Lenguaje C

Algoritmos en tablas

ej1) En una matriz `m` se dispone de 10 números enteros:

```
int m[] = { 5, 3, 2, 6, 3, 4, 1, 6, 4, 8 };
```

Codifica un algoritmo en C que calcule cuál es el valor mínimo.

Una solución:

```
int m[] = { 5, 3, 2, 6, 3, 4, 1, 6, 4, 8 };

int main() {
    int i;
    int minimo = m[0];
    for (i = 1; i < 10; i++)
        if (m[i] < minimo)
            minimo = m[i];
    return 0;
}
```

---

ej2) En una matriz se dispone de varios valores enteros:

```
int m[] = { 15, 13, 2, -6, 17, 4, 1, 6, -9, 8 };  // Matriz con los valores
int nElementos = 10;  // Número de elementos en la matriz
```

Codifica un algoritmo en lenguaje C que determine en variables

```
int posMinimo, posMaximo;  // Posición de mínimo y máximo en la matriz
```

en qué posición está el valor mínimo y en qué posición está el valor máximo. En este ejemplo el mínimo está en la posición 8 y el máximo en la 4.

Una solución:

```
int main() {
    int m[] = { 15, 13, 2, -6, 17, 4, 1, 6, -9, 8 };  // Matriz con los valores
    int nElementos = 10;  // Número de elementos en la matriz
    int iMinimo = 0, iMaximo = 0;  // Candidatos iniciales para posición de mínimo y máximo

    for(int i = 1; i < nElementos; i++) {  // Desde el segundo elemento al último ...
        if (m[i] < m[iMinimo])  // Si se encuentra otro menor
            iMinimo = i;        // se actualiza la posición del mínimo
        if (m[i] > m[iMaximo])  // Si se encuentra otro mayor
            iMaximo = i;        // Se actualiza la posición del máximo
    }
    return 0;
}
```

---

ej3) En una matriz se dispone de varios valores enteros:

```
int m[] = { 15, 13, 2, -6, 17, 4, 1, 6, -9, 8 };  // Matriz con los valores
int nElementos = 10;  // Número de elementos en la matriz
```

Codifica un algoritmo en lenguaje C que los ordene de menor a mayor.
Para ello se puede determinar la posición donde se encuentra el valor mínimo de todos los valores, desde la posición 0 al final. Una vez encontrada la posición del valor mínimo, se intercambia el valor mínimo con el de la posición 0. Luego se hace lo mismo, pero analizando desde la posición 1 al final. Y así sucesivamente.

Una solución:

```c
int main() {
    int m[] = { 15, 13, 2, -6, 17, 4, 1, 6, -9, 8 }; // Matriz con los valores
    int nElementos = 10; // Número de elementos en la matriz
    int posMinimo, i, j, temp;

    for (i = 0; i < nElementos-1; i++) {
        posMinimo = i;
        for (j = i; j < nElementos; j++) if (m[j] < m[posMinimo]) posMinimo = j;
        temp = m[i]; m[i] = m[posMinimo]; m[posMinimo] = temp;
    }

    return 0;
}
```

---

ej4) En una matriz `m1` se dispone de 10 números enteros:

```c
int m[] = { 5, 3, 2, 6, 3, 4, 1, 6, 4, 8 };
```

Codifica en C un algoritmo que ordene los valores de menor a mayor utilizando el método de la burbuja. Se hacen pasadas sucesivas por la matriz y se intercambian los valores de dos posiciones consecutivas cuando están desordenados. El algoritmo finaliza cuando no se encuentran valores consecutivos desordenados.

Una solución:

```c
int m[] = { 5, 3, 2, 6, 3, 4, 1, 6, 4, 8 };

int main() {
    int i, modificado, auxiliar;
    do {
        modificado = 0;
        for (i = 0; i < 9; i++)
            if (m[i] > m[i + 1]) {
                auxiliar = m[i];
                m[i] = m[i + 1];
                m[i + 1] = auxiliar;
                modificado = 1;
            }
    } while (modificado);
    return 0;
}
```

---

ej32) En una factoría de fabricación de productos farmacéuticos hay 10 máquinas idénticas, desde la máquina 1 hasta la máquina 10. Para ejecutar la próxima orden de producción en una máquina determinada, se elige la máquina libre que menos órdenes ha ejecutado previamente. Codifica un algoritmo en lenguaje C que determine la máquina que tiene que ejecutar la siguiente orden, teniendo en cuenta la información indicada en las siguientes variables:

```c
int ordenes[] = { 5, 3, 2, 6, 3, 4, 1, 6, 4, 8 };  // Número de órdenes ejecutadas por cada máquina
int libres[] =   { 1, 1, 0, 0, 0, 1, 0, 0, 1, 1 };   // Buleanos que indican qué máquinas están libres
int nMaquinas = 10;  // Número de máquinas existentes
```

Una solución:

```c
#include <stdio.h>

int main()
{
    int ordenes[] = { 5, 3, 2, 6, 3, 4, 1, 6, 4, 8 };  // Número de órdenes ejecutadas por cada máquina
    int libres[] =  { 1, 1, 0, 1, 0, 1, 0, 0, 1, 0 };   // Buleanos que indican máquinas libres
    int nMaquinas = 10;  // Número de máquinas existentes
    int i;
    int minimo;
    int seleccionada;

    seleccionada = 0;
    minimo = 100000;
    for(i = 0; i < nMaquinas; i++)
        if (ordenes[i] < minimo && libres[i]) {
            minimo = ordenes[i];
            seleccionada = i + 1;
        }

    if (seleccionada)
        printf("Máquina seleccionada: %d\n", seleccionada);
    else printf("Todas las máquinas están ocupadas\n");

    return 0;
}
```

Otra solución:

```c
#include <stdio.h>

int ordenes[] = { 5, 3, 2, 6, 3, 4, 1, 6, 4, 8 };  // Número de órdenes ejecutadas por cada máquina
int libres[] =   { 1, 1, 0, 0, 0, 1, 0, 0, 1, 1 };   // Buleanos que indican qué máquinas están libres
int nMaquinas = 10;  // Número de máquinas existentes


int main() {

    int maquina = 0;
    int encontrada = 0;
    int i;

    // Busca la primera máquina lilbre
    while (maquina < nMaquinas && !encontrada) {
        encontrada = libres[maquina];
        if (!encontrada)
            maquina++;
    }

    if (encontrada) {  // Si la encontró ...
        if (maquina < nMaquinas - 1)  // Si no es la última ...
            for (i = maquina + 1; i < nMaquinas; i++)  // Busca otra libre que tenga menos trabajos
                if (libres[i] && ordenes[i] < ordenes[maquina])
                    maquina = i;
        return printf("Máquina seleccionada: %d\n", maquina + 1);
    }
    else printf("Todas las máquinas están ocupadas\n");

    return 0;
}
```

ej29) Codifica en lenguaje C un módulo **`bloquememoria.c`** con su correspondiente archivo de declaraciones **`bloquememoria.h`** donde se definan herramientas para la reserva dinámica de bloques de memoria en un microcontrolador con memoria RAM de tamaño reducido, de manera que se puede utilizar como se indica en este ejemplo:

```
#include "bloquememoria.h"

uint8_t m[65536];
// Variable donde se van a realizar las operaciones


int main() {

    TBloqueMemoria zona;  // Representa a una zona de memoria donde se van a realizar reservas

    inicializaBloqueMemoria(m, 65536, &zona);
    // Crea un objeto para manejar una zona de memoria en la variable 'm'
    // con un tamaño de 65536 bytes

    double* pd = (double*) reservaBloqueMemoria(100 * sizeof(double), & zona);
    // Reserva un bloque de memoria para almacenar 100 reales de 64 bits. Devuelve
    // la dirección donde se almacena el primer número real.

    pd[0] = 3.25;  // Guarda un dato en el bloque

    uint16_t* p16 = (uint16_t*) reservaBloqueMemoria(5 * sizeof(uint16_t), & zona);
    // Reserva un bloque de memoria para almacenar 5 enteros de 16 bits sin signo.
    // Devuelve la dirección del primer entero.

    p16[0] = 14;  // Guarda un dato en el bloque

    if (errorReservandoBloqueMemoria(& zona))  // Si hubo algún error durante las reservas,
        while(1);                 // bloquea la ejecución del programa

    liberaBloquesMemoria(& zona);
    // Libera todos los bloques de memoria reservados

    uint32_t* p32 = (uint32_t*) reservaBloqueMemoria(30 * sizeof(uint32_t), & zona);
    // Reserva un bloque de memoria para almacenar 30 enteros de 32 bits sin signo.
    // Devuelve la dirección del primero de ellos.

    p32[0] = 25;  // Guarda un dato en el bloque

    return 0;
}
```

Una solución:

Fichero **`bloquememoria.h`**:

```
#ifndef BLOQUEMEMORIA_H
#define BLOQUEMEMORIA_H

#include <stdint.h>

typedef struct  {  // Bloque de memoria donde realizar reservas
    uint8_t * pComienzo;  // Puntero al comienzo de la zona de memoria
    uint32_t capacidad;   // Tamaño de la zona de memoria
    uint32_t reservados;  // Número de bytes reservados
    int error;  // Cierto si hubo algún error en alguna reserva
} TBloqueMemoria;


void inicializaBloqueMemoria(void* m, uint32_t tamano, TBloqueMemoria* bm);
    // En el primer parámetro se le indica dónde comienza
    // la zona de memoria y en el segundo se le indica su tamaño en bytes.
```

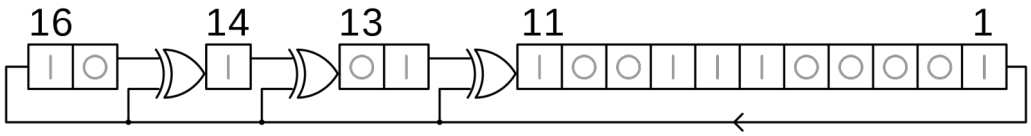
```
void * reservaBloqueMemoria(uint32_t tamano, TBloqueMemoria* bm);
    // Reserva un bloque de 'tamano' bytes y devuelve su dirección de comienzo.
    // Si no se pudo hacer la reserva, devuelve NULL.

void liberaBloquesMemoria(TBloqueMemoria* bm);  // Libera todos los bloques de memoria reservados.

int errorReservandoBloqueMemoria(TBloqueMemoria* bm);
    // Devuelve un valor cierto si hubo algún problema en alguna reserva

#endif // BLOQUEMEMORIA_H
```

Fichero `bloquememoria.c`:

```
#include "bloquememoria.h"
#include <stdlib.h>


void inicializaTBloqueMemoria(void* m, uint32_t tamano, TBloqueMemoria* bm) {
    // En el primer parámetro se le indica dónde comienza
    // la zona de memoria y en el segundo se le indica su tamaño en bytes.

    bm->capacidad = tamano;  // Guarda la capacidad de la zona
    bm->reservados = 0;  // Inicialmente no hay ningún byte reservado
    bm->pComienzo = (uint8_t *) m;  // Apunta al comienzo de la zona de memoria.
    bm->error = 0;  // Inicialmente no hay ningún error por reservas.
}


void * reservaBloqueMemoria(uint32_t tamano, TBloqueMemoria* bm) {
    // Reserva un bloque de 'tamano' bytes y devuelve su dirección de comienzo.
    // Si no se pudo hacer la reserva, devuelve nullptr.

    if (bm->error)  // Si hubo algún error en una reserva anterior
        return NULL;  // devuelve NULL
    else {
        bm->error = bm->reservados + tamano > bm->capacidad;  // Comprueba si hay bytes suficientes.
        if (bm->error)  // Si no se pudo realizar la reserva
            return NULL;  // devuelve nullptr
        else {
            bm->reservados += tamano;  // Reserva los bytes solicitados.
            return bm->pComienzo + bm->reservados - tamano;  // Devuelve direcciónde1 bloque reservado.
        }
    }
}


void liberaBloquesMemoria(TBloqueMemoria* bm) {  // Libera todos los bloques de memoria reservados.
    bm->reservados = 0;  // Indica que no hay ningún byte reservado
    bm->error = 0;  // Inicialmente no hay error de reserva
}


int errorReservandoBloqueMemoria(TBloqueMemoria* bm) {
    // Devuelve un valor cierto si hubo algún problema en alguna reserva

    return bm->error;
}
```

Operaciones con bits

ej5) En una ciudad la alcaldía contrata a una ingeniería para la mejora de la secuencia pseudoaleatoria de encendido y apagado de 16 bombillas instaladas en un dinoseto para la próxima campaña de Navidad de forma que su funcionamiento corresponda al siguiente sistema digital compuesto por puertas XOR y biestables D situados en cascada. El estado inicial de los biestables se indica en la figura. El sistema realiza a cada segundo las operaciones XOR indicadas y un desplazamiento.



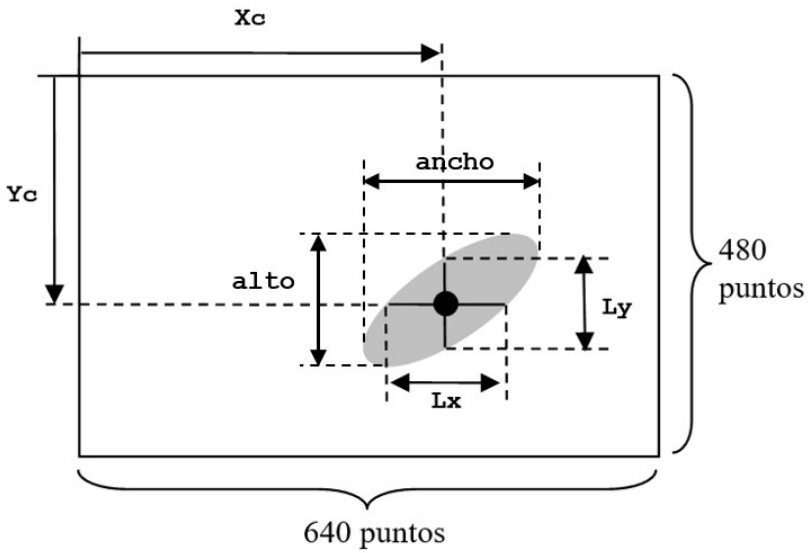
Las bombillas se manejan con las salidas digitales de dos puertos de 8 bits del microcontrolador, que se pueden manejar utilizando los identificadores **`PORTA`** (para los 8 biestables más significativos) y **`PORTB`** (para los 8 menos significativos). Existe una variable predefinida **`MS`** de tipo **`unsigned long`** que se incrementa automáticamente mediante una interrupción de un temporizador a cada milisegundo desde el comienzo de la ejecución del programa.

Una solución:

```
#include <stdint.h>  // Para poder uitlizar uint16_t y uint8_t
unsigned long MS;
uint8_t PORTA, PORTB;

int main() {
    uint16_t biestables = 0xACE1;  // Para representar a los 16 biestables
    while (1) {  // Repite contínuamente ...
        while (MS < 1000);  // Espera 1 s.
        MS = 0;  // Resetea la variable que cuenta tiempo transcurrido
        if (biestables & 0x0001) {
            biestables ^= 0x6800; // XOR con 0110 1000 0000 0000
            biestables >>= 1;  // Desplazamiento una posición a la derecha
            biestables |= 0x8000;  // En el bit más significativo entra un 1
        }
        else biestables >>= 1;  // Desplazamiento una posición a la derecha
        PORTA = biestables >> 8;  // Los 8 bits más significativos al puerto PORTA
        PORTB = biestables & 0x00FF;  // Los 8 bits menos significativos al puerto PORTB
    }
}
```

Visión artificial

ej6) Un sistema embebido está conectado a una cámara de vídeo de 640x480 puntos de resolución. Cada punto de una imagen se representa en niveles de gris mediante un byte sin signo (desde 0=negro hasta 255=blanco). Se supone que la imagen ya se ha capturado y todos los bytes están disponibles en una matriz **`uint8_t m[307200]`** creada en el archivo **`imagen.h`** suministrado.
Todos los bytes se organizan en esta matriz por filas de puntos en la imagen, desde la fila superior a la inferior y dentro de cada fila, de izquierda a derecha. Codifica un algoritmo que calcule una serie de características del único objeto (de color oscuro, con valor menor que 100) existente en la escena (el fondo es de color claro, mayor que 200).
Hay que obtener las coordenadas **`(Xc, Yc)`** del su centro para determinar su posición, así como su dimensión **`Lx`** en el eje horizontal y **`Ly`** en el vertical, ejes que pasan por dicho centro. También su ancho y alto y también su superficie.
Para la imagen suministrada, el resultado tiene que ser: ancho=303, alto=191, superficie=39115, Lx=221, Ly=53, Xc=444.39, Yc=282.75.



Una solución:

```
#include <stdint.h>
#include "imagen.h"

int main() {
    uint32_t fila, columna;  // Para barrer filas y columnas en la imagen
    uint32_t filaMin, filaMax, columnaMin, columnaMax;  // Región ocupada por el objeto
    float Xc = 0, Yc = 0;  // Coordenadas del centro de gravedad del objeto
    uint32_t area = 0;  // Area del objeto medida en puntos
    int primeroDetectado = 0;  // Buleano: se detectó el primer punto
    uint32_t ancho, alto, Lx, Ly;

    for (fila = 0; fila < 480; fila ++)
        for (columna = 0; columna < 640; columna++) {
            if (m[fila * 640 + columna] < 100) {
                area++;
                Xc += columna;
                Yc += fila;
                if (primeroDetectado) {
                    if (filaMin > fila) filaMin = fila;
                    if (filaMax < fila) filaMax = fila;
                    if (columnaMin > columna) columnaMin = columna;
                    if (columnaMax < columna) columnaMax = columna;
                }
                else {
                    filaMin = filaMax = fila;
                    columnaMin = columnaMax = columna;
                    primeroDetectado = 1;
                }
            }
        }


    if (area) {
        Xc = (float)Xc / area;
        Yc = (float)Yc / area;
        ancho = columnaMax - columnaMin;
        alto = filaMax - filaMin;
        Lx = Ly = 0;
        for (fila = 0; fila < 480; fila++)
            if (m[fila * 480 + (int)Xc] < 100) Ly++;
        for (columna = 0; columna < 640; columna++)
            if (m[(int)Xc * 480 + columna] < 100) Lx++;
    }
    return 0;
}
```

ej7) Codifica un programa en lenguaje C para que un sistema embebido pueda determinar el contorno de un objeto en una imagen capturada mediante una cámara de 640x480 puntos de resolución. Cada punto se representa en tono de gris en blanco y negro mediante un único byte, con valores desde el color negro (valor 0), hasta el blanco (valor 255).

Se parte de la imagen original (como la de la fig. 1) obtenida con la cámara y guardada en una matriz `uint8_t imagenCamara[307200]`.

Se genera como resultado otra imagen (como la de la fig. 2) en la matriz `uint8_t imagenProcesada[307200]` en la que los puntos que pertenecen al contorno del objeto son de color blanco y en el resto de la imagen son de color negro. Las imágenes se representan en memoria mediante un bloque de 640x480=307200 bytes, guardando la información por filas, desde la esquina superior izquierda a la inferior derecha.

Para determinar si un punto (x,y) pertenece al contorno, hay que analizar su color y el de los puntos circundantes, en total 9 puntos, como se muestra en la fig. 3. Hay que determinar la media de color de los tres puntos más oscuros (para este ejemplo, (32+33+34)/3=33) y la media de color de los tres puntos más claros (en este caso, (111+126+153)/3=130). Cuando la diferencia entre ambas medias supera 10 unidades, entonces el punto (x,y) pertenece al contorno (en este caso es así, ya que 130-33=97). El objeto nunca toca los bordes de la imagen.

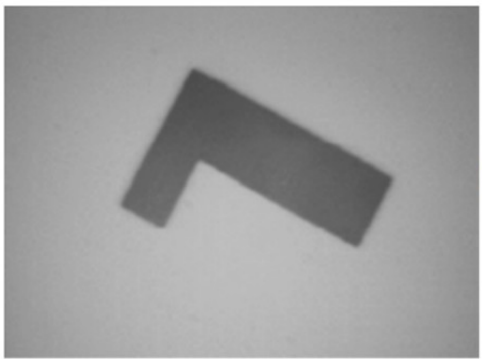


fig. 1

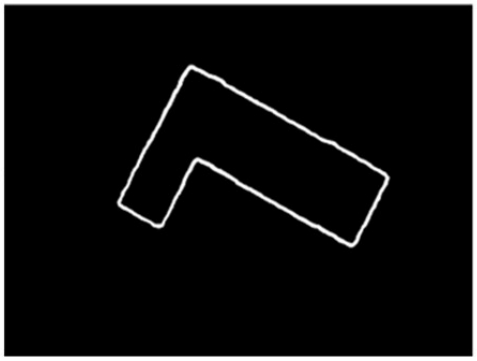


fig.2

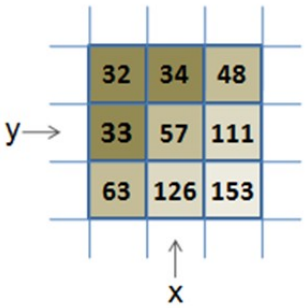


fig.3

Una solución:

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

uint8_t imagenCamara[307200];
uint8_t imagenProcesada[307200];

int main() {
    int i,j,f,c;
    uint8_t temp[9];

    for(i=0; i<640; i++)
        imagenProcesada[i]=0;

    j=0;
    for(i=0; i<480; i++){
        imagenProcesada[j]=0;
        j+=640;
    }

    for(i=307199; i>307199-640; i--)
        imagenProcesada[i]=0;
```

```
    j=639;
    for(i=0; i<480; i++){
        imagenProcesada[j]=0;
        j+=640;
    }

    for(f=1; f<479; f++){
        for(c=1; c<639; c++){
            temp[0] = imagenCamara[f*640+c];
            temp[1] = imagenCamara[f*640+c+1];
            temp[2] = imagenCamara[f*640+c-1];
            temp[3] = imagenCamara[f*640+c-640];
            temp[4] = imagenCamara[f*640+c-640+1];
            temp[5] = imagenCamara[f*640+c-640-1];
            temp[6] = imagenCamara[f*640+c+640];
            temp[7] = imagenCamara[f*640+c+640+1];
            temp[8] = imagenCamara[f*640+c+640-1];

            //Los ordenamos de menor a mayor
            int minimo;
            for(i = 0; i<8; i++) {
                int posMinimo = i;
                for(j=i; j<9; j++) {
                    if(temp[j] < temp[posMinimo]) {
                        posMinimo = j;
                        minimo = temp[posMinimo];
                    }
                 }
                 if(temp[posMinimo]!=temp[i]) {
                        temp[posMinimo] = temp[i];
                        temp[i] = minimo;
                 }
            }

            double ngran, npeque, diff;
            //variables donde estaran las medias de los grandes y los pequeños y la diferencia

            npeque = (temp[0]+temp[1]+temp[2])/3;
            ngran = (temp[6]+temp[7]+temp[8])/3;
            diff = ngran-npeque;
            if(diff<=10)
                imagenProcesada[f*640+c]=0;
            else
                imagenProcesada[f*640+c]=255;
        }
    }
    return 0;
}
```

---

ej8) Prepara el programa desarrollado en el ejercicio ej7) dividiéndolo en dos funciones:

- Una para ordenar de menor a mayor una tabla.

- Otra para procesar la imagen obtenida por la cámara y generar la imagen resultado de la detección del objeto.

Una solución:

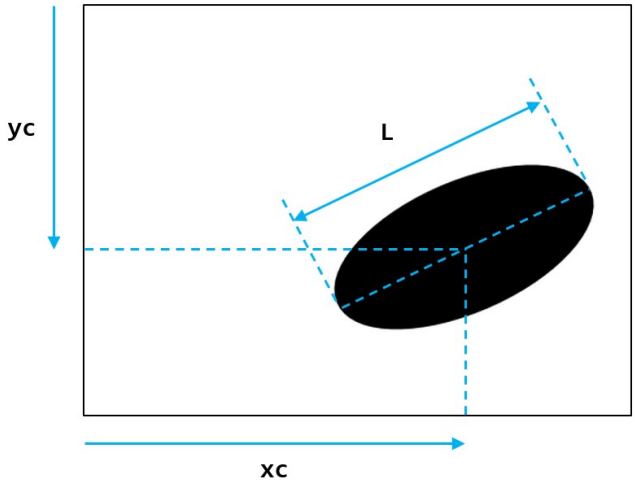
```
#include <stdint.h>
```

```
void ordenaMatrizBytes(uint8_t * matriz, int nDatos) {
    int i, j;

    //Los ordenamos de menor a mayor
    int minimo;
    for(i = 0; i<nDatos-1; i++) {
        int posMinimo = i;
        for(j=i; j<nDatos; j++) {
            if(matriz[j] < matriz[posMinimo]) {
                posMinimo = j;
                minimo = matriz[posMinimo];
            }
         }
         if(matriz[posMinimo]!=matriz[i]) {
                matriz[posMinimo] = matriz[i];
                matriz[i] = minimo;
         }
    }
}

void determinaContorno(uint8_t * pCamara, uint8_t * pProcesada) {

    int i,j,f,c;
    uint8_t temp[9];

    for(i=0; i<640; i++)
        pProcesada[i]=0;

    j=0;
    for(i=0; i<480; i++){
        pProcesada[j]=0;
        j+=640;
    }

    for(i=307199; i>307199-640; i--)
        pProcesada[i]=0;

    j=639;
    for(i=0; i<480; i++){
        pProcesada[j]=0;
        j+=640;
    }

    for(f=1; f<479; f++){
        for(c=1; c<639; c++){
            temp[0] = pCamara[f*640+c];
            temp[1] = pCamara[f*640+c+1];
            temp[2] = pCamara[f*640+c-1];
            temp[3] = pCamara[f*640+c-640];
            temp[4] = pCamara[f*640+c-640+1];
            temp[5] = pCamara[f*640+c-640-1];
            temp[6] = pCamara[f*640+c+640];
            temp[7] = pCamara[f*640+c+640+1];
            temp[8] = pCamara[f*640+c+640-1];

            ordenaMatrizBytes(temp, 9);

            double ngran, npeque, diff;
            //variables donde estaran las medias de los grandes y los pequeños y la diferencia

            npeque = (temp[0]+temp[1]+temp[2])/3;
            ngran = (temp[6]+temp[7]+temp[8])/3;
            diff = ngran-npeque;
            if(diff<=10)
                pProcesada[f*640+c]=0;
            else
                pProcesada[f*640+c]=255;
        }
    }
}
```

```c
uint8_t imagenCamara[640 * 480];  // Imagen original obtenida con la cámara
uint8_t imagenProcesada[640 * 480];  // Imagen con el contorno del objeto en blanco y fondo negro

int main() {
    determinaContorno(imagenCamara, imagenProcesada);
    return 0;
}
```

---

ej33) Un sistema embebido está conectado a una cámara de vídeo de 640x480 puntos de resolución. Cada punto de una imagen se representa en niveles de gris mediante un byte sin signo (desde 0=negro hasta 255=blanco). Se supone que la imagen ya se ha capturado y todos los bytes están disponibles en una matriz `uint8_t m[307200]` creada en el archivo `imagen.h` suministrado.
Todos los bytes se organizan en esta matriz por filas de puntos en la imagen, desde la fila superior a la inferior y dentro de cada fila, de izquierda a derecha. Codifica un algoritmo que calcule una serie de características de objetos elipsoidales (de color oscuro, con valor menor que 100) existente en la escena (el fondo es de color claro, mayor que 200). Para cada objeto hay que obtener su posición `xc`, `yc`, su dimensión máxima `L`. y su área (número de puntos ocupados por el objeto).



El resultado debería ser:

Objeto	xc	yc	L	área
1	319.01	109.17	49.28	4759
2	157.88	144.37	52.53	2710
3	401.88	246.30	79.88	7210
4	224.10	290.94	84.89	11887

Una solución:

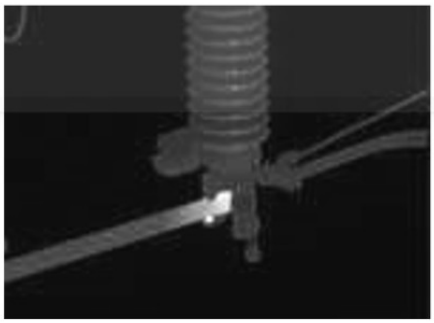
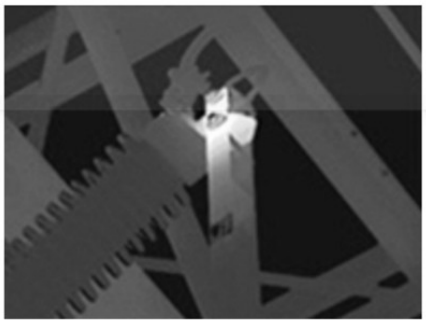
```c
#include "imagen.h"
#include <stdint.h>
#include <math.h>
#include <stdlib.h>

typedef struct {
    float area, xc, yc, L;
} Objeto;
// Estructura para representar los datos de cada objeto detectado,
// con su posición (xc, yc) y su tamaño L
```

```
int main() {
    float sumaX = 0;
    float sumaY = 0;
    int n = 0;

    for (int f = 0; f < 480; f++)  // Por cada fila de puntos en la imagen
        for(int c = 0; c < 640; c++) // Por cada columna
            if (m[f * 640 + c] > 200)  // Si es un punto claro
                m[f * 640 + c] = 255;  // lo pone en blanco
            else m[f * 640 + c] = 0;  // si no, en negro

    int nObjetos = 0;  // Contador de objetos detectados

    for (int f = 1; f < 480-1; f++) {  // Desde la segunda a la penúltima fila

        int enObjeto = 0;
        // Cierto si al recorrer los puntos de la fila, se están recorriendo puntos de un objeto

        int cInicial = 0, cFinal = 0;
        // Columnas inicial y final de detección del objeto en la fila

        for (int c = 1; c < 640-1; c++) {  // Desde la segunda columna a la penúltima
            if ((! enObjeto) && (m[f*640+c] == 0)) {  // Si aún no se detectó objeto y es punto negro
                cInicial = c;  // Guarda columna de primer punto del objeto en la fila
                enObjeto = 1;  // Indica que se detectó objeto
            }
            if (enObjeto && m[f*640+c] == 255) {  // Si ya se detectó objeto y es punto blanco
                enObjeto = 0;  // Indica que ya no es punto de objeto
                cFinal = c-1;  // Guarda la última columna del objeto en la fila
                int objetoEncima = 0;  // Buleano que indica si el objeto se detectó en fila superior
                for(int c = cInicial; c <= cFinal; c++) {  // Para todos los puntos del objeto
                    if (m[(f-1)*640+c] > 0 && m[(f-1)*640+c] < 255)  // Si encima se detectó objeto
                        objetoEncima = m[(f-1)*640+c];  // Guarda el número de objeto detectado encima
                }
                if (objetoEncima) {  // Si se detectó objeto en la fila superior
                    for (int c = cInicial; c <= cFinal; c++)  // Para los puntos detectados en fila
                        m[f*640+c] = objetoEncima;  // Guarda el número de objeto
                } else {  // Si no
                    nObjetos ++;  // Indica que se detectó un nuevo objeto
                    for (int c = cInicial; c <= cFinal; c++)  // Para todos los puntos donde se detectó
                        m[f*640+c] = nObjetos;  // Guarda el nuevo número de objeto
                }
            }
        }
    }

    Objeto * pObjetos = (Objeto *) malloc(nObjetos * sizeof(Objeto));
    // Reserva un nuevo bloque de memoria dinámica con tantas estructuras como objetos detectados

    for (int iObjeto = 0; iObjeto < nObjetos; iObjeto++) {  // Por cada objeto detectado
        Objeto * p = pObjetos + iObjeto;  // Puntero a la estructura del objeto
        p->xc = 0;  // Inicializa el cálculo de su coordenada X
        p->yc = 0;  // Inicializa el cálculo de su coordenada Y
        int nPuntos = 0;  // Contador de puntos del objeto
        for(int f = 0; f < 480; f++)  // Para cada fila de la imagen
            for(int c = 0; c < 640; c++)  // Para cada columna de la imagen
                if (m[f*640+c] == iObjeto+1) {  // Si es un punto del objeto
                    p->xc += c;  // Acumula las coordenadas X
                    p->yc += f;  // Acumula las coordenadas Y
                    nPuntos++;  // Indica que se detecó un nuevo punto
                }
        p->xc /= nPuntos;  // Halla la media
        p->yc /= nPuntos;
        p->area = nPuntos;  // El área es el número de puntos del objeto
        p->L = 0;  // Inicializa el cálculo de la longitud del objeto
        for(int f = 0; f < 480; f++)  // Por cada fila de la imagen
            for(int c = 0; c < 640; c++) {  // Por cada columna de la imagen
                if (m[f*640+c] == iObjeto+1) {  // Si es punto del objeto
                    float dx = c - p->xc;  // Distancia en eje X al centro
                    float dy = f - p->yc;  // Distancia en eje Y al centro
```

```
                    float d = sqrt(dx*dx+dy*dy);  // Distancia euclídea
                    if (p->L < d)  // Si la nueva distancia es mayor
                        p->L = d;  // la guarda
                }
            }
    }

    return 0;
}
```

---

ej65) Una compañía eléctrica desea inspeccionar los puntos de conexión de cables en torres de una línea de alta tensión. Para ello utiliza los servicios de un helicóptero en el que se carga una cámara de video termográfica y un sistema embebido que analiza las imágenes tomadas por la cámara y avisa cuando detecta problemas de sobrecalentamiento por una conexión mal realizada en la línea o por problemas debidos a la corrosión. Esta cámara utiliza un sensor que detecta únicamente radiación infrarroja, de manera que genera imágenes de 1024x1024 puntos en tonos de gris, donde cada punto se expresa en un byte, que van desde 0=negro cuando detecta temperatura baja hasta 254 indicando diferentes valores de temperatura y un color 255=blanco cuando detecta una temperatura excesiva que señala un posible problema. Elabora un programa en lenguaje C para este computador de forma que recoja imágenes contínuamente y las analice para avisar cuando detecte un problema de este tipo en la línea. Para capturar una imagen hay que llamar a la función **`void capturaImagen (unsigned char *)`** a la que hay que pasar la dirección de memoria a partir de la cual va a guardar todos los bytes que componen la imagen, por filas, desde la esquina superior izquierda a la inferior derecha. Para aumentar la fiabilidad de la detección y debido a la baja velocidad con la que se va a mover el helicóptero en el momento de la inspección y también debido a la gran velocidad con la que se van a poder capturar y analizar las imágenes, impanta el siguiente algoritmo: se considera que se ha detectado un sobrecalentamiento cuando en diez imágenes consecutivas hay al menos 10 puntos en blanco y sucede que entre una imagen y la siguiente, más del 50% de los puntos de color blanco de la primera también están en blanco en la segunda en las mismas posiciones de la imagen. Mientras se siga detectando el problema, hay que mantener encendido un aviso acústico que podemos activar pasando un buleano cierto a la función **`void alarma (int)`**. Para apagar el aviso acústico hay que llamar a esa misma función pasándole un buleano falso.

Una solución:

