

Examen IO, cuatrimestre 1A, (15 Junio 2006)

Notas provisionales 22/06/06 Período Alegaciones 23/06/06 (Mañana) Notas definitivas 26/06/06

Nombre y Apellidos: _____ DNI: _____

Instrucciones: Duración examen 3 horas. Las preguntas 2 y 3 deben entregarse en folios separados.

1. Cuestiones (3.0 puntos, 35 minutos)

Responda y justifique brevemente cada una de las siguientes preguntas en el mismo folio.

a) Dado el siguiente código:

```
#include "tortuga.h"
#define pi 180
void x(int j, int k)
{
    int i;
    for(i=0;i<j;i++)
    {
        avanzar(j);
        girar(k);
    }
    girar(k/2);
}

void main()
{
    int t=4;
    inicializar();
    x(t,pi/2);
    x(t/2,pi/2);
    finalizar();
}
```

Dibuje la salida que saldrá por pantalla al ejecutar el código anterior.

b) Completa el siguiente fragmento de código colocando las dos invocaciones a f1() desde f2(). Coloca tu respuesta sobre los puntos suspensivos de f2().

```
void f1(int *a)
{
    (*a)++ ;
}

void f2( int *p)
{
    int q=2 ;

    ..... /* Aplicar f1() a q */

    ..... /* Aplicar f1() a p */

}
```

c) Dado el siguiente fragmento de código, cuál sería la salida por pantalla?

```
#include <stdio.h>
#define N 7
void f1(char v1[], char v2[])
{
    int i=0;
    do
    {
        v1[i]=v2[i];
        i++;
    } while(v2[i]!='\0');
}

void main()
{
    char line1[N]='\L','I','N','E','A','1','\0';
    char line2[]="linea2";
    f1(line1,line2);
    printf("%s",line1);
}
```

Respuesta:

d) Construya el fragmento de código de la función *Tras Matriz* para trasponer la matriz A[5][5].

```
void Tras_Matriz(int A[5][5])
```

e) ¿Qué sentencia colocarías dentro de la función `Imprimir_Cam` para acceder al tercer elemento del vector `vecInt`?

```
#define M 20
typedef struct
{
    float a;
    int vecInt[M];
}Ttipus;

typedef struct
{
    int A;
    Ttipus B;
}Tdades;

void Imprimir_Cam(Tdades *dades)
{
    printf("%d",.....);
}

void main()
{
    Tdades vecDades;
    Imprimir_Cam(&vecDades);
}
```

2. Problema 2. El juego de la oca (2 puntos, 50 minutos)

El texto que se muestra a continuación es una versión simplificada del juego de la oca, basada en el reglamento oficial del ministerio de cultura y deportes.

REGLAS DEL JUEGO DE LA OCA (Simplificadas)

Normas: El tablero está compuesto por 63 casillas numeradas del 1 al 63. Hay algunas casillas especiales:

- Todas las casillas múltiples de 6 tienen representadasocas.

Si la ficha del jugador cae en una casilla "Oca" salta a la "oca" inmediatamente siguiente y vuelva a tirar. El jugador acostumbra decir: "De oca a oca y tiro por que me toca".

En esta versión del juego (simplificada) el turno finaliza cuando el jugador alcanza o supera la casilla 63.

El objetivo de este problema es realizar una función `int turno(int pos_actual);` que simule *un turno completo de un jugador*. La función debe tener en cuenta la posición actual del jugador, y la distribución de las casillas especiales en el tablero (ocas), según las reglas anteriores. El resultado de la función debe ser la posición final del jugador al finalizar su turno, incluyendo en el turno los saltos y repeticiones consecuentes de caer en las casillas especiales. No hace falta hacer control de errores.

Al mismo tiempo, la función debe mostrar por pantalla la evolución de la jugada. Ejemplo:

```
Empiezo el turno en la casilla 10.
Tiro el dado y saco un 2. Caigo en la casilla 12.
De oca en oca y tiro porque me toca. Salto a la casilla 18.
Tiro el dado y saco un 3. Caigo en la casilla 21.
Termino el turno en la casilla 21.
```

En este ejemplo, la posición inicial es 10 y el resultado de la función, la posición final, debe ser 21.

Para simular el dado puede utilizarse la función `int tirar_dado();` que devuelve un entero al azar entre 1 y 6.

3. Parque Móvil (90 minutos, 5 puntos)

Debes realizar un programa que junta ficheros cuyo contenido son listas de vehículos (parque móvil). Los ficheros que el programa juntará pueden contener errores (vehículos repetidos, datos incompletos, etc) y las listas de vehículos que contienen no están ordenadas. El objetivo del programa es juntar todos los ficheros en un único fichero que esté ordenado y que no contenga errores.

