (fecha_inicial);
    dias_fecha2 = Dias_transcurridos (fecha_final);
    // Comprobamos si las fechas son del mismo año o no
    if (fecha_inicial.anio == fecha_final.anio)
    {
        // Las fechas son del mismo año
        total_dias = dias_fecha2 - dias_fecha1;
    }
    else
    {
        // Las fechas son de distinto año
        // Cálculo de los días del primer año
        if (fecha_inicial.anio%4 == 0 &&
          !(fecha_inicial.anio%100 == 0 && fecha_inicial.anio%400 != 0))
            total_dias = 366 - dias_fecha1;
        else
            total_dias = 365 - dias_fecha1;
        // Cálculo de los días de años intermedios (si los hay)
        for(anio=fecha_inicial.anio+1; anio<=fecha_final.anio-1; anio++)
            if (anio%4 == 0 && !(anio%100 == 0 && anio%400 != 0))
                total_dias = total_dias + 366;
            else
                total_dias = total_dias + 365;
        // Cálculo de los días del último año
        total_dias = total_dias + dias_fecha2;
    }
```

```
    // Devolvemos los días totales entre las fechas recibidas
    return total_dias;
}

/*
    ...
*/
int Dias_transcurridos (T_FECHA fecha)
{
    // Variables locales
    int dias, mes;
    // Primero cogemos los días del mes en curso
    dias = fecha.dia;
    // Añadimos los días de los meses anteriores al mes en curso
    for (mes=1; mes<=fecha.mes-1; mes++)
    {
        switch (mes)
        {
            case 2: if (fecha.anio%4 == 0 &&
                        !(fecha.anio%100 == 0 &&
                              fecha.anio%400 != 0))
                        dias = dias + 29;
                    else
                        dias = dias + 28;
                    break;

            case 4:
            case 6:
            case 9:
            case 11: dias = dias + 30;
                     break;
            default: dias = dias +31;
        }
    }
    return dias;
}
```

Apartado 3. (**2 puntos**) Ejercicio de ficheros.

Disponemos de un fichero binario **biblioteca.dat** que contiene información de cada uno de los libros de una bilioteca (**c:\temp\biblioteca.dat**). Cada registro contiene los datos de un ejemplar de un libro. Se sabe que de un mismo libro puede haber más de un ejemplar. El fichero está ordenado por materia y dentro de cada materia por título. En este caso se sabe que no hay títulos repetidos. La estructura del registro es la siguiente:

```
Materia   (15 caracteres)
Título    (20 caracteres)
Autor     (20 caracteres)
Editorial (20 caracteres)
```

```
typedef struct
{
    char  materia  [15];
    char  titulo   [20];
    char  autor    [20];
    char  editorial[20]
}T_LIBRO;
```

a) **Se pide:** Codificar un programa que, a partir del fichero, saque un listado en pantalla de libros con el siguiente formato:

```
                LISTADO DE EJEMPLARES POR MATERIA Y POR TITULO

Materia: XXXXXXXXXXXX
TITULO               AUTOR                EDITORIAL        EJEMPLARES
XXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXX       X
...                  ...                  ...              ...
XXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXX       X
Total libros materia: XXX

...

Materia: XXXXXXXXXXXX
TITULO               AUTOR                EDITORIAL        EJEMPLARES
XXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXX       X
...                  ...                  ...              ...
XXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXX       X
Total libros materia: XXX

Total libros en la biblioteca: XXXX
```

```
/*
    Programa:
    Descripción:
    Autor/a:
    Fecha:
*/
// Directivas del precompilador
#include <stdio.h>
#include <string.h>
// Programa principal
int main (void)
{
    // Declaración de la estructura
    typedef struct
    {
        char  materia  [15];
        char  titulo   [20];
        char  autor    [20];
        char  editorial[20];
    }T_LIBRO;
    // Declaración de variables
    char mat_anterior [15];
    char tit_anterior [20];
    int  ejemplares, ejemplares_materia, ejemplares_totales;
    FILE *libros;            // Puntero al fichero
    T_LIBRO libro;           // Registro buffer
```