```
unsigned char imagen[1024*1024], imagenAnterior[1024*1024];
// Para guardar la última y penúltima imágenes capturadas

void main() {
    uint32_t nPuntos, nPuntosAnterior, nPuntosCoincidentes, nDetecciones;

    nDetecciones = 0;  // Número de coincidencias entre dos imágenes consecutivas
    capturaImagen(imagenAnterior);  // Recoge una primera imagen
    nPuntosAnterior = 0;  // Para contar número de puntos bancos
    for(uint32_t i = 0; i < 1024*1024; i++) {  // Recorriendo la imagen ...
        if (imagen[i] == 255)   // Si es punto blanco ...
            nPuntosAnterior ++;  // Incrementa contador
```

```
    while(1) {  // Repite contínuamente ...
        capturaImagen(imagen);  // Recoge una nueva imagen
        nPuntos = 0;  // Para contar número de puntos en blanco
        nPuntosCoincidentes = 0;  // Cuenta número de puntos blancos coincidentes con imagen anterior
        for(uint32_t i = 0; i < 1024*1024; i++) {  // Recorriendo la imagen ...
            if (imagen[i] == 255) {  // Si es punto blanco ...
                nPuntos ++;  // Incrementa número de puntos blancos
                if (imagenAnterior[i] == 255)  // Si también es blanco en la imagen anterior ...
                    nPuntosCoincidentes ++;  // Incrementa número de puntos blancos coincidentes
            }
        }
        if (nPuntosAnterior >= 10 && nPuntos >= 10 && nPuntosCoincidentes >= nPuntos / 2)
            nDetecciones ++;  // Incrementa si 10 puntos blancos y 50% coincidentes
        else nDetecciones = 0;  // Si no, resetea número de detecciones
        alarma(nDetecciones >= 10);  // Sonido si hubo 10 detecciones
        for (uint32_t i = 0; i < 1024*1024; i++)
            imagenAnterior[i] = imagen[i];  // Copia en la imagen anterior
        nPuntosAnterior = nPuntos;  // Copia contador de puntos blancos
    }
}
```

Automatización

ej9) Codifica un programa en lenguaje C para un microcontrolador que tiene que controlar las operaciones de un pulmón de almacenamiento temporal de hasta 24 cajas (4 torres de 6 cajas cada una como máximo) que hay que llevar de una cinta transportadora de entrada a otra cinta de salida, según se indica en la figura.

El microcontrolador dispone de los puertos de entrada/salida digital A, B, C y D, cada uno de ellos con 8 señales digitales A0 a A7, B0 a B7, C0 a C7 y D0 a D7. Las señales de los puertos A, B y C están preconfiguradas como entradas y las del puerto D como salidas. Para medir las señales de los puertos A, B y C, hay que realizar lecturas en las direcciones de memoria 0x100, 0x110 y 0x120, respectivamente. Para modificar las salidas del puerto D hay que realizar escrituras en la dirección 0x200.

Las cintas transportadoras de entrada CE y salida CS funcionan de forma automática, no las controlamos desde nuestro programa, están en funcionamiento siempre y cuando haya al menos una caja sobre ellas y salvo que alguna caja haya llegado al final.

Un operario deposita cajas al comienzo de la cinta de entrada CE. Las coloca de forma que siempre hay una cierta separación entre ellas. La grúa se encarga de realizar las siguientes operaciones:
a) cuando llega una caja al final de CE, si no hay caja al comienzo de la cinta de salida CS, hay que utilizar la grúa para transportar la caja allí,
b) si no hay caja al comienzo de CS y hay alguna caja en el pulmón, la lleva al comienzo de CS
c) cuando hay una caja al final de CE, hay que utilizar la grúa para transportarla al pulmón (en el caso de que haya algún hueco).
Las operaciones a) son las más prioritarias, luego las operaciones b) y las operaciones c) son las menos prioritarias. Al final de la cinta CS hay otro operario que retira cada caja para llevarla a otro lugar.

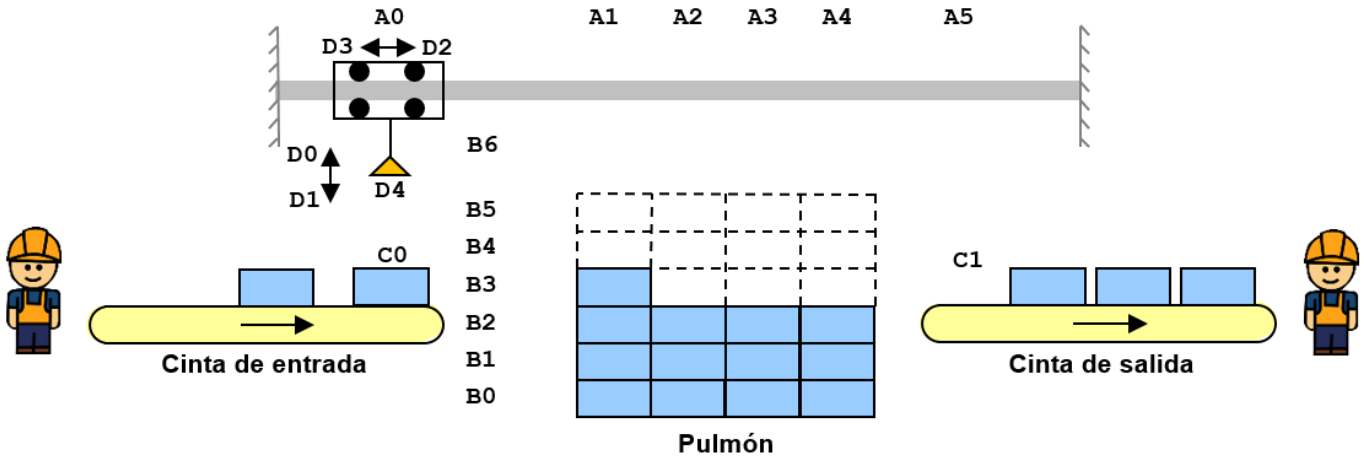
Al final de CE hay un sensor conectado a la línea C0 (puerto C, bit 0) del microcontrolador. Cuando este sensor detecta una caja, genera un 1. Al comienzo de CS también hay otro sensor de presencia conectado a C1.

Las cajas se transportan mediante una grúa consistente en un carro que se puede mover a la izquierda y la derecha a lo largo de un carril horizontal. Dispone de una ventosa para recoger los paquetes, que se puede subir y bajar.
Existen sensores de presencia conectados a las señales A0 a A5 para detectar si la grúa se sitúa horizontalmente en su carril sobre el final de la cinta de entrada (señal A0), sobre alguna posición de

apilamiento de cajas en el pulmón (señales A1 a A4) o sobre el comienzo de la cinta de salida (señal A5).

En el movimiento vertical de la ventosa, extendiendo o recogiendo el cable de la grúa, se activan sensores conectados a las señales B0 a B6 del puerto B del microcontrolador. Cuando la ventosa está a una altura adecuada para coger o depositar un paquete en las filas 0 a 5 del pulmón, se pone a 1 la señal correspondiente B0 a B5. La señal B3 también corresponde a una altura adecuada para coger o depositar cajas en las cintas. La altura correspondiente a B6 es adecuada para mover la grúa horizontalmente sin provocar colisiones.

Se utilizan también las salidas digitales del puerto D para recoger el cable y subir la ventosa (sube mientras mantenemos D0=1), para bajarla (mientras D1=1), para coger una caja activando la ventosa (mientras D4=1), para mover el carro hacia la derecha (mientras D2=1) y hacia la izquierda (mientras D3=1).



Una solución:

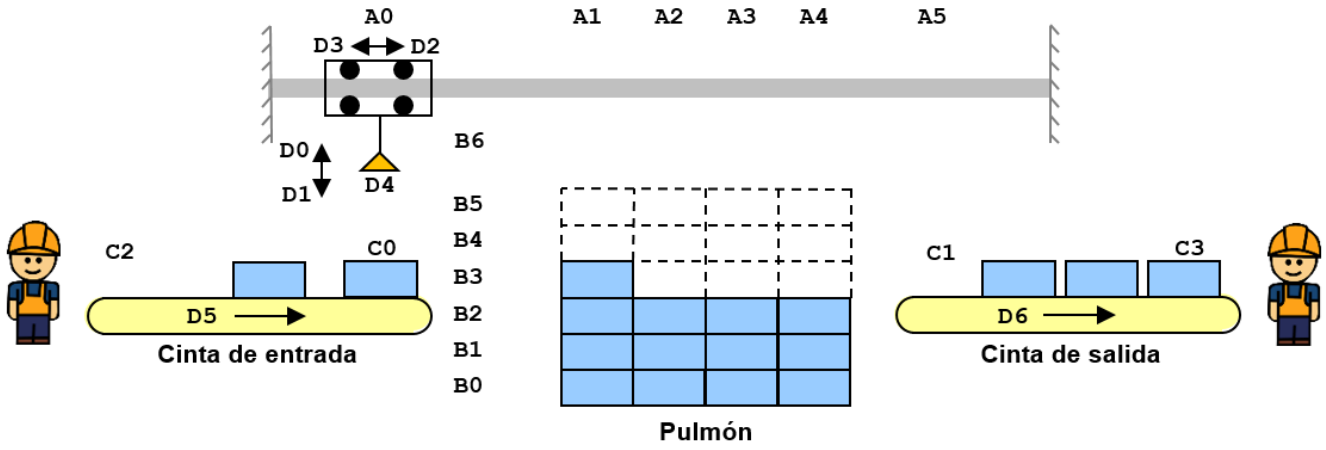
```
#include <stdint.h>
uint8_t* A = (uint8_t*)0x100;
uint8_t* B = (uint8_t*)0x110;
uint8_t* C = (uint8_t*)0x120;
uint8_t* D = (uint8_t*)0x200;
// Punteros para acceder a los puertos de E/S digital

int leeBit(uint8_t* puerto, int nBit) {  // Devuelve un buleano cierto si un bit de un puerto está a 1
    return (*puerto) & (1 << nBit);
}

void escribeBit(uint8_t* puerto, int nBit, int valor) { // Modifica un bit de un puerto
    if (valor)
        *puerto |= 1 << nBit;
    else *puerto &= ~(1 << nBit);
}

void mueveGruaVertical(int posicion) {  // LLeva la grúa a esa posición vertical
    static int posicionActual = 6;  // Recuerda en qué posición está
    if (posicion == posicionActual) return;  // Si ya está allí, termina
    if (posicion > posicionActual)  // Si hay que subir ...
        escribeBit(D, 0, 1);  // Activa subida
    else
        escribeBit(D, 1, 1);  // Activa bajada
    while (!leeBit(B, posicion));  // Espera a llegar a la posición
    escribeBit(D, 0, 0);  // Desactiva subida
    escribeBit(D, 1, 0);  // Desactiva bajada
    posicionActual = posicion;  // Recuerda dónde está
}
```

```
void mueveGruaHorizontal(int posicion) {  // Lleva a la grúa a esa posición horizontal
    static int posicionActual = 0;  // Recuerda en qué posición está
    if (posicion == posicionActual) return;  // Si ya está allí, termina
    if (posicion > posicionActual)  // Si hay que ir a la derecha ...
        escribeBit(D, 2, 1);  // Activa movimiento a la derecha
    else
        escribeBit(D, 3, 1);  // Activa movimiento a la izquierda
    while (!leeBit(A, posicion));  // Espera a llegar a la posición
    escribeBit(D, 2, 0);  // Desactiva movimiento a la derecha
    escribeBit(D, 3, 0);  // Desactiva movimiento a la izquierda
}

void ventosa(int activar) {  // Activa o desactiva la ventosa
    escribeBit(D, 4, activar);
}

void inicializaGrua() {  // Sitúa la grúa arriba a la izquierda
    escribeBit(D, 0, 1);  // Activa grúa subir
    while (!leeBit(B, 6));  // Espera a que llegue arriba
    escribeBit(D, 0, 0);  // Desactiva subida
    escribeBit(D, 3, 1); // Activa grúa izquierda
    while (!leeBit(A, 0));  // Espera a que llegue a la izquierda
    escribeBit(D, 3, 0);  // Desactiva grúa izquierda
}

int cajaEnCintaEntrada() {  // Devuelve buleano cierto si hay caja al final de la cinta de entrada
    return leeBit(C, 0);
}

int cajaEnCintaSalida() {  // Devuelve buleano cierto si hay caja al comienzo de la cinta de salida
    return leeBit(C, 1);
}

void main() {
    int nCajasEnPulmon = 0;  // Número de cajas almacenadas en el pulmón

    *D = 0; // Todas las salidas desactivadas
    inicializaGrua();   // Sitúa la grúa arriba a la izquierda

    while (1) {  // Bucle infinito

        if (cajaEnCintaEntrada() && !cajaEnCintaSalida()) {
            // Si hay pieza al final de cinta de entrada pero no al principio de cinta de salida ...

            mueveGruaHorizontal(0);  // Sitúa la grúa sobre cinta entrada
            mueveGruaVertical(3);  // Baja
            ventosa(1);  // Coge caja
            mueveGruaVertical(6);  // Sube
            mueveGruaHorizontal(5);  // Sitúa la grúa sobre cinta salida
            mueveGruaVertical(3);  // Baja
            ventosa(0);  // Deja caja
            mueveGruaVertical(6);  // Sube
        }

        else if (nCajasEnPulmon > 0 && !cajaEnCintaSalida()) {
            // Si hay cajas en pulmón que se pueden llevar a la cinta de salida ...

            int fila = nCajasEnPulmon / 4;  // Fila en el pulmón, de 0 a 5
            int columna = nCajasEnPulmon % 4;  // Columna en el pulmón, de 0 a 5
            mueveGruaHorizontal(columna + 1);  // Sitúa la grúa sobre la columna
            mueveGruaVertical(fila);  // Baja la grúa hasta la caja del pulmón
            ventosa(1);  // Coge la caja
            mueveGruaVertical(6);  // Sube la grúa
            mueveGruaHorizontal(5);  // Sitúa la grúa sobre cinta salida
            mueveGruaVertical(3);  // Baja
```

```
            ventosa(0);  // Deja la caja
            mueveGruaVertical(6);  // Sube
            nCajasEnPulmon--;  // Decrementa contador de cajas en el pulmón
        }

        else if (nCajasEnPulmon < 24 && cajaEnCintaEntrada()) {
            // Si hay caja al final de cinta de entrada y hay hueco en pulmón ...

            nCajasEnPulmon++;  // Va a haber una caja más en el pulmón
            int fila = nCajasEnPulmon / 4;  // Fila donde depositar la caja, de 0 a 5
            int columna = nCajasEnPulmon % 4;  // Columna donde depositar la caja, de 0 a 3
            mueveGruaHorizontal(0);  // Sitúa grúa sobre cinta de entrada
            mueveGruaVertical(3);  // Baja
            ventosa(1);  // Coge caja
            mueveGruaVertical(6);  // Sube
            mueveGruaHorizontal(columna + 1);  // Sitúa caja sobre posición en el pulmón
            mueveGruaVertical(fila);  // Baja hasta la fila
            ventosa(0);  // Deja caja
            mueveGruaVertical(6);  // Sube
        }
    }
}
```

---

ej10) Modifica el programa del ejercicio ej9) para que, además de controlar las operaciones de la grúa, se controlen las cintas de entrada y salida. Cualquiera de estas cintas tiene que estar en movimiento (siempre transportando cajas hacia la derecha) si hay alguna caja sobre ella y si no llegó ninguna caja al final de la cinta. Para la cinta de entrada tenemos un sensor adicional de presencia conectado a la entrada digital C2 de nuestro microcontrolador, para comprobar si hay alguna caja al comienzo de la cinta. Para la cinta de salida hay un sensor conectado a C3 para comprobar si hay alguna caja en la posición final. Con las salidas digitales D5 y D6 podemos poner en movimiento o detener las cintas de entrada y salida, mientras alguna de estas salidas está a 1, la cinta correspondiente está en movimiento.



Una solución:

A la solución de ej9) hay que añadir funciones para atender a las cintas:

```
void atiendeCintaEntrada() {
    static int enMovimiento = 0, sensor1Anterior = 0, sensor2Anterior = 0, nCajas = 0;
    int sensor1 = leeBit(C, 2);
    int sensor2 = leeBit(C, 0);
    if (!sensor1Anterior && sensor1)  // Si apareció una caja al principio
        nCajas++;  // Incrementa contador de cajas
    if (sensor2Anterior && !sensor2)  // Si desapareció una caja al final
        nCajas--;  // Decrementa contador de cajas
```

```
    if (enMovimiento && sensor2) {  // Si en movimiento y llegó caja al final
        escribeBit(D, 5, 0);  // Para la cinta
        enMovimiento = 0;
    } else if (!sensor2 && nCajas > 0 && !enMovimiento) { // Si cajas y ninguna al final
        escribeBit(D, 5, 1);  // Pone en marcha la cinta
        enMovimiento = 1;
    }
    sensor2Anterior = sensor2;
    sensor1Anterior = sensor1;
}

void atiendeCintaSalida() {
    static int enMovimiento = 0, sensor1Anterior = 0, sensor2Anterior = 0, nCajas = 0;
    int sensor1 = leeBit(C, 1);
    int sensor2 = leeBit(C, 3);
    if (!sensor1Anterior && sensor1)  // Si apareció una caja al principio
        nCajas++;  // Incrementa contador de cajas
    if (sensor2Anterior && !sensor2)  // Si desapareció una caja al final
        nCajas--;  // Decrementa contador de cajas
    if (enMovimiento && sensor2) {  // Si en movimiento y llegó caja al final
        escribeBit(D, 6, 0);  // Para la cinta
        enMovimiento = 0;
    } else if (!sensor2 && nCajas > 0 && !enMovimiento) { // Si cajas y ninguna al final
        escribeBit(D, 6, 1);  // Pone en marcha la cinta
        enMovimiento = 1;
    }
    sensor2Anterior = sensor2;
    sensor1Anterior = sensor1;
}

void atiendeCintas() {
    atiendeCintaEntrada();
    atiendeCintaSalida();
}
```

En las siguientes funciones, donde se ejecutan bucles de espera, hay que realizar llamadas a `atiendeCintas()` para que en esos bucles no se desatiendan a las cintas:

```
void mueveGruaVertical(int posicion) {  // LLeva la grúa a esa posición vertical
    static int posicionActual = 6;  // Recuerda en qué posición está
    if (posicion == posicionActual) return;  // Si ya está allí, termina
    if (posicion > posicionActual)  // Si hay que subir ...
        escribeBit(D, 0, 1);  // Activa subida
    else
        escribeBit(D, 1, 1);  // Activa bajada
    while (!leeBit(B, posicion)) atiendeCintas();  // Espera a llegar a la posición
    escribeBit(D, 0, 0);  // Desactiva subida
    escribeBit(D, 1, 0);  // Desactiva bajada
    posicionActual = posicion;  // Recuerda dónde está
}

void mueveGruaHorizontal(int posicion) {  // Lleva a la grúa a esa posición horizontal
    static int posicionActual = 0;  // Recuerda en qué posición está
    if (posicion == posicionActual) return;  // Si ya está allí, termina
    if (posicion > posicionActual)  // Si hay que ir a la derecha ...
        escribeBit(D, 2, 1);  // Activa movimiento a la derecha
    else
        escribeBit(D, 3, 1);  // Activa movimiento a la izquierda
    while (!leeBit(A, posicion)) atiendeCintas();  // Espera a llegar a la posición
    escribeBit(D, 2, 0);  // Desactiva movimiento a la derecha
    escribeBit(D, 3, 0);  // Desactiva movimiento a la izquierda
}
```

```
void inicializaGrua() {  // Sitúa la grúa arriba a la izquierda
    escribeBit(D, 0, 1);  // Activa grúa subir
    while (!leeBit(B, 6)) atiendeCintas();  // Espera a que llegue arriba
    escribeBit(D, 0, 0);  // Desactiva subida
    escribeBit(D, 3, 1); // Activa grúa izquierda
    while (!leeBit(A, 0)) atiendeCintas();  // Espera a que llegue a la izquierda
    escribeBit(D, 3, 0);  // Desactiva grúa izquierda
}
```

En el programa principal, dentro del bucle, hay que incluir una llamada a `atiendeCintas()`:

```
void main() {
    int nCajasEnPulmon = 0;  // Número de cajas almacenadas en el pulmón

    *D = 0; // Todas las salidas desactivadas
    inicializaGrua();   // Sitúa la grúa arriba a la izquierda

    while (1) {  // Bucle infinito

        atiendeCintas();  // Atiende tambien a las cintas

        if (cajaEnCintaEntrada() && !cajaEnCintaSalida()) {
            // Si hay pieza al final de cinta de entrada pero no al principio de cinta de salida ...

            mueveGruaHorizontal(0);  // Sitúa la grúa sobre cinta entrada
            mueveGruaVertical(3);  // Baja
            ventosa(1);  // Coge caja
            mueveGruaVertical(6);  // Sube
            mueveGruaHorizontal(5);  // Sitúa la grúa sobre cinta salida
            mueveGruaVertical(3);  // Baja
            ventosa(0);  // Deja caja
            mueveGruaVertical(6);  // Sube
        }

        else if (nCajasEnPulmon > 0 && !cajaEnCintaSalida()) {
            // Si hay cajas en pulmón que se pueden llevar a la cinta de salida ...

            int fila = nCajasEnPulmon / 4;  // Fila en el pulmón, de 0 a 5
            int columna = nCajasEnPulmon % 4;  // Columna en el pulmón, de 0 a 5
            mueveGruaHorizontal(columna + 1);  // Sitúa la grúa sobre la columna
            mueveGruaVertical(fila);  // Baja la grúa hasta la caja del pulmón
            ventosa(1);  // Coge la caja
            mueveGruaVertical(6);  // Sube la grúa
            mueveGruaHorizontal(5);  // Sitúa la grúa sobre cinta salida
            mueveGruaVertical(3);  // Baja
            ventosa(0);  // Deja la caja
            mueveGruaVertical(6);  // Sube
            nCajasEnPulmon--;  // Decrementa contador de cajas en el pulmón
        }

        else if (nCajasEnPulmon < 24 && cajaEnCintaEntrada()) {
            // Si hay caja al final de cinta de entrada y hay hueco en pulmón ...

            nCajasEnPulmon++;  // Va a haber una caja más en el pulmón
            int fila = nCajasEnPulmon / 4;  // Fila donde depositar la caja, de 0 a 5
            int columna = nCajasEnPulmon % 4;  // Columna donde depositar la caja, de 0 a 3
            mueveGruaHorizontal(0);  // Sitúa grúa sobre cinta de entrada
            mueveGruaVertical(3);  // Baja
            ventosa(1);  // Coge caja
            mueveGruaVertical(6);  // Sube
            mueveGruaHorizontal(columna + 1);  // Sitúa caja sobre posición en el pulmón
            mueveGruaVertical(fila);  // Baja hasta la fila
            ventosa(0);  // Deja caja
            mueveGruaVertical(6);  // Sube
```

```
        }
    }
}
```

---

ej19) Un sistema embebido con microcontrolador dispone de los siguientes elementos:

Tres puertos de señales digitales, cada uno con 8 señales: el puerto A con salidas digitales y los puertos B y C con entradas digitales, mapeados en memoria en las direcciones 0x100, 0x101 y 0x102, respectivamente.

Un sistema de adquisición analógica conectado a un sensor de temperatura en un horno. Dispone de un convertidor analógico/digital de 12 bits sin signo. El número entero resultado de la medición expresa la temperatura del horno en grados centígrados. Este convertidor se maneja utilizando las siguientes señales:
- CONVERTIR (conectada a la señal 0 del puerto A del microcontrolador): es una entrada para el convertidor y a través de ella generamos un flanco descendente para solicitar el comienzo de una conversión.
- FIN (conectada a la señal 4 del puerto C del microcontrolador): es una salida para el convertidor y la utiliza para indicar fin de conversión mediante un flanco descendente. Antes de generar este flanco, el convertidor pone en D0-D11 el resultado de la medida. Vuelve a estado 1 cuando ponemos CONVERTIR a 1.
- D0-D7 (conectadas a las señales 0 a 7 del puerto B): son salidas para el convertidor e indican los 8 bits menos significativos de la última conversión realizada.
- D8-D11 (conectadas a las señales 0 a 3 del puerto C): son salidas para el convertidor e indican los 4 bits más significativos de la última conversión realizada.

Dos lámparas de emergencia que tendrán que estar encendidas en los siguientes casos:
- Lámpara de "temperatura alta": tiene que estar encendida siempre y cuando la temperatura actual sea mayor de 1000 grados centígrados. Está encendida si la señal 1 del puerto A del microcontrolador está a 1.
- Lámpara de "temperatura media alta": tiene que estar encendida siempre y cuando la media de la temperatura calculada desde hace 100 segundos hasta el momento actual sea superior a 900 grados centígrados. Está encendida si la señal 2 del puerto A del microcontrolador está a 1.

Podemos utilizar la variable **`MS`** predefinida y de tipo **`unsigned long`** que indica el tiempo transcurrido desde el comienzo del programa expresado en milisegundos. Esta variable se actualiza automáticamente, desde el programa sólo consultamos su valor.

Codifica un programa para este sistema embebido de forma que esté continuamente pendiente de la temperatura del horno (muestreándola a intervalos de un segundo) y encienda o apague las lámparas cuando sea necesario.

Una solución:

```
#include <stdint.h>

void main() {
    unsigned long tAnterior = MS;  // Para controlar los intervalos de tiempo de 1 s.
    uint8_t* pA, * pB, * pC;  // Punteros para manejar los puertos
    uint16_t medidas[100];  // Medidas de la temperatura en los últimos 100 segundos
    int nMedidas = 0;  // Número de medidas realizadas
    int i;

    pA = (uint8_t*)0x100;  // Apunta a la dirección 0x100 para manejar el puerto A
    pB = (uint8_t*)0x101;  // Apunta a la dirección 0x101 para manejar el puerto B
    pC = (uint8_t*)0x102;  // Apunta a la dirección 0x102 para manejar el puerto C
    *pA = 0x01; // Inicialmente señal CONVERTIR a 1 y lámparas apagadas
```

```
    while (1) {  // Repite continuamente ...
        while (MS - tAnterior < 1000);  // Espera al siguiente segundo
        tAnterior = MS;  // Actualiza esta variable para el siguiente bucle
        *pA &= ~0x01;  // Flanco descendente en señal CONVERTIR
        while (*pC & 0x10);  // Espera a flanco descendente en señal FIN
        if (nMedidas < 100) { // Si aún no se realizaron 100 medidas ...
            medidas[nMedidas] = *pB | ((*pC & 0x0F) << 8);  // Obtiene la medida en 12 bits
            nMedidas++;  // Se realizó una medida más
        } else {
            for (i = 0; i < 99; i++) medidas[i] = medidas[i + 1];  // Desplaza medidas
            medidas[99] = *pB | ((*pC & 0x0F) << 8);  // Obtiene la medida en 12 bits
        }
        *pA |= 0x01;  // Flanco ascendente en señal CONVERTIR
        if (medidas[nMedidas - 1] > 1000)  // Si la última medida supera los 1000 grados
            *pA |= 0x02;  // Enciende la lámpara de temperatura alta
        else *pA &= ~0x02;  // Si no, la apaga
        if (nMedidas == 100) { // Si se dispone de 100 medidas
            float suma = 0;
            for (i = 0; i < 100; i++) suma += medidas[i];  // Las suma
            if (suma / 100 > 900)  // Si la media supera los 900 grados ...
                *pA |= 0x04;  // Enciende la lámpara de temperatura media alta
            else *pA &= ~0x04;  // Si no, la apaga
        }
    }
}
```

---

ej28) Codifica un programa para un sistema embebido que tiene que controlar las operaciones de un surtidor de gasolinera que puede servir los siguientes productos:

- Manguera número 0: Diesel 10e+, precio 1.359 €/litro
- Manguera número 1: Gasolina 98, 1.602 €/litro
- Manguera número 2: Gasolina 95, 1.445 €/litro
- Manguera número 3: Diesel e+, 1.282 €/litro

No se pueden servir varios productos al mismo tiempo. Hay que visualizar el volumen cargado en litros y el importe (a medida que se va cargando el depósito del vehículo), además del precio por litro del combustible seleccionado.

El sistema embebido está gobernado mediante un microcontrolador que dispone de los siguientes puertos de señales digitales:

- Puerto A de 8 entradas digitales.
- Puerto B de 8 entradas digitales.
- Puerto C de 8 salidas digitales.

Para utilizar estos puertos se pueden usar los siguientes recursos de programación ya definidos:

```
enum PuertoESDigital {PUERTOA = 0, PUERTOB, PUERTOC};
// Enumerado para indicar uno de los tres puertos de E/S digital

int entradaDigital(enum PuertoESDigital puerto, int bit);
// Devuelve un buleano indicando el estado de la entrada digital correspondiente a
// un determinado bit (0 a 7) de un puerto

void salidaDigital(enum PuertoESDigital puerto, int bit, int valor);
// Establece un nuevo valor (un buleano) para una salida digital que corresponde a
// un determinado bit (0 a 7) de un puerto
```

En el surtidor existe un sensor por manguera que indica si está depositada en su posición de reposo (en cuyo caso genera un 0 lógico). El sensor de la manguera m está conectado a la entrada digital m del puerto A.
En el gatillo de cada manguera también hay un sensor que permite saber si está accionado (en ese caso genera un 1 lógico). Para la manguera m el sensor correspondiente está conectado a la entrada digital m+4 del puerto A.

En la boca de salida de cada manguera hay un sensor que detecta si el depósito que se está cargando está repleto de combustible (genera un 1 lógico cuando está lleno), en cuyo caso habrá que impedir que se desborde. Para la manguera m su sensor está conectado a la entrada m del puerto B.

Para cada producto hay una bomba independiente, de forma que la bomba de la manguera m la podemos activar o desactivar poniendo a 0 o a 1 la salida digital m del puerto C.

En el surtidor existen tres visualizadores:

- en el primero se muestra el importe del combustible cargado en el vehículo. Habrá que ponerlo a cero cuando se coja una manguera y se actualizará a medida que se esté cargando el producto. Se maneja mediante la función `void visualizaImporte(float valor)`, indicando por parámetro la cantidad a mostrar.

- en el segundo se visualizará el volumen en litros del combustible cargado. También parte de cero cuando se coge una manguera y hay que actualizarlo a medida que se va suministrando combustible. Se maneja mediante la función `void visualizaVolumen(float valor)`.

- en el terecero se mostrará, desde que se coge una manguera, el precio por litro del combustible correspondiente. Se maneja mediante `void visualizaPrecioLitro(float valor)`.

Para cada tipo de combustible hay también un sensor de caudal conectado a una unidad de contaje de combustible.La función `float contadorLitros(int manguera)` devuelve los litros totales suministrados por el surtidor de gasolinera a través de una manguera.

Una solución:

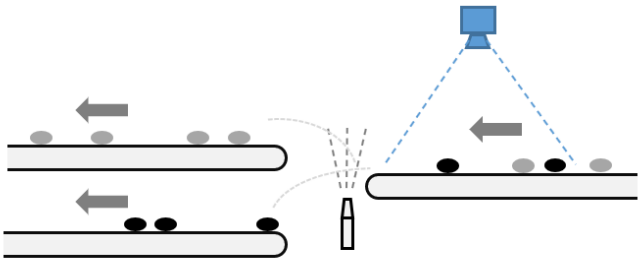
```
float precioLitro[] = {1.359, 1.369, 1.239, 1.099};
// Matriz con el precio por litro de cada tipo de combustible

int depositoLleno (int manguera) {
// Devuelve un buleano cierto si se detecta el depósito lleno a través de la manguera
    return entradaDigital(PUERTOB, manguera);
}
```

```
int gatilloPulsado (int manguera) {
// Devuelve un buleano cierto si se detecta el gatillo pulsado en la manguera
    return entradaDigital(PUERTOA, manguera + 4);
}


int mangueraCogida (int manguera) {
// Devuelve un buleano cierto si se detecta que se ha cogido la manguera
    return entradaDigital(PUERTOA, manguera);
}

void bomba (int manguera, int accionar) {
// Llamamos a esta function para actuar sobre la bomba de la manguera según el buleano 'accionar'
    salidaDigital(PUERTOC, manguera, accionar);
}

void main() {
    int manguera;
    float inicial, volumen;

    while(1) {

        for(manguera = 0; manguera < 4; manguera++) {  // Recorre todas las mangueras

            if (mangueraCogida(manguera)) {  // Si se cogió la manguera ...

                visualizaVolumen (0.0);
                visualizaImporte (0.0);
                // Pone a cero los visualizadores

                visualizaPrecioLitro (precioLitro [manguera]);
                // Visualiza el precio por litro del producto correspondiente a la manguera

                inicial = contadorLitros(manguera);  // Inicializa el volumen cargado

                while (mangueraCogida(manguera)) {
                // Mientras no se haya depositado la manguera ...

