Fundamentos de Programación – 1º ASI

Tema 6 – Ejercicios resueltos de RECURSIVIDAD

6-01

Escribe una función recursiva que **sume los N primeros números naturales**. Es decir, que calcule N + (N - 1) + (N - 2) + ... + 2 + 1

```
#include <stdio.h>
#include <stdlib.h>

int sumatorio(int n);

int main(void)
{
    int n, suma;

    printf("Sumar números desde 1 hasta...");
    scanf("%i", &n);

    suma = sumatorio(n);

    printf("La suma desde 1 hasta %i es %i\n", n, suma);
    system("PAUSE");

    return 0;
}


int sumatorio(int n)
{
    int r;

    if (n == 0) r = 0;
    else r = n + sumatorio(n-1);

    return r;
}
```

6-02

Escribe un programa que pida dos números naturales, A y B, y calcule la **potencia** de A^B mediante una función recursiva

```
#include <stdio.h>
#include <stdlib.h>

long potencia(int b, int e);

int main(void)
{
    int b,e;
    long pot;

    printf("Cáculo recursivo de una potencia\n");
    printf("Introduzca base: "); scanf("%i", &b);
    printf("Introduzca exponente: "); scanf("%i", &e);

    pot = potencia(b,e);

    printf("%i elevado a %i es igual a %li\n", b, e, pot);
    system("PAUSE");

    return 0;
}


long potencia(int b, int e)
{
    long r;

    if (e == 0) r = 1;
    else r = b * potencia(b, e-1);
```

```
    return r;
}
```

6-03

La secuencia de números de **Fibonacci** 1, 1, 2, 3, 5, 8, 13, 21, 34, etc, se obtiene sumando a cada número natural el anterior (0 + 1 = 1, 1 + 1 = 2, 2 + 3 = 5, 3 + 5 = 8, etc). Escribe un módulo que escriba los números de Fibonacci hasta que sobrepasen las 5 cifras. Haz una versión iterativa y otra recursiva

```
#include <stdio.h>
#include <stdlib.h>

void fibonacci(int n, int n2);

int main(void)
{
    int b,e;
    long pot;

    printf("Cálculo recursivo de la secuencia de Fibonacci hasta 5 cifras:\n\n");

    printf("0 - 1 - ");
    fibonacci(0,1);

    printf("\n\n");
    system("PAUSE");

    return 0;
}


void fibonacci(int n, int n2)
{
    int r;

    r = n + n2;
    if (r < 99999)
    {
        printf("%i - ", r);
        fibonacci(n2, r);
    }
}
```

6-05

Escribe un programa que, por medio de funciones recursivas, sea capaz de calcular, en un vector de enteros, la suma, la media, el mínimo y el máximo.

```
#include <stdio.h>
#include <stdlib.h>

#define TAM 10

int calcula_maximo(int v[TAM], int max, int pos);
int calcula_minimo(int v[TAM], int min, int pos);
int calcula_suma(int v[TAM], int pos);
float calcula_media(int v[TAM], int pos);

int main(void)
{
    int i,b,e;
    long t;
    int v[TAM];
    int maximo, minimo, suma;
    float media;

    printf("Cálculos recursivo con un vector\n\n");

    printf("Generando un vector aleatorio de %i elementos...\n", TAM);
    t = time(NULL);
    srand(t);
    for (i=0; i<TAM; i++)
```

```
    {
        v[i] = rand()%20;
        printf("%i - ", v[i]);
    }
    printf("\n");

    maximo = calcula_maximo(v, -99999, 0);
    minimo = calcula_minimo(v, 99999, 0);
    suma = calcula_suma(v, 0);
    media = calcula_media(v, 0);

    printf("Máximo = %i, Mínimo = %i\n", maximo, minimo);
    printf("Suma = %i, Media = %.2f\n", suma, media);
    system("PAUSE");

    return 0;
}


int calcula_maximo(int v[TAM], int max, int pos)
{
    int r;

    if (v[pos] > max) max = v[pos];
    if (pos == TAM-1)
        r = max;
    else
        r = calcula_maximo(v, max, pos+1);

    return r;
}

int calcula_minimo(int v[TAM], int min, int pos)
{
    int r;

    if (v[pos] < min) min = v[pos];
    if (pos == TAM-1)
        r = min;
    else
        r = calcula_minimo(v, min, pos+1);

    return r;
}


int calcula_suma(int v[TAM], int pos)
{
    int r;

    if (pos == TAM-1)
        r = v[pos];
    else
        r = v[pos] + calcula_suma(v, pos+1);

    return r;
}


float calcula_media(int v[TAM], int pos)
{
    float r;

    if (pos == TAM-1)
        r = (float)v[pos]/TAM;
    else
        r = ((float)v[pos]/TAM) + calcula_media(v, pos+1);

    return r;
}
```

