Fundamentos de Programación. Grado de Telecomunicación. Convocatoria de Febrero. Curso 2014/2015

Examen Escrito Febrero (60 % de la nota final)

Tiempo: dos horas y media.
IMPORTANTE: Los algoritmos han de ir correctamente explicados.

1. (1 puntos) Hacer una función para pasar un número entero positivo a binario. La función recibirá el número entero positivo y debe devolver una cadena de caracteres con el número pasado a binario. Hay que decidir como se devuelve esa cadena de caracteres y hacer un pequeño programa principal que ilustre como se utiliza la función realizada.

```
/*************************************************************************/
/* Programa para pasar un numero entero positivo a binario. Pide el numero  */
/* y imprime el numero en binario.                                       */
/*************************************************************************/

#include<stdio.h>
#include<alloc.h>

/*************************************************************************/
/*                                PROTOTIPOS                             */
/*************************************************************************/

int numero_cifras(int numero);
char *pasabinario(int numero);

/*************************************************************************/
/* int numero_cifras(int numero). Funcion que recibe un numero entero posi- */
/* tivo y devuelve el numero de cifras que tendra este numero representado  */
/* en binario. Realiza un bucle y va haciendo division entera entre 2 hasta */
/* que el resultado sea < 2.                                             */
/*************************************************************************/

int numero_cifras(int numero)
{
  int cifras=1;

  while(numero > 1)
  {
    numero = numero / 2;
    cifras++;
  }

  return(cifras);
}
```

```
/*************************************************************************/
/* char *pasabinario(numero). Funcion que recibe un numero entero positivo  */
/* y devuelve una cadena de caracteres con el numero representado en bina-  */
/* rio. Utiliza a funcion numero_cifras para calcular el numero de cifras   */
/* que tendra la cadena en binario y pedir dinamicamente el vector para al- */
/* macenar la cadena. El usuario debera liberar esta cadena de caracteres.  */
/* calcula las cifras haciendo utilizando modulo y division entera hasta que*/
/* numero es < 2.                                                           */
/*************************************************************************/

char *pasabinario(int numero)
{
  char *vector;
  int cifras, i = 0;

  cifras = numero_cifras(numero);
  vector = (char*) malloc((cifras +1)*sizeof(char));

  while (numero >1)
  {
    vector[cifras-1-i] = '0' + numero % 2;
    numero = numero / 2;
    i++;
  }

  vector[0] = '0' + numero;
  vector[cifras] = '\0';

  return(vector);

}

/*************************************************************************/
/*                          PROGRAMA PRINCIPAL                           */
/*************************************************************************/

int main()
{
  int numero;
  char *binario;

  printf("\nintroduce un numero = ");
  scanf("%d", &numero);

  binario=pasabinario(numero);

  printf("El numero %d en binario es %s\n", numero, binario);

  free(binario);
  return(0);
}
```

2. (2 puntos) Hacer una función que implemente el método de ordenación de la burbuja para ordenar una lista simplemente enlazada. Donde el campo info son números enteros positivos. Realizando una ordenación ascendente por el campo info.

```
struct Nodo{
int info;
struct Nodo *sig;
};

/*************************************************************************/
/* ordena(struct Nodo *inicio). Funcion que ordena una lista enlazada de    */
/* numeros enteros positivos. Recibe el puntero incio de la lista y recorre */
/* la lista intercambiando hasta que en una pasada no se cambia ningun valor*/
/*************************************************************************/

void ordena(struct Nodo *inicio)
{
  struct Nodo *cursor;
  int cambio = 1;

  if (inicio != NULL)
  {
    while(cambio)
    {
      cursor = inicio;
      cambio = 0;
      while(cursor->sig != NULL)
      {
        if(cursor->info > cursor->sig->info)
        {
          intercambia(&(cursor->info), &(cursor->sig->info))
          cambio = 1;
        }
        cursor = cursor->sig;
      }
    }  /* while(cambio)*/
  }    /* if */
}
/*************************************************************************/
/* Funcion que intercambia los valores de las variables apuntadas por a y b */
/*************************************************************************/

void intercambia(int *a, int *b)
{
  int aux;

  aux = *a;
  *a = *b;
  *b = aux;
}
```

3. (3 puntos) Una matriz de enteros se dice que es mágica, si es cuadrada (de dimensiones NxN), y si:

   a) Aparecen todos los números enteros desde 1 a N^2

   b) La suma de los elementos de cada fila, cada columna, y las dos diagonales principales dan el mismo valor.