                    bomba (manguera, gatilloPulsado(manguera) && ! depositoLleno(manguera));
                    // Suministra combustible siempre que el gatillo esté pulsado y no se haya llenado
                    // el depósito

                    volumen = contadorLitros(manguera) - inicial;  // Volumen cargado en el depósito

                    visualizaVolumen (volumen);
                    visualizaImporte (volumen * precioLitro [manguera]);
                    // Actualiza los visualizadores del volumen cargado y del importe
                }

                bomba (manguera, 0);  // En cualquier caso, desactiva la bomba
            }
        }
    }
}
```

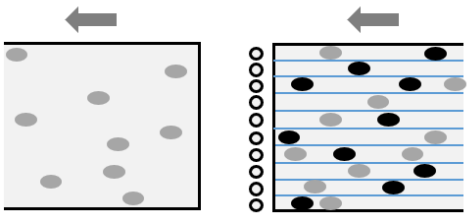
---

ej62) Codifica un programa en lenguaje C para un microcontrolador que controla una instalación de tratamiento de aceituna en el que existe un sistema de visión artificial para la separación de las aceitunas negras de las verdes similar al que se muestra en el vídeo **https://www.youtube.com/watch?v=Nwy0plI_JWE** . Hay una cinta transportadora de entrada de 20cm de ancho que se mueve a 1m/s y en la que existen unas acanaladuras de 2cm de ancho donde se han depositado aceitunas de forma desordenada, por ejemplo tal y como se indica en la figura. Se supone que cada aceituna tiene un diámetro mínimo de 1cm. El sistema de visión consiste en una cámara situada sobre la cinta y a 0.5m de unos sopladores, que permite obtener hasta 200 imágenes/s en blanco y negro de 1024x1024 puntos y que corresponden en la escena a un cuadrado de 20cm de lado. Cada imagen se recoge

llamando a la función `void capturaImagen(unsigned char *)` a la que se le pasa la dirección de memoria a partir de la cual guarda los niveles de gris de todos los puntos de la imagen, cada punto en un byte (0=negro, 255=blanco) y por filas. El eje horizontal de la imagen tiene la misma dirección que la cinta transportadora. Los puntos de la imagen correspondientes a las aceitunas negras corresponden a un nivel de gris inferior a 100 y el fondo de la imagen y las aceitunas verdes aparecen con un nivel de gris superior a 150. Al final de esta cinta hay una fila de 10 sopladores separados entre sí 2cm y centrados con las acanaladuras de la cinta, con el objetivo de crear una corriente ascendente de aire para expulsar las aceitunas verdes a una cinta de salida a un nivel superior. Las aceitunas verdes deben recibir el soplido, para llevarlas otra cinta transportadora superior. Las aceitunas negras no deben de recibir el soplido para que caigan a una cinta inferior. Para activar/desactivar a cada soplador, utilizamos una salida digital de nuestro microcontrolador. Los 8 primeros sopladores están controlados por las 8 salidas digitales de un puerto que se pueden establecer realizando escrituras en un SFR de 8 bits mapeado en la dirección 0x0E00. Los dos sopladores restantes se controlan con las dos salidas menos significativas de otro puerto mapeadoen la dirección 0x0E01. Existe una función `unsigned long systick()` que devuelve el tiempo transcurrido desde el comienzo de la ejecución del programa expresado en milisegundos.



Vista lateral



Vista superior

Una solución:

```
unsigned long systick();
void capturaImagen(unsigned char *);

unsigned char m[10][100];
unsigned char imagen[1024*1024];

int main() {

    int a = 10 * 1024 / 200;
    int b = 20 * 1024 / 200;

    for (int i = 0; i < 10; i++)
        for (int j = 0; j < 100; j++)
            m[i][j] = 0;

    unsigned long tiempoAnterior = systick();
    unsigned char * p = (unsigned char *) 0xE00;
    unsigned char salidas1, salidas2;

    while(1) {

        capturaImagen(imagen);

        for (int canal = 0; canal < 10; canal++)
            for (int columna = 0; columna <= 98; columna++)
                m[canal][columna] = m[canal][columna+1];

        int f = a;
        for (int canal = 0; canal < 10; canal ++) {
            m[canal][99] = imagen[f * 1024 + 512] < 100;
            f += b;
        }
```

```
        for (int canal = 0; canal < 10; canal ++) {
            if (m[canal][0]) {
                if (canal < 8)
                    salidas1 &= ~(1 << canal);
                else salidas2 &= ~(1 << (canal - 8));
            } else {
                if (canal < 8)
                    salidas1 |= (1 << canal);
                else salidas2 |= (1 << (canal - 8));
            }
        }
        p[0] = salidas1;
        p[1] = salidas2;

        while (systick() - tiempoAnterior < 5);
        tiempoAnterior = systick();
    }

    return 0;
}
```

Lenguaje C++

ej11) En un sistema embebido se dispone en un microcontrolador que dispone de varios puertos de entrada/salida digital, cada uno de 8 señales: puerto A, puerto B, etc. 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. También exsiten SFR para realizar las operaciones de entrada/salida de señales, mapeando estos registros en memoria en direcciones consecutivas para los puertos A, B, etc. y en el mismo orden. Codifica una clase con métodos 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.

- un método para configurar varias líneas consecutivas como entrada o como salida

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

- un método para modificar una línea de salida, otro método para modificar 8 líneas de salida consecutivas y otro método para modificar 16 salidas consecutivas.

Archivo **esdigital.h**:

```
#pragma once

#include <cstdint>  // Para uint8_t y uint16_t

class ESDigital {

    // Datos privados:
    uint8_t * puertos;   // Dirección de comienzo de los SFR para E/S con puertos
    uint8_t * configuracion;  // Dirección de comienzo de SFR para configuración de puertos
    unsigned int maxNBit;  // Número máximo de bit utilizado
```

```
public:  // Métodos públicos

    ESDigital(uint8_t* puertos, uint8_t* configuracion);
    // Constructor al que se le indica 'puertos' la dirección donde comienzan los SFR para E/S
    // y en 'configuracion' donde comienzan los SFR para configuración de los puertos

    ~ESDigital();
    // Destructor que configura a todas las E/S digitales como entradas

    void configura(unsigned int n, int entradaOSalida);
    // Configura la señal 'n' como entrada o salida. Si 'entradaOSalida' es cierto,
    // entonces la configura como entrada, si no, como salida.

    void configura(unsigned int primero, unsigned int ultimo, int entradaOSalida);
    // Configura varias señales, desde 'primero' hasta 'ultimo', como entradas o salidas

    int leeBit(unsigned int n);
    // Devuelve el estado de la señal 'n' en un buleano

    uint8_t lee8Bits(unsigned int n);
    // Devuelve en un byte el estado de las 8 señales que comienzan en la señal 'n'

    uint16_t lee16Bits(unsigned int n);
    // Devuelve en 16 bits el estado de las 16 señales que comienzan en la señal 'n'.

    void escribeBit(unsigned int n, int valor);
    // Modifica la salida digital en la señal 'n' y la pone al valor 'valor'.

    void escribe8Bits(unsigned int n, uint8_t valor);
    // Modifica 8 salidas digitales comenzando en la señal 'n' y las pone según los
    // bits de 'valor'

    void escribe16Bits(unsigned int n, uint16_t valor);
    // Modifica 16 salidas digitales comenzando en la señal 'n' y las pone según los
    // bits de 'valor.
};
```

Archivo **esdigital.cpp**:

```
#include "esdigital.h"

ESDigital::ESDigital(uint8_t* puertos, uint8_t* configuracion) {
    // Constructor al que se le indica 'puertos' la dirección donde comienzan los SFR para E/S
    // y en 'configuracion' donde comienzan los SFR para configuración de los puertos

    this->puertos = puertos;
    this->configuracion = configuracion;
    // Guarda los parámetros

    this->maxNBit = 0;  // Por ahora no se configuró ninguna señal
}


ESDigital::~ESDigital() {
    // Destructor que configura a todas las E/S digitales como entradas

    configura(0, maxNBit, 1);
    // Configura las señales desde la 0 hasta 'maxNBit' como entradas
}


void ESDigital::configura(unsigned int n, int entradaOSalida) {
    // Configura la señal 'n' como entrada o salida. Si 'entradaOSalida' es cierto,
    // entonces la configura como entrada, si no, como salida.

    if (entradaOSalida)  // Si es entrada
        configuracion[n / 8] |= (1 << n % 8);  // Hay que poner el bit a 1
```

```
    else configuracion[n / 8] &= ~(1 << n % 8);  // Si no, a 0

    if (n > maxNBit) maxNBit = n;
    // En maxNBit se guarda el número de la última señal configurada
}


void ESDigital::configura(unsigned int primero, unsigned int ultimo, int entradaOSalida) {
    // Configura varias señales, desde 'primero' hasta 'ultimo', como entradas o salidas

    for (unsigned int i = primero; i <= ultimo; i++)  // Desde la primera a la última
        configura(i, entradaOSalida);  // Configura cada señal
}


int ESDigital::leeBit(unsigned int n) {
    // Devuelve el estado de la señal 'n' en un buleano

    return puertos[n / 8] & (1 << n % 8);  // Aplica una máscara para determinar su valor
}


uint8_t ESDigital::lee8Bits(unsigned int n) {
    // Devuelve en un byte el estado de las 8 señales que comienzan en la señal 'n'

    uint8_t resultado = 0;  // Para calcular el resultado a devolver
    if (n % 8 == 0)  // Si n es múltiplo de 8
        resultado = puertos[n / 8];  // Lee un puerto completo
    else  // Si no
        for (unsigned int i = 0; i < 8; i++)
            if (leeBit(n + i))  // Si una señal está a 1
                resultado |= 1 << i;  // Guarda un 1 en su posición del resultado
    return resultado;
}


uint16_t ESDigital::lee16Bits(unsigned int n) {
    // Devuelve en 16 bits el estado de las 16 señales que comienzan en la señal 'n'.

    uint16_t resultado = 0;  // Para calcular el resultado a devolver
    if (n % 8 == 0)  // Si n es múltiplio de 8
        resultado = puertos[n / 8] | (puertos[n / 8 + 1] << 8);  // Combina dos puertos
    else   // Si no
        for (unsigned int i = 0; i < 16; i++)
            if (leeBit(n + i))  // Lee cada señal y si está a 1
                resultado |= 1 << i;  // Pon un 1 en su posición
    return resultado;
}


void ESDigital::escribeBit(unsigned int n, int valor) {
    // Modifica la salida digital en la señal 'n' y la pone al valor 'valor'.

    if (valor)  // Si hay que ponerla a 1 ...
        puertos[n / 8] |= 1 << (n % 8);  // Aplica una ma´scara con una OR
    else  // Si hay que ponerla a 0
        puertos[n / 8] &= ~(1 << (n % 8));  // Aplica una máscara con un AND
}


void ESDigital::escribe8Bits(unsigned int n, uint8_t valor) {
    // Modifica 8 salidas digitales comenzando en la señal 'n' y las pone según los
    // bits de 'valor'

    if (n % 8 == 0)  // Si n es múltiplo de 8 ...
        puertos[n / 8] = valor;  // Guarda los 8 bits en un puerto
    else  // Si no ...
        for (int i = 0; i < 8; i++)
            escribeBit(n + i, valor & (1 << i));  // Modifica de bit en bit
}
```

```
void ESDigital::escribe16Bits(unsigned int n, uint16_t valor) {
    // Modifica 16 salidas digitales comenzando en la señal 'n' y las pone según los
    // bits de 'valor.

    if (n % 8 == 0) {  // Si n es múltiplo de 8 ...
        puertos[n / 8] = valor & 0xFF;  // Guarda los 8 bits menos significativos en un puerto
        puertos[n / 8 + 1] = valor >> 8;  // Guarda los 8 bits menos significativos en el siguiente
    } else  // Si no ...
        for (int i = 0; i < 16; i++)
            escribeBit(n + i, valor & (1 << i));  // Modifica de bit en bit
}
```

---

ej12) 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 el ejercicio ej11 para manejo de E/S digital. Las señales del convertidor se conectan a señales de E/S digital del microcontrolador.

La línea $\overline{CS}$ está cableada a masa y $\overline{SHDN}$ 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 un puntero 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:

Fichero **maxim197.h**:

```
#pragma once

#include <cstdint>
#include "esdigital.h"
```

```
class MAX197 {

public:

    enum RANGO {
        DESACTIVADA = 0,   // Entrada analógica no configurada
        BIPOLAR10 = 1,   // Entrada con rango de -10V a 10V
        BIPOLAR5 = 2,    // Entrada con rango de -5V a 5V
        UNIPOLAR10 = 3,  // Entrada con rango de 0V a 10V
        UNIPOLAR5 = 4    // Entrada con rango de 0V a 5V
    };
    // Enumerado para expresar los diferentes rangos de entrada con los que
    // pueden operar cada una de las 8 entradas analógicas.

private:

    RANGO rangos[8] = { DESACTIVADA, DESACTIVADA, DESACTIVADA, DESACTIVADA,
                        DESACTIVADA, DESACTIVADA, DESACTIVADA, DESACTIVADA };
    // Para indicar el rango utilizado en cada una de las 8 entradas analógicas

    int WR, RD, HBEN, INT, D0;
    // Para indicar el número de señal digital donde se ha conectado cada señal
    // digital del convertidor

    ESDigital* pESDigital;
    // Puntero al objeto utilizado para el manejo de la E/S digital

public:

    MAX197(int WR, int RD, int HBEN, int INT, int D0, ESDigital* pESDigital);
    // Constructor. Se le indica el número de señal digital que corresponden a las señales WR,
    // RD, HB, INT y D0. Se le pasa un puntero 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 (0 a 7) 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 (0 a 7).
};
```

Fichero **maxim197.cpp**:

```
#include "max197.h"


MAX197::MAX197(int WR, int RD, int HBEN, int INT, int D0, ESDigital* pESDigital) {
    // Constructor. Se le indica el número de señal digital que corresponden a las señales WR,
    // RD, HB, INT y D0. Se le pasa un puntero a un objeto de la clase ESDigital para manejar
    // las entradas/salidas digitales del microcontrolador.

    this->WR = WR;
    this->RD = RD;
    this->HBEN = HBEN;
    this->INT = INT;
    this->D0 = D0;
    this->pESDigital = pESDigital;
    // Guarda los parámetros

    pESDigital->configura(WR, 0);
    pESDigital->escribeBit(WR, 1);
    // Configura la señal WR como salida y la pone a 1
```

```
    pESDigital->configura(RD, 0);
    pESDigital->escribeBit(RD, 1);
    // Configura la señal RD como salida y la pone a 1

    pESDigital->configura(HBEN, 0);
    pESDigital->escribeBit(HBEN, 0);
    // Configura la señal HBEN como salida y la pone a 0

    pESDigital->configura(INT, 1);
    // Configura la señal INT como entrada

    pESDigital->configura(D0, D0 + 7, 1);
    // Configura 8 señales digitales a partir de la señal D0 como entradas
}


void MAX197::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.

    if (linea >= 0 && linea < 8)  // Si es una línea de 0 a 7 ...
        rangos[linea] = rango;  // recuerda el rango elegido
}


void MAX197::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

    int j = 0;
    for (int i = 0; i < 8; i++)  // Recorriendo las 8 entradas analógicas ...
        if (rangos[i] != DESACTIVADA)  // Si la entrada no está desactivada ...
            pMedidas[j++] = mide(i);  // Mide la señal y guarda el resultado
}


float MAX197::mide(int linea) {
// Devuelve el resultado de medir la entrada analógica indicada por parámetro (0 a 7).

    pESDigital->configura(D0, D0 + 7, 0);  // Configura el bus de datos como salidas

    uint8_t b = linea;  // Guarda en un byte el número de línea

    if (rangos[linea] == BIPOLAR10) b |= 0x58;
    else if (rangos[linea] == BIPOLAR5) b |= 0x48;
    else if (rangos[linea] == UNIPOLAR10) b |= 0x50;
    else if (rangos[linea] == UNIPOLAR5) b |= 0x40;
    // Pone en el byte b determinados bits a 1 dependiendo del rango de medida

    pESDigital->escribe8Bits(D0, b);  // Envía la línea y el rango por el bus de datos

    pESDigital->escribeBit(WR, 0);
    pESDigital->escribeBit(WR, 1);
    // Pulso a nivel bajo en la señal WR para transferir el dato al convertidor

    pESDigital->configura(D0, D0 + 7, 1);  // Configura el bus de datos como entrada

    pESDigital->escribeBit(HBEN, 0);
    // Pone HBEN=0 para recibir los 8 bits menos significativos de la medida

    while (pESDigital->leeBit(INT));  // Espera a que finalice la medida

    pESDigital->escribeBit(RD, 0);  // Señal RD=0 para recibir los datos
    int16_t lectura = pESDigital->lee8Bits(D0);  // Recoge los 8 bits menos significativos

    pESDigital->escribeBit(HBEN, 1);  // Pone HBEN=1 para recibir los 4 bits más significativos
    lectura |= pESDigital->lee8Bits(D0) << 8;  // Los 4 bits más significativos y los pone en su lugar
```

```
    float resultado = 0;
    if (rangos[linea] == BIPOLAR10) resultado = lectura * 10.0f / 2048;
    else if (rangos[linea] == BIPOLAR5) resultado = lectura * 5.0f / 2048;
    else if (rangos[linea] == UNIPOLAR10) resultado = lectura * 10.0f / 4096;
    else if (rangos[linea] == UNIPOLAR5) resultado = lectura * 5.0f / 4096;
    // Obtiene la tensión medida en función del rango configurado en la línea

    pESDigital->escribeBit(RD, 1);  // Pone RD=1 para finalizar la lectura de datos

    return resultado;  // Devuelve la tensión medida por el convertidor
}
```

Fichero **principal.cpp**:

```
#include "esdigital.h"  // Para utilizar la clase ESDigital
#include "max197.h"  // Para utilizar la clase MAX197


int main() {

    ESDigital esDigital((uint8_t*)0xF80, (uint8_t*)0xF92);
    // Objeto para manejar las E/S digitales. Los SFR de E/S comienzan en 0xF80 y
    // los de configuración de las líneas en 0xF92.

    MAX197 convertidorAD(26, 24, 25, 27, 5, &esDigital);
    // Objeto para manejar la conversión A/D mediante un MAX197. El convertidor se
    // conecta a las siguientes señales digitales del microcontrolador: WR=26, RD=24,
    // HBEN=24, INT=27, D0=5.

    convertidorAD.configura(2, MAX197::RANGO::BIPOLAR10);
    // Configura la entrada analógica 2 para que trabaje con un rango de -10V a 10V

    convertidorAD.configura(5, MAX197::RANGO::BIPOLAR5);
    // Configura la entrada analógica 2 para que trabaje con un rango de -5V a 5V

    float m[2];
    convertidorAD.mide(m);
    // Mide las dos entradas analógicas configuradas y guarda los valores en la matriz m.

    float medida = convertidorAD.mide(2);
    // Mide la entrada analógica 2

    return 0;
}
```

---

ej13) Codifica una clase 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 $\overline{CLR}$ 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. 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**:

```
#pragma once

#include "esdigital.h"
```

```
class AD5725 {

    ESDigital* pESDigital;  // Puntero al objeto de manejo de la E/S digital
    int A0, A1, RW, CS, LDAC, DB0;  // Líneas digitales donde se conectó el convertidor

    void escribeBusDatos(uint16_t dato);  // Establece los 12 bits del bus de datos

    float valores[4];  // Recuerda los valores de las 4 salidas analógicas

public:

    AD5725(int A0, int A1, int RW, int CS, int LDAC, int DB0, ESDigital* p);
    // Constructor al que se le indica en qué señales digitales del microcontrolador se han
    // conectado las señales A0, A1, RW, CS, LDAC y DB0 del convertidor. El resto de señales del
    // bus de datos del convertidor se conectan a señales digitales consecutivas, a partir de
    // DB0.

    void salida(float AD0, float AD1, float AD2, float AD3);
    // Actualiza las cuatro salidas analógicas de forma simultánea según los valores pasados
    // por parámetro. Se comunica esta información al convertidor según se indica en la
    // figura 5 de su manual.

    void salida(int nSalida, float valor);
    // Modifica la salida analógica indicada en el primer parámetro (0 a 3) con el valor de
    // tensión indicado en el segundo parámetro
};
```

Fichero `ad5725.cpp`:

```
#include "ad5725.h"

AD5725::AD5725(int A0, int A1, int RW, int CS, int LDAC, int DB0, ESDigital* pESDigital) {
    // Constructor al que se le indica en qué señales digitales del microcontrolador se han
    // conectado las señales A0, A1, RW, CS, LDAC y DB0 del convertidor. El resto de señales del
    // bus de datos del convertidor se conectan a señales digitales consecutivas, a partir de DB0.

    this->A0 = A0;
    this->A1 = A1;
    this->RW = RW;
    this->CS = CS;
    this->DB0 = DB0;
    this->LDAC = LDAC;
    this->pESDigital = pESDigital;
    // Guarda los parámetros en miembros del objeto

    pESDigital->configura(A0, 0);
    pESDigital->escribeBit(A0, 0);
    pESDigital->configura(A1, 0);
    pESDigital->escribeBit(A1, 0);
    pESDigital->configura(CS, 0);
    pESDigital->escribeBit(CS, 1);
    pESDigital->configura(RW, 0);
    pESDigital->escribeBit(RW, 1);
    pESDigital->configura(LDAC, 0);
    pESDigital->escribeBit(LDAC, 1);
    for(int i = 0; i < 12; i++)
        pESDigital->configura(DB0 + i, 0);
    // Configura las salidas digitales y las inicializa

    for (int i = 0; i < 4; i++)
        valores[i] = 0.0f;
    salida(0, 0, 0, 0);
    // Inicializa las salidas analógicas a 0 V.
}
```

```
void AD5725::escribeBusDatos(uint16_t dato) {
    // Establece valores en el bus de datos de 12 bits para comunicarle al convertidor
    // un valor para una salida analógica

    for (int i = 0; i < 12; i++)
        pESDigital->escribeBit(DB0 + i, dato & (1 << i));
}


void AD5725::salida(float AD0, float AD1, float AD2, float AD3) {
    // Actualiza las cuatro salidas analógicas de forma simultánea según los valores pasados
    // por parámetro. Se comunica esta información al convertidor según se indica en la
    // figura 5 de su manual.

    pESDigital->escribeBit(RW, 0);
    pESDigital->escribeBit(A0, 0);
    pESDigital->escribeBit(A1, 0);
    escribeBusDatos((uint16_t)(4096.0f * AD0 / 5.0f));
    pESDigital->escribeBit(CS, 0);
    pESDigital->escribeBit(CS, 1);
    escribeBusDatos((uint16_t)(4096.0f * AD1 / 5.0f));
    pESDigital->escribeBit(A0, 1);
    pESDigital->escribeBit(A1, 0);
    pESDigital->escribeBit(CS, 0);
    pESDigital->escribeBit(CS, 1);
    escribeBusDatos((uint16_t)(4096.0f * AD2 / 5.0f));
    pESDigital->escribeBit(A0, 0);
    pESDigital->escribeBit(A1, 1);
    pESDigital->escribeBit(CS, 0);
    pESDigital->escribeBit(CS, 1);
    escribeBusDatos((uint16_t)(4096.0f * AD3 / 5.0f));
    pESDigital->escribeBit(A0, 1);
    pESDigital->escribeBit(A1, 1);
    pESDigital->escribeBit(CS, 0);
    pESDigital->escribeBit(LDAC, 0);
    pESDigital->escribeBit(CS, 1);
    pESDigital->escribeBit(LDAC, 1);
    pESDigital->escribeBit(RW, 1);
    valores[0] = AD0;
    valores[1] = AD1;
    valores[2] = AD2;
    valores[3] = AD3;
}

void AD5725::salida(int nSalida, float valor) {
    // Modifica la salida analógica indicada en el primer parámetro (0 a 3) con el valor de
    // tensión indicado en el segundo parámetro

    valores[nSalida] = valor;
    salida(valores[0], valores[1], valores[2], valores[3]);
}
```

Un programa que utiliza esta clase:

```
#include "ad5725.h"

int main() {

    ESDigital esDigital((uint8_t*)0xF80, (uint8_t*)0xF92);
    // Objeto para manejar las E/S digitales. Los SFR de E/S comienzan en 0xF80 y
    // los de configuración de las líneas en 0xF92.

    AD5725 convertidorDA(30, 31, 32, 33, 34, 35, &esDigital);
    // Objeto para manejar un convertidor DA AD5725. Se indica a qué líneas digitales
    // del microcontrolador están conectadas las líneas
```

```
    convertidorDA.salida(0.0, 1.0, 1.0, 2.0);
    // Establece la tensión en las 4 salidas analógicas del convertidor

    convertidorDA.salida(2, 1.5);
    // Establece 1.5 V en la salida analógica 2

    return 0;
}
```

Adaptación de la clase `AD5725` para que pueda manejar el convertidor DA con cualquier rango de tensión de salida:

Fichero `ad5725.h`:

```
#pragma once
#include "esdigital.h"

class AD5725 {

    ESDigital* pESDigital;  // Puntero al objeto de manejo de la E/S digital
    int A0, A1, RW, CS, LDAC, DB0;  // Líneas digitales donde se conectó el convertidor
    float vMin, vMax;  // Rango de tensión analógica
    void escribeBusDatos(uint16_t dato);  // Establece los 12 bits del bus de datos

    float valores[4];  // Recuerda los valores de las 4 salidas analógicas

public:

    AD5725(int A0, int A1, int RW, int CS, int LDAC, int DB0, float vMin, float vMax, ESDigital* p);
    // Constructor al que se le indica en qué señales digitales del microcontrolador se han
    // conectado las señales A0, A1, RW, CS, LDAC y DB0 del convertidor. El resto de señales del
    // bus de datos del convertidor se conectan a señales digitales consecutivas, a partir de
    // DB0. Se indica el rango de tensión analógica de salida, entre vMin y vMax.

    void salida(float AD0, float AD1, float AD2, float AD3);
    // Actualiza las cuatro salidas analógicas de forma simultánea según los valores pasados
    // por parámetro. Se comunica esta información al convertidor según se indica en la
    // figura 5 de su manual.

    void salida(int nSalida, float valor);
    // Modifica la salida analógica indicada en el primer parámetro (0 a 3) con el valor de
    // tensión indicado en el segundo parámetro
};
```

Fichero `ad5725.cpp`:

```
#include "ad5725.h"

AD5725::AD5725(int A0, int A1, int RW, int CS, int LDAC, int DB0, float vMin, float vMax,
    ESDigital* pESDigital) {
    // Constructor al que se le indica en qué señales digitales del microcontrolador se han
    // conectado las señales A0, A1, RW, CS, LDAC y DB0 del convertidor. El resto de señales del
    // bus de datos del convertidor se conectan a señales digitales consecutivas, a partir de DB0.
    // Se indica el rango de tensión analógica de salida, entre vMin y vMax.

    this->A0 = A0;
    this->A1 = A1;
    this->RW = RW;
    this->CS = CS;
    this->DB0 = DB0;
    this->LDAC = LDAC;
    this->vMin = vMin;
    this->vMax = vMax;
    this->pESDigital = pESDigital;
    // Guarda los parámetros en miembros del objeto
```

```
    pESDigital->configura(A0, 0);
    pESDigital->escribeBit(A0, 0);
    pESDigital->configura(A1, 0);
    pESDigital->escribeBit(A1, 0);
    pESDigital->configura(CS, 0);
    pESDigital->escribeBit(CS, 1);
    pESDigital->configura(RW, 0);
    pESDigital->escribeBit(RW, 1);
    pESDigital->configura(LDAC, 0);
    pESDigital->escribeBit(LDAC, 1);
    for(int i = 0; i < 12; i++)
        pESDigital->configura(DB0 + i, 0);
    // Configura las salidas digitales y las inicializa

    salida(0, 0, 0, 0);
    // Inicializa las salidas analógicas a 0 V.
}


void AD5725::escribeBusDatos(uint16_t dato) {
    // Establece valores en el bus de datos de 12 bits para comunicarle al convertidor
    // un valor para una salida analógica

    for (int i = 0; i < 12; i++)
        pESDigital->escribeBit(DB0 + i, dato & (1 << i));
}


void AD5725::salida(float AD0, float AD1, float AD2, float AD3) {
    // Actualiza las cuatro salidas analógicas de forma simultánea según los valores pasados
    // por parámetro. Se comunica esta información al convertidor según se indica en la
    // figura 5 de su manual.

    pESDigital->escribeBit(RW, 0);
    pESDigital->escribeBit(A0, 0);
    pESDigital->escribeBit(A1, 0);
    escribeBusDatos((uint16_t)(4096.0f * (AD0 - vMin) / (vMax - vMin)));
    pESDigital->escribeBit(CS, 0);
    pESDigital->escribeBit(CS, 1);
    pESDigital->escribeBit(A0, 1);
    pESDigital->escribeBit(A1, 0);
    escribeBusDatos((uint16_t)(4096.0f * (AD1 - vMin) / (vMax - vMin)));
    pESDigital->escribeBit(CS, 0);
    pESDigital->escribeBit(CS, 1);
    pESDigital->escribeBit(A0, 0);
    pESDigital->escribeBit(A1, 1);
    escribeBusDatos((uint16_t)(4096.0f * (AD2 - vMin) / (vMax - vMin)));
    pESDigital->escribeBit(CS, 0);
    pESDigital->escribeBit(CS, 1);
    pESDigital->escribeBit(A0, 1);
    pESDigital->escribeBit(A1, 1);
    escribeBusDatos((uint16_t)(4096.0f * (AD3 - vMin) / (vMax - vMin)));
    pESDigital->escribeBit(CS, 0);
    pESDigital->escribeBit(LDAC, 0);
    pESDigital->escribeBit(CS, 1);
    pESDigital->escribeBit(LDAC, 1);
    pESDigital->escribeBit(RW, 1);
    valores[0] = AD0;
    valores[1] = AD1;
    valores[2] = AD2;
    valores[3] = AD3;
}
```

```
void AD5725::salida(int nSalida, float valor) {
    // Modifica la salida analógica indicada en el primer parámetro (0 a 3) con el valor de
    // tensión indicado en el segundo parámetro

    valores[nSalida] = valor;
    salida(valores[0], valores[1], valores[2], valores[3]);
}
```

---

ej63) Una vez se han desarrollado funciones en lenguaje C para el manejo de una pantalla LCD en una placa de desarrollo STM32F429I-DISC1:

```
typedef struct {
    uint16_t ancho, alto;  // Ancho y alto de la pantalla en puntos
    uint32_t* buffers[2];  // Direcciones de memoria donde residen los frame buffers
    int bufferActual;  // Número de buffer donde se dibuja, el 0 o el 1
} PantallaLCD;
// Estructura con información sobre la pantalla LCD

void inicializaPantalla1Buffer(uint32_t colorFondo, PantallaLCD* pantallaLCD);
// Inicializa la pantalla LCD para manejarla utilizando un único frame buffer. Cubre
// la pantalla con el color indicado por parámetro. Guarda información sobre la
// pantalla en la estructura de datos apuntada por 'pantallaLCD'.

void dibujaPunto(uint16_t x, uint16_t y, uint32_t color, PantallaLCD * pPantalla);
// Dibuja un punto en las coordenadas (x, y) con el color indicado

void dibujaLinea(int x0, int y0, int x1, int y1, uint32_t color, PantallaLCD *pPantalla);
// Dibuja una línea desde el punto (x0, y0) hasta el punto (x1, y1) con el color indicado

void dibujaCirculo(int xc, int yc, int r, uint32_t color, PantallaLCD * pPantalla);
// Dibuja un círculo con centro en (xc, yc) con el color indicado

void dibujaTexto(uint16_t x, uint16_t y, const char * texto, uint32_t color,
    const uint8_t * juegoCaracteres, PantallaLCD * pantallaLCD);
// Dibuja el texto apuntado por 'texto' a partir de la posición (x, y) utilizando un color
// y un juego de caracteres.
```

codifica clases en C++ para realizar esas operaciones utilizando programación orientada a objetos y definiendo una clase base abstracta donde recoger las acciones comunes a los diferentes elementos gráficos que se van a representar en la pantalla: acción de dibujado y acción de traslación utilizando para ello las coordenadas de un vector.

Una solución:

En el proyecto añadimos los archivos **`dibujo.hpp`** y **`dibujo.cpp`** para la declaración y definición de las clases.

Archivo **`dibujo.hpp`**:

```
#ifndef INC_DIBUJO_HPP_
#define INC_DIBUJO_HPP_

#include "pantalla.h"
// Para utilizar el tipo PantallaLCD y las funciones de inicialización y
// dibujo en la pantalla, desarrolladas en lenguaje C

#include <cstdint>  // Para uint16_t, uint32_t

// ----------------------------------------------------------------------------
```

```
class Vector {
    // Clase para representar a un vector utilizado para trasladar la posición
    // un punto en pantalla.

public:
    int x, y;  // Coordenadas del vector
    Vector(int x, int y);  // Constructor para crear un vector
};

// ----------------------------------------------------------------------

class ElementoGrafico {
    // Clase abstracta con métodos virtuales puros no definidos. Se utiliza como
    // clase base para poder crear otras clases derivadas utilizadas para manejar
    // diferentes tipos de elementos gráficos: puntos, líneas, textos, etc.

public:

    virtual void dibuja(PantallaLCD & pantallaLCD) = 0;
    // Todos los elementos gráficos (objetos de clases derivadas de esta) tienen
    // que implantar este método de dibujado, al que se le pasa por parámetro una
    // referencia a una estructura de tipo PantallaLCD con información sobre la
    // pantalla

    virtual void traslada(Vector & v) = 0;
    // Todos los elementos gráficos tienen que implantar este método de traslación
    // donde se aplica un desplazamiento de posición fijado por un vector cuyas
    // coordenadas se indican en el objeto que se pasa por referencia
};

// ----------------------------------------------------------------------

class Punto : public ElementoGrafico {
    // Cada objeto de esta clase representa a un punto. Es una clase derivada de
    // ElementoGrafico y tiene que implantar los métodos de dibujado y traslación.

protected:  // Datos protegidos para que sean accesibles a métodos de esta clase
            // y de clases derivadas

    uint16_t x, y;  // Posición del punto en pantalla
    uint32_t color;  // Color de dibujado

public:

    Punto(uint16_t x, uint16_t y, uint32_t color = 0xFF000000);
    // Constructor al que se le pasa las coordenadas (x, y) del punto y su
    // color. El color tiene un valor por defecto a color negro.

    void dibuja(PantallaLCD & pantallaLCD) override;
    // Define el método de dibujado, recibido mediante herencia de la clase
    // base ElementoGrafico. Dibuja el punto en la pantalla representada por la
    // estructura de tipo PantallaLCD pasada por referencia.

    void traslada(Vector & v) override;
    // Define el método de traslación recibido mediante herencia de la clase
    // base ElementoGrafico. Suma las coordenadas del vector pasado por
    // referencia.