6-06

Escribe un programa que lea un número natural N por teclado y que, recursivamente, muestre todas las formas posibles de calcular N como sumas de otros números naturales.

Por ejemplo, el número 12 puede calcularse como 1+11, 2+10, 3+9, 4+8, 5+7 y 6+6

```
#include <stdio.h>
#include <stdlib.h>

int calcula_sumas(int n, int a, int b);

int main(void)
{
    int n;

    printf("Calcular todas las sumas que dan lugar a otro número\n\n");

    printf("Introduce un número entero positivo: ");
    scanf("%i", &n);

    calcula_sumas(n, 1, n-1);

    system("PAUSE");

    return 0;
}


int calcula_sumas(int n, int a, int b)
{
  if (a <= b)
  {
    if (a + b == n)
        printf("%i + %i = %i\n", a, b, n);

    calcula_sumas(n, a+1, b-1);
  }
}
```

6-07

Utiliza funciones recursivas para resolver el problema de **convertir números binarios a decimales**.

```
#include <stdio.h>
#include <stdlib.h>

int calcula_decimal(char nbin[100], int pos, int exponente);
int potencia(int base, int exponente);

int main(void)
{
    char nbin[100];
    int ndec;

    printf("Convertir de decimal a binario\n\n");

    printf("Introduce un número BINARIO (no se comprobará la entrada): ");
    gets(nbin);

    ndec = calcula_decimal(nbin, strlen(nbin)-1, 0);
    printf("El número decimal es: %i\n", ndec);

    system("PAUSE");

    return 0;
}


int calcula_decimal(char nbin[100], int pos, int exponente)
{
  int valor, resultado;

  if (pos < 0)
     resultado = 0;
  else
  {
     if (nbin[pos] == '1')
          valor = potencia(2, exponente);
     else
          valor = 0;+
     resultado = valor + calcula_decimal(nbin, pos-1, exponente+1);
  }
```

```
  return resultado;
}


int potencia(int base, int exponente)
{
    int i, r = 1;

    for (i=1; i<=exponente; i++) r = base * r;

    return r;
}
```

6-08

Escribe un procedimiento recursivo capaz de **ordenar un vector** v de n de números enteros mediante este algoritmo recursivo:

a. Si el vector tiene 1 elemento, ya está ordenado
b. Si el vector tiene 2 elementos, comprobar si están en orden. Si no lo están, intercambiarlos.
c. Si el vector tiene más de 2 elementos:
    i. Calcular el elemento medio (llamémosle k)
    ii. Ordenar el subvector que va de v[0] a v[k-1]
    iii. Ordenar el subvector que va de v[k] a v[n-1]
    iv. Mezclar los dos subvectores

```
#include <stdio.h>
#include <stdlib.h>

#define TAM 10

void ordenar(int v[TAM], int inf, int sup);
void mezclar(int v[TAM], int inf, int med, int sup);

int main(void)
{
    int i,b,e;
    long t;
    int v[TAM];
    int maximo, minimo, suma;
    float media;

    printf("Ordenación por métodos recursivos: MERGESORT\n\n");

    printf("Generando un vector aleatorio de %i elementos...\n", TAM);
    t = time(NULL);
    srand(t);
    for (i=0; i<TAM; i++)
    {
        v[i] = rand()%20;
        printf("%i - ", v[i]);
    }
    printf("\n");

    printf("\nOrdenando...");
    ordenar(v, 0, TAM-1);

    printf("\n\nVector ordenado. Ha quedado así:\n");
    for (i=0; i<TAM; i++)
        printf("%i - ", v[i]);
    printf("\n");

    system("PAUSE");

    return 0;
}


void ordenar(int v[TAM], int inf, int sup)
```

```
{
    int aux, med;

    if (sup == inf)
        return;

    if (sup - inf == 1)
        return;

    if (sup - inf == 2)
    {
        if (v[sup] < v[inf])
        {
                aux = v[sup];
                v[sup] = v[inf];
                v[inf] = aux;
        }
    }
    else
    {
        med = (inf + sup) / 2;
        ordenar(v, inf, med);
        ordenar(v, med, sup);
        mezclar(v, inf, med, sup);
    }
}


void mezclar(int v[TAM], int inf, int med, int sup)
{
    int salir = 0;
    int aux, i = 0;

    while (!salir)
    {
        if ((inf+i >= med) || (med+i >= sup))
           salir = 1;
        else if (v[inf+i] > v[med+i])
        {
           aux = v[inf+i];
           v[inf+i] = v[med+i];
           v[med+i] = aux;
        }
        i++;
    }
}
```