   Por ejemplo, la matriz cuadrada siguiente de dimensiones 5x5 contiene todos los elementos del 1 al 25, y tiene como suma común 65.

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

   Haga una función booleana que reciba una matriz cuadrada como argumento, y devuelva 1 si la matriz es mágica y 0 si no lo es.

```
Suponemos que tenemos un define global N que nos da la maxima dimension de la
matriz y que ademas tenemos un utilm que es el util que ahora estamos
utilizando para filas y columnas. El mismo valor pues las matrices siempre
son cuadradas. Por supuesto utilm ha de ser <= N.

#define N 100

/*************************************************************************/
/* Funcion que suma de una matriz cuadrada de util utilm la fila fil y de- */
/* vuelve el valor de esta suma. Hace un bucle for para recorrer todos los */
/* elementos de la fila y calcular la suma.                              */
/*************************************************************************/

int sumaFila(int matriz[][N], int utilm, int fil)
{
  int i, salida = 0;

  for(i=0; i<utilm ; i++)
    salida = salida + matriz[fil][i];

  return(salida);
}

/*************************************************************************/
/* Funcion que suma de una matriz cuadrada de util utilm la columna col y  */
/* devuelve el valor de la suma. Utiliza un bucle for para recorrer la     */
/* columna indicada e ir sumando todos los valores.                      */
/*************************************************************************/

int sumaColumna(int matriz[][N], int utilm, int col)
{
  int i, salida = 0;
```

```
  for(i=0; i<utilm ; i++)
    salida = salida + matriz[i][col];

  return(salida);
}


/*************************************************************************/
/* Funcion que suma de una matriz cuadrada de until utilm una diagonal y    */
/* devuelve el valor. Si diagonal vale 1 suma la diagonal principal y si es */
/* distinto de 1 suma la diagonal secundaria. Hace un for para recorrer la  */
/* diagonal e ir sumando sus valores.                                       */
/*************************************************************************/

int sumaDiagonal(int matriz[][N], int utilm, int diagonal)
{
  int i, salida = 0;

  for(i=0; i<utilm ; i++)
    if (diagonal == 1)
      salida = salida + matriz[i][i];
    else
      salida = salida + matriz[i][utilm-1-i];

  return(salida);
}

/*************************************************************************/
/* Funcion booleana que comprueba si todas las filas de una matriz          */
/* cuadrada de utilm suman el mismo valor comun. Se hace un for para reco-  */
/* rrer todas las filas y haciendo uso de la funcion sumaFila comprobar     */
/* si suma el mismo valor comun.                                            */
/*************************************************************************/

int comprueba_filas(int matriz[][N], int utilm, int comun)
{
 int i, salir = 1;

 for(i=0;i<utilm && salir;i++)
 {
   if (sumaFila(matriz, utilm, i) != comun)
     salir = 0;
 }

 return(salir);
}

/*************************************************************************/
/* Funcion booleana que comprueba si todas las columnas de una matriz       */
/* cuadrada de utilm suman el mismo valor comun. Se hace un for para reco-  */
/* rrer todas las columnas y haciendo uso de la funcion sumaColumna         */
```

```
/* comprobar si suma el mismo valor comun.                                */
/************************************************************************/

int comprueba_columnas(int matriz[][N], int utilm, int comun)
{
 int i, salir = 1;

 for(i=0;i<utilm && salir;i++)
 {
   if (sumaColumna(matriz, utilm, i) != comun)
     salir = 0;
 }

 return(salir);
}

/************************************************************************/
/* Funcion booleana que recibe una matriz cuadrada y su util y un valor */
/* comun si las dos diagonales suman el valor comun devuelve 1 si alguna */