};

// ----------------------------------------------------------------------

class Linea: public Punto {
    // Un objeto de esta clase representa a una línea. Recibe mediante herencia de
    // la clase Punto el punto origen (x, y) de la línea y en esta clase se añaden
    // las coordenadas (x2, y2) del punto final de la línea. También recibe mediante
    // herencia el color de la línea.

    uint16_t x2, y2;  // Punto final de la línea
```

```
public:
    Linea(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2, uint32_t color);
    // Constructor para crear un objeto que represente a una línea que va del punto
    // (x1, y1) al punto (x2, y2) indicando también su color.

    void dibuja(PantallaLCD & p) override;
    // Redefine el método de dibujado recibido mediante herencia desde la clase
    // base Punto. Dibuja la línea en la pantalla representada por la
    // estructura de tipo PantallaLCD pasada por referencia.


    void traslada(Vector & v) override;
    // Redefine el método de traslación recibido mediante herencia desde la clase
    // Punto. Aplica a la línea una traslación definida por un vector indicado en
    // el objeto punto pasado por referencia.
};

// ----------------------------------------------------------------------------

class Circulo: public Punto {
    // Cada objeto de esta clase representa a un círculo. Es una clase derivada
    // de la clase Punto para recibir mediante herencia las coordenadas del
    // centro del círculo y su color. También recibe mediante herencia el método
    // de traslación, que tal y como se define en la clase Punto, es válido
    // para esta clase derivada Circulo.

    int radio;  // En la clase Circulo se añade el radio

public:

    Circulo(uint16_t x, uint16_t y, uint16_t radio, uint32_t color);
    // Constructor que crea un círculo de centro en las coordenadas (x, y) y con
    // un cierto radio y color.

    void dibuja(PantallaLCD & pantallaLCD) override;
    // Redefine el método de dibujado recibido mediante herencia desde la clase
    // Punto. Dibuja el círculo en la pantalla representada por la
    // estructura de tipo PantallaLCD pasada por referencia.
};

// ----------------------------------------------------------------------------

class Texto: public Punto {
    // Cada objeto de esta clase representa a una cadena de caracteres que se
    // muestra en pantalla. Recibe mediante herencia de la clase Punto la posición
    // (x, y) del texto y su color. También recibe mediante herencia el método de
    // traslación de la clase Punto que es válido para esta clase Texto.

    const char * texto;  // Puntero a la cadena de caracteres
    const uint8_t * juegoCaracteres;  // Matriz que define el juego de caracteres

public:

    Texto(uint16_t x, uint16_t y, const char * texto, uint32_t color,
        const uint8_t * juegoCaracteres);
    // Constructor al que se le pasa la posición (x, y) del texto, un puntero a la
    // cadena de caracteres, su color y la matriz donde se define el juego de caracteres.

    void dibuja(PantallaLCD & pantallaLCD) override;
    // Redefine el método de dibujado recibido de la clase Punto, para dibujar el
    // texto en la pantalla representada por la estrcutura de tipo PantallaLCD
    // pasada por referencia.
};

// ----------------------------------------------------------------------------
```

```
class Dibujo: public ElementoGrafico {
    // Un objeto de esta clase representa a un conjunto de elementos gráficos (objetos
    // de clases derivadas de la clase ElementoGrafico) que se manejan en bloque.
    // Un dibujo es a su vez un elemento gráfico, ya que la clase Dibujo es derivada
    // de la clase ElementoGrafico, por lo que un dibujo puede contener a otros
    // dibujos.

    ElementoGrafico ** ppElementos;
    // Puntero a una matriz de punteros a objetos de clases derivadas de ElementoGrafico.
    // De esta forma, en un dibujo se pueden cargar direcciones de objetos de diferentes
    // clases como Punto, Linea, Texto, etc.

    int nElementos;  // Número de elementos gráficos cargados en el dibujo

public:

    Dibujo(uint16_t tamano);
    // Constructor que crea un dibujo en el que se pueden cargar hasta 'tamano'
    // elementos gráficos

    void carga(ElementoGrafico * p);
    // Carga un nuevo elemento gráfico en el dibujo. Se pasa por parámetro un puntero
    // al objeto que representa ese elemento gráfico. Puede ser cualquier objeto de cualquier
    // clase derivada de ElementoGrafico.

    void dibuja(PantallaLCD & p) override;
    // Redefine el método de dibujado recibido mediante herencia desde la clase base
    // ElementoGrafico. Con este método se dibujan todos los elementos gráficos cargados en
    // el dibujo.

    void traslada(Vector & v) override;
    // Redefine el método de traslación recibido mediante herencia desde la clase base
    // ElementoGrafico. Con este método se trasladan todos los elementos gráficos cargados en
    // el dibujo aplicando el vector de traslación pasado por referencia.
};


#endif /* INC_DIBUJO_HPP_ */
```

Archivo **dibujo.cpp**:

```
#include <dibujo.hpp>

// -----------------------------------------------------------------

Vector::Vector(int x, int y) {  // Constructor para crear un vector
    this->x = x;  // Guarda la coordenada x en el objeto creado
    this->y = y;  // Guarda la coordenada y en el objeto creado
}

// -----------------------------------------------------------------

Punto::Punto(uint16_t x, uint16_t y, uint32_t color) {
    // Constructor al que se le pasa las coordenadas (x, y) del punto y su
    // color. El color tiene un valor por defecto a color negro.

    this->x = x;  // Guarda la coordenada X
    this->y = y;  // Guarda la coordenada Y
    this->color = color;  // Guarda el color
}


void Punto::dibuja(PantallaLCD & pantallaLCD) {
    // Define el método de dibujado, recibido mediante herencia de la clase
    // base ElementoGrafico. Dibuja el punto en la pantalla representada por la
    // estructura de tipo PantallaLCD pasada por referencia.
```

```
    dibujaPunto(x, y, color, &pantallaLCD);
    // Llama a la funcion dibujaPunto() definida en lenguaje C pasándole las coordenadas
    // del punto, su color y un puntero a la estructura.
}


void Punto::traslada(Vector & v) {
    // Define el método de traslación recibido mediante herencia de la clase
    // base ElementoGrafico. Suma las coordenadas del vector pasado por
    // referencia.

    x += v.x;  // Acumula la coordenada X del vector en la coordenada X del punto
    y += v.y;  // Acumula la coordenada Y del vector en la coordenada Y del punto
}

// ----------------------------------------------------------------

Linea::Linea(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2, uint32_t color): Punto(x1,y1,color) {
    // Constructor para crear un objeto que represente a una línea que va del punto
    // (x1, y1) al punto (x2, y2) indicando también su color. Llama al constructor
    // de la clase base Punto para inicializar las coordenadas del punto inicial (x1, y1) y
    // el color de la línea

    this->x2 = x2;  // Guarda la coordenada X del punto final de la línea
    this->y2 = y2;  // Guarda la coordenada Y del punto final de la línea
    // En el constructor de la clase derivada Linea se añaden las instrucciones
    // necesarias para inicializar los datos adicionales añadidos en esa clase.
}


void Linea::dibuja(PantallaLCD & pantallaLCD) {
    // Redefine el método de dibujado recibido mediante herencia desde la clase
    // base Punto. Dibuja la línea en la pantalla representada por la
    // estructura de tipo PantallaLCD pasada por referencia.

    dibujaLinea(x, y, x2, y2, color, &pantallaLCD);
    // Llama a la función dibujaLinea() definida en lenguaje C, pasándole las coordenadas
    // del punto inicial (x, y), el punto final (x2, y2), el color y un puntero a la
    // estructura con datos sobre la pantalla.
}


void Linea::traslada(Vector & v) {
    // Redefine el método de traslación recibido mediante herencia desde la clase
    // Punto. Aplica a la línea una traslación definida por un vector indicado en
    // el objeto punto pasado por referencia.

    Punto::traslada(v);
    // Llama al método de traslación de la clase Punto para aplicar el vector de
    // traslación v al punto inicial (x, y) de la línea

    x2 += v.x;  // Traslada la coordenada X del punto final de la línea
    y2 += v.y;  // Traslada la coordenada Y del punto final de la línea
}

// ----------------------------------------------------------------

Circulo::Circulo(uint16_t x, uint16_t y, uint16_t radio, uint32_t color) : Punto(x, y, color) {
    // Constructor que crea un círculo de centro en las coordenadas (x, y) y con
    // un cierto radio y color. Llama al constructor de la clase base Punto para inicializar
    // la posición del círculo (x, y) y su color.

    this->radio = radio;  // Guarda el radio del círculo
}


void Circulo::dibuja(PantallaLCD & pantallaLCD) {
    // Redefine el método de dibujado recibido mediante herencia desde la clase
    // Punto. Dibuja el círculo en la pantalla representada por la
    // estructura de tipo PantallaLCD pasada por referencia.
```

```
    dibujaCirculo(x, y, radio, color, &pantallaLCD);
    // Llama a la función dibujaCirculo() definida en lenguaje C pasándole las coordenadas
    // (x, y) del centro, el radio y un puntero a la estructura con datos de la pantalla.
}

// ------------------------------------------------------------------

Texto::Texto(uint16_t x, uint16_t y, const char * texto, uint32_t color,
    const uint8_t * juegoCaracteres) : Punto(x, y, color) {
    // Constructor al que se le pasa la posición (x, y) del texto, un puntero a la
    // cadena de caracteres, su color y la matriz donde se define el juego de caracteres.
    // Llama al constructor de la clase base Punto para inicializar
    // la posición del texto (x, y) y su color.

    this->texto = texto;  // Guarda en el objeto un puntero a la cadena

    this->juegoCaracteres = juegoCaracteres;
    // Guarda la dirección de la matriz que describe el juego de caracteres empleado
}


void Texto::dibuja(PantallaLCD & pantallaLCD) {
    // Redefine el método de dibujado recibido de la clase Punto, para dibujar el
    // texto en la pantalla representada por la estrcutura de tipo PantallaLCD
    // pasada por referencia.

    dibujaTexto(x, y, texto, color, juegoCaracteres, &pantallaLCD);
    // Llama a la función dibujaTexto() definida en lenguaje C pasándole las coordenadas
    // (x, y) del centro, el texto, el color, la matriz donde se define el juego de
    // caracteres y un puntero a la estructura con datos de la pantalla.
}

// ------------------------------------------------------------------

Dibujo::Dibujo(uint16_t tamano) {
    // Constructor que crea un dibujo en el que se pueden cargar hasta 'tamano'
    // elementos gráficos

    ppElementos = new ElementoGrafico* [tamano];
    // Crea una matriz de punteros a los elementos gráficos que se van a cargar en el dibujo.
    // El tamaño de esa matriz se indica por parámetro.

    nElementos = 0;  // Inicialmente no hay ningún elemento gráfico cargado en el dibujo.
}


void Dibujo::carga(ElementoGrafico * elemento) {
    // Carga un nuevo elemento gráfico en el dibujo. Se pasa por parámetro puntero
    // al objeto que representa ese elemento gráfico. Puede ser cualquier objeto de cualquier
    // clase derivada de ElementoGrafico.

    ppElementos[nElementos] = elemento;
    // Copia en la matriz de punteros la dirección del elemento gráfico indicado por parámetro

    nElementos ++;  // Incrementa el contador de elementos gráficos cargados en el dibujo
}


void Dibujo::dibuja(PantallaLCD & lcd) {
    // Redefine el método de dibujado recibido mediante herencia desde la clase ElementoGrafico.
    // Con este método se dibujan todos los elementos gráficos cargados en el dibujo.

    for(int i = 0; i < nElementos; i++)
       ppElementos[i]->dibuja(lcd);
    // Recorre en un bucle la matriz de punteros a los elementos gráficos y en cada uno de
    // ellos ejecuta la llamada al método dibuja(). En esta instrucción se utiliza polimorfismo,
    // ya que la llamada al método dibuja() va a ejecutar diferente código dependiendo del tipo
    // de elemento gráfico (Punto, Linea, Circulo, etc).
}
```

```
void Dibujo::traslada(Vector & v) {
    // Redefine el método de traslación recibido mediante herencia desde la clase base
    // ElementoGrafico. Con este método se trasladan todos los elementos gráficos cargados en
    // el dibujo aplicando el vector de traslación pasado por referencia.

    for(int i = 0; i < nElementos; i++)
       ppElementos[i]->traslada(v);
    // Recorre en un bucle la matriz de punteros a los elementos gráficos y en cada uno de
    // ellos ejecuta la llamada al método traslada() para trasladarlos según las coordenadas
    // del vector referenciado por v. La llamada a traslada() es polimórfica porque ejecuta
    // código diferente según el tipo de elemento que se está trasladando (Punto, Línea, etc.)
}
```

El programa principal tiene que utilizar objetos de estas clases, por lo que es necesario renombrar el archivo `main.c` a `main.cpp` para que se utilice un compilador de lenguaje C++. En este archivo, en la función que define la tarea de manejo de la pantalla podríamos ejecutar instrucciones como las siguientes:

```
PantallaLCD pantallaLCD;  // Estructura para los datos de la pantalla

inicializaPantalla1Buffer(0xFF0000FF, &pantallaLCD);  // Inicializa la pantalla en azul

Punto p1(10, 10, 0xFFFFFFFF);
// Objeto que representa a un punto en (10, 10) y color blanco

Linea l1(10, 10, 20, 20, 0xFFFFFF00);
// Línea de (10, 10) a (20, 20) en amarillo

Circulo c1(100, 100, 30, 0xFFFFFFFF);
// Círculo de centro (100, 100), radio 30 y color blanco

Dibujo d1(10);  // Dibujo donde se pueden acumular hasta 10 elementos gráficos
d1.carga(&l1);  // Carga la línea en el dibujo
d1.carga(&c1);   // carga el círculo en el dibujo
d1.carga (new Punto(50, 50, 0xFFFFFFFF));  // Carga un punto en el dibujo
d1.carga (new Texto(100, 100, "Temperatura = 23", 0xFFFFFFFF, juego8x11));  // Carga un texto

Dibujo d2(5);  // Crea otro dibujo donde se pueden acumular hasta 5 elementos gráficos
d2.carga(&d1);  // Carga el primer dibujo
d2.carga(new Linea(10, 10, 20, 20, 0xFF00000));  // Carga una línea

Vector t(10, 0);  // Vector de traslación para mover elementos gráficos 10 puntos en eje X
d2.traslada(t);   // Traslada el segundo dibujo
d2.dibuja(pantallaLCD);  // Muestra en pantalla el segundo dibujo
```

Ajuste parámetros de controladores

ej14) Codifica una subrutina para un controlador digital PID industrial gobernado por un microcontrolador con el objetivo de realizar el ajuste automático (autotuning) de sus parámetros K_p, K_i, K_d. El regulador dispone de dos entradas analógicas. En la primera entrada se le suministra el valor de consigna $r(t)$. La segunda entrada se conecta la salida de la planta $y(t)$. A partir de ambas el regulador calcula la señal de error $e(t) = r(t) - y(t)$ de seguimiento de la consigna. Mediante un algoritmo PID el controlador calcula la actuación sobre la planta $u(t)$ que se lleva a una salida analógica. Todas las señales tienen un rango de -10 V a 10 V.

$$u(t) = K_p e(t) + K_i \int_{\tau=0}^{t} e(\tau) d\tau + K_d \dot{e}(t)$$

El autoajuste de los parámetros K_p, K_i, K_d se realizará mediante el método del relé, utilizando el siguiente procedimiento:

- Se supone que el sistema parte de un estado inicial en el que todas las señales están a 0 V antes de que se llame a esta subrutina.

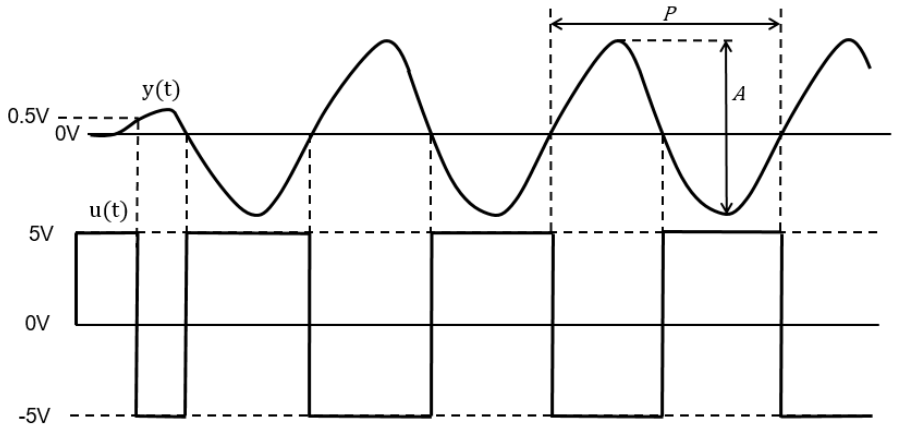
- Primero se aplicará una actuación de 5V hasta que la salida de la planta alcance 0.5 V.

- A continuación el controlador se convierte en un relé que genera una actuación de 5V si la salida de la planta es negativa y que genera -5 V si la salida de la planta es positiva.

- El proceso de autoajuste finaliza cuando la oscilación en la salida de la planta se estabiliza presentando una amplitud A y período P relativamente constantes en períodos de oscilación sucesivos. Se considera que cualquiera de estos parámetros se mantiene relativamente constante cuando los 5 valores obtenidos de A y P en los últimos 5 períodos difieren de la media de esos 5 valores en menos del 2% de la media. El valor final de A o P corresponde a la media de esos 5 últimos valores de A o P.

- Una vez determinados A y P, se aplican las siguientes fórmulas para calcular los parámetros del PID, donde es $D = 5$ la amplitud de la actuación aplicada en este proceso de autoajuste

$$K_u = \frac{4D}{\pi A}, \quad K_p = 0.6K_u \quad K_i = 1.2\frac{K_u}{P} \quad K_d = 0.075K_u$$



Supón que la señal de entrada analógica está filtrada y no presenta ningún ruido apreciable.
El microcontrolador dispone de un temporizador que se ha programado previamente y que incrementa automáticamente a cada milisegundo la variable global `MS` de tipo `unsigned int`. Esta variable se puede leer y escribir.
Se dispone de la función `double entradaAnalogica(int nEntrada)` que devuelve el resultado de la medición en V de la entrada analógica indicada por parámetro (0=referencia, 1=salida de la planta).
También se dispone de la función `void salidaAnalogica(double valor)` a la que podemos pasar el valor que queremos establecer en la salida analógica.

Una solución:

```
#include <cmath>
#define PI 3.14159265358979238463

void autoajustePID(float* Kp, float* Ki, float* Kd) {

    salida(5);
    while (entrada(0) < 0.5);
    salida(-5);

    float P[5], A[5];
    float minimo = 10, maximo = -10;
```

```
    int nPeriodos = 0;

    while (entrada(0) > 0);
    salida(5);
    while (entrada(0) < 0);

    bool finalizar = false;
    double medidaAnterior = entrada(0);
    double medida;

    MS = 0;
    float mediaA = 0, mediaP = 0;

    while (!finalizar) {

        medida = entrada(0);
        if (medida < minimo) minimo = medida;
        if (medida > maximo) maximo = medida;

        if (medidaAnterior < 0 && medida >= 0) {

            if (nPeriodos == 5)
                for (int i = 0; i <= 3; i++) {
                    A[i] = A[i + 1];
                    P[i] = P[i + 1];
                }
            else nPeriodos++;

            P[nPeriodos-1] = MS;
            MS = 0;

            A[nPeriodos-1] = maximo - minimo;
            maximo = -10;
            minimo = 10;

            if (nPeriodos == 5) {

                mediaA = 0;
                mediaP = 0;

                for (int i = 0; i < 5; i++) {
                    mediaA += A[i];
                    mediaP += P[i];
                }
                mediaA /= 5;
                mediaP /= 5;

                bool estable = true;
                for (int i = 0; i < 5; i++) {
                    if (fabs((A[i] - mediaA) / mediaA) > 0.02) estable = false;
                    if (fabs((P[i] - mediaP) / mediaP) > 0.02) estable = false;
                }
                if (estable) finalizar = true;
            }
        }
    }

    float Ku = 20 / PI / mediaA;
    *Kp = 0.6 * Ku;
    *Ki = 1.2 * Ku / mediaP;
    *Kd = 0.075 * Ku;
}
```

Se utilizaría en un programa:

```
void main() {
    double Kp, Ki, Kd;

    autoajustePID(&Kp, &Ki, &Kd);
}
```

Controlador y comunicaciones

ej15) Codifica aplicaciones en C++ para dos sistemas embebidos A y B conectados mediante un canal de comunicaciones.

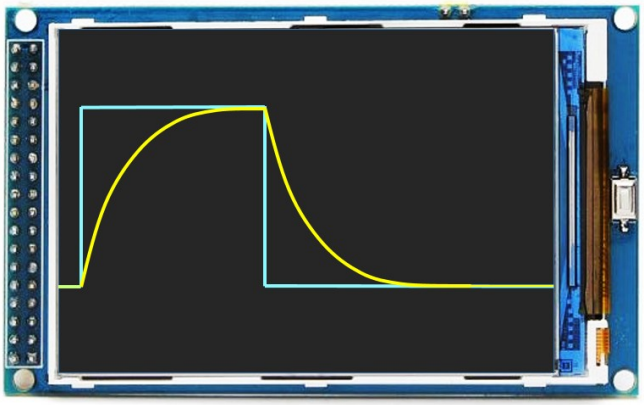
En A se reliza un control PI con período de muestreo de 0.1 s, $K_p = 0.1, K_i = 0.1$. Se controla una planta SISO, utilizando:
- una entrada analógica para medir la salida de la planta mediante la función **`double entrada()`**.
- una salida analógica para fijar la actuación con la función **`void salida(double valor)`**.
- un interruptor que se puede consultar con la función **`int interruptor()`**, que devuelve un buleano cierto si está en posición ON y falso si está en posición OFF. Un operario acciona este interruptor para fijar una consigna para la planta de valor 1 en posición ON y de valor 0 en posición OFF.

En A y B se pueden utilizar las siguientes clases:
- la clase **`Temporizador`** con un constructor al que se le pasa el período de tiempo en ms con el que se inicializa un temporizador que genera eventos contínuamente con ese intervalo. Esta clase dispone del método **`void espera()`** que bloquea hasta que ocurra el siguiente evento de temporización.
- la clase **`Canal`** para manejar las comunicaciones, con 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 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.

El sistema A envía a B contínuamente a cada segundo la consigna y la salida de la planta.

En B existe una pantalla LCD de una resolución de 480 puntos en horizontal por 320 puntos en vertical en la cual se dibujará gráficamente la consigna en azul y la salida de la planta en amarillo. En vertical se representarán valores en el intervalo de -0.5 a 1.5. En horizontal cada punto corresponde a un intervalo de tiempo de un segundo. Para manejar la pantalla se dispone de la clase **`LCD`** con los métodos:
- **`void borra()`** para borrar la pantalla
- **`void punto(int x, int y, int color)`** para colorear el punto de coordenadas (x, y), desde (0, 0) para la esquina inferior izquierda hasta (479, 319) para la esquina superior derecha, en amarillo (indicando en **`color`** el valor **`0xFFFF00`**) o azul (indicando **`0x7FFFFF`**).

Cuando las gráficas ocupen en horizontal toda la pantalla, la imagen tiene que ir desplazándose hacia la izquierda a medida que se añaden más puntos.



Una solución:

```
// Programa para el sistema A

void main() {
    Canal canal;  // Crea un objeto para manejar las comunicaciones
    Temporizador t(100);  // Arranca un temporizador con período de 0.1 s
    int nPeriodos = 0;  // Para contar períodos de temporización
    double consigna, y, u, u1 = 0, e, e1 = 0;  // Para el controlador PI
    while (1) {
        t.espera();  // Espera a la siguiente temporización
        if (interruptor())  // Establece la consigna según el interruptor
            consigna = 1;
        else consigna = 0;
        y = entrada();  // Mide la salida del sistema
        e = consigna - y;  // Error de seguimiento de consigna
        u = u1 + 0.11 * e - 0.01 * e1;  // Algoritmo PI
        salida(u);  // Actúa sobre el sistema
        e1 = e;  // Recuerda el error para el siguiente período
        u1 = u;  // Recuerda la actuación para el siguiente período
        nPeriodos++;
        if (nPeriodos == 10) {  // Si transcurrió 1 segundo ...
            canal.envia(&consigna, sizeof(double));  // Envía la consigna
            canal.envia(&y, sizeof(double));  // Envía la medida en la salida de la planta
            nPeriodos = 0;  // Resetea contador de períodos
        }
    }
}

// Programa para el sistema B

int consignas[480], salidas[480];  // Almacenan las consignas y salidas a dibujar

void main() {
    Canal canal;  // Crea un objeto para manejar las comunicaciones
    LCD lcd;  // Crea un objeto para manejar la pantalla
    double consigna, salida;
    int nPuntos = 0;  // Número de puntos en gráfica
    lcd.borra();  // Borra la pantalla
    while (1) {
        canal.recibe(&consigna);  // Recibe la consigna
        canal.recibe(&salida);  // Recibe la salida de la planta
        if (nPuntos < 480) {  // Si la gráfica no ocupa toda la pantalla ...
            consignas[nPuntos] = (consigna - -0.5) * 319 / 2;  // Escala la señal y guarda
            salidas[nPuntos] = (salida - -0.5) * 319 / 2;  // Escala la señal y guarda
            lcd.punto(nPuntos, consignas[nPuntos], 0x7FFFFF);  // Dibuja nuevo punto de consigna
            lcd.punto(nPuntos, salidas[nPuntos], 0xFFFF00);  // Dibuja nuevo punto de salida
            nPuntos++;  // Hay un nuevo punto en pantalla
        }
        else {  // Si la gráfica ocupa toda la pantalla ...
            lcd.borra();  // Borra la pantalla
            for (int i = 0; i < 480 - 1; i++) {  // Desplaza todas las señales en las matrices
                consignas[i] = consignas[i + 1];  // una posición y elimina la primera medida
                salidas[i] = salidas[i + 1];
            }
            consignas[479] = (consigna - -0.5) * 319 / 2;  // Añade las nuevas medidas
            salidas[479] = (salida - -0.5) * 319 / 2;      // al final
            for (int i = 0; i < 480; i++) {  // Dibuja los valores guardados
                lcd.punto(i, consignas[i], 0x7FFFFF);
                lcd.punto(i, salidas[i], 0xFFFF00);
            }
        }
    }
}
```

ej16) Codifica aplicaciones en C++ para dos sistemas embebidos S1 y S2 con sistema operativo en tiempo real multitarea conectados mediante un canal de comunicaciones.

En S1 se dispone de una consola compuesta por un teclado confeccionado con varios pulsadores y una pantalla de cristal líquido alfanumérica. Para su manejo se dispone de la clase `Consola` con un constructor sin parámetros y un método `void introduce(char * cadena)` que espera a que el operario introduzca una cadena de caracteres compuesta por varios dígitos numéricos y un punto decimal y a que el operario pulse el botón verde. Para la conversión de cadena de caracteres a número real se puede utilizar la función `double atof(const char *).`

En S2 se reliza un control PI $(T = 0.1s.,\ K_p = 0.1,\ K_i = 0.1)$ de una planta SISO, utilizando:
- una entrada analógica que se puede medir mediante la función `double entrada()`.
- una salida analógica que se puede establecer con la función `void salida(double valor)`.

Existe la clase `Temporizador` con un constructor al que se le pasa el período de tiempo en ms con el que se inicializa un temporizador que genera eventos contínuamente con ese intervalo. Esta clase dispone del método `void espera()` que bloquea hasta que ocurra el siguiente evento de temporización.

Existe la clase `Canal` para manejar las comunicaciones, con 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 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.

S1 envía el valor de consigna a S2 contínuamente a cada segundo. Un operario podrá modificar en S1 la consigna en cualquier momento y cuantas veces sea necesario utilizando el teclado y la pantalla. La comunicación entre S1 y S2 comienza cuando se introduce un valor de consigna por primera vez en S1. En S2 se realiza el control utilizando la consigna enviada por S1 siempre y cuando se estén recibiendo consignas a cada segundo, si no, hay que llevar la salida del sistema a cero.

Una solución:

$$u_k = K_p e_k + K_i T \sum_{i=0}^{k} e_k$$

$$u_{k-1} = K_p e_{k-1} + K_i T \sum_{i=0}^{k-1} e_k$$

$$u_k = u_{k-1} + K_p(e_k - e_{k-1}) + K_i T e_k$$

$$u_k = u_{k-1} + a e_k - K_p e_{k-1}$$

$$a = K_p + K_i T$$

Para S1:

```
Consola consola;
double consigna;
semaforo sComienzo = 0, sConsigna = 1;
```

```
void hConsola() {
    char cadena[10];
    while (1) {
        consola.introduce(cadena);
        wait(sConsigna);  // Entra en región crítica
        consigna = atof(cadena);  // Convierte a real
        signal(sConsigna);  // Sale de región crítica
        if (primero) {  // Si es la primera consigna
            signal(sComienzo);  // despierta al otro hilo
            primero = 0;  // Ya no es la primera
        }
    }
}

void hComunicacion() {
    wait(sComienzo);  // Espera a primera consigna
    Temporizador t(1000);  // Lanza temporizador
    Canal canal;
    double copiaConsigna;
    while (1) {
        t.espera();  // Espera temporización
        wait(sConsigna);  // Entra en región crítica
        copiaConsigna = consigna;  // Haz una copia de la consigna
        signal(sConsigna);  // Sale de región crítica
        canal.envia(&copiaConsigna, sizeof(double));  // Envía la consignaa S2
    }
}

void main() {
    concurrente {
        hConsola();  // Hilo para manejo de teclado y pantalla
        hComunicacion ();  // Hilo para enviar consigna a cada segundo
    }
}
```

Para S2:

```
semaforo sComienzo = 0, sConsigna = 1;
double consigna;
int nTemporizaciones;

void hComunicacion () {
    Canal canal;
    double consignaRecibida;
    int primero = 1;  // Es la primera vez que se va a recibir la consigna
    while (1) {
        canal.recibe(&consignaRecibida);  // Espera a recibir la consigna
        wait(sConsigna);  // Entra en región crítica
        consigna = consignaRecibida;  // Haz copia de la consigna recibida
        nTemporizaciones = 10;  // Número de tempoerizaciones hasta recibir la siguiente
        signal(sConsigna);  // Sale de región crítica
        if (primero) {  // Si es la primera vez ...
            signal(sComienzo);  // Despierta al hilo de control
            primero = 0;  // Ya no es la primera vez
        }
    }
}

void hControl() {
    wait(sComienzo);  // Espera a la primera consigna
    Temporizador t(100);  // Lanza temporizador de 0.1 s.
    double copiaConsigna, u, u1 = 0, e, e1 = 0;
    while (1) {
        t.espera();  // Espera temporización
        wait(sConsigna);  // Entra en región crítica
```

```
        if (nTemporizaciones > 0) {  // Si no transcurrió 1 s. desde última recepción
            nTemporizaciones--;  // Decrementa contador de temporizaciones
            copiaConsigna = consigna;  // Establece la consigna para el PI
        } else copiaConsigna = 0;  // Si no, hay que llevar la salida del sistema a 0
        signal(sConsigna);  // Sale de región crítica
        e = copiaConsigna - entrada();  // Error de seguimiento de consigna
        u = u1 + 0.11 * e - 0.01 * e1;  // Controlador PI
        salida(u);  // Actuación
        e1 = e;  // Recuerda valores anteriores para el siguiente período
        u1 = u;
    }
}

void main() {
    concurrente {
        hComunicacion();  // Hilo para recibir la consigna a cada segundo
        hControl();  // Hilo para control PI a cada 0.1 s.
    }
}
```

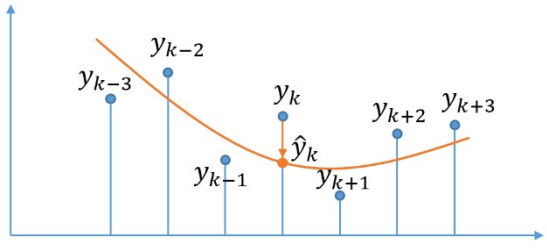
Filtro

ej17) Codifica una función para la aplicación de un filtro Savitzky-Golay sobre una matriz de float donde se guardaron los valores de un cierto número de muestras de una señal medidas con un período fijo, con el objetivo de suavizarla y reducir el ruido. Este filtro se basa en la aplicación de una interpolación polinómica en torno a cada muestra, ajustada mediante técnicas de optimización basadas en el método de los mínimos cuadrados.

Dada una secuencia de n muestras $y_1,\ldots,y_k,\ldots,y_n$ obtenidas con un período de muestreo T, para obtener el valor $\hat{y}_k$ de la señal suavizada en el instante k, se utiliza una aproximación polinomial óptima centrada en k con un tamaño de ventana de 5, 7 o 9 muestras. Por ejemplo, si se aplica una interpolación polinomial con una ventana total de 7 muestras, es decir, utilizando la muestra y_k, tres muestras anteriores $y_{k-3}, y_{k-2}, y_{k-1}$ y tres muestras posteriores $y_{k+1}, y_{k+2}, y_{k+3}$, el valor de la señal suavizada $\hat{y}_k$ es:

$$\hat{y}_k = \frac{1}{21}\left(-2\,y_{k-3}+3\,y_{k-2}+6\,y_{k-1}+7\,y_k+6\,y_{k+1}+3\,y_{k+2}-2\,y_{k+3}\right)$$

Tamaño de ventana	5	7	9
k-4			-21
k-3		-2	14
k-2	-3	3	39
k-1	12	6	54
k	17	7	59
k+1	12	6	54
k+2	-3	3	39
k+3		-2	14
k+4			-21
Cociente	35	21	231



Los coeficientes óptimos para diferentes tamaños de ventana se indican en la tabla.

Para calcular los primeros y últimos valores de la señal suavizada, se considera $y_k = y_1$ para $k<1$ e $y_k = y_n$ para $k>n$

Codifica otra función para aplicar este filtro en tiempo real.

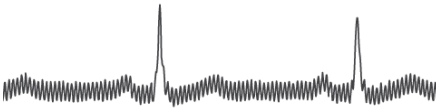
Una solución:

```
float medida(int k, float* pMedidas, int nMedidas) {
    if (k < 0) return pMedidas[0];
    else if (k >= nMedidas) return pMedidas[nMedidas - 1];
    else return pMedidas[k];
}
```

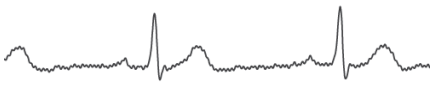
```
void SavitzkyGolay(float* pMedidas, unsigned int nMedidas, float* pFiltradas, unsigned int ventana) {
    // Aplica un filtro Savitzky-Golay sobre nMedidas medidas guardadas a partir de pMedidas y guarda
    // el resultado del filtrado a partir de pFiltradas

    static float m[3][9] = {  // Matriz de coeficientes del filtro
        {0, 0, -3, 12, 17, 12, -3, 0, 0},
        {0, -2, 3, 6, 7, 6, 3, -2, 0},
        {-21, 14, 39, 45, 49, 45, 39, 14, -21}
    };

    int i, n;
    if (ventana == 5) {
        i = 0;  // Índice para acceder a los coeficientes
        n = 2;  // Número de medidas anteriores y posteriores a tener en cuenta
    } else if (ventana == 7) {
        i = 1;
        n = 3;
    }
    else {
        i = 2;
        n = 4;
    }

    for (int k = 0; k < nMedidas; k++) {  // Aplica el filtro a todas las medidas

        pFiltradas[k] = 0;  // Inicializa el resultado del filtro

        for(int j = -n; j <= n; j++)
            pFiltradas[k] += m[i][4 + j] * medida(k + j, pMedidas, nMedidas);
        // Añade el resultado de multplicar los coeficientes por las n medidas anteriores y posteriores
    }

}
```

---

ej34) Un electrocardiógrafo recibe una señal proveniente de un sensor que genera una señal con nivel de tensión muy bajo y que necesita de una etapa de amplificación analógica para poder llevarla a un convertidor A/D. Este amplificador amplifica también el ruido debido a interferencias electromagnéticas. La señal amplificada tiene un rango máximo de -3V a 3V. Para medir esta señal amplificada, en el microcontrolador de este aparato disponemos de un sistema de desarrollo con la función `float mideSenal()` que devuelve un número real en ese rango.



Hay que muestrear la señal con un período de muestreo T = 1 ms y hay que aplicarle un filtro Butterworth de 5º orden de tipo paso bajo y frecuencia de corte de 100 Hz para poder reducir el ruido. La función de transferencia digital de este filtro es:



$$G(z) = \frac{0.0013z^5 + 0.0064z^4 + 0.0128z^3 + 0.0128z^2 + 0.0064z + 0.0013}{z^5 - 2.9754z^4 + 3.8060z^3 - 2.5453z^2 + 0.8811z - 0.1254}$$

El filtro parte de condiciones iniciales nulas.

Disponemos también de un temporizador y dos funciones para su manejo:

- `void arrancaTemporizador(int s, int ns)`: arranca el temporizador, que genera temporizaciones sucesivas con un intervalo de `s` segundos y `ns` nanosegundos.

- `void esperaTemporizador()`: la llamada a esta función no devuelve el control hasta que finalice una temporización.

El electrocardiógrafo dispone de una pantalla LCD de una resolución de 1024x768 puntos donde se muestra la señal filtrada correspondiente a los últimos 4 segundos. Hay que refrescar la gráfica a cada 2 segundos.
Para el manejo de la pantalla disponemos de las siguientes funciones:

- `void borraPantalla()`: borra las líneas dibujadas en la pantalla.

- `void dibujaLinea(int x1, int y1, int x2, int y2)`: dibuja una línea en la pantalla uniendo los puntos (x1, y1) y (x2, y2). El rango de coordenadas disponibles en la pantalla para dibujar líneas es de 0 a 1023 en el eje X y de 50 a 717 en el eje Y.

Codifica la aplicación que hay que instalar en este aparato.

Una solución:

$U(z)$ señal de entrada al filtro, $Y(z)$ señal de salida del filtro

$$G(z) = \frac{Y(z)}{U(z)} = \frac{0.0013z^5 + 0.0064z^4 + 0.0128z^3 + 0.0128z^2 + 0.0064z + 0.0013}{z^5 - 2.9745z^4 + 3.8060z^3 - 2.5453z^2 + 0.8811z - 0.1254}$$
$$= \frac{0.0013 + 0.0064z^{-1} + 0.0128z^{-2} + 0.0128z^{-3} + 0.0064z^{-4} + 0.0013z^{-5}}{1 - 2.9754z^{-1} + 3.8060z^{-2} - 2.5453z^{-3} + 0.8811z^{-4} - 0.1254z^{-5}}$$

$$y_k = 2.9745y_{k-1} - 3.8060y_{k-2} + 2.5453y_{k-3} - 0.8811y_{k-4} + 0.1254y_{k-5}$$
$$+ 0.0013u_k + 0.0064u_{k-1} + 0.0128u_{k-2} + 0.0128u_{k-3} + 0.0054u_{k-4} + 0.0013u_{k-5}$$

Los valores desde y_{k-5} hasta y_k de la señal filtrada se guardan en la matriz `y` del programa en las posiciones `y[3994]` hasta `y[3999]`.
Los valores desde u_{k-5} hasta u_k de la señal medida se guardan en la matriz `u` en `u[0]` hasta `u[5]`.

