SCTR
Ejercicios de aula

1) En un proceso productivo en un momento dado se sabe que se están procesando como máximo 10 piezas, cada una identificada mediante un código numérico. Se proporciona en una matriz **`incidencias`** las incidencias que se han producido, indicando para cada incidencia el número de pieza. También se indica en el entero **`numIncidencias`** cuántas incidencias se han producido. Elabora un algoritmo que guarde en una matriz los códigos de las piezas y en otra matriz el número de incidencias por pieza.

```
int incidencias[] = {85, 72, 91, 72, 72, 37, 37, 91, 37, 72, 37, 85, 37, 91};
int numIncidencias = 14;
```

Una solución:

```
#include <stdio.h>

int main()
{   int incidencias[] = {85, 72, 91, 72, 72, 37, 37, 91, 37, 72, 37, 85, 37, 91};
    int numIncidencias =14;

    int i,j,k=0;
    int contador=0; //para el numero de incidencias
    int repetido=0; //variable para saber si el numero esta en la matriz
    int piezas[10]={0,0,0,0,0,0,0,0,0,0}; //aqui voy a guardar los codigos de las piezas
    int repeticiones[10]={0,0,0,0,0,0,0,0,0,0};
    //la matriz tiene diez posiciones y se va a guardar aqui el numero de incidencias por pieza

    for(i=0;i<numIncidencias;i++)
    {for(j=0;j<10;j++)
        {if (incidencias[i]==piezas[j])
            {  repetido=1; //el codigo de la incidencia esta repetido
                break; //sale del bucle
            }
            else repetido=0; //el numero no esta guardado en la matriz y hay que añadirlo

        }

        if (repetido==0)
        { piezas[k]=incidencias[i]; //guardo el numero de la pieza en la matriz
            k++; //incremento la posicion para la siguiente vez
        }
    }

    //hago los bucles para contar el numero de incidencias por pieza

    for(j=0;j<k;j++)
    {for(i=0;i<numIncidencias;i++)
            if (incidencias[i]==piezas[j])
                contador++;
        repeticiones[j]=contador;
        contador=0; //reinicio el contador para la siguiente pieza
    }
    return 0;
}
```

Otra solución:

```
int incidencias[] = {85, 72, 91, 72, 72, 37, 37, 91, 37, 72, 37, 85, 37, 91};
int numIncidencias = 14;

int piezas[10];  // Para guardar los identificadores de las piezas
int numPiezas = 0;  // Para contar cuántas piezas se han detectado
int repeticiones[10];  // Para guardar cuántas incidencias se producen por pieza
```

```
for (int i = 0; i < numIncidencias; i++) {  // Recorriendo todas las incidencias ...
    int encontrada = 0;  // Buleano para indicar si se encontró la pieza en la matriz piezas
    int posicion;  // Posición de la matriz donde se encontró la pieza
    for (int j = 0; j < numPiezas && ! encontrada; j++) {  // Recorriendo las piezas registradas ...
        encontrada = incidencias[i] == piezas[j];  // Indica si se encontró la pieza
        posicion = j;  // Recuerda la posición
    }
    if (encontrada)  // Si se encontró ...
        repeticiones[posicion]++;  // Incrementa su número de incidencias
    else {  // Si no ...
        piezas[numPiezas] = incidencias[i];  // Registra una nueva pieza
        repeticiones[numPiezas] = 1;  // Indica que tuvo una incidencia
        numPiezas ++;  // Incrementa el contador de piezas
    }
}
```

En Python:

```
incidencias = [85, 72, 91, 72, 72, 37, 37, 91, 37, 72, 37, 85, 37, 91]
piezas = list(set(incidencias))  # Elimina los valores repetidos
repeticiones = []  # Para guardar el número de incidencias por pieza
for pieza in piezas:  # Para cada pieza ...
    repeticiones.append(incidencias.count(pieza))  # Añade el número de veces que se repite
```

2) Codifica herramientas para el manejo de puntos y vectores en $\mathbb{R}^3$

Se dispondrá de las siguientes operaciones:

- módulo de un vector $\mathbf{v} = (x, y, z)$

$$|\mathbf{v}| = \sqrt{x^2 + y^2 + z^2}$$

- vector unitario en la dirección de un vector $\mathbf{v} = (x, y, z)$

$$\mathbf{u} = \frac{\mathbf{v}}{|\mathbf{v}|}$$

- suma de dos vectores $\mathbf{v}_1 = (x_1, y_1, z_1)$ y $\mathbf{v}_2 = (x_2, y_2, z_2)$

$$\mathbf{v}_1 + \mathbf{v}_2 = (x_1 + x_2, y_1 + y_2, z_1 + z_2)$$

- resta de dos vectores $\mathbf{v}_1 = (x_1, y_1, z_1)$ y $\mathbf{v}_2 = (x_2, y_2, z_2)$

$$\mathbf{v}_1 - \mathbf{v}_2 = (x_1 - x_2, y_1 - y_2, z_1 - z_2)$$

- multiplicación de un vector $\mathbf{v} = (x, y, z)$ por un escalar a

$$a\mathbf{v} = (ax, ay, az)$$

- producto escalar de dos vectores $\mathbf{v}_1 = (x_1, y_1, z_1)$ y $\mathbf{v}_2 = (x_2, y_2, z_2)$

$$\mathbf{v}_1 \cdot \mathbf{v}_2 = x_1x_2 + y_1y_2 + z_1z_2$$

- producto vectorial de dos vectores $\mathbf{v}_1 = (x_1, y_1, z_1)$ y $\mathbf{v}_2 = (x_2, y_2, z_2)$

$$\mathbf{v}_1 \times \mathbf{v}_2 = (y_1z_2 - y_2z_1, x_2z_1 - x_1z_2, x_1y_2 - x_2y_1)$$

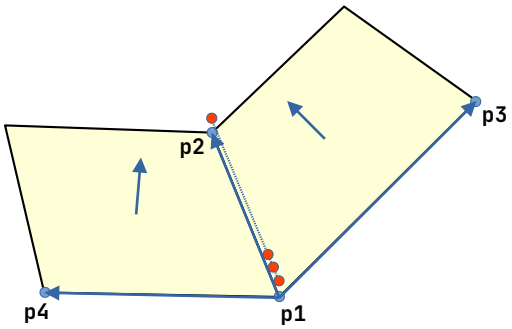
Codifica también un programa para generar la secuencia de 20 posiciones donde tiene que situarse sucesivamente el cabezal de un robot de soldadura para unir dos planchas de hierro. En cada posición el cabezal realiza una soldadura puntual, para lo que tiene que situarse a una distancia de 0.2 unidades de la línea de unión de ambas

planchas y a la misma distancia de ambas. La primera plancha está anclada en los puntos p1, p2 y p3 cuyas coordenadas se indican a continuación. La segunda plancha está anclada en p1, p2 y p4. La línea de soldadura va de p1 a p2, la primera posición de soldadura es p1 y la última p2.

p1 = (5.07, 3.28, 2.16)
p2 = (10.53, 12.19, 17.72)
p3 = (21.01, 15.63, 6.97)
p4 = (-3.15, 4.96, 0.32)