/* no suma comun devuelve 0. Hace uso de la funcion sumaDiagonal que suma */
/* la diagonal de la matriz que se pida.                                  */
/************************************************************************/

int comprueba_diagonales(int matriz[][N], int utilm, int comun)
{
 int sumadiagoprincipal, sumadiagonalsecundaria;

 sumadiagonalprincipal = sumaDiagonal(matriz, utilm, 1);
 sumadiagonalsecundaria = sumaDiagonal(matriz, utilm, -1);

 if (sumadiagonalprincipal == comun && sumadiagonalsecundaria == comun)
     return(1);

 return(0);
}

/************************************************************************/
/* Funcion que recibe una matriz cuadrada de dimension utilm y devuelve 1 si*/
/* todas las filas columnas y diagonales suman lo mismo. Hace uso de las */
/* funciones comprueba_filas, comprueba_columnas y comprueba_diagonales para*/
/* comprobar cada caso si todos dan true la funcion devolvera true.       */
/************************************************************************/

int todas_sumas(int matriz[][N], int utilm)
{
  int comun;

  comun = sumaFila(matriz, utilm, 0);

  if(comprueba_filas(matriz, utilm, comun) &&
     comprueba_columnas(matriz, utilm, comun) &&
```

```
     comprueba_diagonales(matriz, utilm, comun))
     return(1);

  return(0);
}

/*************************************************************************/
/* Funcion booleana que devuelve 1 si en la matriz recibida como argumento  */
/* estan todos los numeros entre 1 y utilm*utilm. Se hace una copia de la   */
/* matriz pasada y sobre esa copia se van marcando los numeros usando la    */
/* funcion marca.                                                           */
/*************************************************************************/

int todos_numeros(int matriz[][N], utilm)
{
  int matriz_copia[N][N], i, tope, existe = 1;

  copia_matriz(matriz, matriz_copia, utilm);
  tope = utilm*utilm;
  for(i=1;i< tope && existe;i++)
    existe=marca(matriz_copia, utilm, i);

  return(exite);
}

/*************************************************************************/
/* Funcion que busca en una matriz cuadrada de util utilm si el elemento    */
/* elem esta en la matriz si esta marca la posicion con -1 y devuelve 1 si  */
/* esta devuelve 0. Hace un doble bucle recorriendo toda la matriz y viendo */
/* si alguno de los elementos no marcados coincide con el elemento pasado.  */
/*************************************************************************/

int marca(int matriz[][N], int utilm, int elem)
{
 int i, j, salida = 1;

 for(i=0;i<utilm && salida;i++)
   for(j=0;j<utilm && salida;j++)
     if(matriz[i][j]!= -1 && matriz[i][j] == elem)
     {
       matriz[i][j]= -1;
       salida = 0;
     }

 return(!salida);
}

/*************************************************************************/
/* int copia_matriz(int matriz[][N], int matriz_copia[][N], utilm). Funcion */
/* que copia una matriz cuadrada a otra matriz_copia. utilm es el util en   */
/* filas y columnas. Hace un doble bucle copiando todos los elementos.      */
```

```
/*************************************************************************/

void copia_matriz(int matriz[][N], int matriz_copia[][N], utilm)
{
  int i, j;

  for(i=0;i<utilm;i++)
    for(j=0;j<utilm;j++)
      matriz_copia[i][j] = matriz[i][j];
}

/*************************************************************************/
/* int magica(int matriz[][N], int utilm). Funcion que recibe una matriz    */
/* cuadrada de enteros positivos y el util de la matriz que como es cuadrada*/
/* solo es un numero. Llama a todos_numeros y todas_sumas para comprobar las*/
/* dos condiciones para ver si la matriz es magica.                         */
/*************************************************************************/

int magica(int matriz[][N], int utilm)
{
 if(todos_numeros(matriz, utilm) && todas_sumas(matriz, utilm))
   return(1);

 return(0);
}
```