```
// Declaración de las funciones
void arrancaTemporizador(int, int);
void esperaTemporizador();
float mideSenal();
void borraPantalla();
void dibujaLinea(int x1, int y1, int x2, int y2);


void main() {
    int i = 0;
    int nDatosDisponibles = 0;  // Para esperar a disponer de 4000 medidas
    int nPeriodos = 0;  // Contaje de períodos para intervalo de 2 segundos
    float y[4000];  // Guarda las últimas 4000 salidas del filtro
    float u[6];  // Guarda la medida actual y 5 anteriores

    for (i = 0; i < 6; i++) {  // El filtro parte de condiciones iniciales nulas
        u[i] = 0;
        y[3999 - i] = 0;
    }

    float escalaY = (717 - 50) / 6.0f;
    // Escala que hay que aplicar para llevar el rango de la señal entre -3V y 3V a
    // la coordenada Y en la pantalla de 50 a 717

    float ceroY = 768 / 2;  // Coordenada Y de pantalla donde se representa 0V
```

```
    arrancaTemporizador(0, 1000000);
    // Arranca un temporizador con período de 1 ms

    while (1) {  // Repite contínuamente ...

        esperaTemporizador();   // Espera a la siguiente temporización

        for (i = 0; i <= 3998; i++)
            y[i] = y[i + 1];   // Desplaza la señal filtrada
        for (i = 0; i <= 4; i++)
            u[i] = u[i + 1];  // Desplaza las medidas

        u[5] = mideSenal();  // Añade la medida actual

        y[3999] = 2.9754 * y[3998] - 3.8060 * y[3997] + 2.5453 * y[3996] - 0.8811 * y[3995] +
            0.1254 * y[3994] + 0.0013 * u[5] + 0.0064 * u[4] + 0.0128 * u[3] + 0.0128 * u[2] +
            0.0064 * u[1] + 0.0013 * u[0];
        // Aplica la ecuación en diferencias del filtro

        if (nDatosDisponibles < 4000)
            nDatosDisponibles++;  // Para saber si se realizaron al menos 4000 medidas

        nPeriodos++;  // Para saber si transcurrió 1 s.

        if (nDatosDisponibles == 4000 && nPeriodos == 2000) { // Si 4000 datos y transcurrieron 2 s.
            nPeriodos = 0;  // Para contar otros 2 segundos

            borraPantalla();  // Borra la gráfica

            int x1 = 0;
            int y1 = ceroY + y[0] * escalaY;
            // Coordenadas del primer punto en la pantalla

            for(int i = 1; i < 4000; i++) {  // Por cada dato de la señal filtrada, desde segundo dato
                int x2 = i * 1023.0f / 4000.0f;  // Coordenada X en pantalla
                int y2 = ceroY + y[i] * escalaY;  // Coordenada Y en pantalla
                dibujaLinea(x1, y1, x2, y2);  // Dibuja una línea desde el punto anterior
                x1 = x2;  // Actualiza la coordenada X del punto anterior para la siguiente línea
                y1 = y2;  // Actualiza la coordenada Y del punto anterior para la siguiente línea
            }
        }
    }
}
```

---

ej63) Codifica una clase para implantar el filtro Butterworth de 5º orden indicado en el anterior ejercicio ej62) y utilízala para la codificación de la aplicación a ejecutar en el electrocardiógrafo.

Una solución:

Fichero `filtro.h`:

```
#ifndef FILTRO_H
#define FILTRO_H

class Filtro {
    float yAnt[5];
    float uAnt[5];
public:
    Filtro();
    float filtra(float uk);
};

#endif
```

Fichero `filtro.cpp`:

```
#include "filtro.h"

Filtro::Filtro() {
    for(int i = 0; i < 5; i ++) {
        uAnt[i] = 0;
        yAnt[i] = 0;
    }
}

float Filtro::filtra(float uk) {

    float yk = 2.9754 * yAnt[0] - 3.8060 * yAnt[1] + 2.5453 * yAnt[2] -
           0.8811 * yAnt[3] + 0.1254 * yAnt[4] +
           0.0013 * uk + 0.0064 * uAnt[0] + 0.0128 * uAnt[1] +
           0.0128 * uAnt[2] + 0.0064 * uAnt[3] + 0.0013 * uAnt[4];

    for (int i = 3; i >= 0; i--) {
        uAnt[i+1] = uAnt[i];
        yAnt[i+1] = yAnt[i];
    }

    uAnt[0] = uk;
    yAnt[0] = yk;
    return yk;
}
```

Fichero `main.cpp`:

```
void arrancaTemporizador(int, int);
void esperaTemporizador();
float mideSenal();
void borraPantalla();
void dibujaLinea(int x1, int y1, int x2, int y2);

#include "filtro.h"

int main() {
    int i = 0;
    int nDatosDisponibles = 0;  // Para esperar a disponer de 4000 medidas
    int nPeriodos = 0;  // Contaje de períodos para intervalo de 2 segundos
    float y[4000];  // Guarda las últimas 4000 salidas del filtro
    float u[6];  // Guarda la medida actual y 5 anteriores
    Filtro f;


    float escalaY = (717 - 50) / 6.0f;
    // Escala que hay que aplicar para llevar el rango de la señal entre -3V y 3V a
    // la coordenada Y en la pantalla de 50 a 717

    float escalaX = 1023.0 / 3999.0;
    // Escala que hay que aplicar en el eje X

    float ceroY = 768 / 2;  // Coordenada Y de pantalla donde se representa 0V

    arrancaTemporizador(0, 1000000);
    // Arranca un temporizador con período de 1 ms

    while (1) {  // Repite contínuamente ...

        esperaTemporizador();   // Espera a la siguiente temporización

        for (i = 0; i <= 3998; i++)
            y[i] = y[i + 1];   // Desplaza la señal filtrada

        y[3999] = f.filtra(mideSenal());
```

```
        if (nDatosDisponibles < 4000)
            nDatosDisponibles++;  // Para saber si se realizaron al menos 4000 medidas

        nPeriodos++;  // Para saber si transcurrió 1 s.

        if (nDatosDisponibles == 4000 && nPeriodos == 2000) { // Si 4000 datos y transcurrieron 2 s.
            nPeriodos = 0;  // Para contar otros 2 segundos

            borraPantalla();  // Borra la gráfica

            int x1 = 0;
            int y1 = ceroY + y[0] * escalaY;
            // Coordenadas del primer punto en la pantalla

            for(int i = 1; i < 4000; i++) {  // Por cada dato de la señal filtrada, desde segundo dato
                int x2 = i * escalaX;  // Coordenada X en pantalla
                int y2 = ceroY + y[i] * escalaY;  // Coordenada Y en pantalla
                dibujaLinea(x1, y1, x2, y2);  // Dibuja una línea desde el punto anterior
                x1 = x2;  // Actualiza la coordenada X del punto anterior para la siguiente línea
                y1 = y2;  // Actualiza la coordenada Y del punto anterior para la siguiente línea
            }
        }
    }

    return 0;
}
```

Control

ej57) Dado un sistema SISO lineal expesando en variables de estado mediante $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}u,\ y = \mathbf{C}\mathbf{x}$ donde intervienen n estados $\mathbf{x}$, una entrada u y una salida y, elabora funciones en lenguaje C para determinar si el sistema es controlable y/o es observable. Para ello se determinará si la matriz de controlabilidad $\mathcal{C}$ y/o la matriz de observabilidad $\mathcal{O}$ tienen rango máximo n.

$$\mathcal{C} = \begin{pmatrix} \mathbf{B} & \mathbf{AB} & \mathbf{A}^2\mathbf{B} & \cdots & \mathbf{A}^{n-1}\mathbf{B} \end{pmatrix}, \qquad \mathcal{O} = \begin{pmatrix} \mathbf{C} \\ \mathbf{CA} \\ \mathbf{CA}^2 \\ \vdots \\ \mathbf{CA}^{n-1} \end{pmatrix}$$

Los elementos de las matrices se almacenan guardándolos en números reales de tipo **`double`** en posiciones consecutivas de memoria, por filas. Ej. para

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 \\ -1 & -1 & -1 \\ 0 & 0 & -1 \end{pmatrix}, \qquad \mathbf{B} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \qquad \mathbf{C} = \begin{pmatrix} 1 & 0 & 1 \end{pmatrix}, \qquad n = 3$$

```
double A[] = { 0, 1, 0, -1, -1, -1, 0, 0, -1 };
double B[] = { 0, 1, 1 };
double C[] = { 1, 0, 1 };
```

Se supone que disponemos de una función `int rango(double * m, int nf, int nc)` que devuelve el rango de la matriz cuyos elementos están guardados donde indica **`m`** y que tiene **`nf`** filas y **`nc`** columnas.

Una solución:

```c
#include <stddef.h>  // Para NULL
#include <stdio.h>   // Para printf

int rango(double* p, int filas, int columnas);

void calculaCPorA(double* C, double* A, int n, double* CPorA) {
    int i, j;
    for (i = 0; i < n; i++) {
        double x = 0;
        for (j = 0; j < n; j++)
            x += C[j] * A[j * n + i];
        CPorA[i] = x;
    }
}

void matrizObservabilidad(double* A, double* C, int n, double* mO) {
/* Guarda los elementos de la matriz de observabilidad a partir de mO.
   La calcula para el sistema con las matrices A y C, con n estados. */

    double* CA = (double*)malloc(n * sizeof(double));
    double* CA2 = (double*)malloc(n * sizeof(double));
    int i, j;
    for (i = 0; i < n; i++)
        CA[i] = C[i];
    i = 0;
    while (i < n) {
        for (j = 0; j < n; j++)
            mO[i * n + j] = CA[j];
        calculaCPorA(CA, A, n, CA2);
        for (j = 0; j < n; j++)
            CA[j] = CA2[j];
        i++;
    }
    free(CA);
    free(CA2);
}

void calculaAPorB(double* A, double* B, int n, double* APorB) {
    int i, j;
    for (i = 0; i < n; i++) {
        double x = 0;
        for (j = 0; j < n; j++)
            x += A[i * n + j] * B[j];
        APorB[i] = x;
    }
}

void matrizControlabilidad(double* A, double* B, int n, double* mC) {
    /* Guarda los elementos de la matriz de controlabilidad a partir de mC.
       La calcula para el sistema con las matrices A y B, con n estados. */

    double* AB = (double*)malloc(n * sizeof(double));
    double* AB2 = (double*)malloc(n * sizeof(double));
    int i, j;
    for (i = 0; i < n; i++)
        AB[i] = B[i];
    i = 0;
    while (i < n) {
        for (j = 0; j < n; j++)
            mC[j * n + i] = AB[j];
        calculaAPorB(A, AB, n, AB2);
        for (j = 0; j < n; j++)
            AB[j] = AB2[j];
        i++;
```

```
    }
    free(AB);
    free(AB2);
}

int controlable(double * mC, int n) {
// Devuelve un buleano cierto si la matriz de controlabilidad guardada
// en mC para un sistema con n estados, indica que el sistema es
// controlable

    return rango(mC, n, n) == n;
}

int observable(double* mO, int n) {
    // Devuelve un buleano cierto si la matriz de observabilidad guardada
    // en mO para un sistema con n estados, indica que el sistema es
    // observable

    return rango(mO, n, n) == n;
}

double A[] = { 0, 1, 0, -1, -1, -1, 0, 0, -1 };
double B[] = { 0, 1, 1 };
double C[] = { 1, 0, 1 };
// Matrices del sistema representado en variables de estado

int n = 3;  // Número de estados

double mC[9], mO[9];
// Reserva espacio para los elementos de las matrices de
// controlabilidad y observabilidad

int main() {

    matrizControlabilidad(A, B, n, mC);
    // Calcula la matriz de controlabilidad

    matrizObservabilidad(A, C, n, mO);
    // Calcula la matriz de observabilidad

    if (controlable(mC, n))
        printf("Controlable\n");
    else printf("No controlable\n");

    if (observable(mO, n))
        printf("Observable\n");
    else printf("No observable\n");

    return 0;
}
```

---

ej20) Codifica una clase en C++ para implantar un regulador digital SISO genérico para cualquier n y m

$$G_r(z) = \frac{U(z)}{E(z)} = \frac{b_m z^m + b_{m-1} z^{m-1} + \cdots + b_1 z + b_0}{z^n + a_{n-1} z^{n-1} + \cdots + a_1 z + a_0}$$

con entrada $E(z)$ y salida $U(z)$ y con $m \leq n$, de forma que se pueda utilizar como se indica en el siguiente ejemplo:

$$G_r(z) = \frac{0.35z^2 + 0.889z + 1.5}{z^3 + 0.99z^2 + 0.863z + 0.6}$$

```
#include "reguladorh"

double b[] = {0.35, 0.889, 1.5};  // Coeficientes del polinomio del numerador
double a[] = {0.99, 0.863, 0.6};  // Coeficientes del polinomio del denominador
int m = 2;  // Grado del polinomio del numerador
int n = 3;  // Grador del polinomio del denominador

int main() {
    Regulador regulador(m, n, b, a);  // Crea un regulador digital SISO
    while(1) {
        esperaPeriodo();  // Bloquea hasta siguiente período de muestreo
        double e = referencia() - mideSalida();  // Obtiene la señal de error
        double u = regulador.actuacion(e);  // Calcula la actuación a partir del error
        aplicaActuacion(u);  // Aplica la actuación a la planta
    }
    return 0;
}
```

No hay que codificar las funciones **esperaPeriodo()**, **mideSalida()**, **referencia()** y **aplicaActuacion()**.

Una solución:

$$G_r(z) = \frac{U(z)}{E(z)} = \frac{b_m z^{m-n} + b_{m-1} z^{m-1-n} + \cdots + b_1 z 1 - n + b_0 z^{-n}}{1 + a_{n-1} z^{-1} + \cdots + a_1 z^{1-n} + a_0 z^{-n}}$$

$$u_k + a_{n-1}u_{k-1} + \cdots + a_1 u_{k+1-n} + a_0 u_{k-n} = b_m e_{k+m-n} + b_{m-1} e_{k+m-1-n} + \cdots + b_1 e_{k+1-n} + b_0 e_{k-n}$$

$$u_k = \sum_{i=0}^{m} b_i e_{k+i-n} - \sum_{i=0}^{n-1} a_i u_{k+i-n}$$

numerador

a_{n-1}	0
a_{n-2}	1
a_{n-3}	2
⋮	⋮
a_2	n-3
a_1	n-2
a_0	n-1

salidasAnteriores

u_{k-1}	0
u_{k-2}	1
u_{k-3}	2
⋮	⋮
u_{k+2-n}	n-3
u_{k+1-n}	n-2
u_{k-n}	n-1

denominador

b_m	0
b_{m-1}	1
b_{m-2}	2
⋮	⋮
b_2	m-2
b_1	m-1
b_0	m

entradasAnteriores

e_k	0
e_{k-1}	1
e_{k-2}	2
⋮	⋮
e_{k+m-n}	n-m
$e_{k+m-1-n}$	n-m+1
$e_{k+m-2-n}$	n-m+2
⋮	
e_{k+2-n}	n-2
e_{k+1-n}	n-1
e_{k-n}	n

Fichero `regulador.h`:

```
#ifndef REGULADOR_H
#define REGULADOR_H

class Regulador {  // Un objeto de esta clase representa a un regulador digital SISO

    int gradoNumerador, gradoDenominador;  // Grados de los polinomios del numerador y denominador
    double * numerador, * denominador;  // Matrices con los coeficientes de los polinomios
    double * entradasAnteriores, * salidasAnteriores;  // Matrices para entradas y salidas anteriores

public:

    Regulador(int gradoNumerador, int gradorDenominador, double * numerador, double * denominador);
    // Constructor para inicializar un regulador a partir de su función de transferencia:
    // - 'gradoNumerador': grado del polinomio del numerador
    // - 'gradoDenominador': grado del polinomio del denominador
    // - 'numerador': matriz de coeficientes del polinomio del numerador, de mayor a menor exponente
    // - 'denominador': matriz de coeficientes del polinomio del denominador, de mayor a menor
    //   exponente

    ~Regulador();  // Destructor para liberar zonas de memoria reservadas en el constructor

    double actuacion(double entrada);
    // Devuelve la salida del regulador cuando se le aplica la entrada pasada por parámetro
};

#endif // REGULADOR_H
```

Fichero `regulador.cpp`:

```
#include "regulador.h"

Regulador::Regulador(int gradoNumerador, int gradoDenominador, double * numerador, double *
denominador) {
    // Constructor para inicializar un regulador SISO digital a partir de su función de transferencia:
    // - 'gradoNumerador': grado del polinomio del numerador
    // - 'gradoDenominador': grado del polinomio del denominador
    // - 'numerador': matriz de coeficientes del polinomio del numerador, de mayor a menor exponente
    // - 'denominador': matriz de coeficientes del polinomio del denominador, de mayor a menor
exponente

    this->gradoNumerador = gradoNumerador;
    this->gradoDenominador = gradoDenominador;
    this->numerador = numerador;
    this->denominador = denominador;
    // Guarda los parámetros

    this->entradasAnteriores = new double[gradoDenominador + 1];
    // Matriz para guardar la entrada de error actual y las anteriores

    this->salidasAnteriores = new double[gradoDenominador];
    // Matriz para guardar las salidas anteriores

    for (int i = 0; i < gradoNumerador; i ++) {
        this->entradasAnteriores[i] = 0.0;
        this->salidasAnteriores[i] = 0.0;
    }
    this->entradasAnteriores[gradoDenominador] = 0.0;
    // Inicializa las matrices con ceros
}


Regulador::~Regulador() {  // Destructor para liberar zonas de memoria reservadas en el constructor
    delete[] entradasAnteriores;
    delete[] salidasAnteriores;
}
```

```
double Regulador::actuacion(double entrada) {
    // Devuelve la salida del regulador cuando se le aplica la entrada pasada por parámetro

    this->entradasAnteriores[0] = entrada;  // Guarda la entrada actual en la matriz

    double resultado = 0.0;  // Variable local para calcular la salida

    for (int i = 0; i <= gradoNumerador; i++) {
        double b = numerador[i];  // Coeficiente del polinomio del numerador
        double e = entradasAnteriores[gradoDenominador - gradoNumerador + i];  // Entrada
        resultado += b * e;  // Suma el término correspondiente a esa entrada
    }
    for (int i = 0; i < gradoDenominador; i++) {
        double a = denominador[i];  // Coeficiente del polinomio del denominador
        double u = salidasAnteriores[i];  // Salida
        resultado -= a * u;  // Resta el término corrrespondiente a esa salida
    }

    for (int i = gradoDenominador; i >= 1; i--)
        entradasAnteriores[i] = entradasAnteriores[i - 1];
    // Desplaza la matriz de entradas

    for (int i = gradoDenominador - 1; i >= 1; i--)
        salidasAnteriores[i] = salidasAnteriores[i - 1];
    salidasAnteriores[0] = resultado;
    // Desplaza la matriz de salidas y pone la nueva salida en la primera posición

    return resultado;  // Devuelve el cálculo de la salida
}
```

Multitarea

ej58) En un sistema operativo en tiempo real existen los semáforos binarios (sólo pueden valer 0 o 1).

- Existe el tipo de dato `bsem` para poder declarar una variable que represente a un semáforo binario.

- La función `void bsemInit(bsem &)` para inicializar un semáforo binario.

- Las funciones `void bsemWait(bsem &)` y `void bsemSignal(bsem &)` para realizar las operaciones wait y signal.

Crea un tipo de dato `semaforo` y las siguientes funciones para poder utilizar semáforos convencionales que puedan tener un valor mayor que 1:

- `void init(semaforo &, int valor)` para inicializarlo a un determinado valor

- `void wait(semaforo &)` y `void signal(semaforo &)` para las operaciones wait y signal

Una solución:

```
typedef struct {
    bsem acceso, bloqueo;
    int valor, nBloqueados;
} semaforo;
```

```
void inicializaSemaforo(int valor, semaforo * ps) {
    inicializaBSem(&(ps->acceso), 1);
    inicializaBSem(&(ps->bloqueo), 0);
    ps->valor = valor;
    ps->nBloqueados = 0;
}

void wait(semaforo * ps) {
    bWait(&(ps->acceso));
    if (ps->valor == 0) {
        ps->nBloqueados ++;
        bSignal(&(ps->acceso));
        bWait(&(ps->bloqueo));
    } else {
        ps->valor --;
        bSignal(&(ps->acceso));
    }
}

void signal(semaforo * ps) {
    bWait(&(ps->acceso));
    if (ps->nBloqueados > 0) {
        ps->nBloqueados --;
        bSignal(&(ps->bloqueo));
    } else
        ps->valor ++;
    bSignal(&(ps->acceso));
}
```

---

ej21) Codifica el programa elaborado para el ejercicio ej10) de forma que se atienda a los diferentes dispositivos utilizando multitarea mediante hilos.

Una solución:

```
#include <stdint.h>
uint8_t* A = (uint8_t*)0x100;
uint8_t* B = (uint8_t*)0x110;
uint8_t* C = (uint8_t*)0x120;
uint8_t* D = (uint8_t*)0x200;
// Punteros para acceder a los puertos de E/S digital

int leeBit(uint8_t* puerto, int nBit) {  // Devuelve un buleano cierto si un bit de un puerto está a 1
    return (*puerto) & (1 << nBit);
}

void escribeBit(uint8_t* puerto, int nBit, int valor) { // Modifica un bit de un puerto
    if (valor) *puerto |= 1 << nBit;
    else *puerto &= ~(1 << nBit);
}

void hCintaEntrada() {  // Función que ejecuta el hilo que atiende a la cinta de entrada
    int enMovimiento = 0, sensor1Anterior = 0, sensor2Anterior = 0, nCajas = 0;

    while (1) {  // Repite indefinidamente ...
        int sensor1 = leeBit(C, 2);
        int sensor2 = leeBit(C, 0);
        if (!sensor1Anterior && sensor1)  // Si apareció una caja al principio
            nCajas++;  // Incrementa contador de cajas
        if (sensor2Anterior && !sensor2)  // Si desapareció una caja al final
            nCajas--;  // Decrementa contador de cajas
        if (enMovimiento && sensor2) {  // Si en movimiento y llegó caja al final
            escribeBit(D, 5, 0);  // Para la cinta
```

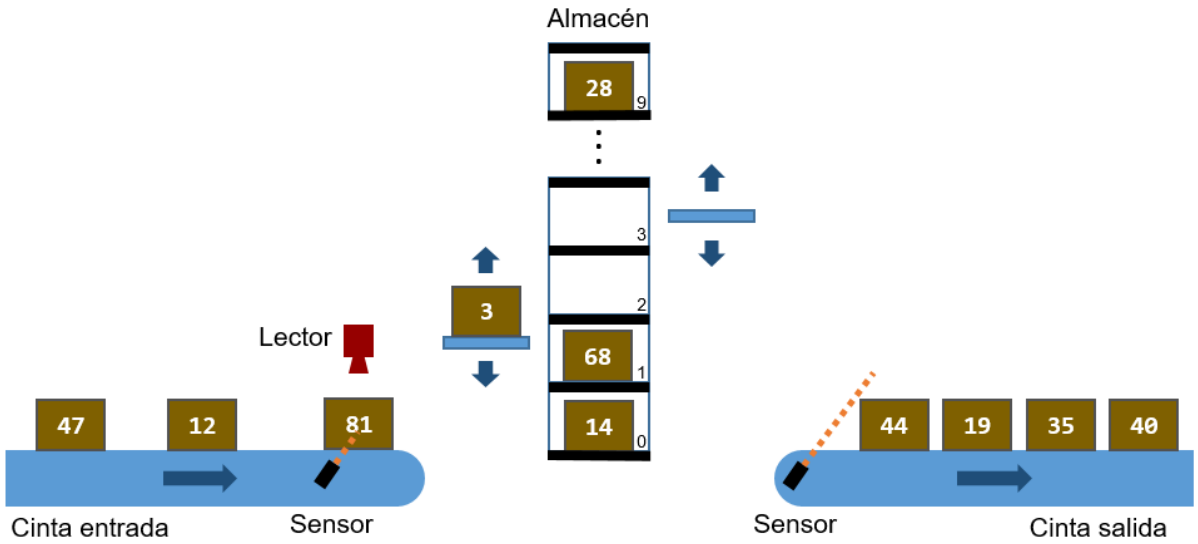
```
            enMovimiento = 0;
        } else if (!sensor2 && nCajas > 0 && !enMovimiento) { // Si cajas y ninguna al final
            escribeBit(D, 5, 1);  // Pone en marcha la cinta
            enMovimiento = 1;
        }
        sensor2Anterior = sensor2;
        sensor1Anterior = sensor1;
    }
}

void hCintaSalida() {  // Función que ejecuta el hilo que atiende a la cinta de salida
    int enMovimiento = 0, sensor1Anterior = 0, sensor2Anterior = 0, nCajas = 0;

    while (1) {  // Repite indefinidamente ...
        int sensor1 = leeBit(C, 1);
        int sensor2 = leeBit(C, 3);
        if (!sensor1Anterior && sensor1)  // Si apareció una caja al principio
            nCajas++;  // Incrementa contador de cajas
        if (sensor2Anterior && !sensor2)  // Si desapareció una caja al final
            nCajas--;  // Decrementa contador de cajas
        if (enMovimiento && sensor2) {  // Si en movimiento y llegó caja al final
            escribeBit(D, 6, 0);  // Para la cinta
            enMovimiento = 0;
        } else if (!sensor2 && nCajas > 0 && !enMovimiento) { // Si cajas y ninguna al final
            escribeBit(D, 6, 1);  // Pone en marcha la cinta
            enMovimiento = 1;
        }
        sensor2Anterior = sensor2;
        sensor1Anterior = sensor1;
    }
}

void mueveGruaVertical(int posicion) {  // LLeva la grúa a esa posición vertical
    static int posicionActual = 6;  // Recuerda en qué posición está
    if (posicion == posicionActual) return;  // Si ya está allí, termina
    if (posicion > posicionActual)  // Si hay que subir ...
        escribeBit(D, 0, 1);  // Activa subida
    else escribeBit(D, 1, 1);  // Activa bajada
    while (!leeBit(B, posicion));  // Espera a llegar a la posición
    escribeBit(D, 0, 0);  // Desactiva subida
    escribeBit(D, 1, 0);  // Desactiva bajada
    posicionActual = posicion;  // Recuerda dónde está
}

void mueveGruaHorizontal(int posicion) {  // Lleva a la grúa a esa posición horizontal
    static int posicionActual = 0;  // Recuerda en qué posición está
    if (posicion == posicionActual) return;  // Si ya está allí, termina
    if (posicion > posicionActual)  // Si hay que ir a la derecha ...
        escribeBit(D, 2, 1);  // Activa movimiento a la derecha
    else escribeBit(D, 3, 1);  // Activa movimiento a la izquierda
    while (!leeBit(A, posicion));  // Espera a llegar a la posición
    escribeBit(D, 2, 0);  // Desactiva movimiento a la derecha
    escribeBit(D, 3, 0);  // Desactiva movimiento a la izquierda
}

void ventosa(int activar) {  // Activa o desactiva la ventosa
    escribeBit(D, 4, activar);
}

void inicializaGrua() {  // Sitúa la grúa arriba a la izquierda
    escribeBit(D, 0, 1);  // Activa grúa subir
    while (!leeBit(B, 6));  // Espera a que llegue arriba
    escribeBit(D, 0, 0);  // Desactiva subida
    escribeBit(D, 3, 1); // Activa grúa izquierda
    while (!leeBit(A, 0));  // Espera a que llegue a la izquierda
    escribeBit(D, 3, 0);  // Desactiva grúa izquierda
}
```

```
int cajaEnCintaEntrada() {  // Devuelve buleano cierto si hay caja al final de la cinta de entrada
    return leeBit(C, 0);
}

int cajaEnCintaSalida() {  // Devuelve buleano cierto si hay caja al comienzo de la cinta de salida
    return leeBit(C, 1);
}

void hPulmon() {  // Función que ejecuta el hilo que atiende al pulmón
    int nCajasEnPulmon = 0;  // Número de cajas almacenadas en el pulmón
    *D = 0; // Todas las salidas desactivadas
    inicializaGrua();   // Sitúa la grúa arriba a la izquierda

    while (1) {  // Bucle infinito

        if (cajaEnCintaEntrada() && !cajaEnCintaSalida()) {
            // Si hay pieza al final de cinta de entrada pero no al principio de cinta de salida ...

            mueveGruaHorizontal(0);  // Sitúa la grúa sobre cinta entrada
            mueveGruaVertical(3);  // Baja
            ventosa(1);  // Coge caja
            mueveGruaVertical(6);  // Sube
            mueveGruaHorizontal(5);  // Sitúa la grúa sobre cinta salida
            mueveGruaVertical(3);  // Baja
            ventosa(0);  // Deja caja
            mueveGruaVertical(6);  // Sube
        }

        else if (nCajasEnPulmon > 0 && !cajaEnCintaSalida()) {
            // Si hay cajas en pulmón que se pueden llevar a la cinta de salida ...