Como resultado se deberían de generar las siguientes posiciones de soldadura:

(5.10, 3.10, 2.25)
(5.39, 3.57, 3.07)
(5.67, 4.04, 3.89)
(5.96, 4.51, 4.71)
(6.25, 4.98, 5.53)
(6.54, 5.45, 6.35)
(6.82, 5.92, 7.16)
(7.11, 6.39, 7.98)
(7.40, 6.86, 8.80)
(7.68, 7.32, 9.62)
(7.97, 7.79, 10.44)
(8.26, 8.26, 11.26)
(8.55, 8.73, 12.08)
(8.83, 9.20, 12.90)
(9.12, 9.67, 13.72)
(9.41, 10.14, 14.54)
(9.70, 10.61, 15.35)
(9.98, 11.08, 16.17)
(10.27, 11.55, 16.99)
(10.56, 12.01, 17.81)



a) Una solución utilizando estructuras:

Archivo **vectores.h**:

```
#ifndef VECTORES_H
#define VECTORES_H

typedef struct {
    float x, y, z;
} Vector;

float modulo(Vector vector);

Vector unitario(Vector vector);

Vector suma(Vector v1, Vector v2);

Vector resta(Vector v1, Vector v2);

Vector multiplica(Vector v1, float a);

float producto(Vector v1, Vector v2);

Vector vectorial(Vector v1, Vector v2);

#endif // VECTORES_H
```

Archivo **vectores.c**:

```
#include <vectores.h>
#include <math.h>
```

```
float modulo(Vector vector) {
    return sqrt(vector.x * vector.x + vector.y * vector.y + vector.z * vector.z);
}

Vector unitario(Vector vector) {
    Vector uni;
    float m = modulo(vector);
    uni.x = vector.x / m;
    uni.y = vector.y / m;
    uni.z = vector.z / m;
    return uni;
}

Vector suma(Vector v1, Vector v2){
    Vector suma;
    suma.x=v1.x+v2.x;
    suma.y=v1.y+v2.y;
    suma.z=v1.z+v2.z;
    return suma;
}

Vector resta(Vector v1, Vector v2){
    Vector resta;
    resta.x=v1.x-v2.x;
    resta.y=v1.y-v2.y;
    resta.z=v1.z-v2.z;
    return resta;
}

Vector multiplica(Vector v1, float a){
    Vector mult;

    mult.x=v1.x*a;
    mult.y=v1.y*a;
    mult.z=v1.z*a;
    return mult;
}

float producto(Vector v1, Vector v2){
    float producto;

    producto=v1.x*v2.x+ v1.y*v2.y +v1.z*v2.z;

    return producto;
}

Vector vectorial(Vector v1, Vector v2){
    Vector vect;

    vect.x=v1.y*v2.z-v1.z*v2.y;
    vect.y=v2.x*v1.z-v1.x*v2.z;
    vect.z=v1.x*v2.y-v2.x*v1.y;
    return vect;
}
```

Archivo `main.c`:

```
#include <stdio.h>
#include <math.h>
#include <vectores.h>


void puntosSoldadura(Vector p1, Vector p2, Vector p3, Vector p4, int nPuntos, float separacion,
    Vector * m) {

    Vector v13 = resta(p3, p1);
    Vector v12 = resta(p2, p1);
    Vector v14 = resta(p4, p1);
```

```
    Vector v123 = vectorial(v13, v12);
    Vector v124 = vectorial(v12, v14);

    Vector u123 = unitario(v123);
    Vector u124 = unitario(v124);

    Vector b = suma(u123, u124);
    Vector bu = unitario(b);

    Vector desplazamiento = multiplica(bu, 0.2);

    float d12 = modulo(v12);
    float d = d12 / 19;
    Vector u12 = unitario(v12);
    Vector vd = multiplica(u12, d);

    m[0] = suma(p1, desplazamiento);
    for(int i = 1; i < nPuntos; i++)
        m[i] = suma(m[i-1], vd);
}


int main() {

    Vector p1 = {5.07, 3.28, 2.16};
    Vector p2 = {10.53, 12.19, 17.72};
    Vector p3 = {21.01, 15.63, 6.97};
    Vector p4 = {-3.15, 4.96, 0.32};

    Vector m[20];
    puntosSoldadura(p1, p2, p3, p4, 20, 0.2, m);

    return 0;
}
```

b) Una solución utilizando matrices:

Archivo `vectores.h`:

```
#ifndef VECTORES_H
#define VECTORES_H

float modulo(float * vector);

void unitario(float * vector, float * resultado);

void suma(float * v1, float * v2, float * resultado);

void resta(float * v1, float * v2, float * resultado);

void multiplica(float * v1, float a, float * resultado);

float producto(float * v1, float * v2);

void vectorial(float * v1, float * v2, float * resultado);

#endif // VECTORES_H
```

Archivo `vectores.c`:

```
#include <vectores.h>
#include <math.h>

float modulo(float * vector) {
    return sqrt(vector[0] * vector[0] + vector[1] * vector[1] + vector[2] * vector[2]);
}
```

```
void unitario(float * vector, float * uni) {
    float m = modulo(vector);
    uni[0] = vector[0] / m;
    uni[1] = vector[1] / m;
    uni[2] = vector[2] / m;
}

void suma(float * v1, float * v2, float * resultado){
    resultado[0] = v1[0] + v2[0];
    resultado[1] = v1[1] + v2[1];
    resultado[2] = v1[2] + v2[2];
}

void resta(float * v1, float * v2, float * resultado){
    resultado[0] = v1[0] - v2[0];
    resultado[1] = v1[1] - v2[1];
    resultado[2] = v1[2] - v2[2];
}

void multiplica(float * v1, float a, float * resultado){
    resultado[0] = v1[0] * a;
    resultado[1] = v1[1] * a;
    resultado[2] = v1[2] * a;
}

float producto(float * v1, float * v2){
    return v1[0] * v2[0] + v1[1] * v2[1] + v1[2] * v2[2];
}

void vectorial(float * v1, float * v2, float * resultado){
    resultado[0] = v1[1] * v2[2] - v1[2] * v2[1];
    resultado[1] = v2[0] * v1[2] - v1[0] * v2[2];
    resultado[2] = v1[0] * v2[1] - v2[0] * v1[1];
}
```

Archivo `main.c`:

```
#include <stdio.h>
#include <math.h>
#include <vectores.h>


void puntosSoldadura(float * p1, float * p2, float * p3, float * p4, int nPuntos, float separacion,
    float * m) {

    float v13[3], v12[3], v14[3], v123[3], v124[3], u123[3], u124[3], b[3], bu[3], desplazamiento[3];

    resta(p3, p1, v13);
    resta(p2, p1, v12);
    resta(p4, p1, v14);

    vectorial(v13, v12, v123);
    vectorial(v12, v14, v124);

    unitario(v123, u123);
    unitario(v124, u124);

    suma(u123, u124, b);
    unitario(b, bu);

    multiplica(bu, 0.2, desplazamiento);

    float d12 = modulo(v12);
    float d = d12 / 19;

    float u12[3], vd[3];
    unitario(v12, u12);
    multiplica(u12, d, vd);
```

```
    suma(p1, desplazamiento, m);
    for(int i = 1; i < nPuntos; i++)
        suma(m + (i-1) * 3, vd, m + i * 3);
}


int main() {

    float p1[] = {5.07, 3.28, 2.16};
    float p2[] = {10.53, 12.19, 17.72};
    float p3[] = {21.01, 15.63, 6.97};
    float p4[] = {-3.15, 4.96, 0.32};

    float m[20 * 3];
    puntosSoldadura(p1, p2, p3, p4, 20, 0.2, m);

    return 0;
}
```

3) En el manual de un módulo GPS se describe el protocolo de comunicación utilizado para transferir datos contínuamente con una cierta cadencia a través de un canal de comunicación serie asíncrono a un sistema embebido que controla el seguimiento de una trayectoria preprogramada en un dron.