Un parque móvil es una lista ordenada de vehículos. Un vehículo es una serie de informaciones como matrícula, número de bastidor, modelo y lista de propietarios. Ya se han definido los siguientes tipos de datos:

```
#define CIERTO 1
#define FALSO 0
#define MAXCAR 255
```

```

#define MAXVE 1000
#define MAXPRO 10

typedef char Tcadena[MAXCAR+1];
typedef struct
{
    int    numero;      /*El número de la matrícula*/
    char  letras[4];   /*Las 3 letras de la matrícula*/
} Tmatricula;
typedef struct
{
    Tmatricula  matricula;      /*Matricula*/
    long        bastidor;      /*Número de bastidor*/
    Tcadena     modelo;        /*Modelo del vehículo*/
    int         np;            /*Número de propietarios*/
    Tcadena     propietario[MAXPRO]; /*Lista de propietarios*/
} Tvehiculo;
typedef struct
{
    int         nve;           /*Número de vehículos en la lista*/
    Tvehiculo  ve[MAXVE];    /*Lista de vehículos*/
} TparqueMovil;

```

Un fichero que contenga un parque móvil guarda la información de un vehículo en cada línea. La estructura de una línea del fichero es la siguiente:

```

<numeroMatri>-<letrasMatri> <numBastidor> <modelo>:<nomPropietario1>:<nomPropietario2>:
...

```

Aquí puedes ver un ejemplo de 3 ficheros de entrada y el fichero de salida que generará el programa:

```

FICHERO 1
1234-ccc 223456789 seat toledo tdi:juan perez:pepe perez
1134-aaa 123456789 seat ibiza tdi:juan perez:luis perez

FICHERO 2
1254-aaa 123426789 seat toledo tdi:juan palomo
1534-aaa 123452700 seat toledo tdi:juan sanchez:pepe garcia
1134-aaa 123456789 seat ibiza tdi:juan perez:luis perez

FICHERO 3
1235-bbb 100256789 seat toledo tdi:juan xxx
1114-aaa 122456789 seat alambra tdi:juan garcia:pepe garcia:luis garcia

FICHERO SALIDA
1114-aaa 122456789 seat alambra tdi:juan garcia:pepe garcia:luis garcia
1134-aaa 123456789 seat ibiza tdi:juan perez:luis perez
1254-aaa 123426789 seat toledo tdi:juan palomo
1534-aaa 123452700 seat toledo tdi:juan sanchez:pepe garcia
1235-bbb 100256789 seat toledo tdi:juan xxx
1234-ccc 223456789 seat toledo tdi:juan perez:pepe perez

```

La lista de vehículos del fichero de salida debe estar ordenada. El criterio para ordenar los vehículos es: las letras de la matrícula, el número de la matrícula, el número de bastidor y el modelo.

Se pide que hagas las siguientes funciones:

a) int comparaVehiculos(Tvehiculo v1, Tvehiculo v2)

Retorna un número menor que 0 si $v1 < v2$, retorna 0 si $v1 = v2$, y retorna un número mayor que 0 si $v1 > v2$. La comparación entre vehículos debe primero comparar las letras de la matrícula, después el número de la matrícula, después el número de bastidor y finalmente el modelo.

b) int leeVehiculo(FILE *fp, Tvehiculo *v)

Lee una línea del fichero apuntado por `fp` y almacena la información leída en la variable apuntada por `v`. La función retorna CIERTO si ha podido leer la línea sin problemas, en caso contrario retorna FALSO. El formato de la línea es el descrito en el enunciado.

c) void escribirVehiculo(FILE *fp, Tvehiculo v)

Escribe en una línea del fichero apuntado por `fp` la información del vehículo `v`. El formato es el descrito en el enunciado.

d) int buscarVehiculo(TparqueMovil p, Tvehiculo v, int *pos)

Busca el vehículo `v` en el parque móvil `p`. Si encuentra el vehículo `v` retorna `CIERTO`, en caso contrario retorna `FALSO`. En la variable apuntada por `pos` escribe la posición de `v` en `p`, o bien la posición donde debería estar para mantener el orden en `p`. El parque móvil `p` está ordenado de menor a mayor.

e) void insertaVehiculo(TparqueMovil *p, Tvehiculo v, int pos)

Inserta el vehículo `v` en la posición `pos` del parque móvil apuntado por `p`. Esta función no comprueba ninguna condición de error, se limita a insertar el vehículo.

f) void resetParqueMovil(TparqueMovil *p)

Inicializa a vacío el parque móvil apuntado por `p`.

g) int leerParqueMovil(TparqueMovil *p)