            int fila = nCajasEnPulmon / 4;  // Fila en el pulmón, de 0 a 5
            int columna = nCajasEnPulmon % 4;  // Columna en el pulmón, de 0 a 5
            mueveGruaHorizontal(columna + 1);  // Sitúa la grúa sobre la columna
            mueveGruaVertical(fila);  // Baja la grúa hasta la caja del pulmón
            ventosa(1);  // Coge la caja
            mueveGruaVertical(6);  // Sube la grúa
            mueveGruaHorizontal(5);  // Sitúa la grúa sobre cinta salida
            mueveGruaVertical(3);  // Baja
            ventosa(0);  // Deja la caja
            mueveGruaVertical(6);  // Sube
            nCajasEnPulmon--;  // Decrementa contador de cajas en el pulmón
        }

        else if (nCajasEnPulmon < 24 && cajaEnCintaEntrada()) {
            // Si hay caja al final de cinta de entrada y hay hueco en pulmón ...

            nCajasEnPulmon++;  // Va a haber una caja más en el pulmón
            int fila = nCajasEnPulmon / 4;  // Fila donde depositar la caja, de 0 a 5
            int columna = nCajasEnPulmon % 4;  // Columna donde depositar la caja, de 0 a 3
            mueveGruaHorizontal(0);  // Sitúa grúa sobre cinta de entrada
            mueveGruaVertical(3);  // Baja
            ventosa(1);  // Coge caja
            mueveGruaVertical(6);  // Sube
            mueveGruaHorizontal(columna + 1);  // Sitúa caja sobre posición en el pulmón
            mueveGruaVertical(fila);  // Baja hasta la fila
            ventosa(0);  // Deja caja
            mueveGruaVertical(6);  // Sube
        }
    }
}
```

```
void main() {
    pthread_t idHCintaEntrada, idHCintaSalida, idHPulmon;  // Identificadores de hilos

    pthread_create(&idHCintaEntrada, NULL, hCintaEntrada, NULL);
    pthread_create(&idHCintaSalida, NULL, hCintaSalida, NULL);
    pthread_create(&idHPulmon, NULL, hPulmon, NULL);
    // Lanza los hilos para atender a los dispositivos

    pthread_join(idHCintaEntrada, NULL);
    pthread_join(idHCintaSalida, NULL);
    pthread_join(idHPulmon, NULL);
    // Espera a que finalicen los hilos
}
```

---

ej22) 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.

Codifica una aplicación de forma que se estén realizando simultaneamente el mayor de operaciones posibles con estos dispositivos.

Una solución:

```
int posicionesOcupadas[10];
int nPosicionesOcupadas = 0;
uint32_t numerosSerie[10];

void hCintaEntrada() {
    while (1) {
        cintaEntrada(!sensorEntrada());
    }
}

void hElevadorEntrada() {
    while (1) {
        elevadorEntrada(0);
        while (!sensorEntrada());
        uint32_t num_serie = leeCodigo();
        cogeEntrada();
        while (nPosicionesOcupadas == 10);
        int posicion = 0;
        while (posicionesOcupadas[posicion]) posicion++;
        elevadorEntrada(posicion);
        dejaEntrada();
        posicionesOcupadas[posicion] = 1;
        numerosSerie[posicion] = num_serie;
        nPosicionesOcupadas++;
    }
}

void hElevadorSalida() {
    while (1) {
        while (nPosicionesOcupadas == 0);
        int primera = 1, seleccionada;
        for (int posicion = 0; posicion < 10; posicion++)
            if (posicionesOcupadas[posicion]) {
                if (primera) {
                    seleccionada = posicion;
                    primera = 0;
                } else if (numerosSerie[posicion] < numerosSerie[seleccionada])
                    seleccionada = posicion;
            }
        elevadorSalida(seleccionada);
        cogeSalida();
        posicionesOcupadas[seleccionada] = 0;
        nPosicionesOcupadas --;
        elevadorSalida(0);
        while (sensorSalida());
        dejaSalida();
    }
}

void hCintaSalida() {
    while (1)
        cintaSalida(sensorSalida());
}
```

```
int main(void) {
    concurrente {
        hCintaEntrada();
        hCintaSalida();
        hElevadorEntrada();
        hElevadorSalida();
    }
}
```

---

ej23) Utiliza herramientas de sincronización para optimizar el uso de la CPU en el ejercicio ej22)

Una solución:

```
int posicionesOcupadas[10];
uint32_t numerosSerie[10];
semaforo spEntrada = 1, scEntrada = 0;
semaforo spAlmacen = 10, scAlmacen = 0;
semaforo spSalida = 1, scSalida = 0;

void hCintaEntrada(void) {
    while (1) {
        wait(spEntrada);
        cintaEntrada(1);
        while (!sensorEntrada());
        cintaEntrada(0);
        signal(scEntrada);
    }
}

void hElevadorEntrada(void) {
    while (1) {
        elevadorEntrada(0);
        wait(scEntrada);
        uint32_t num_serie = leeCodigo();
        cogeEntrada();
        signal(spEntrada);
        wait(spAlmacen);
        int posicion = 0;
        while (posicionesOcupadas[posicion]) posicion++;
        elevadorEntrada(posicion);
        dejaEntrada();
        posicionesOcupadas[posicion] = 1;
        numerosSerie[posicion] = num_serie;
        signal(scAlmacen);
    }
}

void hElevadorSalida(void) {
    while (1) {
        wait(scAlmacen);
        int primera = 1, seleccionada;
        for (int posicion = 0; posicion < 10; posicion++)
            if (posicionesOcupadas[posicion]) {
                if (primera) {
                    seleccionada = posicion;
                    primera = 0;
                } else if (numerosSerie[posicion] < numerosSerie[seleccionada])
                    seleccionada = posicion;
            }
        elevadorSalida(seleccionada);
        cogeSalida();
        posicionesOcupadas[seleccionada] = 0;
        signal(spAlmacen);
```

```
        elevadorSalida(0);
        wait(spSalida);
        dejaSalida();
        signal(scSalida);
    }
}

void hCintaSalida(void) {
    while (1) {
        wait(scSalida);
        cintaSalida(1);
        while (sensorSalida());
        cintaSalida(0);
        signal(spSalida);
    }
}

int main(void) {
    concurrente {
        hCintaEntrada();
        hCintaSalida();
        hElevadorEntrada();
        hElevadorSalida();
    }
}
```

---

ej24) En una célula de fabricación flexible disponemos de un robot de configuración cilíndrica, una prensa hidráulica (0, 100), una máquina de control numérico (-100, 0) y una posición de entrada (100, 0) y otra de salida (0,-100) de piezas. Se indica para cada una de ellas su posición según el sistema de coordenadas del robot. Todos estos dispositivos se pueden controlar desde un computador con sistema operativo multitarea.

Para manejar el robot, disponemos de las siguientes funciones:

- **`void coge(float, float)`:** Durante la ejecución de esta función se mueve el robot de forma que la pinza se sitúe en el punto cuyas coordenadas se indican por parámetro, coge una pieza en esa posición y vuelve a una posición de reposo.

- **`void deja(float, float)`**: El robot deja la pieza que tiene cogida en la posición indicada por parámetro y luego vuelve a la posición de reposo.

Cuando colocamos una pieza en la prensa hidráulica, podemos efectuar un estampado sobre la misma mientras ejecutamos la función **`void estampa()`**.

Si introducimos una pieza en la máquina de control numérico, podemos efectuar una operación de mecanizado sobre la misma mientras ejecutamos la función **`void mecaniza()`**.

Además, disponemos de barreras ópticas en las posiciones de entrada y salida, de forma que podemos llamar a las funciones **`int barrera_entrada()`** o **`int barrera_salida()`**, que devuelven un buleano verdadero cuando hay alguna pieza situada en las mismas.

Utilizando todas estas herramientas, define un programa que opere sobre todos los dispositivos que componen la célula de fabricación flexible, de forma que las piezas que van apareciendo una a una en la posición de entrada se estampen primero y luego se mecanicen, colocándose finalmente en la posición de salida. La capacidad de la posición de entrada, de la prensa, de la máquina de control numérico y de la posición de salida es de una pieza cada una.

Una solución: Un hilo por cada posición. El robot es región crítica para mover piezas entre posiciones.

```
semaforo spEntradaPrensa = 1, scEntradaPrensa = 0;
semaforo spPrensaCNC = 1, scPrensaCNC = 0;
semaforo spCNCSalida = 1, scCNCSalida = 0;
semaforo sRobot = 1;

void hEntrada() {
    while (1) {
        while (!barrera_entrada());
        wait(spEntradaPrensa);
        wait(sRobot);
        coge(100, 0);
        deja(0, 100);
        signal(sRobot);
        signal(scEntradaPrensa);
    }
}

void hPrensa() {
    while (1) {
        wait(scEntradaPrensa);
        estampa();
        wait(spPrensaCNC);
        wait(sRobot);
        coge(0, 100);
        deja(-100, 0);
        signal(sRobot);
        signal(spEntradaPrensa);
        signal(scPrensaCNC);
    }
}

void hCNC() {
    while (1) {
        wait(scPrensaCNC);
        mecaniza();
        wait(spCNCSalida);
        wait(sRobot);
        coge(-100, 0);
        deja(0, -100);
        signal(sRobot);
        signal(spPrensaCNC);
        signal(scCNCSalida);
    }
}

void hSalida() {
    while (1) {
        wait(scCNCSalida);
        while (barrera_salida());
        signal(spCNCSalida);
    }
}

void main() {
    concurrente{
        hEntrada();
        hPrensa();
        hCNC();
        hSalida();
    }
}
```

Otra solución: eliminando los hilos de entrada y salida, integrándolos en los de la prensa y CNC. Robot región crítica.

```
semaforo sp = 1, sc = 0;
semaforo sRobot = 1;
```

```
void hPrensa() {
    while (1) {
        while (!barrera_entrada());
        wait(sRobot);
        coge(100, 0);
        deja(0, 100);
        signal(sRobot);
        estampa();
        wait(sp);
        wait(sRobot);
        coge(0, 100);
        deja(-100, 0);
        signal(sRobot);
        signal(sc);
    }
}

void hiloCNC() {
    while (1) {
        wait(sc);
        mecaniza();
        while (barrera_salida());
        wait(sRobot);
        coge(-100, 0);
        deja(0, -100);
        signal(sRobot);
        signal(sp);
    }
}

void main() {
    concurrente{
        hPrensa();
        hCNC();
    }
}
```

---

ej25) Codifica una aplicación en C++ para un sistema embebido con el objetivo de implantar en un único microcontrolador cinco reguladores PI que se pueden operar de forma remota en un canal de comunicaciones. En este sistema embebido existen cinco entradas analógicas y cinco salidas analógicas con rango de -10V a 10V que se pueden operar mediante:

- `double entradaAnalogica(int nEntrada)`: mide la entrada indicada por parámetro (0 a 4) y devuelve el resultado.

- `void salidaAnalogica(int nSalida, double valor)`: lleva el `valor` a la salida indicada en `nSalida` (0 a 4).

La actuación de cada regulador tiene que ser 0 hasta que se reciban por el canal sus parámetros (la ganancia K_p, el tiempo integral T_i y el período de muestreo T) y su primera consigna. Los parámetros de cada regulador se reciben una única vez. La consigna se va a recibir múltiples veces.

La función `void establecePrioridad(int prioridad)` establece la prioridad del hilo que realiza la llamada, indicando un valor de prioridad por parámetro, de 0 a 255, cuanto más alto mayor prioridad.

Se dispone de las siguientes clases:

- la clase `Temporizador` con un constructor al que se le pasa el período de tiempo en ms con el que se inicializa un temporizador que genera a partir de ese momento eventos contínuamente con ese intervalo. Esta clase dispone del método `void espera()` que bloquea hasta que ocurra el siguiente evento de temporización.

- la clase `Canal` para manejar las comunicaciones, con 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 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.

Por cada petición recibida en el canal, nuestro sistema embebido envía una respuesta. Las posibles peticiones y respuestas son las siguientes:

- *Recepción de parámetros de un regulador*: se recibe un paquete de bytes que contiene un byte a 0, a continuación otro byte que indica el número de regulador (0 a 4) y finalmente la ganancia proporcional K_p, el tiempo integral T_i y el período de muestreo T de ese regulador, estos tres últimos valores en formato `double`. Los tiempos expresados en segundos. Nuestro sistema embebido responde enviando un byte con el valor 1.

- *Recepción de consigna*: se recibe un paquete que comienza con un byte con el valor 1, a continuación un byte que indica el número de regulador (0 a 4), y luego un `double` con el valor de consigna a establecer en ese regulador. Nuestro sistema embebido responde con un byte a 1.

- *Petición de datos*: se recibe un byte con el valor 2 y a continuación un byte que indica el número de regulador (0 a 4) para solicitar a ese regulador el valor de actuación y el valor de la señal de salida de la planta. Nuestro sistema embebido devuelve un paquete de bytes con dos `double` con esos valores, en ese orden.

Si es necesario y siempre que sea posible, usa herramientas de sincronización para codificar operaciones de espera.

Una solución:

$$u_k = K_p \left(e_k + \frac{1}{T_i} T \sum_{i=0}^{k} e_k \right)$$

$$u_{k-1} = K_p \left(e_{k-1} + \frac{1}{T_i} T \sum_{i=0}^{k-1} e_k \right)$$

$$u_k = u_{k-1} + K_p \left(e_k - e_{k-1} + \frac{T}{T_i} e_k \right)$$

$$u_k = u_{k-1} + a e_k - K_p e_k$$

$$a = K_p \left(1 + \frac{T}{T_i} \right)$$

```
double Kp[5], Ti[5], T[5];  // Parámetros de cada PI
double consigna[5], actuacion[5], salida[5];  // Consigna, actuación y salida de la planta para cada PI
int consignaRecibida[5] = { 0, 0, 0, 0, 0 };  // Indican si se recibió la primera consigna de cada PI
semaforo sParametrosRecibidos[5] = { 0, 0, 0, 0, 0 };  // Semáforos espera recepción de parámetros
semaforo sConsignaRecibida[5] = { 0, 0, 0, 0, 0 };  // Semáforos espera primera consigna de cada PI
semaforo sConsigna[5] = { 1, 1, 1, 1, 1 };  // Para región crítica de acceso a la consigna de cada PI
semaforo sDatos[5] = { 1, 1, 1, 1, 1 };  // Para región crítica de acceso a la actuación y salida

void hiloComunicacion() {  // Hilo que se encarga de las comunicaciones

    Canal canal;  // Crea un objeto para manejar las comunicaciones
    uint8_t buffer[100];  // Buffer para recibir y enviar bytes

    establecePrioridad(10);  // Establece una prioridad más baja para este hilo de comunicación
```

```
    while (1) {  // Repite indefinidamente ...

        int nRecibidos = canal.recibe(buffer);  // Espera hasta recibir un paquete de bytes
        int nRegulador = buffer[1];  // Número de PI

        switch (buffer[0]) {  // En función del valor del primer byte ...

        case 0: // Recepción de los parámetros de un PI

            Kp[nRegulador] = *(double*)(buffer + 2);  // Recoge la ganancia
            Ti[nRegulador] = *(double*)(buffer + 2 + sizeof(double));  // Recoge el tiempo integral
            T[nRegulador] = *(double*)(buffer + 2 + 2 * sizeof(double));  // Recoge período de muestreo
            signal(sParametrosRecibidos[nRegulador]);  // Indica que se recibieron parámetros del PI

            buffer[0] = 1;
            canal.envia(buffer, 1);  // Envía la respuesta

            break;

        case 1: // Recepción de la consigna de un PI

            wait(sConsigna[nRegulador]);  // Entra en región crítica
            consigna[nRegulador] = *(double*)(buffer + 1);  // Modifica la consigna de un regulador
            signal(sConsigna[nRegulador]);  // Sale de región crítica

            // Si no se había recibido la primera consigna, avisa que ya se ha recibido
            if (!consignaRecibida[nRegulador]) {
                signal(sConsignaRecibida[nRegulador]);
                consignaRecibida[nRegulador] = 1;
            }

            buffer[0] = 1;
            canal.envia(buffer, 1);  // Envía la respuesta

            break;

        case 2:  // Petición de actuación y salida de la planta para un PI

            double* p = (double*)buffer;  // Para guardar valores double en el buffer

            wait(sDatos[nRegulador]);  // Entra en región crítica
            p[0] = actuacion[nRegulador];  // Guarda la actuación del PI
            p[1] = salida[nRegulador];  // Guarda la salida de la planta para ese PI
            signal(sDatos[nRegulador]);  // Salida de región crítica

            canal.envia(buffer, 2 * sizeof(double));  // Envía la actuación y salida
        }
    }
}

void hiloPI(int i) {  // Hilo para implantar el PI i-ésimo
    double consignaLocal;
    establecePrioridad(11);  // Establece una prioridad mayor que la del hilo de comunicaciones

    wait(sParametrosRecibidos[i]);  // Espera hasta que se reciban los parámetros
    wait(sConsignaRecibida[i]);  // Espera hasta que se reciba la primera consigna

    double a = Kp[i] * (1 + T[i] / Ti[i]);  // Coeficiente de la ecuación en diferencias
    double e1 = 0;  // Error anterior
    double u1 = 0;  // Actuación anterior

    Temporizador t(T[i] * 1000);  // Inicializa el temporizador del PI con su período de muestreo
```

```
    while (1) {  // Repite indefinidamente ...

        t.espera();  // Espera hasta la siguiente temporización

        wait(sConsigna[i]);  // Entra en región crítica
        consignaLocal = consigna[i];  // Hace una copia de la consigna
        signal(sConsigna[i]);  // Sale de la región crítica

        double y = entradaAnalogica(i);  // Mide la salida de la planta
        double e = consignaLocal - y;  // Error de seguimiento de la consigna
        double u = u1 + a * e - Kp[i] * e1;  // Actuación generada por el PI

        if (u > 10) u = 10;
        else if (u < -10) u = -10;
        // Satura a los límites de la salida analógica

        salidaAnalogica(i, u);  // Aplica la actuación a la planta

        wait(sDatos[i]);  // Entra en región crítica
        salida[i] = y;  // Guarda la salida de la planta
        actuacion[i] = u;  // Guarda la actuación
        signal(sDatos[i]);  // Sale de la región crítica
        e1 = e;  // Guarda el valor anterior del error
        u1 = u;  // Guarda el valor anterior de la actuación
    }
}

void main() {
    for (int i = 0; i < 5; i++) salidaAnalogica(i, 0);  // Inicialmente actuación a 0
    concurrente {   // Lanza concurrentemente ...
        hiloComunicacion();  // Un hilo para atender a las comunicaciones
        for (int i = 0; i < 4; i++) hiloPI(i);  // Un hilo por cada PI
    }
}
```

---

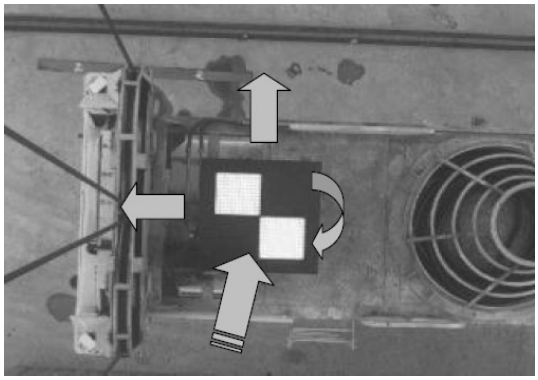
ej26) En un puente grúa existe un sistema embebido que dispone de una cámara en el carro superior apuntando hacia abajo para tomar imágenes de dos elementos reflectores cuadrados dispuestos en la plataforma de las garras tal y como se muestra en la figura. La cámara tiene una resolución de 1280x1024 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.

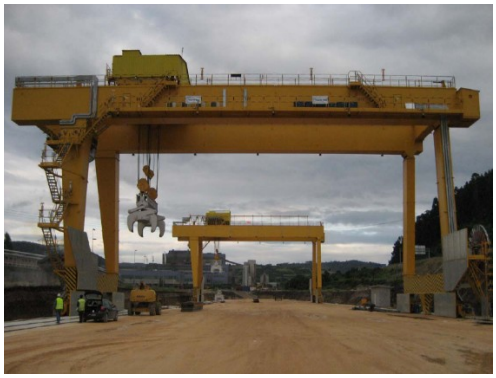
Se dispone de la función `void capturaImagen(uint8_t* p)` que acciona la cámara, espera a que tome una imagen y a que se almacene en la dirección indicada por parámetro. Cada punto de la imagen se guarda en un byte en blanco y negro expresando su nivel de gris (desde 0=negro hasta 255=blanco), recorriendo 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).

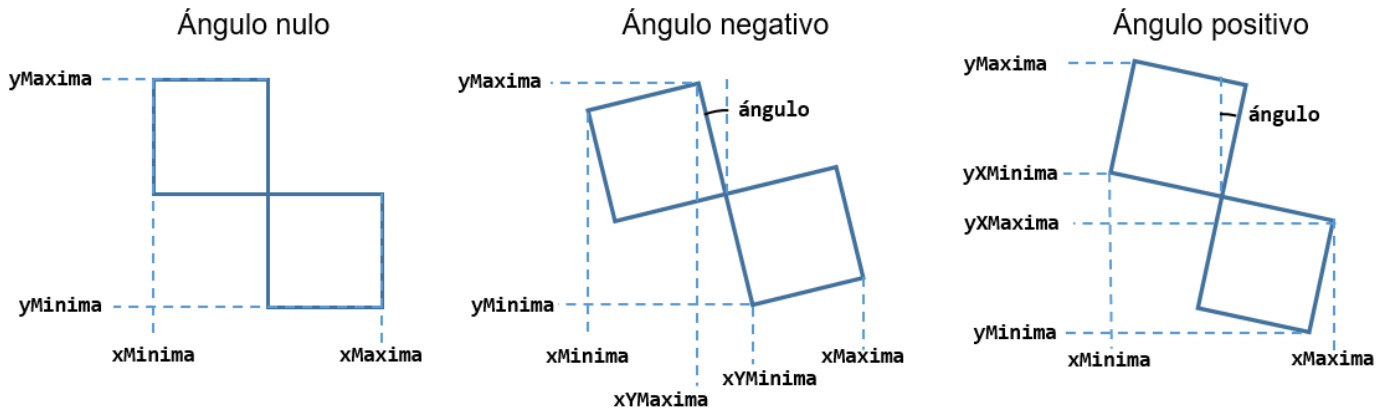
Existe un canal de comunicaciones en el que nuestro sistema embebido actúa como esclavo. Cuando se recibe un paquete con un único byte con valor 1, hay que contestar inmediatamente con un paquete de bytes donde se envían los desplazamientos y el ángulo de la carga en tres `float` obtenidos en el último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

Existe la clase `Canal` para manejar las comunicaciones, con 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 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:




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

semaforo s = 1;  // Semáforo para establecer una región crítica
float datos[3];  // Datos a enviar: desplazamientos horizontal y vertical y ángulo

void hCamara() {  // Hilo de detección del movimiento
    uint8_t imagen[1280 * 1024];  // Matriz para reccoger la imagen
    int xMinima, xMaxima, xYMinima, xYMaxima;
    int yMinima, yMaxima, yXMinima, yXMaxima;
    int x, y;

    while (1) {

        capturaImagen(imagen);  // Solicita una nueva imagen a la cámara

        xMinima = 1280;
        xMaxima = 0;
        yMinima = 1024;
        yMaxima = 0;
        // Inicializa estas variables con valores extremos
```

```
        for (x = 0; x < 1280; x++)  // Para cada coordenada X en la imagen
            for (y = 0; y < 1024; y++) {  // Para cada coordenada Y en la imagen
                if (imagen[x + y * 1280] == 255) {
                    if (x < xMinima) {  // Obtiene coordenadas del punto con X menor
                        xMinima = x;
                        yXMinima = y;
                    }
                    if (x > xMaxima) { // Obtiene coordenadas del punto con X mayor
                        xMaxima = x;
                        yXMaxima = y;
                    }

                    if (y < yMinima) { // Obtiene coordenadas del punto con Y menor
                        yMinima = y;
                        xYMinima = x;
                    }
                    if (y > yMaxima) { // Obtiene coordenadas del punto con Y mayor
                        yMaxima = y;
                        xYMaxima = x;
                    }
                }
            }

        double xCentro = (xMaxima - xMinima) / 2;
        double yCentro = (yMaxima - yMinima) / 2;
        // Coordenadas del punto de contacto de los cuadrados

        wait(s);  // Entra en región crítica
        datos[0] = (1280 / 2 - xCentro) * 0.2 / 50;  // Desplazamiento longitudinal en m.
        datos[1] = (1024 / 2 - yCentro) * 0.2 / 50;  // Desplazamiento transversal en m.
        if (xMaxima - xMinima == 100 && yMaxima - yMinima == 100)
            datos[2] = 0;  // Ángulo de giro nulo
        else if (xMaxima - xMinima > yMaxima - yMinima)
            datos[2] = -atan2(xYMinima - xYMaxima, yMaxima - yMinima);  // Ángulo de giro negativo
        else datos[2] = atan2(yXMinima - yXMaxima, xMaxima - xMinima);  // Ángulo de giro positivo
        signal(s);  // Sale de región crítica
    }
}

void hComunicacion() {  // Hilo que atiende a las comunicaciones
    Canal canal();  // Objeto para la comunicación
    float datosEnviados[3];  // Copia de los datos para enviarlos
    uint8_t peticion[1];  // Petición recibida
    while (1) {
        canal.recibe(peticion);  // Espera a recibir una petición de datos
        wait(s);  // Entra en región crítica
        memcpy(datosEnviados, datos, 3 * sizeof(float));  // Copia los datos a enviar
        signal(s);  // Sale de la región crítica
        canal.envia(datosEnviados, 3 * sizeof(float));  // Envía los datos
    }
}

void main() {
    concurrente {
        hCamara();  // Hilo para procesado de imágenes y detección de movimiento
        hComunicacion();  // Hilo para atender a las comunicaciones
    }
}
```

---

ej27) Codifica una aplicación para que un operario pueda manejar 10 máquinas idénticas (numeradas de la 1 a la 10), cada una de ellas controlada por un autómata programable.

Si existe alguna máquina libre (inicialmente ninguna está ocupada), la aplicación le indica al operario con la función `void visualiza(char* mensaje)` qué máquina se ha seleccionado y se le pide al operario

mediante la función `int introduce()` el número de fichero que contiene los datos de producción que hay que transferir al autómata que controla la máquina, por ejemplo si introduce un `300` el fichero es `300.dat`.

Cuando existan varias máquinas libres, la aplicación selecciona la que menos trabajos haya realizado.

Con una llamada a `int ejecuta(int maquina, int fichero)` se envían los datos de producción del fichero a la máquina, se ejecuta el trabajo programado y cuando finaliza el trabajo la función finaliza su ejecución y devuelve un 0 si no hubo ninguna incidencia y devuelve un código diferente de 0 indicando el tipo de incidencia en el caso de que hubiese algún problema durante la ejecución del trabajo.

Hay que indicar al operario el resultado de los trabajos mediante llamadas a la función `visualiza`.
Para formatear mensajes en cadenas de caracteres se puede utilizar la función `sprintf`, como en el siguiente ejemplo:

```
int maquina = 2;
int fichero = 300;
char mensaje[30];
sprintf(mensaje, "Fichero %d en máquina %d", fichero, maquina);
// Guarda en mensaje el texto "Fichero 300 en máquina 2"
```

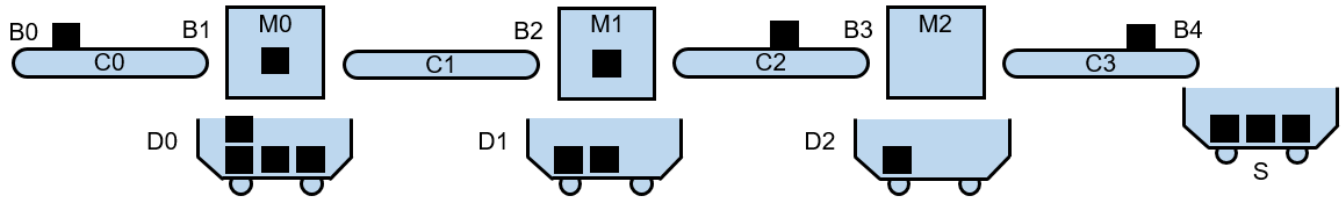
Una solución:

```
int ficheros[10];
int trabajos[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
int libre[10] = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
semaforo sOperario = 10;
semaforo sMaquina[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
semaforo sPantalla = 1;
semaforo sTrabajos = 1;

void hOperario() {
    while(1) {
        wait(sOperario);
        int fichero = introduce();
        int seleccionada = 0;
        while(!libre[seleccionada])
            seleccionada++;
        int i = seleccionada + 1;
        wait(sTrabajos);
        while (i < 10) {
            if (libre[i] && trabajos[i] < trabajos[seleccionada])
                seleccionada = i;
            i++;
        }
        libre[seleccionada] = 0;
        signal(sTrabajos);
        ficheros[seleccionada] = fichero;
        char mensaje[50];
        sprintf(mensaje, "Máquina seleccionada = %d", seleccionada + 1);
        wait(sPantalla);
        visualiza(mensaje);
        signal(sPantalla);
        signal(sMaquina[seleccionada]);
    }
}

void hMaquina(int maquina) {
    while(1) {
        wait(sMaquina[maquina]);
        int resultado = ejecuta(maquina, ficheros[maquina]);
        char mensaje[50];
        if (resultado == 0)
            sprintf(mensaje, "Fichero %d ejecutado en máquina %d", ficheros[maquina], maquina+1);
        else
            sprintf(mensaje, "Error %d en ejecución de fichero %d en máquina %d", resultado,
                ficheros[maquina], maquina+1);
```

```
        wait(sPantalla);
        visualiza(mensaje);
        signal(sPantalla);
        wait(sTrabajos);
        trabajos[maquina]++;
        libre[maquina] = 1;
        signal(sTrabajos);
        signal(sOperario);
    }
}

int main() {
    concurrente {
        hOperario();
        for(int i = 0; i < 10; i ++)
            hMaquina(i);
    }
    return 0;
}
```

---

ej30) Codifica un programa para un sistema embebido con sistema operativo de tiempo real multitarea que controla una cadena automatizada en la que se efectuan secuencialmente tres operaciones mediante tres máquinas M0, M1 y M2. Existen cuatro cintas transportadoras C0, C1, C2 y C3 que permiten transferir piezas entre la posición de entrada, las máquinas y la posición de salida. Ninguna cinta puede contener más que una pieza. Al final de todas las cintas existe una barrera fotoeléctrica (B1, B2, B3 y B4) que permite saber si llegó alguna pieza a esa posición. También existe otro detector B0 al comienzo de la primera cinta para comprobar si se depositó alguna pieza a la entrada del sistema.



Para manejar esta planta se disponen de las siguientes funciones:

- `void cinta(int cual, int movimiento)` hace que la cinta indicada como primer parámetro (0 a 3) se ponga en marcha o se detenga según se pase un buleano cierto o falso como segundo parámetro.
- `int barrera(int cual)` devuelve un buleano que es cierto cuando se detectó una pieza en la barrera correspondiente (0 a 4) y falso si no detecta pieza.
- `void entrada(int maquina)` hace que la máquina indicada por parámetro (0 a 2) recoja una pieza en el final de la cinta presente en su entrada. Dicha cinta tiene que estar detenida.
- `int procesado(int maquina)` hace que la máquina correspondiente (0 a 2) mecanice la pieza que ha recogido. Cuando finaliza su ejecución, devuelve un buleano que es cierto si la operación se efectuó sin ningún problema y falso cuando hubo algún error.
- `void salida(int maquina)` provoca que la máquina (0 a 2) deje la pieza en su cinta de salida (que tendrá que estar detenida).
- `void rechazo(int maquina)` activa un cilindro neumático en la máquina (0 a 2) que empuja la pieza a un contenedor de piezas defectuosas.

Cada máquina dispone de su contenedor de piezas defectuosas D0, D1 y D2, cada contenedor de 100 piezas de capacidad. En la salida de la planta existe otro contenedor S con la misma capacidad para recoger las piezas mecanizadas correctamente. Cuando se llene un contenedor, el programa tendrá que dar las indicaciones oportunas al operario para que lo sustituya por otro vacío. Para ello se dispone de una única pantala de cristal líquido afanumérica y un pulsador para toda la instalación.
Una llamada `void mensaje(char * cadena)` borra la pantalla y visualiza en ella la cadena de caracteres pasada por parámetro.
La función `int pulsador()` devuelve un buleano cierto mientras el operario está presionando el pulsador.