El módulo GPS envía paquetes de bytes con el siguiente contenido:

- Bytes 0 y 1: dos bytes con los valores 0x04 y 0x24 como cabecera del paquete.

- Bytes 2 a 5: cuatro bytes para expresar el instante de tiempo (dentro de un día de 24h) en el que se genera el paquete. Ejemplo: los bytes 0x03, 0xDF, 0x15, 0x37 indican el valor 0x03DF1537 = 64951607. Dividiendo este valor por 1000 se obtiene el instante en formato hhmmss.sss. En el ejemplo, 64951607 / 1000 = 64951.607 = 6 horas, 49 minutos y 51.607 segundos.

- Bytes 6 a 9: cuatro bytes para expresar la latitud en grados decimales multiplicada por 1000000. Ejemplo: 0x01, 0x60, 0x74, 0xCC indican 0x016074CC / 1000000 = 23.098572 grados.

- Byte 10: con el valor 0x01 indica latitud N y con el valor 0x02 indica latitud S.

- Bytes 11 a 14: cuatro bytes para expresar la longitud en grados decimales multiplicada por 1000000. Ejemplo: 0x07, 0x2B, 0x64, 0xDF indican 0x072B64DF / 1000000 = 120.284383 grados.

- Byte 15: con el valor 0x01 indica longitud E y con el valor 0x02 indica longitud W.

- Byte 16: con el valor 0x01 indica que los datos del paquete son válidos y con 0x02 que no lo son.

- Bytes 17 a 20: cuatro bytes para expresar la dirección de movimiento con respecto al N. Ejemplo: 0x02, 0xB0, 0xE5, 0x7A indican 0x02B0E57A = 45147514 indica que se está moviendo con un ángulo 45147514 / 1000000 = 45.1474514 grados.

- Bytes 21 a 24: cuatro bytes para expresar la velocidad de movimiento en nudos/s. Ejemplo: 0x00, 0x00, 0xE8, 0x24 indica 0x0000E824 / 1000 = 59.428 nudos/s.

- Byte 25: el valor 0x4E indica que la velocidad se expresó en nudos/s.

- Bytes 26 a 29: cuatro bytes para expresar la velocidad de movimiento en Km/h. Ejemplo: 0x00, 0x01, 0xAD, 0xED indican 0x0001ADED / 1000 = 110.061 Km/h.

- Byte 30: el valor 0x4B indica que la velocidad se expresó en Km/h.

- Byte 31: es el checksum de todos los bytes anteriores salvo los de la cabecera, obtenido como el resultado de una O-exclusiva de todos ellos. En este ejemplo, 0x14.

- Bytes 32 y 33: valores 0x0D y 0x0A que finalizan el paquete.

Una vez recibido por un canal de comunicaciones un paquete, el programa lo almacena en una matriz de bytes como la que se indica a continuación. Codifica una función a la que le pasamos este paquete de bytes y nos indica si el paquete es válido y, en el caso de serlo, devuelve en una estructura la información extraida.

```
uint8_t mensaje[] = { 0x04, 0x24, 0x03, 0xDF, 0x15, 0x37, 0x01, 0x60, 0x74, 0xCC, 0x01, 0x07, 0x2B,
    0x64, 0xDF, 0x01, 0x01, 0x02, 0xB0, 0xE5, 0x7A, 0x00, 0x00, 0xE8, 0x24, 0x4E, 0x00, 0x01, 0xAD,
    0xED, 0x4B, 0x14, 0x0D, 0x0A };
```

Una solución:

```
// http://docs.mirifica.eu/GlobalTop_Technology/features/binary_sentence/binary_protocol.html

#include <stdint.h>  // Para utilizar uint8_t y uint32_t

uint8_t mensaje[] = {
    0x04, 0x24,   // Cabecera de paquete
    0x03, 0xDF, 0x15, 0x37,  // Tiempo
    0x01, 0x60, 0x74, 0xCC,  // Latitud
    0x01,  // E o W
    0x07, 0x2B, 0x64, 0xDF,  // Longitud
    0x01,  // N o S
    0x01,  // Datos válidos
    0x02, 0xB0, 0xE5, 0x7A,  // Dirección movimiento
    0x00, 0x00, 0xE8, 0x24,  // Velocidad
    0x4E,  // En nudos/h
    0x00, 0x01, 0xAD, 0xED,  // Velocidad
    0x4B,  // En Km/h
    0x14,  // Checksum
    0x0D, 0x0A};  // Fin de paquete


uint32_t lee32BitsBigEndian(uint8_t * p) {
    // Devuelve un entero de 32 bits recogido a partir de la dirección de memoria
    // apuntada por p y guardado en formato Big endian
    uint32_t resultado;  // Para montar el entero de 32 bits
    resultado = p[0] << 24;  // Recoge los 8 bits más significativos y los desplaza a su lugar
    resultado |= p[1] << 16; // Los siguientes 8 bits, desplazados a su lugar
    resultado |= p[2] << 8;  // Los 8 siguientes
    resultado |= p[3];       // Y los 8 bits menos significativos
    return resultado;
}


typedef struct {
    int hora;
    int minuto;
    double segundo;
    double latitud, longitud, direccion;
    double velocidadNudos, velocidadKmH;
} DatosGPS;
// Datos extraidos del mensaje enviado por el GPS
```

```c
int extraeDatosGPS(uint8_t * mensaje, DatosGPS * pDatos) {
    // Devuelve un buleano cierto si en el mensaje indicado como primer parámetro hay un
    // paquete válido enviado por un GPS. Si el paquete es válido, extrae los datos y lo
    // guarda en la estructura cuya dirección se pasa como segundo parámetro.

    uint8_t checksum = 0;  // Variable de 8 bits para calcular el checksum

    for (int i = 2; i < 31; i++)

        checksum ^= mensaje[i];  // O-exclusiva de todos los bytes menos cabecera y final

    if (checksum == mensaje[31] && mensaje[16] == 0x01) {
        // Si el checksum es correcto y en el paquete se indica que los datos son válidos ...

        uint32_t uTiempo = lee32BitsBigEndian(mensaje + 2);
        // Entero de 32 bits donde se expresa el tiempo

        double tiempo = uTiempo / 1000.0;
        pDatos->hora = (int)(tiempo / 10000);
        pDatos->minuto = (int)((tiempo - pDatos->hora * 10000) / 100);
        pDatos->segundo = tiempo - pDatos->hora * 10000 - pDatos->minuto * 100;
        // Extrae hora, minuto y segundo

        pDatos->latitud = lee32BitsBigEndian(mensaje + 6) / 1000000.0;
        if (mensaje[10] == 0x02)
            pDatos->latitud = -(pDatos->latitud);
        // Extrae la latitud, positiva si hacia el N y negativa si hacia el S

        pDatos->longitud = lee32BitsBigEndian(mensaje + 11) / 1000000.0;
        if (mensaje[15] == 0x02)
            pDatos->longitud = -(pDatos->longitud);
        // Extrae la longitud, positiva si hacia el E, negativa si hacia el W

        pDatos->direccion = lee32BitsBigEndian(mensaje + 17) / 1000000.0;
        // Ángulo de dirección de movimiento

        pDatos->velocidadNudos = lee32BitsBigEndian(mensaje + 21) / 1000.0;
        pDatos->velocidadKmH = lee32BitsBigEndian(mensaje + 26) / 1000.0;
        // Velocidad de movimiento en nudos/s y en Km/h

        return 1;  // Devuelve un buleano cierto indicando que el mensaje es válido
    } else return 0;  // Devuelve un buleano falso indicando que el mensaje no es válido
}


int main() {

    DatosGPS datos;  // Estructura para recoger los datos recibidos del GPS

    int paqueteValido = extraeDatosGPS(mensaje, & datos);

    return 0;
}
```