Lee de un fichero un parque móvil y lo deja en la variable apuntada por `p`. La función pedirá por pantalla el nombre del fichero. Leerá todo el fichero dejando los vehículos leídos ordenados dentro del parque móvil. Recuerda que los vehículos **NO** están ordenados en el fichero. La lectura del fichero será línea a línea hasta que el fichero no se acabe. Si una línea da un error de lectura, se mostrará un mensaje en pantalla indicando línea del fichero incompleta y se pasará a la siguiente línea. Si la función encuentra vehículos repetidos en el fichero sacará un mensaje por pantalla indicando esta situación, y sólo insertará en el parque móvil una vez el vehículo repetido. La función retorna `CIERTO` si no se puede abrir el fichero o la lista de vehículos no cabe en el parque móvil, en caso contrario retorna `FALSO`.

h) void escribirParqueMovil(TparqueMovil p)

Escribe en un fichero el parque móvil `p`. La función pedirá por pantalla el nombre del fichero. El formato del fichero es el descrito por el enunciado.

i) int unionParques(TparqueMovil p1, TparqueMovil p2, TparqueMovil *p3)

Deja en la variable apuntada por `p3` la unión de las listas `p1` y `p2`. La lista unión debe estar ordenada y no debe contener vehículos repetidos. La función retorna `CIERTO` si ha habido error, es decir si no ha podido realizar la unión y `FALSO` en caso contrario.

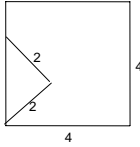
j) void main()

El programa principal ira leyendo ficheros de entrada y realizando la unión con un parque móvil ordenado. Esto se repetirá mientras la lectura y la unión puedan realizarse sin errores. Si se produce un error el programa mostrará en pantalla el motivo del error (lectura fallida o unión fallida).

Antes de acabar el programa debe escribir en un fichero el parque móvil ordenado que exista en ese momento.

1) SOLUCION P1

a)



b) f1(&q), f1(p)

c) linea2

```
d) void Tras_Matriz(int A[5][5])
{
    int i,j,aux;
    for (i=0;i<5;i++)
        for (j=i+1;j<5;j++)
        {
            aux=A[i][j];
            A[i][j]=A[j][i];
            A[j][i]=aux;
        }
}
```

e) dades->B.vecInt[2]

2) SOLUCION P2

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
void init_dado(){
    srand(time(0)); //inicializa el generador de numeros aleatorios.
}
int tirar_dado(){
    return rand()%6+1; //rand devuelve un entero >=0 al azar.
}
int turno(int pos_actual){
    int dado;
    int destino;
    int repetir;
    destino=pos_actual;
    printf("Empiezo el turno en la casilla %d\n",pos_actual);
    do{
        repetir=0;
        dado=tirar_dado();
        printf("Saco un %d. ",dado);
        destino=destino+dado;
        printf("Caigo en la casilla %d\n", destino);
        if (destino<63 && destino%6==0){
            printf("De oca en oca y tiro porque me toca\n");
            destino=destino+6;
            if (destino<63)
                repetir=1;
            printf("Salto a la casilla %d\n", destino);
        }
    }while (repetir);
    printf("Termino el turno en la casilla %d\n",destino);
    return destino;
}
void main(){
    int pos=1;
    init_dado();
    do{
        pos=turno(pos);
    }while (pos<63);
}
```

3) SOLUCIÓN P3

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

```

#define CIERTO 1
#define FALSO 0
#define MAXCAR 255
#define MAXVE 100
#define MAXPRO 10

typedef char Tcadena[MAXCAR+1];

typedef struct
{
    int numero;
    char letras[4];
} Tmatricula;

typedef struct
{
    Tmatricula matricula;
    long bastidor;
    Tcadena modelo;
    int np;
    Tcadena propietario[MAXPRO];
} Tvehiculo;

typedef struct
{
    int nve;
    Tvehiculo ve[MAXVE];
} TparqueMovil;

int comparaVehiculos( Tvehiculo v1, Tvehiculo v2 )
{
    int k;

    k = strcmp( v1.matricula.letras, v2.matricula.letras );
    if (k == 0)
    {
        k = v1.matricula.numero - v2.matricula.numero;
        if (k == 0)
        {
            k = (int)(v1.bastidor - v2.bastidor);
            if (k == 0)
            {
                k = strcmp( v1.modelo, v2.modelo );
            }
        }
    }
    return k;
}

int leeVehiculo( FILE *fp, Tvehiculo *v )
{
    int i, k, leido;
    char cc;

    leido = FALSO;
    k = fscanf( fp, "%d-%s %ld %[^:]:", &v->matricula.numero,
                v->matricula.letras,
                &v->bastidor, v->modelo );

    if (k == 4)
    {
        i = 0;
    }
}

```

```

do
{
    k = fscanf( fp, "%[^\n:]%c", &v->propietario[i][0], &cc );
    if (k == 2)
        i++;
}while (cc != '\n' && k==2);
v->np = i;
leido = CIERTO;
}