Una solución:

```
semaforo sCintaP[] = {1, 1, 1}; //Como solo las cintas 0, 1 y 2 actuan como productores, solo hay tres
cintas en la matriz
semaforo sMaquinaC[] = {0, 0, 0};

semaforo sMaquinaP[] = {1, 1, 1};
semaforo sCintaC[] = {0, 0, 0};  //Como solo las cintas 1, 2 y 3 actuan como consumidores, solo hay
tres cintas en la matriz

semaforo sMensaje = 1;//Region critica del mensaje en pantalla

void hCinta0(){
    while(1){
        cinta(0, 0);
        wait(sCintaP[0]);//Espera a que la cinta este vacia
        while(! barrera(0));
        cinta(0,1);
        while(! barrera(1));
        cinta(0,0);
        signal(sMaquinaC[0]);//Le indica a la siguiente maquina que ya tiene una pieza para recoger
    }
}

void hMaquina(int maquina){
    int numpiezas=0;
    char aviso[30];
    while(1){
        wait(sMaquinaC[maquina]);//Espera a que en la anterior cinta haya una pieza para recoger
        entrada(maquina);
        signal(sCintaP[maquina]);//Le indica a la anterior cinta que puede preparar otra pieza
        if(procesado(maquina)){
            wait(sMaquinaP[maquina]);//Espera a que la siguiente cinta este vacia
            salida(maquina);
            signal(sCintaC[maquina]);//Le indica a la siguiente cinta que ya tiene pieza
        }
        else{
            rechazo(maquina);
            numpiezas++;
            if(numpiezas == 100){
                sprintf(aviso, "Contenedor D %d lleno",maquina);
                wait(sMensaje);
                mensaje(aviso);
                while(! pulsador());//Mientras no se pulse el pulsador, esta maquina esta desactivada
                signal(sMensaje);
                numpiezas = 0;
            }
        }
    }
}

void hCinta(int c){
    while(1){
        cinta(c, 0);
        wait(sCintaP[c]);//Espera a que la cinta este vacia
        wait(sCintaC[c-1]);//Espera a que haya pieza en la cinta
        cinta(c,1);
        while(! barrera(c+1));
        cinta(c, 0);
        signal(sMaquinaC[c]);//Le indica a la siguiente maquina que ya tiene una pieza para recoger
        signal(sMaquinaP[c-1]);//Le indica a la maquina anterior que puede dejar una pieza
    }
}
```

```
void hCinta3(){
    int numpiezas = 0;
    while(1){
        cinta(3, 0);
        wait(sCintaC[2]);//Espera a que haya pieza en la cinta
        cinta(3,1);
        while(! barrera(4));
        while(barrera(4));
        cinta(3, 0);
        numpiezas++;
        signal(sMaquinaP[2]);//Le indica a la maquina anterior que puede dejar una pieza
        if(numpiezas == 100){
            wait(sMensaje);
            mensaje("Contenedor S lleno");
            while(! pulsador());//Mientras no se pulse el pulsador, esta cinta esta desactivada
            signal(sMensaje);
            numpiezas = 0;
        }
    }
}

int main() {
    concurrente {
        hCinta0();
        for(i=1; i<=2; i++) hCinta(i); //Hilos de las cintas 1 y 2 porque funcionan igual
        hCinta3();
        for(i=0; i<=2; i++) hMaquina(i); //Hilos de las maquinas porque funcionan igual
    }
}
```

---

ej31) En una nave industrial hay 20 robots móviles para transporte de mercancías. Existe una función `int ejecutaTransportes(int n)` a la que se le pasa el número de robot `n` (0 a 19) para que ese robot realice las operaciones de transporte de mercancías programadas para él. Esta función devuelve el número de transportes realizados. Cuando esta función finaliza, el robot está desocupado. La programación de los transportes la realiza otra aplicación, por lo que una llamada posterior a esta función puede implicar la ejecución de más transportes.
En la nave hay una sala donde se realiza el mantenimiento sobre robots desocupados y donde pueden permanecer hasta 4 robots simultáneamente. Un robot desocupado intentará entrar en esta sala cuando haya acumulado al menos 100 transportes desde su último mantenimiento. En un determinado instante sólo se puede realizar el mantenimiento sobre un robot, los otros robots que estén en la sala esperarán por su turno. Los mantenimientos se harán por orden temporal de concesión de permiso para realizarlo.
Si un robot que está fuera de la sala necesita mantenimiento cuando en la sala ya hay otros 4 robots, seguirá intentando ejecutar más transportes pendientes e intentará realizar el mantenimiento más adelante.
La función `void entrarSala(int n)` hace que el robot `n` (0 a 19) entre en la sala de mantenimiento y la función `void salirSala(int n)` hace que salga.
Durante la ejecución de la función `void realizaMantenimiento(int n)` un operario realiza el mantenimiento del robot `n`. Hay que ejecutar simultáneamente `void recibeMantenimiento(int n)` para gobernar a ese robot durante el mantenimiento. Cuando ambas funciones finalizan su ejecución, el mantenimiento terminó.
Los robots parten desocupados y ninguno en la sala de mantenimiento.

Una solución:

```
semaforo ms[20] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
int cola[4], carga = 0, descarga = 0;
semaforo sOtroRobot = 0, mutex = 1, sRobotTermina = 0, sOperarioTermina = 0;
int nRobotsEnMantenimiento = 0;
```

```
void hMantenimiento() {
    int nRobot;
    while(1) {
        wait(sOtroRobot);  // Espera a que haya otro robot en la sala de mantenimiento
        nRobot = cola[descarga];  // Quita de la cola el siguiente que hay que atender
        descarga++;  // Acutaliza la posición en la cola
        if (descarga == 4)
            descarga = 0;
        signal(ms[nRobot]);  // Despierta al robot para hacer su mantenimiento
        ejecutaMantenimiento(nRobot);  // Realiza el mantenimiento
        signal(sOperarioTermina);  // Avisa que el operario terminó
        wait(sRobotTermina);  // Espera a que el robot termine
    }
}

int mantenimientoEnRobot(int nRobot) {
// Intenta realizar el mantenimiento del robot indicado por parámetro. Devuelve un buleano
// cierto si lo hizo.

    while (1) {
        wait(mutex);
        if (nRobotsEnMantenimiento == 4) {  // Si ya hay 4 robots en la sala ...
            signal(mutex);
            return 0;  // Termina e indica que no se realizó el mantenimiento
        } else {
            nRobotsEnMantenimiento++;  // Reserva una posición en la sala
            signal(mutex);
            cola[carga] = nRobot;  // Pone el root en la cola de mantenimiento
            carga++;  // Actualiza la posición de carga en al cola
            if (carga == 4)
                carga = 0;
            entrarSala(nRobot);  // El robot va a la sala y entra
            signal(sOtroRobot);  // Avisa al operario que ya hay otro robot
            wait(ms[nRobot]);  // Espera a que el operario atienda a este robot
            recibeMantenimiento(nRobot);  // El robot recibe el mantenimiento
            signal(sRobotTermina);  // Avisa de que ya finalizó
            wait(sOperarioTermina); // Espera a que el perario termine
            salirSala(nRobot);  // El robot sale de la sala
            wait(mutex);
            nRobotsEnMantenimiento--; // Indica que el robot deja la sala
            signal(mutex);
            return 1;
        }
    }
}

void hRobot(int nRobot) {
    int transportes = 0;
    while (1) {
        if (transportes >= 100)  // Si hizo 100 transportes o más
            if (mantenimientoEnRobot(nRobot))  // Si hizo el mantenimiento
                transportes = 0;  // Reset de contador de transportes
        transportes += ejecutaTransportes(nRobot);  // Realiza más transportes
    }
}

int main() {
    concurrente {
        hMantenimiento();  // Un hilo para el operario de mantenimiento
        for (int nRobot = 0; nRobot < 20; nRobot++)
            hRobot(nRobot);  // Un hilo por cada robot
    }
}
```

ej37) Codifica un programa que maneje y sincronice operaciones sobre dos robots Scara, una cinta transportadora y cuatro barreras de infrarrojos, de forma que, cuando se depositen paquetes en la barrera número 1, este sistema los transporte hacia la barrera número 4 (se supone que en una barrera determinada puede haber como máximo un paquete). Para ello, disponemos de las siguientes funciones:

`int barrera(int)` devuelve un buleano que es cierto cuando la presencia de un objeto interrumpe la barrera indicada por parámetro. Por ejemplo, en la situación de la figura, devuelve cierto sobre las barreras 1 y 3 y falso sobre la 2 y la 4. Hay que tener en cuenta que el hardware utilizado para acceder a estas señales no permite comprobar el estado de varias barreras al mismo tiempo.

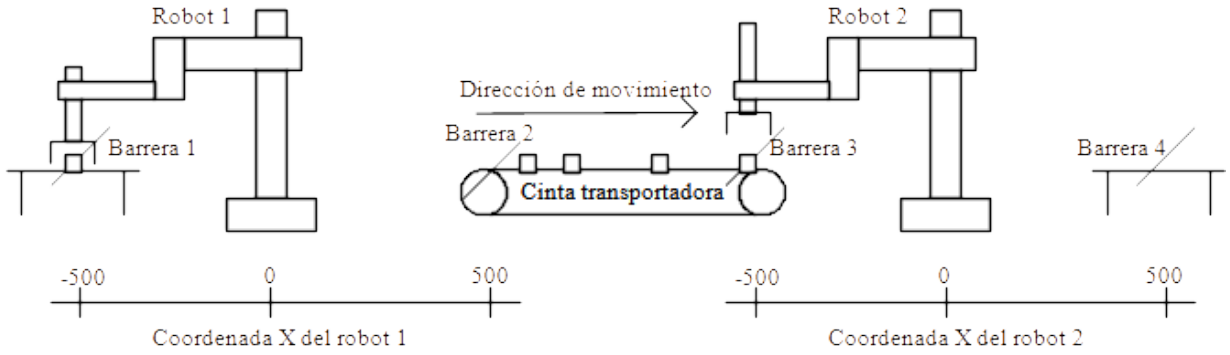
Si llamamos a `void cinta(int)` pasándole un buleano cierto, pone en movimiento la cinta transportadora (sólo se mueve en un sentido) y devuelve el control inmediatamente. Si le pasamos uno falso, para la cinta y devuelve el control inmediatamente. Esta función no es reentrante.

Mediante la función `void muevete(int, double)` podemos hacer que el robot (el 1 o el 2) indicado en el primer parámetro se mueva a la posición cuya coordenada X se indica en el segundo parámetro.

Llamando a `void sube(int)` subimos el último eslabón del robot indicado por parámetro y mediante `void baja(int)` lo bajamos. En la figura, el robot 1 está abajo y el 2 arriba.

Con `void abre(int)` abrimos la pinza del robot indicado por parámetro y `void cierra(int)` coge el objeto situado en su garra.

Todas las funciones para el manejo de los robots son reentrantes. Se pueden ejecutar llamadas concurrentes a la misma función siempre y cuando actúe sobre robots diferentes. Cualquiera de estas funciones de manejo de robots finalizan su ejecución después de que el robot correspondiente haya efectuado la operación ordenada.



Una solución:

```
semaforo scRobot2 = 0, spCinta = 1;  // Para productor/consumidor cinta a robot 2
semaforo scCinta = 0, spRobot1 = 1;  // Para productor/consumidor robot 1 a cinta
semaforo sPiezas = 0;  // Número de piezas en la cinta

void hRobot(int robot) {
    sube(robot);
    muevete(robot, -500);
    abre(robot);
    while(1) {
        if (robot == 1)
            while(! barrera(1));
        else wait(scRobot2);
        baja(robot);
        cierra(robot);
        sube(robot);
        if (robot == 2)
            signal(spCinta);
        muevete(robot, 500);
```

```
        if (robot == 1)
            wait(spRobot1);
        else while(barrera(4));
        baja(robot);
        abre(robot);
        if (robot == 1)
            signal(scCinta);
        sube(robot);
        muevete(robot, -500);
    }
}

void hCintaEntrada() {
    while(1) {
        wait(scCinta);
        signal(sPiezas);
        while(barrera(2));
        signal(spRobot1);
    }
}

void hCintaSalida() {
    while(1) {
        wait(sPiezas);
        wait(spCinta);
        cinta(1);
        while(! barrera(3));
        cinta(0);
        signal(scRobot2);
    }
}

int main() {
    concurrente {
        hRobot(1);
        hRobot(2);
        hCintaEntrada();
        hCintaSalida();
    }
}
```

---

ej44) En una planta industrial queremos distribuir las cajas recibidas mediante una cinta transportadora C0, a otras cinco cintas C1 a C5, de forma que siempre se le dé más prioridad a la que contenga menos cajas. Sobre cualquiera de las cintas puede haber varias cajas. En la entrada de la cinta C0 un operario carga cajas. El transporte de cajas desde el extremo de salida de C0 a la posición de entrada de las demás cintas C1 a C5 se realiza mediante un robot. A la salida de las cintas C1 a C5 hay operarios que extraen las cajas que llegan a esa posición. Se dispone de los siguientes recursos:

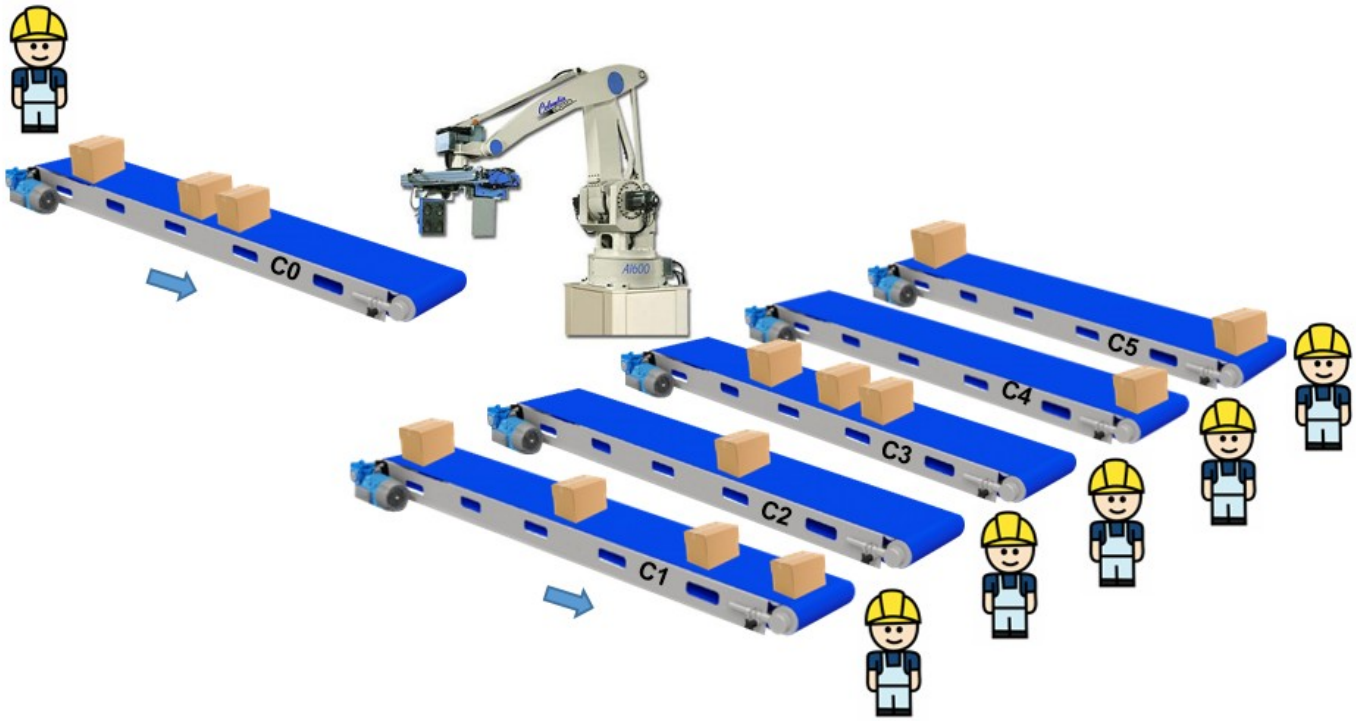
a) Al principio y al final de cada cinta existen sendas barreras fotoeléctricas para detectar la presencia de cajas en dichos puntos. Para consultar el estado de las barreras de entrada utilizaremos la función `int barreraEntrada(int n)` que nos devolverá un buleano que será cierto cuando se está detectando una caja en la barrera de entrada de la cinta Cn, 0<=n<=5. De la misma forma existe una función `int barreraSalida(int n)` para consultar las barreras de salida.

b) `void cinta(int n, int marcha)` pone en funcionamiento o detiene la cinta Cn según se le pase como segundo parámetro un buleano verdadero o falso, respectivamente. Después de poner en funcionamiento o detener una cinta, la llamada a esta subrutina devuelve el control inmediatamente.

c) Para accionar el robot disponemos de las siguientes funciones: `void muevete(int n)` situa la pinza a una cierta altura de seguridad sobre el final (n = 0) o el principio (n = 1, ..., 5) de la cinta Cn. Las funciones `void abre()` y `void cierra()` nos permiten abrir o cerrar la pinza. Mediante `void sube()` y `void baja()` podemos separarnos o acercarnos verticalmente a la cinta sobre la que está la pinza para coger

o depositar cajas. El robot puede depositar una caja en una cinta en movimiento pero no puede depositar una caja en una cinta en la que la posición de entrada está ocupada por otra caja. La llamada a cualquiera de estas subrutinas devuelve el control cuando el robot haya realizado la operación correspondiente.

Siempre que sea posible, las esperas deben realizarse mediante bloqueos con herramientas de sincronización y no con consultas repetidas que consumen tiempo de CPU. Por ahorro de energía, las cintas sobre las cuales no haya cajas no deben estar en movimiento.



Una solución:

```
int numPaquetesCita[6] = {0, 0, 0, 0, 0, 0};
semaforo sProductorCinta0aRobot = 0, sProductorCinta0aCintas = 5, sConsumidorRobot = 0;
semaforo sConsumidorCinta[5] = {0, 0, 0, 0, 0};
int cintaSeleccionada;

void cinta0 () {
int nuevoPaquete, barreraEntrada0Anterior;

barreraEntrada0Anterior = 0;
cinta(0, 1);
while (1) {
    nuevoPaquete = barreraEntrada(0) && ! barreraEntrada0Anterior;
    barreraEntrada0Anterior = barreraEntrada(0);
    if (nuevoPaquete) {
        numPaquetesCinta[0]++;
        if (numPaquetesCinta[0] == 1) cinta(0,1);
        }
    if (barreraSalida(0)) {
        cinta (0, 0);
        wait (sProductorCinta0aCintas);
        cintaEncontrada = 0;
```

```
        for (i = 1; i < 6; i++)
            if (! barreraEntrada(i)) {
                if (cintaEncontrada) {
                    if (numPaquetesCinta[cintaSeleccionada] > numPaquetesCinta[i])
                        cintaSeleccionada = i;
                    } else {
                        cintaSeleccionada = i;
                        cintaEncontrada = 1;
                        }
                    }
                }
        signal (sConsumidorRobot);
        wait (sProductorCinta0aRobot);
        nPaquetesCinta[0]--;
        if (nPaquetesCinta[0])
            cinta (0, 1);
        }
    }
}

void robot () {
int cinta;
abre ();
while (1) {
    robotMuevete(0);
    wait (sConsumidorRobot);
    cinta = cintaSeleccionada;
    baja();
    cierra();
    sube();
    signal (sProductorCinta0aRobot);
    muevete(cintaSeleccionada);
    baja();
    abre();
    sube();
    signal (sConsumidorCinta[cintaSeleccionada]);
    }
}

int entradaCinta (int nCinta) {
while (1) {
    wait (sConsumidorCinta[nCinta]);
    numPaquetesCinta[nCinta]++;
    signal (sConsumidorSalidaCinta[nCinta]);
    while (barreraEntrada[nCinta]);
    signal (sProductorCinta0aCintas);
    }
}

int salidaCinta (int nCinta) {
while (1) {
    wait (sConsumidorSalidaCinta[nCinta]);
    cinta (nCinta, 1);
    while (! barreraSalida(nCinta));
    cinta (nCinta, 0);
    while (barreraSalida(nCinta));
    }
}

main () {
concurrente
    {
    cinta0 ();
    robot ();
    for (i = 1; i <= 5; i ++) {
        entradaCinta(i);
        salidaCinta(i);
        }
```

```
    }
}
```

---

ej46) En una instalación industrial existe una cuba para realizar un tratamiento electrolítico sobre piezas metálicas. Para ello se dispone de 8 robots, numerados desde el 0 al 7, dispuestos alrededor de la cuba, tal y como se muestra en la figura.

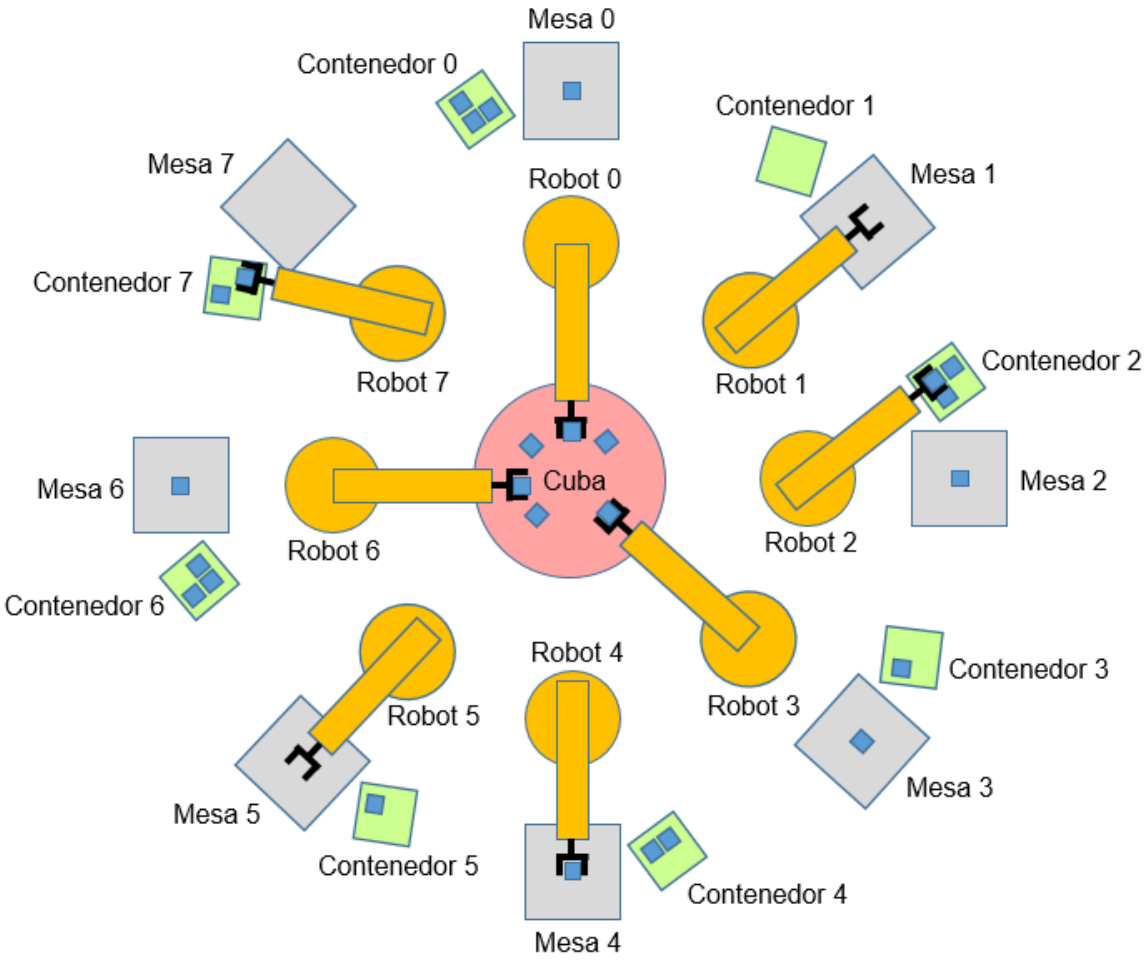
Por cada robot existe una mesa donde un operario deposita una nueva pieza a tratar. Disponemos de la función `int piezaEnMesa(int)` a la que le pasamos el número de robot y que nos devuelve un buleano cierto si un sensor en la mesa detecta pieza.

A continuación el robot tiene que coger la pieza y depositarla en la cuba, pero por limitaciones de espacio, no puede ir a la cuba mientras estén en la cuba el robot situado a su izquierda y/o el que está a su derecha. Por ejemplo, mientras el robot 3 está en la cuba, no pueden ir allí los robots 2 ni 4.

Una vez depositada la pieza en la cuba, el robot se aparta a su mesa y espera 10 s. a que finalice el tratamiento electrolítico. Cuando sea posible, vuelve a la cuba para recoger su pieza para depositarla luego en su contenedor de piezas tratadas, cada robot tiene un contenedor propio.

Para accionar el robot disponemos de varias subrutinas a las que le pasaremos el número de robot: `void abre(int)` abre la pinza, `void cierra(int)` cierra la pinza, `void baja(int)` baja la pinza para recoger o depositar piezas, `void sube(int)` sube la pinza, `void mueveteAMesa(int)` mueve el robot en horizontal para posicionarlo sobre su mesa, `void mueveteACuba(int)` lo posiciona sobre la cuba, `void mueveteAContenedor(int)` lo posiciona sobre su contenedor.

La subrutina `void sleep(int)` bloquea la ejecución durante el tiempo pasado por parámetro, expresado en ms.



Una solución:

```
semaforo sRobot[8] = {1, 1, 1, 1, 1, 1, 1, 1}, sCuba = 7;

void robot (int nRobot) {

    int nRobotAnterior;
    if (nRobot == 0)
        nRobotAnterior = 7;
        else nRobotAnterior = nRobot-1;

    sube(nRobot);
    abre(nRobot);
    mueveteAMesa(nRobot);

    while(1) {

        while(! piezaEnMesa(nRobot));

        baja(nRobot);
        cierra(nRobot);
        sube(nRobot);

        wait(sCuba);
        wait(sRobot[nRobot]);
        wait(sRobot[nRobotAnterior]);

        mueveteACuba(nRobot);
        baja(nRobot);
        abre(nRobot);
        sube(nRobot);
        mueveteAMesa(nRobot);

        signal(sRobot[nRobot]);
        signal(sRobot[nRobotAnterior]);
        signal(sCuba);

        sleep(10000);

        wait(sCuba);
        wait(sRobot[nRobot]);
        wait(sRobot[nRobotAnterior]);

        mueveteACuba(nRobot);
        baja(nRobot);
        cierra(nRobot);
        sube(nRobot);
        mueveteAContenedor(nRobot);

        signal(sRobot[nRobot]);
        signal(sRobot[nRobotAnterior]);
        signal(sCuba);

        baja(nRobot);
        abre(nRobot);
        sube(nRobot);
        mueveteAMesa(nRobot);
    }
}

void main() {
    concurrente {
        robot(0);
        robot(1);
```

```
        robot(2);
        robot(3);
        robot(4);
        robot(5);
        robot(6);
        robot(7);
    }
}
```

---

ej47) En una planta industrial existen 5 carros autoguiados que realizan operaciones de transporte de piezas desde una posición de carga (C1 a C5) a una posición de descarga (D1 a D5). Cada carro tiene su propia vía. Existen cruces 1 a 9 entre las vías. Cada carro repite contínuamente un ciclo consistente en un movimiento de ida en su vía para transportar piezas desde su posición de carga a su posición de descarga y un movimiento de vuelta en la misma vía para volver a la posición de carga para recoger más piezas. Por ejemplo, el carro 1 recoge piezas en su posición de carga C1, tiene un movimiento de ida que pasa por los cruces 1, 2, 6, 7 y 9, llega a su posición de descarga D1, descarga las piezas y luego tiene un movimiento de vuelta pasando por los mismos cruces en orden inverso, para llegar a C1 para recoger más piezas y repetir contínuamente el mismo ciclo de carga/descarga.

Cada carro dispone de un lector RFID mediante radiofrecuencia para detectar y leer tags dispuestos a lo largo de su vía con el objetivo de conocer cuándo llega a una posición determinada. Cada tag tiene grabado un valor numérico. El lector puede detectar y leer tags hasta una distancia de 10 cm. Para el carro n-ésimo (n=1...5) los tags dispuestos en su posición de carga Cn y descarga Dn tienen grabado el número de carro n más 10, por ejemplo, el valor 14 para el carro 4. En cada vía existe un tag 2 metros antes de cada cruce y otro tag 2 metros después de cada cruce que guardan un valor numérico igual al número de cruce el primero y ese mismo valor cambiado de signo el segundo, cuando el movimiento es de ida, con el objetivo de determinar cuándo se aproxima un carro a un cruce y cuándo se aleja, para tratar y evitar posibles colisiones. En la figura se muestran los tags del cruce 6.

Se dispone de las siguientes subrutinas para manejar el sistema:

- `void ida(int nCarro)`: activa el movimiento en sentido de ida al carro indicado por parámetro. Esta subrutina devuelve el control inmediatamente.

- `void vuelta(int nCarro)`: activa el movimiento en sentido de vuelta al carro indicado por parámetro. Esta subrutina devuelve el control inmediatamente.

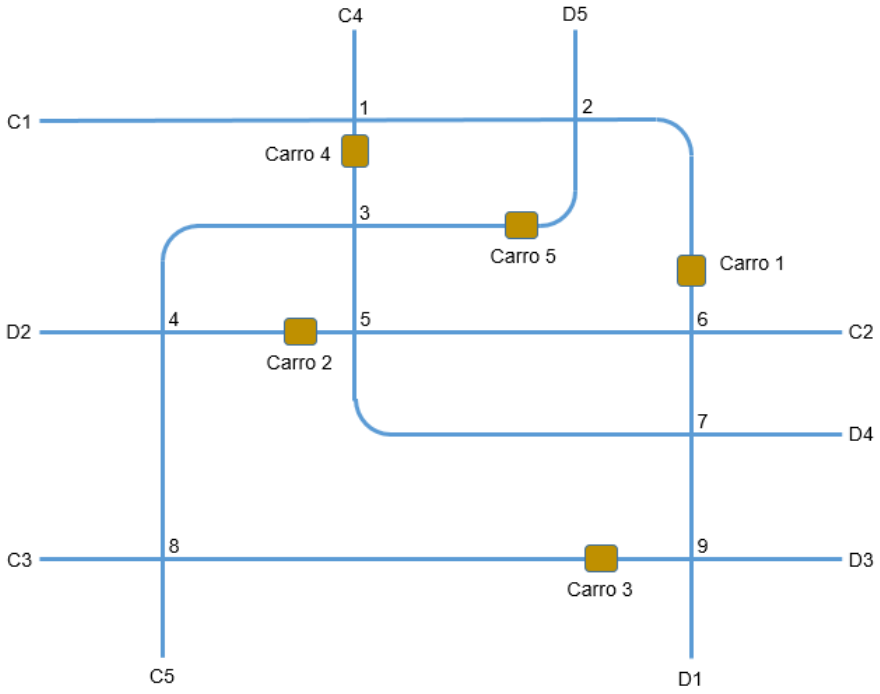
- `void parada(int nCarro)`: detiene el movimiento del carro indicado por parámetro y devuelve el control inmediatamente.

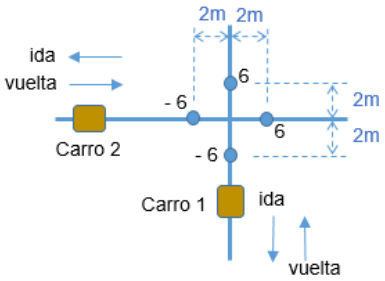
- `void carga(int nCarro)`: mientras se ejecuta su código, realiza la carga de piezas en el carro indicado por parámetro. Cuando finaliza su ejecución, las piezas ya han sido cargadas.

- `void descarga(int nCarro)`: mientras se ejecuta su código, realiza la descarga de piezas en el carro indicado por parámetro. Cuando finaliza su ejecución, las piezas ya han sido descargadas.

- `int leeRFID(int nCarro)`: devuelve el control cuando el lector RFID del carro indicado por parámetro ha detectado y leído un tag. Devuelve el valor numérico guardado en el tag.

Elabora un programa que controle todo el sistema. Codifica en una única subrutina el algoritmo que tienen que ejecutar los hilos que controlan los carros. Codifica las esperas por sincronizaciones mediante herramientas específicas de sincronización y no mediante consulta de variables.





Una solución:

```
int nCrucesCarro[5] = { 5, 3, 2, 4, 4 };
semaforo s[9] = { 1, 1, 1, 1, 1, 1, 1, 1, 1 };

void hiloCarro(int nCarro, int nCruces) {
// Hilo para todos los carros

    int i, nCruce;
    while (1) {
        cargaCarro(nCarro);
        ida(nCarro);
        while (leeTag(nCarro) == nCarro + 10);
        // Hasta leer el tag del primer cruce
        for (i = 0; i < nCruces; i++) {  // Para cada cruce ...
            nCruce = leeTag(nCarro);  // Obtén num cruce
            parada(nCarro);  // Parar
            wait(s[nCruce-1]);  // Comprobar si puede cruzar
            ida(nCarro);  // Adelante
            while (leeTag(nCarro) == nCruce);  // Hasta leer el tag de salida del cruce
            signal(s[-leeTag(nCarro)-1]);  // Liberar cruce
            while (leeTag(nCarro) == -nCruce);  // Hasta leer el siguiente tag
        }
        parada(nCarro);
        descarga(nCarro);  // Para y descarga
        vuelta(nCarro);  // Vuelta
        while (leeTag(nCarro) == nCarro + 10);
        // Hasta leer el tag del primer cruce
        for (i = 0; i < nCruces; i++) {  // Para cada cruce ...
            nCruce = -leeTag(nCarro);  // Obtén num cruce
            parada(nCarro);  // Parar
            wait(s[nCruce-1]);  // Comprobar si puede cruzar
            vuelta(nCarro);  // Adelante
            while (leeTag(nCarro) == - nCruce);  // Hasta leer el tag de salida del cruce
            signal(s[-leeTag(nCarro)-1]);  // Libera cruce
            while (leeTag(nCarro) == nCruce);  // Hasta leer el siguiente tag
        }
        parada(nCarro);  // Para
    }
}
```

```
void main() {
    int iCarro;
    concurrente {
        for (iCarro = 1; iCarro <= 5; iCarro++)
            hiloCarro(iCarro, nCrucesCarro[iCarro-1]);
    }
}
```

---

ej52) Codifica una aplicación en C++ para un sistema embebido con el objetivo de implantar un servidor bajo protocolo UDP, recibiendo peticiones de un cliente en el puerto 10000, para poder disponer de un regulador PI que se pueda operar de forma remota en una red de area local.

El regulador puede estar en uno de estos estados:

- **`Reposo`**: estado inicial. La salida del regulador es 0. En este estado es posible enviarle desde el cliente los parámetros del regulador PI: la ganancia K_p, el tiempo integral T_i y el período de muestreo T. El cliente también puede enviarle la consigna.

- **`Operación`**: el regulador está en funcionamiento y aplica una actuación a la planta calculada mediante un algoritmo PI con los parámetros enviados previamente. En este estado no se permite modificar los parámetros del regulador, pero sí la consigna.

Se dispone de las siguientes funciones:

- **`void establecePrioridad(int prioridad)`**: establece la prioridad del hilo que realiza la llamada, indicando un valor de prioridad por parámetro, de 0 a 255, cuanto más álto mayor prioridad.

- **`double entradaAnalogica()`**: mide la salida de la planta a controlar.

- **`void salidaAnalogica(double valor)`**: establece la salida del regulador.

Se dispone de la clase **`ServidorUDP`** para implantar la comunicación UDP con los siguientes métodos públicos:

- **`ServidorUDP::ServidorUDP(int puerto)`**: constructor que reserva el puerto indicado por parámetro para poder recibir datagramas.

- **`int ServidorUDP::recibe(uint8_t * buffer, uint16_t tamano)`**: espera bloqueado a recibir (por el puerto indicado previamente al constructor) un datagrama con información que guarda a partir de **`buffer`** que dispone de un tamaño de **`tamano`** bytes. Devuelve el número de bytes recibidos.