4) En un sistema embebido se dispone en un microcontrolador que dispone de cinco puertos de entrada/salida digital, cada uno de 8 señales: puerto A, puerto B, hasta puerto E. Existen SFR en el microcontrolador para configurar cada señal de cada puerto para que actúe como entrada (poniendo un bit a 1 en el SFR) o como salida digital (poniéndolo a 0). Estos SFR se mapean en memoria en direcciones consecutivas para los puertos A, B, etc. en el mismo orden a partir de la dirección 0xF89. También exsiten SFR para realizar las operaciones de entrada/salida de señales, mapeando estos registros en memoria en direcciones consecutivas a partir de 0xF80 para los puertos A, B, etc. y en el mismo orden. Codifica un módulo en lenguaje C con herramientas que permitan:

- manejar todas las líneas digitales numerándolas consecutivamente desde la línea 0 (bit menos significativo del puerto A) hasta la línea 7 (bit más significativo del puerto A), siguiendo con la línea 8 (bit menos significativo del puerto B) hasta la línea 15 (bit más significativo del puerto B), etc.

- configurar varias líneas consecutivas como entrada o como salida

- leer el estado de una línea de entrada, leer el estado de 8 líneas de entrada consecutivas y leer 16 entradas consecutivas.

- modificar una línea de salida, modificar 8 líneas de salida consecutivas y modificar 16 salidas consecutivas.

Una solución:

Fichero **ESDigital.h**:

```
#ifndef ESDIGITAL_H
#define ESDIGITAL_H

#include <stdint.h>

#define ENTRADA 1
#define SALIDA 0

void configuraESDigitales(int primera, int cuantas, int entradaOSalida);
// Configura 'cuantas' señales, desde 'primera', como entradas o salidas según el buleano
// 'entradaOSalida'

int leeEntrada(int linea);
// Devuelve el estado de la señal 'linea' en un buleano

uint8_t lee8Entradas(int primera);
// Devuelve en un byte el estado de las 8 señales que comienzan en la señal 'primera'

uint16_t lee16Entradas(int primera);
// Devuelve en 16 bits el estado de las 16 señales que comienzan en la señal 'primera'.

void escribeSalida(int valor, int linea);
// Modifica la salida digital en la señal 'linea' y la pone al valor 'valor'.

void escribe8Salidas(uint8_t valor, int primera);
// Modifica 8 salidas digitales comenzando en la señal 'primera' y las pone según los
// bits de 'valor'

void escribe16Salidas(uint16_t valor, int primera);
 // Modifica 16 salidas digitales comenzando en la señal 'primera' y las pone según los
 // bits de 'valor.

#endif // ESDIGITAL_H
```

Fichero **ESDigital.c**:

#include <stdint.h>

```
static uint8_t * TRIS = (uint8_t *) 0xF92;
static uint8_t * PORT = (uint8_t *) 0xF80;

void configuraESDigitales(int primera, int cuantas, int entradaOSalida) {
    for (int senal = primera; senal < primera + cuantas; senal++) {
        if (entradaOSalida)
            TRIS[senal / 8] |= (1 << (senal % 8));
        else TRIS[senal / 8] &= ~(1 << (senal % 8));
    }
}

int leeEntrada(int linea) {
    return PORT[linea/8] & (1 << (linea % 8));
}
```

```
uint8_t lee8Entradas(int primera) {
    uint8_t resultado = 0;
    for(int i = 0; i < 8; i++)
        if (leeEntrada(primera + i))
            resultado |= (1 << i);
    return resultado;
}

uint16_t lee16Entradas(int primera) {
    uint16_t resultado = 0;
    for(int i = 0; i < 16; i++)
        if (leeEntrada(primera + i))
            resultado |= (1 << i);
    return resultado;
}

void escribeSalida(int valor, int linea) {
    if (valor)
        PORT[linea/8] |= (1 << (linea % 8));
    else PORT[linea/8] &= ~(1 << (linea % 8));
}

void escribe8Salidas(uint8_t valor, int primera) {
    for(int i = 0; i < 8; i++)
        escribeSalida(valor & (1 << i), primera + i);
}

void escribe16Salidas(uint16_t valor, int primera) {
    for(int i = 0; i < 16; i++)
        escribeSalida(valor & (1 << i), primera + i);
}
```

5) Codifica una clase en C++ para preparar las herramientas de manejo de E/S digitales elaboradas en el anterior ejercicio, de forma que un objeto de esa clase nos permita manejar las señales digitales del microcontrolador.

Una solución:

Archivo **digitales.h**:

```
#ifndef DIGITALES_H
#define DIGITALES_H


#include <cstdint>


class ESDigitales {

    uint8_t * TRIS, * PORT;

public:

    enum Tipo {ENTRADA, SALIDA};

    ESDigitales();
    ~ESDigitales();

    void configuraESDigitales(int primera, int cuantas, Tipo entradaOSalida);
    // Configura 'cuantas' señales, desde 'primera', como entradas o salidas según el buleano
    // 'entradaOSalida'

    int leeEntrada(int linea);
    // Devuelve el estado de la señal 'linea' en un buleano

    uint8_t lee8Entradas(int primera);
    // Devuelve en un byte el estado de las 8 señales que comienzan en la señal 'primera'
```

```
    uint16_t lee16Entradas(int primera);
    // Devuelve en 16 bits el estado de las 16 señales que comienzan en la señal 'primera'.

    void escribeSalida(int valor, int linea);
    // Modifica la salida digital en la señal 'linea' y la pone al valor 'valor'.

    void escribe8Salidas(uint8_t valor, int primera);
    // Modifica 8 salidas digitales comenzando en la señal 'primera' y las pone según los
    // bits de 'valor'

    void escribe16Salidas(uint16_t valor, int primera);
    // Modifica 16 salidas digitales comenzando en la señal 'primera' y las pone según los
    // bits de 'valor.

};

#endif // DIGITALES_H
```