```
    // Inicializaciones
    ejemplares_materia = 0;
    ejemplares_totales = 0;
    // Apertura del fichero para lectura
    libros = fopen ("c:\\temp\\biblioteca.dat", "rb");
    if (libros != NULL)
    {
        // Posicionamiento de la cabeza de lectura sobre el primer registro
        rewind (libros);
        // Cabecera del listado
        printf ("\n\n      LISTADO DE EJEMPLARES POR MATERIA Y POR TITULO\n");
        printf ("      ========================================== \n\n");
        // Lectura del primer registro
        fread (&libro, sizeof(T_LIBRO), 1, libros);
        if (!feof(libros))
        {
            // Retenemos los datos del primer registro del fichero
            strcpy (mat_anterior, libro.materia);
            strcpy (tit_anterior, libro.titulo);
            ejemplares = 1;
            // Se escribe la cabecera de materia
            printf("Materia: %s\n", mat_anterior);
            printf ("TITULO        AUTOR          EDITORIAL      EJEMPLARES\n");
            printf ("------------- ------------- -------------- ----------\n");
            // Linea de detalle materia-libro
            printf ("%-20s %-20s %-20s", libro.titulo, libro.autor,
                                                    libro.editorial);
        }
        // Lectura del segundo registro
        fread (&libro, sizeof(T_LIBRO), 1, libros);
        while (!feof(libros))
        {
            if (strcmp (mat_anterior, libro.materia) == 0 &&
                                strcmp (tit_anterior, libro.titulo) == 0)
            {
                // Tratamiento en secuencia
                ejemplares++;
            }
            else
            {
                if(strcmp (dep_anterior, venta.departamento) == 0)
                {
                    // Tratamiento de ruptura de primer nivel
                    // Final del libro anterior
                    printf (" %3d\n", ejemplares);
                    ejemplares_materia += ejemplares;
                    // Inicio de la nueva secuencia de libro
                    strcpy (tit_anterior, libro.titulo);
                    ejemplares = 1;
                    // Linea de detalle materia-libro
                    printf ("%-20s %-20s %-20s", libro.titulo, libro.autor,
                                                        libro.editorial);
                }
                else
```

```
                {
                    // Ruptura de secuencia de primer y segundo nivel
                    // Final del libro anterior
                    printf (" %3d\n", ejemplares);
                    ejemplares_materia += ejemplares;
                    // Final de la materia anterior
                    printf ("Total libros materia: %3d\n", ejemplares_materia);
                    ejemplares_totales += ejemplares_materia;
                    // Inicio de la nueva materia
                    strcpy (mat_anterior, libro.materia);
                    printf("Materia: %s\n", mat_anterior);
                    // Inicio del nuevo titulo
                    strcpy (tit_anterior, libro.titulo);
                    printf ("TITULO         AUTOR          EDITORIAL      EJEMPLARES\n");
                    printf ("-------------- ------------- -------------- ----------\n");
                    ejemplares = 1;
                    // Linea de detalle materia-libro
                    printf ("%-20s %-20s %-20s", libro.titulo, libro.autor,
                                                              libro.editorial);
                }
            }
            // Lectura del tercer registro y sucesivos
            fread (&libro, sizeof(T_LIBRO), 1, libros);
        }
        // Final del libro anterior
        printf (" %3d\n", ejemplares);
        ejemplares_materia += ejemplares;
        // Final de la materia anterior
        printf ("Total libros materia: %3d\n", ejemplares_materia);
        ejemplares_totales += ejemplares_materia;
        // Pie del listado
        printf ("Total libros en la biblioteca: %5d\n", ejemplares_totales);
        // Cierre del fichero
        fclose (libros);
    }
    else
        printf ("\n\n Error en la apertura del fichero biblioteca.dat ...");
    return 0;
}
```

Apartado 4. (**2 puntos)**. Listas simplemente encadenadas.

Suponemos que ya existe una lista simplemente encadenada con información de las selecciones nacionales de fútbol que están participando en el campeonato mundial de fútbol de Alemania. Los elementos son de tipo **T_SELECCION** que se encuentran en cualquier orden. La dirección del primer elemento de la lista se encuentra en el puntero **inicio**.

```
typedef struct seleccion
{
    char   pais [20];
    char   grupo;
    char   seleccionador[30];
    struct seleccion *siguiente;
}T_SELECCION;
```

Se pide: Codificar una función **T_SELECCION * Crear_nueva_lista(T_SELECCION *)** que reciba el puntero **inicio** y genere una nueva lista en la que se encuentren las selecciones ordenadas por el nombre del país. La función devolverá la dirección del primer elemento de la nueva lista.

Nota: En este caso debe suponerse que la memoria dinámica para los nuevos elementos se pide en la función y se libera en el main.

```
/*
    ...
*/
T_SELECCION * Crear_nueva_lista (T_SELECCION *inicio)
{
    // Declaraciones
    T_SELECCION * inicio2, *actual, *aux, *aux1, *aux2;
    // Inicializamos la lista nueva
    inicio2 = NULL;
    // Se recorre la lista elemento a elemento hasta el final
    actual = inicio;
    while (actual != NULL)
    {
        // Generamos el elemento de la nueva lista
        aux = (T_SELECCION *) calloc (1, sizeof(T_SELECCION));
        if (aux == NULL)
            printf("\nNo hay suficiente memoria...");
        else
        {
            // Copiamos los datos del elemento actual en el aux
            strcpy(aux->pais, actual->pais);
            aux->grupo = actual->grupo;
            strcpy(aux->seleccionador, actual->seleccionador);
            // El elemento aux se inserta en la nueva lista ordenada
            if (inicio2 == NULL)
            {
                // Se trata del primer elemento de la nueva lista
                inicio2 = aux;
                aux->siguiente = NULL;
            }
            else
            {
                // Se trata del segundo o sucesivos elementos.
```