6-09

La mayoría de las **operaciones con árboles** se pueden plantear de forma recursiva (de hecho, es mucho más fácil hacerlo así que iterativamente). Escribe las siguientes funciones recursivas para árboles binarios de números enteros:

a. Mostrar por la pantalla el contenido de todos los nodos-hoja
b. Contar el número de nodos de un árbol
c. Sumar el contenido de todos los nodos del árbol
d. Hallar la profundidad máxima de un árbol
e. Averiguar si un árbol está o no completo

```
#include <stdio.h>
#include <stdlib.h>

#define TAM 100

struct s_nodo
{
    long dato;
    struct s_nodo *hijo_izq, *hijo_der;
};


void insertar(struct s_nodo **nodo, struct s_nodo **padre, long dato);
void inorden(struct s_nodo *nodo);
```

```
void mostrar_hojas(struct s_nodo *nodo);
long int sumar_nodos(struct s_nodo *nodo);
int completo(struct s_nodo *nodo);

int main(void)
{
    struct s_nodo *raiz = NULL;
    long n, t1, t2;
    int i, prof;

    printf("Varios ejemplos de funciones recursivas para árboles\n\n");
    printf("Generando un árbol aleatorio de %i elementos...\n", TAM);
    t1 = time(NULL);
    srand(t1);

    for (i=1; i<=TAM; i++)
    {
        n = (rand() % TAM*3)+1;
        insertar (&raiz, NULL, n);
    }

    printf("\n\nRecorrido IN-ORDEN del árbol:\n");
    inorden(raiz);

    printf("\n\nLos nodos hoja son:\n");
    mostrar_hojas(raiz);

    printf("\n\nSuma de todos los nodos: %li\n", sumar_nodos(raiz));

    printf("\n\nProfundidad del árbol: %i\n", profundidad(raiz, 0));

    if (completo(raiz))
        printf("\n\nEl árbol está COMPLETO\n\n");
    else
        printf("\n\nEl árbol está INCOMPLETO\n\n");


    system("PAUSE");

    return 0;
}


// Inserta un dato en el árbol manteniéndolo ordenado
void insertar(struct s_nodo** nodo, struct s_nodo** padre, long int n)
{
  if (*nodo == NULL)
  {
     *nodo = (struct s_nodo*) malloc(sizeof(struct s_nodo));
     if (!nodo) {
        printf("Error de asignaciï¿½ de memoria\n");
           exit(-1);
     }
     (*nodo)->dato = n;
     (*nodo)->hijo_izq = NULL;
     (*nodo)->hijo_der = NULL;
     if (padre!=NULL)
     {
         if (n > (*padre)->dato) (*padre)->hijo_der = *nodo;
            else (*padre)->hijo_izq = *nodo;
     }
  }
  else
  {
     if (n > (*nodo)->dato) insertar(&(*nodo)->hijo_der, nodo, n);
     else insertar(&(*nodo)->hijo_izq, nodo, n);
  }
}


// Recorre el árbol en orden, mostrando el contenido de los nodos
void inorden(struct s_nodo *nodo)
{
    if (nodo != NULL)
    {
        inorden(nodo->hijo_izq);
        printf("%i - ", nodo->dato);
        inorden(nodo->hijo_der);
    }
}
```

```
// Recorre el árbol mostrando sólo el contenido de los nodos-hoja
void mostrar_hojas(struct s_nodo *nodo)
{
    if ((nodo->hijo_der == NULL) && (nodo->hijo_izq == NULL))  // Es una hoja
    {
        printf("%i - ", nodo->dato);
    }
    else    // No es una hoja: continuar el recorrido
    {
        if (nodo->hijo_izq != NULL)
           mostrar_hojas(nodo->hijo_izq);
        if (nodo->hijo_der != NULL)
           mostrar_hojas(nodo->hijo_der);
    }
}


// Suma el contenido de todos los nodos del árbol
long int sumar_nodos(struct s_nodo *nodo)
{
    if (nodo != NULL)
        return nodo->dato + sumar_nodos(nodo->hijo_der) + sumar_nodos(nodo->hijo_izq);
    else
        return 0;
}


// Calcula la profundidad máxima del árbol (p)
int profundidad(struct s_nodo *nodo, int p)
{
    int p_izq, p_der, prof;

    if (nodo != NULL)
    {
        // Profundidad del subárbol izquierdo
        p_izq = profundidad(nodo->hijo_izq, p+1);
        // Profundidad del subárbol derecho
        p_der = profundidad(nodo->hijo_der, p+1);
        // La profundidad máxima será la mayor entre p_izq y p_der
        if (p_izq > p_der)
                prof = p_izq;
        else
                prof = p_der;
    }
    else  // Si nodo == NULL, ya no hay más nodos, así que la profundidad es p
        prof = p;

    return prof;
}


// Decide si el árbol está completo o no
int completo(struct s_nodo *nodo)
{
    int c;

    if (nodo != NULL)
    {
        if ((nodo->hijo_izq != NULL) && (nodo->hijo_der != NULL))
                // Nodo con DOS hijos. Estará completo si sus dos subárboles también lo están
                c = 1 && completo(nodo->hijo_izq) && completo(nodo->hijo_der);
        else if ((nodo->hijo_izq == NULL) && (nodo->hijo_der == NULL))
                // Nodo con CERO hijos. Está completo por definición
                c = 1;
        else    // Nodo con UN hijo. No está completo por definición
                c = 0;
    }
    else        // Si nodo == NULL, este subárbol sí está completo (porque está vacío)
        c = 1;

    return c;

}
```