Archivo **digitales.cpp**:

```
#include "digitales.h"

ESDigitales::ESDigitales() {
    TRIS = (uint8_t *) 0xF92;
    PORT = (uint8_t *) 0xF80;
}

ESDigitales::~ESDigitales() {
    configuraESDigitales(0, 40, Tipo::ENTRADA);
}

void ESDigitales::configuraESDigitales(int primera, int cuantas,
                                       ESDigitales::Tipo entradaOSalida) {
    for (int senal = primera; senal < primera + cuantas; senal++) {
        if (entradaOSalida)
            TRIS[senal / 8] |= (1 << (senal % 8));
        else TRIS[senal / 8] &= ~(1 << (senal % 8));
    }
}

int ESDigitales::leeEntrada(int linea) {
    return PORT[linea/8] & (1 << (linea % 8));
}

uint8_t ESDigitales::lee8Entradas(int primera) {
    uint8_t resultado = 0;
    for(int i = 0; i < 8; i++)
        if (leeEntrada(primera + i))
            resultado |= (1 << i);
    return resultado;
}

uint16_t ESDigitales::lee16Entradas(int primera) {
    uint16_t resultado = 0;
    for(int i = 0; i < 16; i++)
        if (leeEntrada(primera + i))
            resultado |= (1 << i);
    return resultado;
}

void ESDigitales::escribeSalida(int valor, int linea) {
    if (valor)
        PORT[linea/8] |= (1 << (linea % 8));
    else PORT[linea/8] &= ~(1 << (linea % 8));
}

void ESDigitales::escribe8Salidas(uint8_t valor, int primera) {
    for(int i = 0; i < 8; i++)
        escribeSalida(valor & (1 << i), primera + i);
}
```

```
void ESDigitales::escribe16Salidas(uint16_t valor, int primera) {
    for(int i = 0; i < 16; i++)
        escribeSalida(valor & (1 << i), primera + i);
}
```

6) Codifica una módulo en lenguaje C++ para manejar un convertidor DA modelo Analog Devices AD5725. Dispone de 4 salidas analógicas. Las señales V_{REFN} y V_{REFP} están cableadas a GND y a +5V, respectivamente, configurando de esta forma un rango de 0V a 5V para todas las salidas analógicas. La señal CLR está conectada a 5V, no se utiliza. La transferencia de información desde el microcontrolador al convertidor DA se realizará según se indica en la figura 5 del manual del convertidor. En el microcontrolador se utilizan puertos de señales digitales manejados con la clase desarrollada en el ejercicio anterior.
Este convertidor AD5725 se conecta al microcontrolador de la siguiente forma:
- el bus de datos de 12 bits D0 a D11 se conecta a las señales digitales 0 a 11 del microcontrolador.
- las señales A0, A1, RW, CS y LDAC se conectan a las señales digitales 12 en adelante en ese orden.
En el módulo a desarrollar tiene que existir al menos un método que permita modificar todas las salidas analógicas y un método que permita modificar una única salida analógica.

Una solución:

Fichero **ad5725.h**:

```
#ifndef AD5725_H
#define AD5725_H

#include "digitales.h"

class AD5725 {
    ESDigitales & digitales;
public:
    AD5725(ESDigitales & es);
    void salidaAnalogica(int salida, float valor);
    void salidasAnalogicas(float valor1, float valor2, float valor3, float valor4);
};

#endif // AD5725_H
```

Fichero **ad5725.cpp**:

```
#include "ad5725.h"
#include "digitales.h"

#define A0 12
#define A1 13
#define RW 14
#define CS 15
#define LDAC 16

AD5725::AD5725(ESDigitales & es): digitales(es) {
    es.configuraESDigitales(0, 17, ESDigitales::Tipo::SALIDA);
    es.escribeSalida(1, RW);
    es.escribeSalida(1, CS);
    es.escribeSalida(1, LDAC);
}

void AD5725::salidaAnalogica(int salida, float valor) {
    digitales.escribeSalida(0, RW);
    digitales.escribeSalida((salida-1) & 1, A0);
    digitales.escribeSalida((salida-1) & 2, A1);
    uint16_t x = valor * 4095 / 5;
    for(int i = 0; i < 12; i++)
        digitales.escribeSalida(x & (1 << i), i);
    digitales.escribeSalida(CS, 0);
    digitales.escribeSalida(CS, 1);
```

```
    digitales.escribeSalida(0, LDAC);
    digitales.escribeSalida(1, LDAC);
    digitales.escribeSalida(1, RW);
}

void AD5725::salidasAnalogicas(float v1, float v2, float v3, float v4) {
    digitales.escribeSalida(0, RW);

    digitales.escribeSalida(0, A0);
    digitales.escribeSalida(0, A1);
    uint16_t valor = v1 * 4095 / 5;
    for(int i = 0; i < 12; i++)
        digitales.escribeSalida(valor & (1 << i), i);
    digitales.escribeSalida(CS, 0);
    digitales.escribeSalida(CS, 1);

    digitales.escribeSalida(1, A0);
    digitales.escribeSalida(0, A1);
    valor = v2 * 4095 / 5;
    for(int i = 0; i < 12; i++)
        digitales.escribeSalida(valor & (1 << i), i);
    digitales.escribeSalida(0, CS);
    digitales.escribeSalida(1, CS);

    digitales.escribeSalida(0, A0);
    digitales.escribeSalida(1, A1);
    valor = v3 * 4095 / 5;
    for(int i = 0; i < 12; i++)
        digitales.escribeSalida(valor & (1 << i), i);
    digitales.escribeSalida(0, CS);
    digitales.escribeSalida(1, CS);

    digitales.escribeSalida(1, A0);
    digitales.escribeSalida(1, A1);
    valor = v4 * 4095 / 5;
    for(int i = 0; i < 12; i++)
        digitales.escribeSalida(valor & (1 << i), i);
    digitales.escribeSalida(0, CS);
    digitales.escribeSalida(1, CS);

    digitales.escribeSalida(0, LDAC);
    digitales.escribeSalida(1, LDAC);

    digitales.escribeSalida(1, RW);
}
```

7) Prepara una clase para manejar un convertidor A/D externo MAXIM197 en un sistema embebido gobernardo por un microcontrolador para el cual disponemos de la clase definida en un ejercicio anterior para manejo de E/S digital. Las señales del convertidor se conectan a señales de E/S digital del microcontrolador.

La línea $\overline{\text{CS}}$ está cableada a masa y $\overline{\text{SHDN}}$ a +5V. Las líneas D0 a D7 del convertidor están conectadas a líneas de E/S digital consecutivas en el microcontrolador.

Hay que operar el convertidor utilizando su reloj interno y en *Internal Acquisition Mode*.