return leido;
}

void escribirVehiculo( FILE *fp, Tvehiculo v )
{
    int i;

    fprintf( fp, "%d-%s %ld %s:", v.matricula.numero,
v.matricula.letras,
v.bastidor, v.modelo );
    for (i= 0; i< v.np-1; i++)
        fprintf( fp, "%s:", &v.propietario[i][0] );

    fprintf( fp, "%s\n", &v.propietario[v.np-1][0] );
}

int buscarVehiculo( TparqueMovil p, Tvehiculo v, int *pos )
{
    int encontrado, ele_mayor, i, k;

    encontrado = FALSO;
    ele_mayor = FALSO;
    *pos = p.nve;
    i = 0;

    while (i< p.nve && !encontrado && !ele_mayor)
    {
        k = comparaVehiculos( p.ve[i], v );
        if (k == 0)
        {
            encontrado = CIERTO;
            *pos = i;
        }
        else if (k > 0)
        {
            ele_mayor = CIERTO;
            *pos = i;
        }
        else
            i++;
    }

    return encontrado;
}

void insertaVehiculo( TparqueMovil *p, Tvehiculo v, int pos )
{
    int i;

    for (i= p->nve-1; i>= pos; i--)
        p->ve[i+1] = p->ve[i];
}

```

```

    p->ve[pos] = v;
    p->nve++;
}

void resetParqueMovil( TparqueMovil *p )
{
    p->nve = 0;
}

int leerParqueMovil( TparqueMovil *p )
{
    Tcadena      nombre;
    FILE          *fp;
    Tvehiculo     v;
    int           error, leido, existe, pos, hecho;

    printf( "Fichero entrada: " );
    scanf( "%[^\n]*c", nombre );

    fp = fopen( nombre, "r" );
    if (fp == NULL)
    {
        printf( "El fichero %s no se puede abrir\n", nombre );
        error = CIERTO;
    }
    else
    {
        error = FALSO;
        resetParqueMovil( p );
        while (!feof(fp))
        {
            leido = leeVehiculo( fp, &v );
            if (leido)
            {
                existe = buscarVehiculo( *p, v, &pos );
                if (existe)
                    printf( "Vehiculo repetido en fichero de entrada\n" );
                else
                {
                    if (p->nve < MAXVE)
                        insertaVehiculo( p, v, pos );
                    else
                    {
                        printf( "Parque movil lleno\n" );
                        error = CIERTO;
                    }
                }
            }
        }
        else
            printf( "Linea del fichero incompleta\n" );
        fclose(fp);
    }

    return error;
}

void escribirParqueMovil( TparqueMovil p )
{
    Tcadena      nombre;

```

```

FILE          *fp;
int           i;

do
{
    printf( "Fichero salida: " );
    scanf( "%[^\\n]*c", nombre );

    fp = fopen( nombre, "w" );
    if (fp == NULL)
        printf( "El fichero %s no se puede abrir\\n", nombre );
}while (fp == NULL);

for (i= 0; i< p.nve; i++)
    escribirVehiculo( fp, p.ve[i] );

fclose(fp);
}

int unionParques( TparqueMovil p1, TparqueMovil p2, TparqueMovil *p3 )
{
    int i, j, k, x, error;

    i=0; j=0; k=0;
    while (i< p1.nve && j< p2.nve && k< MAXVE)
    {
        x = comparaVehiculos( p1.ve[i], p2.ve[j]);
        if (x < 0)
        {
            p3->ve[k] = p1.ve[i];
            i++;
        }
        else if (x == 0)
        {
            p3->ve[k] = p1.ve[i];
            i++;
            j++;
        }
        else
        {
            p3->ve[k] = p2.ve[j];
            j++;
        }
        k++;
    }

    while (i< p1.nve && k< MAXVE)
    {
        p3->ve[k] = p1.ve[i];
        i++;
        k++;
    }

    while (j< p2.nve && k< MAXVE)
    {
        p3->ve[k] = p2.ve[j];
        j++;
        k++;
    }

    if (k==MAXVE && (i<p1.nve || j<p2.nve))

```

```
    error = CIERTO;
else
    error = FALSO;

return error;
}

void main()
{
    TparqueMovil  p1, p2;
    int          error;

    resetParqueMovil( &p2 );
do
{
    error = leerParqueMovil( &p1 );
    if (error)
        printf( "Error al leer un parque movil\n" );
    else
    {
        error = unionParques( p1, p2, &p2 );
        if (error)
            printf( "Error al hacer la union de Parques Moviles\n" );
    }
}while (!error);

    escribirParqueMovil( p2 );
}
```