```
                    // Buscamos el hueco
                    aux1 = inicio2;
                    aux2 = inicio2;
                    while (aux1->siguiente != NULL &&
                                    strcmp(aux1->pais, aux->pais) < 0)
                    {
                        aux2 = aux1;
                        aux1 = aux1->siguiente;
                    }
                    // Miramos en que lugar hemos encontrado el hueco
                    if (aux1==inicio2 && strcmp(aux1->pais, aux->pais)>0)
                    {
                        // Le corresponde el primer lugar
                        aux->siguiente = inicio2;
                        inicio2 = aux;
                    }
                    else
                    {
                        if (aux1->siguiente == NULL &&
                                    strcmp(aux1->pais, aux->pais) < 0)
                        {
                            // Le corresponde la última posición
                            aux1->siguiente = aux;
                            aux->siguiente = NULL;
                        }
                        else
                        {
                            // Le corresponde una posición intermedia
                            aux->siguiente = aux1;
                            aux2->siguiente = aux;
                        }
                    }
                }

            }
            actual = actual->siguiente;
        }
        return inicio2;
    }
```

Apartado 5. (**1 punto)**. Manejo de cadenas de caracteres.

Codificar una función **Concatenar** que reciba dos cadenas de caracteres (**cad1** y **cad2**) y las devuelva unidas (la primera seguida de la segunda) en una tercera cadena (**cad3**) de la que se devuelve su dirección de memoria. Se sabe que cada una de las cadenas que se reciben no pueden tener más de 50 caracteres.

```
char * Concatenar (char *, char *);
```

```
/*
    ...
*/
char * Concatenar (char *cadena1, char *cadena2)
{
    // Declaraciones
    char * cadena;
    int i, longitud, longitud1, longitud2;
    // Calculamos las longitudes de las cadenas
    longitud1 = strlen(cadena1);
    longitud2 = strlen(cadena2);
    longitud = longitud1 + longitud2;
    // Pedimos memoria dinámica para la cadena concatenada
    cadena = (char *) calloc (longitud + 1, sizeof(char));
    if (cadena == NULL)
        printf("\nNo hay suficiente memoria ... ");
    else
    {
        // Copiamos en cadena la cadena1
        for(i=0; i<longitud1; i++)
            cadena[i] = cadena1[i];
        // Copiamos en cadena la cadena2 a continuación
        for(i=0; i<longitud2; i++)
            cadena[longitud1 + i] = cadena2[i];
        // Metemos el '\0' al final de cadena
        cadena[longitud] = '\0';
    }
    return cadena;
}
```

Apartado 6. (**1 punto)**. Recursividad.

Diseñar una función recursiva **int Ackerman (int m, int n);** para calcular el valor de la función de Ackerman.

Se sabe que dicha función vale:

Ackerman (m, n) = Ackerman ((m-1), Ackerman (m, (n-1)))

Por otra parte, se conoce que:

Ackerman (m, n) = n + 1 si m = 0
Ackerman (m, n) = Ackerman (m-1, 1) si m > 0 y n = 0

Nota: Se supone que **m** y **n** son números enteros >= 0.

```
/*
   ...
*/
int Ackerman (int m, int n)
{
   int valor;

   if(m == 0)
      valor = n+1;
   else
      if(n == 0)
         valor = Ackerman(m-1,1);
      else
         valor = Ackerman(m-1, Ackerman(m, n-1));

   return valor;
}
```

Apartado 7. (**1 punto)**. Ejercicio de cálculo matemático.

Escribir una función **Cuenta_numeros** que reciba el valor de un número **n** entero y positivo, y devuelva cuántos números hay entre 1 y **n** que estén formados sólo por cifras impares. En caso de recibir un número negativo, la función devolverá el valor -1.

Se pide: a) Prototipo de la función.
b) Código de la función.

Ejemplo: Si se recibe el valor 25, devolverá 10.

1 2 **3** 4 **5** 6 **7** 8 **9** 10 **11** 12 **13** 14 **15** 16 **17** 18 **19** 20 21 22 23 24 25

Prototipo:

```
int Cuenta_numeros (int);
```

Función:

```
/*
    ...
*/
int Cuenta_numeros (int n)
{
    // Declaraciones
    int i, j, valor;
    // Comprobamos si el número es 0 o negativo
    if (n <= 0)
        valor = -1;
    else
    {
        valor = 0;
        for(i=1; i<=n; i++)
        {
            j = i;
            while ((j%10)%2 != 0)
                j = j/10;
            if(j == 0)
                valor++;
        }
    }
    return valor;
}
```