En la clase de manejo del convertidor A/D se utilizará este enumerado para indicar cómo se va a configurar una entrada analógica:

```
    enum RANGO {
        DESACTIVADA = 0,   // Entrada analógica no configurada
        BIPOLAR_10  = 1,   // Entrada con rango de -10V a 10V
        BIPOLAR_5   = 2,   // Entrada con rango de -5V a 5V
        UNIPOLAR_10 = 3,   // Entrada con rango de 0V a 10V
        UNIPOLAR_5  = 4    // Entrada con rango de 0V a 5V
    };
```

La clase dispondrá de los siguientes métodos públicos:

```
    MAX197(int WR, int RD, int HBEN, int INT, int D0, ESDigital& pESDigital);
    // Constructor. Se le indica el número de señal digital del microcontrolador que está conectada
    // a las señales WR, RD, HB, INT y D0. Se le pasa una referencia a un objeto de la clase ESDigital
    // para manejar las entradas/salidas digitales del microcontrolador.

    void configura(int linea, RANGO rango);
    // Configura la entrada analógica indicada en el primer parámetro para que trabaje en
    // el rango indicado en el segundo parámetro.

    void mide(float* pMedidas);
    // Realiza una medida en todas las entradas analógicas configuradas previamente y los
    // valores obtenidos se guardan consecutivamente a partir de la dirección indicada en pMedidas

    float mide(int linea);
    // Devuelve el resultado de medir la entrada analógica indicada por parámetro
```

Una solución:

Archivo **max197.h**:

```
#ifndef MAX197_H
#define MAX197_H

#include "digitales.h"

class MAX197 {

public:
    enum RANGO {DESACTIVADA = 4, BIPOLAR_10 = 3, BIPOLAR_5 = 2, UNIPOLAR_10 = 1, UNIPOLAR_5 = 0};

private:
    RANGO rangos[8];
    int wr, rd, Int, hben, d0;
    ESDigitales & esdigitales;

public:
    MAX197(int WR, int RD, int HBEN, int INT, int D0, ESDigitales& pESDigital);
    void configura(int linea, RANGO rango);
    void mide(float* pMedidas);
    float mide(int linea);
};

#endif // MAX197_H
```

Archivo **max197.cpp**:

```
#include "max197.h"

MAX197::MAX197(int WR, int RD, int HBEN, int INT, int D0,
    ESDigitales& pESDigital): esdigitales(pESDigital) {
    esdigitales.configuraESDigitales(WR, 1, ESDigitales::Tipo::SALIDA);
    pESDigital.configuraESDigitales(RD, 1, ESDigitales::Tipo::SALIDA);
    pESDigital.configuraESDigitales(HBEN, 1, ESDigitales::Tipo::SALIDA);
    pESDigital.configuraESDigitales(INT, 1, ESDigitales::Tipo::ENTRADA);
    pESDigital.configuraESDigitales(D0, 8, ESDigitales::Tipo::SALIDA);
    wr = WR;
    rd = RD;
    Int = INT;
    hben = HBEN;
    d0 = D0;
    pESDigital.escribeSalida(1, wr);
    pESDigital.escribeSalida(1, rd);
    for (int i = 0; i < 8; i++)
        rangos[i] = RANGO::DESACTIVADA;
}
```

```
void MAX197::configura(int linea, RANGO rango) {
    rangos[linea] = rango;
}


float MAX197::mide(int linea) {
    uint8_t byteControl = 0x40 | linea;
    if (rangos[linea] == RANGO::UNIPOLAR_10)
        byteControl |= 0x10;
    else if (rangos[linea] == RANGO::BIPOLAR_5)
        byteControl |= 0x08;
    else if (rangos[linea] == RANGO::BIPOLAR_10)
        byteControl |= 0x18;
    esdigitales.escribe8Salidas(byteControl, d0);
    esdigitales.escribeSalida(0, wr);
    esdigitales.escribeSalida(1, wr);
    while (esdigitales.leeEntrada(Int)) ;
    esdigitales.escribeSalida(0, rd);
    esdigitales.escribeSalida(0, hben);
    uint16_t dato = esdigitales.lee8Entradas(d0);
    esdigitales.escribeSalida(1, hben);
    dato |= ((esdigitales.lee8Entradas(d0) & 0x0F) << 8);
    esdigitales.escribeSalida(1, rd);
    float minimo, maximo;
    if (rangos[linea] == RANGO::UNIPOLAR_5) {
        minimo = 0.0;
        maximo = 5.0;
    } else if (rangos[linea] == RANGO::UNIPOLAR_10) {
        minimo = 0.0;
        maximo = 10.0;
    } else if (rangos[linea] == RANGO::BIPOLAR_5) {
        minimo = -5.0;
        maximo = 5.0;
    } else if (rangos[linea] == RANGO::BIPOLAR_10) {
        minimo = -10.0;
        maximo = 10.0;
    } else return 0.0;
    float tension = (maximo - minimo) * dato / 4095 + minimo;
    return tension;
}


void MAX197::mide(float * pMedidas) {
    int j = 0;
    for (int i = 0; i < 8; i++)
        if (rangos[i] != RANGO::DESACTIVADA)
            pMedidas[j++] = mide(i);
}
```

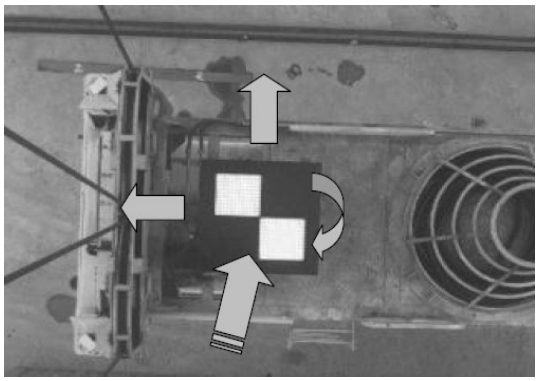
8) En un puente grúa existe un sistema embebido que dispone de una cámara en el carro superior apuntando hacia abajo para tomar imágenes de dos elementos reflectores cuadrados dispuestos en la plataforma de las garras tal y como se muestra en la figura. La cámara tiene una resolución de 1280x1024 puntos. Para determinar el movimiento de balanceo horizontal y vertical, así como el giro de la carga cuando es alzada hasta una altura determinada, es necesario procesar las imágenes capturadas con la cámara, donde la longitud del lado de cada cuadrado es de 0.2 m y corresponde a unos 50 puntos de la imagen en dicha posición. Cuando la carga no se está balanceando y no está girada, los cuadrados reflectores aparecen centrados en la imagen y sus lados paralelos a los bordes de la imagen.

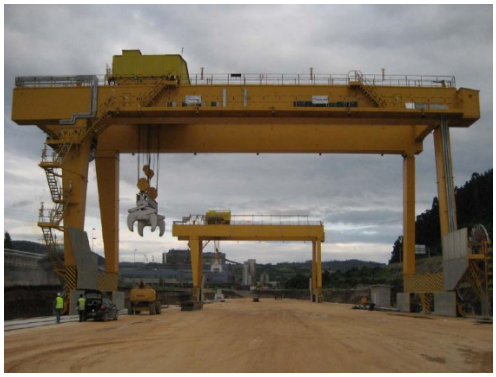
Se dispone de la función `void capturaImagen(uint8_t* p)` que acciona la cámara, espera a que tome una imagen y a que se almacene en la dirección indicada por parámetro. Cada punto de la imagen se guarda en un byte en blanco y negro expresando su nivel de gris (desde 0=negro hasta 255=blanco), recorriendo la imagen por filas, desde la esquina inferior izquierda a la superior derecha. Los cuadrados reflectores son los únicos objetos que aparecen en blanco en las imágenes.