6-10

Escribe un programa capaz de recorrer (y mostrar) los registros de un archivo secuencial *al revés*, es decir, desde el último hasta el primero. Puedes utilizar el archivo de datos del Videoclub (tema 6) para probarlo. Trata de pensar en una solución recursiva al problema.

```
#include <stdio.h>
#include <stdlib.h>

struct s_pelicula
{
   char titulo[50];
   char director[20];
   char reparto[200];
   char genero[20];
   char nacionalidad[10];
   int duracion;
   char borrado;
   int codigo;
};

void listar_datos(void);

FILE *f;      // El flujo será una variable global para evitar problemas con el fichero

int main(void)
{
   printf("EJERCICIOS DE FUNDAMENTOS DE PROGRAMACIÓN - TEMA 8\n");
   printf("Recorrer un archivo secuencial hacia atrás recursivamente\n\n");

   f = fopen("video.dat", "rb");
   if (f == NULL)
   {
      printf("Error al abrir el archivo de datos\n");
      system("PAUSE");
      return -1;
   }
   printf("El contenido del archivo video.dat leído al revés es:\n\n");

   printf("--------------------------------------------------------------------------------\n");
   listar_datos();
   printf("--------------------------------------------------------------------------------\n");
   fclose(f);

   system("PAUSE");
   return 0;
}


// Muestra el contenido del archivo secuencial de datos por la pantalla
// El recorrido se hace HACIA ATRAS y RECURSIVAMENTE
void listar_datos(void)
{
   struct s_pelicula peli;
   int n;

   n = fread(&peli, sizeof(struct s_pelicula), 1, f);   // Leemos un registro
   if ((n > 0) && (peli.borrado == '-'))
   {
      if (feof(f))     // Hemos leído el último registro, así que lo mostramos
          printf("%-3i %s (%s), %s, %s, %i min\n", peli.codigo, peli.titulo, peli.director,
peli.nacionalidad, peli.genero, peli.duracion);
      else
      {
          listar_datos();   // No hemos leído el último: leemos los demás antes de mostrarlo
          printf("%-3i %s (%s), %s, %s, %i min\n", peli.codigo, peli.titulo, peli.director,
peli.nacionalidad, peli.genero, peli.duracion);
      }
   }
}
```