Codifica una aplicación para este sistema embebido que tiene que estar procesando continuamente las imágenes para determinar la la desviación de la carga en m. (como máximo, 1 m.) adelante-atrás e izquierda-derecha, así como el ángulo de giro de la carga (como máximo, ± 20 grados).

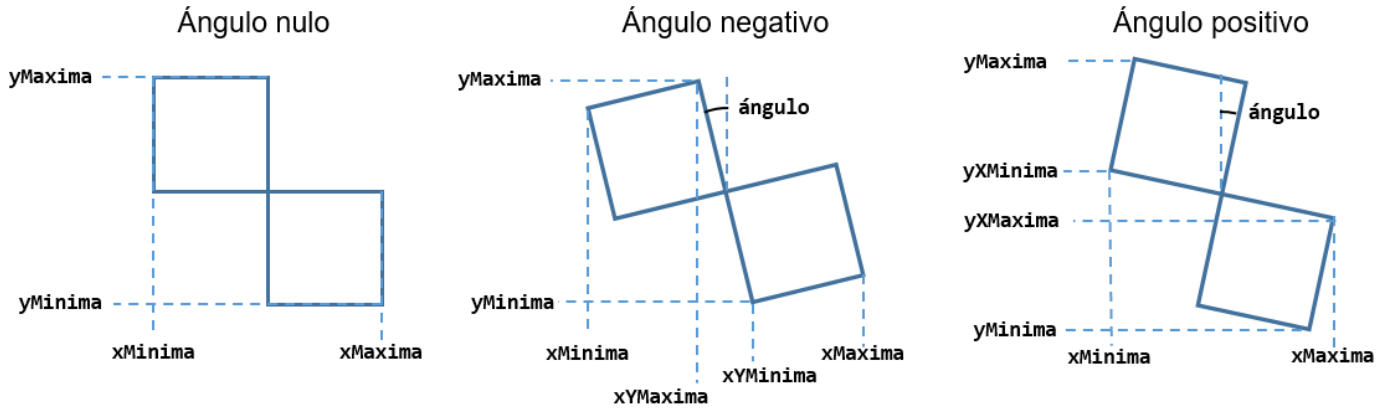
Existe un canal de comunicaciones en el que nuestro sistema embebido actúa como esclavo. Cuando se recibe un paquete con un único byte con valor 1, hay que contestar inmediatamente con un paquete de bytes donde se envían los desplazamientos y el ángulo de la carga en tres `float` obtenidos en el último procesado de imagen realizado.

Existe la clase `Canal` para manejar las comunicaciones, que dispone de un constructor sin parámetros, un método `void envia(void* p, int n)` que transmite un paquete de `n` bytes guardados a partir de `p`, y donde hay un método `int recibe(void* p)` que espera a que se reciba un paquete de bytes guardándolo a partir de `p` y devolviendo su longitud en bytes.





Una solución:




```
typedef struct {
    float dx, dy, angulo;
} DatosGrua;


DatosGrua d;
semaforo s = 1;


void hCamara() {
    uint8_t imagen[1280*1024];
    int xmin = 1280, xmax = 0, ymin = 1024, ymax = 0;
    int xymax, xymin, yxmin, yxmax;
    float xc, yc;

    while(1) {
        capturaImagen(imagen);
```

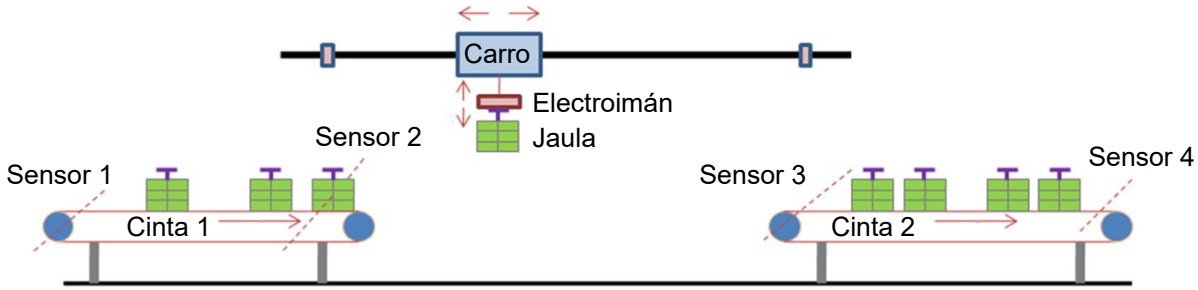
```
        for (int y = 0; y < 1024; y ++)
            for (int x = 0; x < 1280; x++) {
                if (imagen[y * 1280 + x] == 255) {
                    if (x > xmax) {
                        xmax = x;
                        yxmax = y;
                    }
                    if (x < xmin) {
                        xmin = x;
                        yxmin = y;
                    }
                    if (y > ymax) {
                        ymax = y;
                        xymax = x;
                    }
                    if (y < ymin) {
                        ymin = y;
                        xymin = x;
                    }
                }
            }
        xc = (xmax + xmin) / 2;
        yc = (ymax + ymin) / 2;
        wait(s);
        d.dx = 1280/2 - xc;
        d.dy = 1024/2 - yc;
        if (xmax - xmin > ymax - ymin)
            d.angulo = - atan2(xymax - xymin, ymax - ymin);
        else
            d.angulo = atan2(yxmax - yxmin, xmax - xmin);
        signal(s);
    }
}


void hComunicaciones() {
    Canal c;
    uint8_t byteRecibido;
    while(1) {
        c.recibe(& byteRecibido);
        if (byteRecibido == 1) {
            wait(s);
            c.envia(& d, sizeof(d));
            signal(s);
        }
    }
}


int main() {
    concurrente {
        hCamara();
        hComunicaciones();
    }
    return 0;
}
```

9) En una instalación de transporte de jaulas con alimentos se dispone de dos cintas transportadoras 1 y 2 que se pueden poner en marcha (movimiento hacia la derecha) o detener mediante la función **`void cinta(int numCinta, int activar)`**. En la entrada y salida de las cintas hay sensores fotoeléctricos 1, 2, 3 y 4 que se pueden consultar mediante la función **`int sensor(int numSensor)`**, que devuelve un buleano cierto en el caso de que se detecte una jaula. En la entrada de la cinta 1 existe un operario que carga una nueva jaula siempre y cuando haya alguna disponible y la zona de entrada de la esa cinta esté libre. En la salida de la cinta 2 existe otro operario que, cuando puede y cuando hay una jaula al final de la cinta, recoge la jaula y la lleva a otro lugar.

Entre ambas cintas existe un carro que podemos mover horizontalmente a lo largo de una guía. Durante la ejecución de la función **`void izquierda()`** el carro se mueve hacia la izquierda hasta situarse sobre el sensor 2. Durante la ejecución de la función **`void derecha()`** el carro se mueve hacia la derecha hasta situarse sobre el sensor 3. Durante la ejecución de la función **`void baja()`** se desenrolla un cable en el carro para bajar un electroimán hasta situarlo en contacto con una jaula en una cinta. Durante la ejecución de **`void sube()`** el cable se enrolla para subir el electroimán a una altura segura para poder mover el carro horizontalmente a continuación. El electroimán se puede activar o desactivar con la función **`void electroiman(int activacion)`**. Tiene que estar activado mientras el carro transporta una jaula.

Codifica un programa de forma que las jaulas pasen desde el operario de entrada al de salida. En cada cinta transportadora puede haber varias jaulas, pero nunca unas pegadas a otras. El carro sólo puede transportar una jaula de cada vez y puede depositarla en la entrada de la cinta 2 sólo cuando esa posición está libre.

Controla esta instalación de forma que las cintas estén en movimiento (simultáneamente o no) sólo cuando sea necesario. Las esperas hay que codificarlas utilizando semáforos.

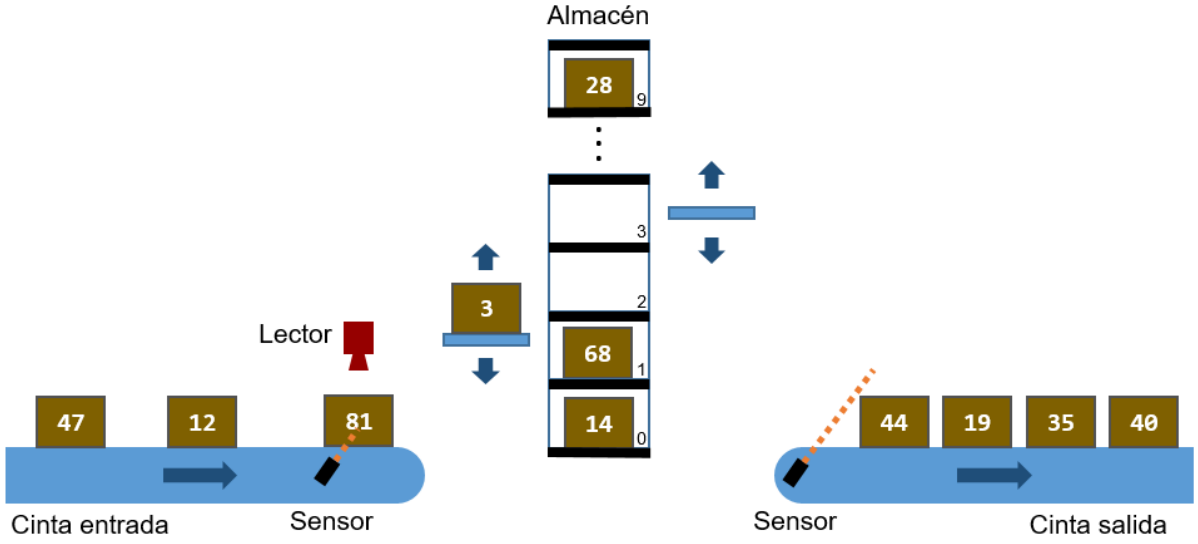


Una solución:

```
semaforo sCarroEntrada=0,sCarroSalida=1;

void hCintaEntrada() {
    cinta(1, 0);
    int numJaulas = 0;
    int sensor1Anterior = 0, sensor2Anterior = 0;
    while(1) {
        if(!sensor1Anterior && sensor(1) )
            numJaulas++;
        sensor1Anterior=sensor(1);
        if (numJaulas>0 && !sensor(2))
            cinta(1, 1);
        if (sensor(2))
            cinta(1, 0);
        if (sensor2Anterior && !sensor(2))
            numJaulas--;
        if(!sensor2Anterior && sensor(2))
            signal(sCarroEntrada);
        sensor2Anterior=sensor(2);
    }
}

void hCarro() {
    electroiman(0);
    sube();
    izquierda();
```

```
    while(1) {
        wait(sCarroEntrada);
        baja();
        electroiman(1);
        sube();
        derecha();
        wait(sCarroSalida);
        baja();
        electroiman(0);
        sube();
        izquierda();
    }
}

void hCintaSalida() {
    int sensor3Anterior = 0, sensor4Anterior = 0;
    int numJaulas = 0;
    cinta(2, 0);
    while(1) {
        if(!sensor3Anterior && sensor(3) )
            numJaulas++;
        if (sensor3Anterior && !sensor(3))
            signal(sCarroSalida);
        if (sensor4Anterior && !sensor(4))
            numJaulas--;
        sensor3Anterior=sensor(3);
        sensor4Anterior=sensor(4);
        if (numJaulas>0 && !sensor(4))
            cinta(2, 1);
        if (sensor(4))
            cinta(2, 0);
    }
}

int main() {
    concurrente {
        hCintaEntrada();
        hCarro();
        hCintaSalida();
    }
}
```

10) En una planta industrial existe una célula de transporte y ordenación de productos compuesta por los siguientes elementos:



a) Una cinta transportadora de entrada en la que se depositan paquetes. Al final de la cinta disponemos de un sensor fotoeléctrico para detectar la llegada de un paquete a esa posición. Una llamada a la función **`int sensorEntrada()`** devuelve un buleano cierto si detecta un paquete. La cinta la podemos poner en funcionamiento o detener si llamamos a **`void cintaEntrada(int)`**, pasándole un buleano cierto o falso. La cinta tiene que estar en funcionamiento mientras no llegue un paquete al final.

b) Una llamada a la función **`uint32_t leeCodigo()`** maneja el lector de códigos de barras que hay al final de la cinta de entrada para leer el número de serie del paquete (entero de 32 bits sin signo) situado en esa posición.

c) Un elevador de entrada que permite recoger un paquete del final de la cinta y colocarlo en cualquiera de los diez estantes de un almacén vertical. Cada estante sólo puede contener un paquete. Cuando se ejecuta una llamada a **`void elevadorEntrada(int)`** el elevador de entrada se mueve hasta la posición indicada por parámetro (0 a 9), la cinta de entrada está a la altura de la posición 0. Con una llamada a **`void cogeEntrada()`** el elevador de entrada recoge un paquete a su izquierda en la cinta y con una llamada a **`void dejaEntrada()`** deja el paquete a su derecha en una estantería.

d) Al otro lado del almacén existe un elevador de salida que se utiliza para recoger del almacén el paquete con el número de serie más bajo y depositarlo en la cinta de salida. Mientras se ejecuta una llamada a **`void elevadorSalida(int)`** el elevador de salida se mueve hasta la posición indicada por parámetro (0 a 9). La cinta de salida está a la altura de la posición 0. Con una llamada a **`void cogeSalida()`** el elevador de salida recoge un paquete a su izquierda en una estantería y con una llamada a **`void dejaSalida()`** deja el paquete a su derecha en la cinta.

e) Una cinta transportadora de salida que se pone en movimiento o se detiene según el buleano pasado a **`void cintaSalida(int)`**. Un sensor fotoeléctrico que detecta la presencia en la entrada de esa cinta, que se puede consultar con **`int sensorSalida()`**, devolviendo un buleano cierto si se detecta paquete. La cinta de salida tiene que estar en movimiento mientras se detecta un paquete en esa posición.

El elevador de salida elige en el pulmón el paquete con el código más pequeño y lo transporta al comienzo de la cinta de salida.