- **`void ServidorUDP::envia(uint8_t * p, uint16_t cantidad)`**: envía al cliente un datagrama con **`cantidad`** bytes recogidos a partir de **`p`**. El cliente es el que realizó el último envío a este servidor.

Se dispone de la clase **`Temporizador`** con los siguientes métodos públicos:

- **`Temporizador::Temporizador()`**: constructor para crear un temporizador.

- **`void Temporizador::configura(uint32_t s, uint32_t ns)`**: configura el período que se utiliza para generar temporizaciones contínuas cuando se ha arrancado el temporizador. Es la suma de **`s`** segundos y **`ns`** nanosegundos.

- `void Temporizador::arranca()`: arranca el temporizador.

- `void Temporizador::para()`: para el temporizador.

- `void Temporizador::espera()`: espera bloqueado a que se produzca una temporización.

A cada petición enviada por un cliente, el servidor envía una respuesta. Las posibles peticiones y respuestas son las siguientes:

- *Envío de parámetros del regulador*: el cliente envía un datagrama que contiene un byte a 0, la ganancia proporcional K_p, el tiempo integral T_i y el período de muestreo T, estos tres últimos en formato `double`. Si se pudieron configurar los parámetros del PI, el servidor devuelve un datagrama con un byte a 1, si no devuelve un byte a 0.

- *Envío de consigna*: el cliente envía un datagrama con un byte con el valor 1 y un `double` con el valor de consigna a establecer en el regulador. El servidor devuelve un byte a 1.

- *Arranque*: el cliente envía un byte con el valor 2 para que el regulador pase a estado `Operación`. El regulador pasa a ese estado si previamente se le ha enviado los parámetros del regulador y la consigna. Si se pudo realizar el arranque, el servidor devuelve un byte a 1, si no devuelve un byte a 0.

- *Parada*: el cliente envía un byte con el valor 3 para que el regulador pase a estado `Reposo`. Si se hizo la parada, el servidor devuelve un byte a 1, si no devuelve un byte a 0.

- *Medidas*: el cliente envía un byte con el valor 4 para solicitar al regulador el valor actuación de actuación y el valor actual de la señal de salida de la planta. El servidor devuelve en un datagrama dos `double` con esos valores, en ese orden.

Siempre que sea posible, usa herramientas de sincronización para codificar operaciones de espera.

Una solución:

$$u_k = K_p e_k + K_i T \sum_{i=0}^{k} e_k$$

$$u_{k-1} = K_p e_{k-1} + K_i T \sum_{i=0}^{k-1} e_k$$

$$u_k = u_{k-1} + K_p(e_k - e_{k-1}) + K_i T e_k$$

$$u_k = u_{k-1} + a e_k - K_p e_{k-1}$$

$$a = K_p + K_i T$$

```
uint8_t buffer[100];
ServidorUDP servidor(10000);

double Kp, Ti, T, consigna, actuacion, salida;
int consignaRecibida = 0;
int parametrosRecibidos = 0;
int controlar = 0;
semaforo sDatos = 1, sArranca = 0;

void main() {
    hiloComunicacion();
```

```
    hiloControl();
}

void hiloComunicacion() {
    uint8_t respuesta;

    establecePrioridad(10);

    while (1) {
        int nRecibidos = servidor.recibe(buffer, 100);
        switch (buffer[0]) {
        case 0: // Envío de parámetros
            wait(sDatos);
            if (controlar)
                respuesta = 0;
            else {
                Kp = *(double*)(buffer + 1);
                Ti = *(double*)(buffer + 1 + sizeof(double));
                T = *(double*)(buffer + 1 + 2 * sizeof(double));
                parametrosRecibidos = 1;
                respuesta = 1;
            }
            signal(sDatos);
            servidor.envia(&respuesta, 1);
            break;
        case 1: // Envío de consigna
            wait(sDatos);
            consigna = *(double*)(buffer + 1);
            consignaRecibida = 1;
            respuesta = 1;
            signal(sDatos);
            servidor.envia(&respuesta, 1);
        case 2: // Arranca control
            if (parametrosRecibidos && consignaRecibida && !controlar) {
                wait(sDatos);
                controlar = 1;
                signal(sDatos);
                signal(sArranca);
                respuesta = 1;
            }
            else respuesta = 0;
            servidor.envia(&respuesta, 1);
            break;
        case 3: // Para control
            wait(sDatos);
            if (controlar) {
                controlar = 0;
                respuesta = 1;
            }
            else respuesta = 0;
            signal(sDatos);
            servidor.envia(&respuesta, 1);
            break;
        case 4: // Pide datos
            wait(sDatos);
            double * p = (double *)buffer;
            p[0] = actuacion;
            p[1] = salida;
            signal(sDatos);
            servidor.envia(buffer, 2 * sizeof(double));
            }
        }
    }
}
```

```
void hiloControl() {
    Temporizador t;
    double KpLocal, TiLocal, TLocal, consignaLocal;

    establecePrioridad(11);

    while (1) {
        t.para();

        wait(sArranca);

        wait(sDatos);
        KpLocal = Kp;
        TiLocal = Ti;
        TLocal = T;
        consignaLocal = consigna;
        signal(sDatos);

        int32_t segundos = TLocal;
        int32_t nanosegundos = (TLocal - segundos) * 10e9;
        t.configura(segundos, nanosegundos);
        t.arranca();

        double a = KpLocal * (1 + TLocal / TiLocal);
        double e1 = 0;
        double u1 = 0;

        int controlarLocal = 1;
        while (controlarLocal) {
            t.espera();
            double y = entradaAnalogica();
            double e = consignaLocal - y;
            double u = u1 + a * e - KpLocal * e1;
            salidaAnalogica(u);
            e1 = e;
            u1 = u;
            wait(sDatos);
            salida = y;
            actuacion = u;
            controlarLocal = controlar;
            signal(sDatos);
        }
    }
}
```

---

ej53) En una instalación automatizada existen 13 máquinas de procesado de piezas (de la máquina 0 a la 12) y 5 carros autónomos (del carro 0 al 4).

Hay una posición donde se pueden depositar dos contenedores A y B simultáneamente mediante la llamada a la función `int entradaContenedores()`. El contenedor A trae un cierto número de piezas que hay que procesar, esta función devuelve el número de piezas que vienen en A. El contenedor B viene vacío. Cada pieza hay que procesarla una sóla vez en una de las máquinas. Todas las máquinas y procesados son iguales. Cada máquina puede contener sólo una pieza. Mientras se ejecuta una llamada a la función `void procesa(int)` se está procesando una pieza en la máquina indicada por parámetro. Una vez se ha procesado una pieza, hay que llevarla al contenedor B. Una vez se han procesado todas las piezas y se han llevado al contenedor B, hay que retirar ambos contenedores simultaneamente mediante una llamada a la función `void salidaContenedores()`. Una vez retirados los contenedores, hay que volver a repetir todo el proceso.

Para mover las piezas entre los contenedores y las máquinas se utilizan los carros.

Mientras se ejecuta una llamada a `void cargaMaquina(int, int)`, el carro indicado en el primer parámetro está llevando una pieza del contenedor A a la máquina indicada en el segundo parámetro.
Mientras se ejecuta una llamada a la función `void descargaMaquina(int, int)`, el carro indicado en el primer parámetro está llevando una pieza de la máquina indicada en el segundo parámetro al contenedor B.

Se dará más prioriodad a la carga de las máquinas que a la descarga. Es decir, si hay algunas máquinas libres y hay algunas máquinas con piezas procesadas, los carros se utilizarán primero para cargar las máquinas libres.

En el caso de que un carro tenga que coger una pieza en el contenedor A y existan varias máquinas libres, llevará la pieza a la máquina que haya hecho menos procesados. En el caso de que un carro vaya a recoger una pieza para llevarla al contenedor B y haya varias piezas procesadas posibles a elegir, recogerá la pieza de la máquina que menos procesados haya realizado.

Considera que las operaciones de lectura o escritura en números enteros son atómicas. Codifica las esperas utilizando herramientas de sincronización.

Una solución:

```
semaforo sEntradas = 0, sSalidas = 0, sCarros = 0;
semaforo sMaquinaCargada[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
semaforo sMaquinaDescargada[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
semaforo sBuscaLibre = 1, sBuscaFinalizada = 1;

int entradaContenedores();
void salidaContenedores();
void procesa(int nMaquina);
void cargaMaquina(int nCarro, int nMaquina);
void descargaMaquina(int nCarro, int nMaquina);

int maquinaLibre[] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 };
int nTrabajosMaquina[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
int recogerEnMaquina[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
int nPiezasAProcesar, nPiezasEnA = 0;
int nMaquinasLibres = 13;

void hEntradaSalida() {  // Hilo de entrada/salida de contenedores
    while (1) {
        nPiezasAProcesar = entradaContenedores();
        nPiezasEnA = nPiezasAProcesar;
        int i;
        for (i = 0; i < nPiezasAProcesar; i++) signal(sEntradas);
        for (i = 0; i < nPiezasAProcesar; i++) wait(sSalidas);
        salidaContenedores();
    }
}

void hCarro(int nCarro) {
    int maquinaEncontrada;
    int nMaquina;

    while (1) {

        wait(sCarros);  // Espera un nuevo encargo

        maquinaEncontrada = 0;

        wait(sBuscaLibre);
        // Región crítica para búsqueda de máquina libre
```

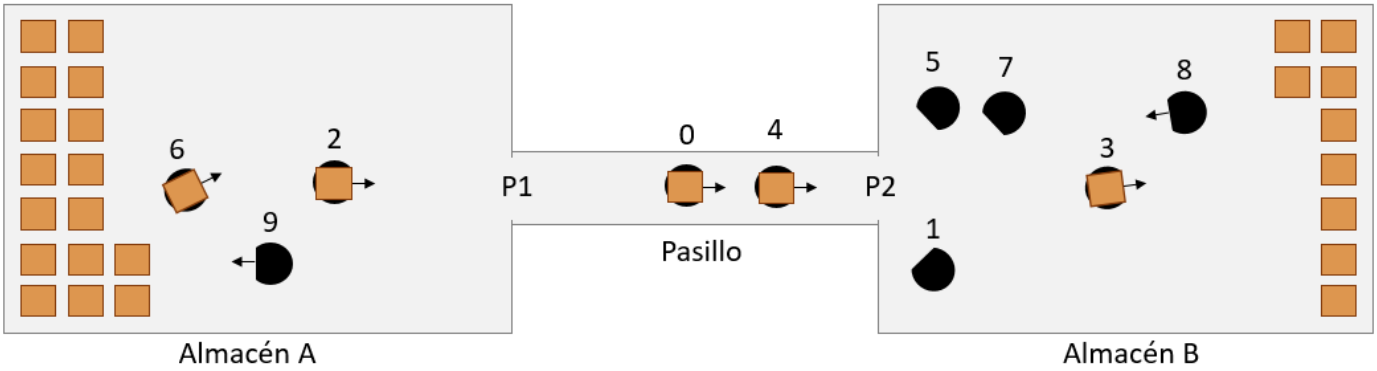
```
        if (nPiezasEnA < nPiezasAProcesar && nMaquinasLibres > 0) {
            // Si falta procesar piezas que están en el contenedor A ...

            for (int i = 0; i < 10 && !maquinaEncontrada; i++) {
                if (maquinaLibre[i]) {
                    maquinaEncontrada = 1;
                    nMaquina = i;
                }
            }
            // Busca una máquina libre

            for (int i = nMaquina + 1; i < 13; i++)
                if (maquinaLibre[i] && nTrabajosMaquina[i] < nTrabajosMaquina[nMaquina]) {
                    nMaquina = i;
                }
            // Busca la libre que tenga menos trabajos hechos

            maquinaLibre[nMaquina] = 0;  // Para reservar la máquina
            nMaquinasLibres--;  // Decrementa el número de máquinas libres
            nPiezasEnA--;  // Y el número de piezas en el contenedor A
            maquinaEncontrada = 1;  // Indica que se encontró una máquina a donde dirigirse

        }

        signal(sBuscaLibre);  // Fin de región crítica para búsqueda de máquina libre

        if (maquinaEncontrada) {  // Si hay máquina libre y hay piezas a procesar ...

            cargaMaquina(nCarro, nMaquina);
            signal(sMaquinaCargada[nMaquina]);
            // Lleva la pieza a la máquina y despierta su hilo
        }

        else {  // Si no hay máquina libre ...

            wait(sBuscaFinalizada);  // Región crítica para búsqueda de procesados finalizados
            maquinaEncontrada = 0;
            for (int i = 0; i < 10 && !maquinaEncontrada; i++) {
                if (recogerEnMaquina[i]) {
                    maquinaEncontrada = 1;
                    nMaquina = i;
                }
            }
            // Busca una máquina que haya finalizado

            for (int i = nMaquina + 1; i < 13; i++)
                if (recogerEnMaquina[i] && nTrabajosMaquina[i] < nTrabajosMaquina[nMaquina]) {
                    nMaquina = i;
                }
            // Busca la que finalizó que tenga menos trabajos hechos

            recogerEnMaquina[nMaquina] = 0;  // Reserva la máquina

            signal(sBuscaFinalizada);

            descargaMaquina(nCarro, nMaquina);
            signal(sMaquinaDescargada[nMaquina]);
            // Descarga la máquina

            nMaquinasLibres++;  // Indica que hay una máquina libre adicional
        }

    }
}
```

```
void hMaquina(int nMaquina) {

    while (1) {

        wait(sEntradas);
        // Reserva una pieza del contenedor de entrada

        signal(sCarros);
        // Da un trabajo a los carros

        wait(sMaquinaCargada[nMaquina]);
        // Espera a que un carro cargue una pieza

        procesa(nMaquina);
        recogerEnMaquina[nMaquina] = 1;
        nTrabajosMaquina[nMaquina]++;
        // Procesa la pieza

        signal(sCarros);
        // Da un trabajo a los carros

        wait(sMaquinaDescargada[nMaquina]);
        // Espera a que un carro la descargue

        maquinaLibre[nMaquina] = 1;
        // Marca la máquina como libre

        signal(sSalidas);
        // Hay una nueva pieza en el contenedor B

    }
}

main() {
    concurrente{
        hEntradaSalida();
        for (int i = 0; i < 13; i++) hMaquina(i);  // Un hilo por cada máquina
        for (int i = 0; i < 5; i++) hCarro(i);  // Un hilo por cada carro
    }
}
```

---

ej54) En una instalación industrial existen dos almacenes A y B. Se dispone de diez AGV (*Automatic Guided Vehicle*), numerados desde el 0 al 9, que transportan contínuamente un número indeterminado y relativamente grande de paquetes desde el almacén A al almacén B. Cada AGV tiene capacidad para un paquete. Inicialmente todos los AGV parten del almacén A y vacíos. Entre ambos almacenes existe un pasillo relativamente estrecho donde pueden circular simultáneamente varios AGV, pero en un único sentido, no se pueden cruzar. El pasillo conecta con el almacén A en el punto P1 y con el almacén B en el punto P2.

Un AGV puede entrar en el pasillo si en el pasillo no hay ningún otro AGV o si los AGV que están en el pasillo se mueven en su mismo sentido. En la figura, los AGV 1, 5 y 7 quieren dirigirse al almacén A y están esperando a que los AGV 0 y 4 pasen por P2 y liberen el pasillo. Si el 2 llega a P1 antes de que el 0 y el 4 salgan por P2, el 2 entra en el pasillo y el 1, 5 y 7 tienen que esperar a que el 2 salga por P2.



Se dispone de las siguientes funciones:

- `void cogePaquete(int agv)` mientras se ejecuta esta función, el AGV indicado por parámetro se dirige a una posición donde haya un paquete a recoger, lo carga, se acerca al punto P1 y se detiene. El carro está entodo momento en el almacén A.

- `void dejaPaquete(int agv)` mientras se ejecuta, el AGV indicado por parámetro va desde P2 a la posición de descarga del paquete que transporta, lo descarga, se acerca a P2 y se detiene. El carro está en todo momento en el almacén B.

- `void cruzaPasillo(int desde, int hasta, int agv)` mientras se ejecuta, hace que el AGV indicado en `agv` circule por el pasillo desde el punto `desde` hasta el punto `hasta`.

Las esperas hay que realizarlas con bloqueos utilizando herramientas de sincronización. Los AGV nunca colisionan, tienen sistemas automáticos de detección y evasión de obstáculos y cuando se detienen en P1 y P2 dejan espacio suficiente para que puedan pasar otros AGV.

Una solución:

```
semaforo sCruzando12 = 1, sCruzando21 = 1, sPasillo = 1;
int nCarrosCruzando12 = 0, nCarrosCruzando21 = 0;

void hCarro(int nCarro) {
    while (1) {
        cogeCaja(nCarro);
        wait(sCruzando12);
        if (nCarrosCruzando12 == 0)
            wait(sPasillo);
        nCarrosCruzando12++;
        signal(sCruzando12);
        cruzaPasillo(1, 2, nCarro);
        wait(sCruzando12);
        nCarrosCruzando12--;
        if (nCarrosCruzando12 == 0)
            signal(sPasillo);
        signal(sCruzando12);
        dejaCaja(nCarro);
        wait(sCruzando21);
        if (nCarrosCruzando21 == 0)
            wait(sPasillo);
        nCarrosCruzando21++;
        signal(sCruzando21);
        cruzaPasillo(2, 1, nCarro);
        wait(sCruzando21);
        nCarrosCruzando21--;
        if (nCarrosCruzando21 == 0)
            signal(sPasillo);
        signal(sCruzando21);
    }
}
```

```
void main() {
    concurrente{
        for (int i = 0; i < 10; i++)
            hCarro(i);
    }
}
```

---

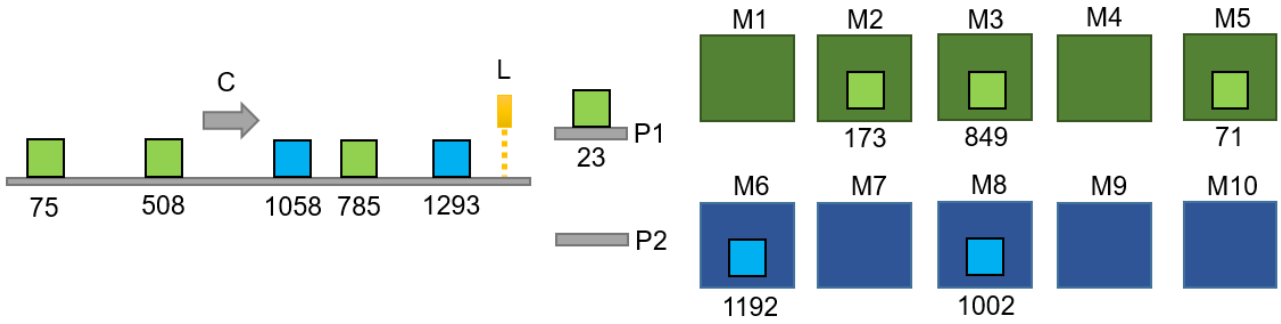
ej59) En una factoría existe una cinta transportadora C, un lector de códigos de barras L, dos plataformas robotizadas P1 y P2 y diez máquinas M1 a M10. Las máquinas M1 a M5 son iguales y pueden procesar piezas cuyo código es menor que 1000, las máquinas M6 a M10 son iguales y pueden procesar piezas con código mayor o igual a 1000. Cada pieza tiene que procesarse en una máquina.

Al comienzo de la cinta C un operario deposita piezas. Podemos poner en marcha o detener a C llamando a la función `void cinta(int marcha)` pasando un buleano cierto o falso como parámetro.

Cada pieza tiene un código numérico diferente escrito en un código de barras. Una llamada a `int lectorCodigoBarras()` espera a que una pieza llegue al lector L y devuelve el código leído.

La plataforma P1 se utiliza para transportar una pieza del final de la cinta a una máquina M1 a M5, la plataforma P2 transporta una pieza de la cinta a las a una máquina M6 a M10. Cada plataforma tiene capacidad para transportar una pieza. Mientras se ejecuta `void plataformaRecoge(int plataforma)`, la plataforma indicada por parámetro recoge una pieza del final de la cinta. Mientras se ejecuta `void plataformaMueveAMaquina(int plataforma, int maquina)`, la plataforma se mueve hasta la máquina. Mientras se ejecuta `void plataformaDeja(int plataforma)`, la plataforma deja una pieza en la máquina en la que está situada. Mientras se ejecuta `void plataformaMueveACinta(int plataforma)`, la plataforma se mueve hacia el final de la cinta. Mientras se ejecuta `void procesa(int maquina)`, la máquina indicada procesa una pieza y luego la deposita en un contenedor.

En el ejemplo de la figura, la cinta C está transportando las piezas 1293, 785, 1058, 508 y 75 y van a llegar en ese orden al lector de códigos de barras L. La plataforma P1 está transportando la pieza 23. La plataforma P2 no está transportando ninguna pieza. Las máquinas M2, M3 y M5 están procesando las piezas 173, 849 y 71, respectivamente. Las máquinas M6 y M8 están procesando las piezas 1192 y 1002, respectivamente. Las máquinas M1, M4, M7, M9 y M10 están desocupadas.



Una solución:

```
semaforo spCinta[2] = { 1, 1 };  // Cinta como productor para plataformas
semaforo scPlataforma[2] = { 0, 0 };  // Plataformas como consumidores de cinta
semaforo spPlataforma[2] = { 5, 5 };  // Plataformas como productores para máquinas
semaforo scMaquina[10] = { 0,0,0,0,0,0,0,0,0,0 };  // Máquinas como consumidores de plataformas
int maquinaOcupada[10] = { 0,0,0,0,0,0,0,0,0,0 };  // Indica si máquina ocupada
semaforo sMaquinaOcupada = 1;  // Para región crítica acceso a la matriz maquinaOcupada
```

```
void hiloCinta() {
    while (1) {
        cinta(1);  // Cinta en movimiento
        int pieza = lectorCodigoBarras();  // Espera a quellegue pieza a L y lee código de barras
        cinta(0);  // Para la cinta
        int plataforma = 1;  // En principio va a plataforma 1
        if (pieza >= 1000)  // Si código mayor o igual a 1000, para plataforma 2
            plataforma = 2;
        wait(spCinta[plataforma - 1]);  // Espera a plataforma preparada para recoger pieza
        plataformaRecoge(plataforma);  // Deposita pieza en plataforma
        signal(scPlataforma[plataforma - 1]);  // Avisa a hilo de plataforma
    }
}

void hiloPlataforma(int plataforma) {
    while (1) {
        wait(scPlataforma[plataforma - 1]); // Espera a que cinta deposite pieza
        wait(spPlataforma[plataforma - 1]); // Espera a que haya alguna máquina libre
        int maquina;  // Máquina destino
        if (plataforma == 1)  // Determina la primera máquina posible
            maquina = 1;
        else maquina = 6;
        wait(sMaquinaOcupada);  // Entra en región crítica
        while (maquinaOcupada[maquina - 1]) maquina++;  // Busca máquina libre
        maquinaOcupada[maquina - 1] = 1;  // La marca como ocupada
        signal(sMaquinaOcupada);  // Sale de región crítica
        plataformaMueve(plataforma, maquina);  // Mueve la plataforma hasta la máquina
        plataformaDeja(plataforma);  // Plataforma deja pieza en máquina
        signal(scMaquina[maquina - 1]);  // Avisa al hilo de la máquina que ya tiene pieza
        plataformaMueve(plataforma, 0);  // Plataforma vuelve a la cinta
        signal(spCinta[plataforma - 1]);  // Plataforma avisa al hilo de la cinta que está libre
    }
}

void hiloMaquina(int maquina) {
    int plataforma = 1;
    if (maquina >= 6)  // Determina su plataforma
        plataforma = 2;
    while (1) {
        wait(scMaquina[maquina - 1]);  // Máquina espera a que su plataforma deje pieza
        mecaniza(maquina);  // La máquina mecaniza la pieza y la deposita en un contenedor
        wait(sMaquinaOcupada);  // Entra en región crítica
        maquinaOcupada[maquina - 1] = 0;  // Marca máquina desocupada
        signal(sMaquinaOcupada);  // Sale de región crítica
        signal(spPlataforma[plataforma - 1]);  // Avisa a su plataforma que hay máquina libre
    }
}

int main() {
    concurrente{
        hiloCinta();  // Un hilo para atender a la cinta
        for (int i = 1; i <= 2; i++) hiloPlataforma(i);  // Un hilo por cada plataforma
        for (int i = 1; i <= 10; i++) hiloMaquina(i);  // Un hilo por cada máquina
    }
    return 0;
}
```

---

ej60) En un almacén existen 40 robots móviles AGV para transporte de mercancías.Todos los robots están numerados, del 0 al 39. Hay cuatro modelos de robots: modelo 0 (robots 0 al 9), modelo 1 (robots 10 al 19), modelo 2 (robots 20 al 29) y modelo 3 (robots 30 al 39).

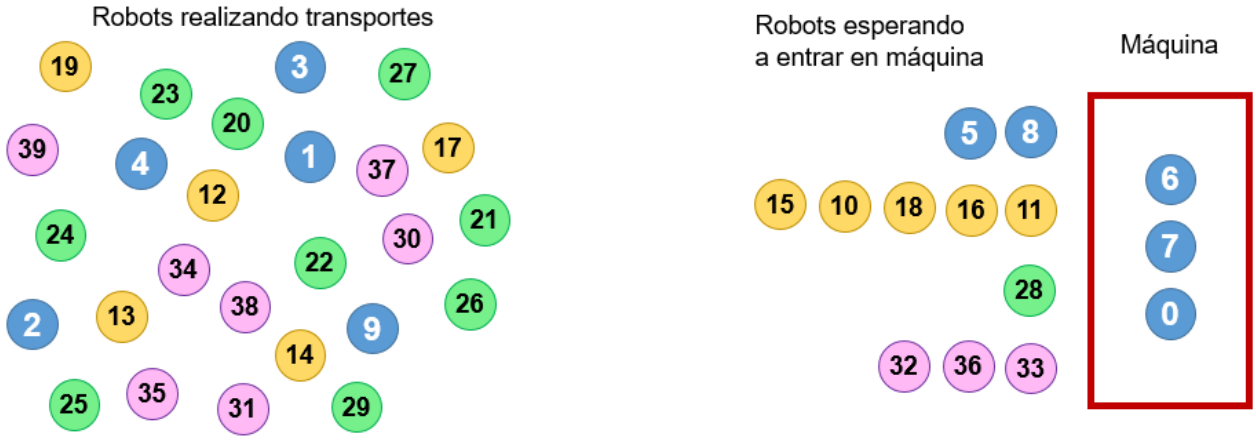
Durante la ejecución de `void transportes(int robot)`, el robot indicado por parámetro realiza varios transportes de mercancías hasta que necesita recibir ciertas operaciones (recarga de baterías,

mantenimiento, etc), para lo cual se acerca a una máquina y espera hasta poder entrar en ella. En la máquina tienen que entrar tres robots del mismo tipo para que puedan recibir simultaneamente las operaciones que necesitan.

En la figura, los robots 6, 7 y 0 están recibiendo esas operaciones, los robots 5, 8, 15, 10, 18, 16, 11, 28, 32, 36 y 33 necesitan recibir esas operaciones pero están esperando hasta que puedan entrar en la máquina y el resto de robots están realizando transportes.

Durante la ejecución de `void entra(int robot)`, el robot indicado por parámetro entra en la máquina, con `void recibeOperaciones(int robot)` se maneja el robot para que reciba las operaciones necesarias y con `void sale(int robot)` sale de la máquina a la zona de transportes.

Durante la llamada a `void opera(int modelo)`, se gobiera la máquina para que realice operaciones simultaneamente sobre tres robots del modelo indicado por parámetro.



Una solución:

```
semaforo sGrupo[4] = { 3, 3, 3, 3 };  // Para formar grupos de 3 robots por modelo
semaforo srcModelo[4] = { 1, 1, 1, 1 };  // Región crítica entre robots del mismo modelo
semaforo srcMaquina = 0;  // Máquina como región crítica para los modelos
semaforo scMaquina = 0;  // Para controlar a la máquina como consumidor de trabajos
semaforo sEntraRobot[4] = { 0, 0, 0, 0 };  // Para esperar a que haya 3 robots para entrar
int nRobots[4] = { 0, 0, 0, 0 };  // Número de robots de un modelo dentro o esperando
int modeloDentro;  // Tipo de modelo dentro de la máquina

void hiloRobot(int robot) {

    int modelo = robot / 10;  // Determina su modelo

    while (1) {

        transportes(robot);  // Transporta hasta que se acerca a la máquina para operaciones

        wait(sGrupo[modelo]);  // Por este wait sólo pasan 3 por modelo, los demás esperan

        wait(srcModelo[modelo]);  // Entra en región crítica
        nRobots[modelo] ++;  // Hay un robot más de ese modelo preparado para entrar
        if (nRobots[modelo] == 3) {  // Si hay 3 esperando de ese modelo ...
            wait(srcMaquina);  // La máquina es región crítica para cada grupo de 3
            modeloDentro = modelo;  // Indica qué modelo de robot entró
            signal(sEntraRobot[modelo]);  // Avisa a los otros 2
            signal(sEntraRobot[modelo]);  // robots para que entren
            signal(srcModelo[modelo]);  // Sale de región crítica
        }
```

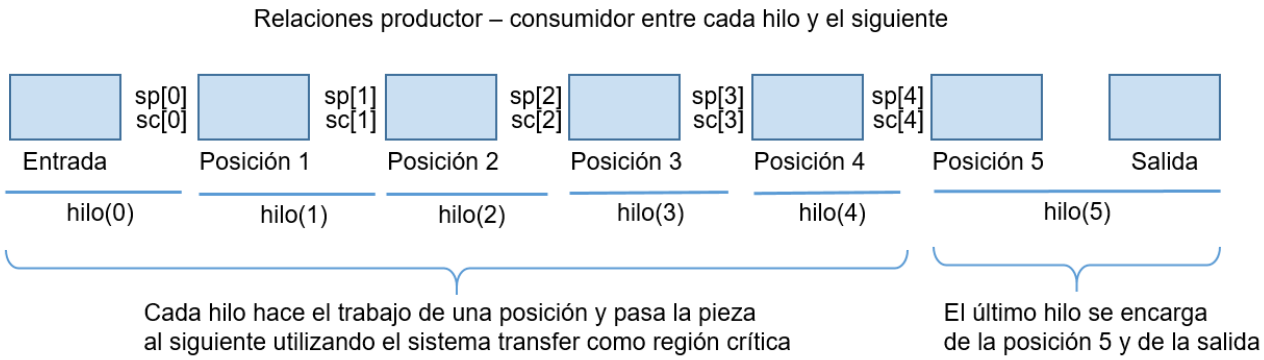
```
        else {
            signal(srcModelo[modelo]);  // Sale de región crítica
            wait(sEntraRobot[modelo]);  // Espera a que haya 3 robots del mismo modelo
        }

        entra(robot);  // Entra en la máquina
        signal(scMaquina);  // Avisa a la máquina que entró un robot
        recibeOperacion(robot);  // Se prepara para recibir operaciones
        sale(robot);  // Sale de la máquina

        wait(srcModelo[modelo]);  // Entra en región crítica
        nRobots[modelo]--;  // Indica que salió un robot
        if (nRobots[modelo] == 0) {  // Si es el último de ese modelo en salir ...
            signal(srcMaquina);  // Libera región crítica de máquina para los grupos de robots
            for (int i = 0; i < 3; i++) signal(sGrupo[modelo]);  // Para formar grupo a otros 3
        }
        signal(srcModelo[modelo]);  // Sale de región crítica

    }
}

void hiloMaquina() {
    while (1) {
        for (int i =0; i < 3; i++) wait(scMaquina);  // Espera a que los 3 robots estén dentro
        operaRobots(modeloDentro);  // Realiza operaciones sobre los 3 robots que entraron
    }
}

void main() {
    concurrente {
        for (int i = 0; i < 40; i++)
            hiloRobot(i);  // Un hilo por robot
        hiloMaquina();  // Un hilo para la máquina
    }
}
```

---

ej61) Una máquina está gobernada por un sistema embebido con sistema operativo multitarea en tiempo real. Un operario deposita piezas metálicas a su entrada con el objetivo de que la máquina las mecanice. En esa posición de entrada sólo puede haber una pieza en un determinado momento y hay un sensor capacitivo para detectarla que podemos consultar llamando a `int entrada()`, que devuelve un buleano cierto si hay pieza en esa posición. En la máquina existe un único sistema transfer neumático que mueve las piezas dentro de la máquina para llevarlas sucesivamente por las posiciones de mecanizado 1, 2, 3, 4 y 5, cada posición tiene capacidad para una pieza. Cada pieza tiene que pasar por todas las posiciones de mecanizado y en ese orden. Con la llamada a `void mecaniza(int posicion)` se ejecuta un mecanizado sobre la pieza que está en la posición indicada por parámetro. Existe una posición de salida donde el sistema transfer deposita las piezas mecanizadas. En esa posición de salida puede haber sólo una pieza en un momento determinado, hay un sensor que se puede consultar con `int salida()`, que devuelve un buleano cierto si detecta pieza. Un operario extrae las piezas de la posición de salida. El sistema transfer tiene capacidad para transportar una pieza dentro de la máquina, para ello utilizaremos la función `void transfiere(int posicionOrigen, int posicionDestino)` para llevar una pieza desde una posición hasta otra posición. La posición de entrada de piezas es la número 0 y la de salida la número 6. Codifica una aplicación para que las piezas que se depositan a la entrada se mecanicen llevándolas sucesivamente desde la posición 1 a la 5 y se lleven a la salida, de forma que pueda haber varias posiciones de mecanizado trabajando simultáneamente sobre piezas diferentes, mientras el sistema transfer puede estar moviendo una pieza entre dos posiciones. Las operaciones de espera hay que realizarlas utilizando herramientas de sincronización.

Una solución:

```
semaforo sc[5] = { 0,0,0,0,0 };  // Para controlar a los consumidores
semaforo sTransfer = 1;  // El sistema transfer es región crítica para todos los hilos

void hilo(int n) {
    while (1) {
        if (n == 0)  // Si es la posición de entrada ...
            while (!entrada());  // Espera a que se deposite una pieza
        else wait(sc[n - 1]);  // Si no, espera como consumidor
        if (n > 0)  // Si no es la posición de entrada ...
            mecaniza(n);  // mecaniza la pieza
        if (n < 5)  // Si no es la posición de salida ...
            wait(sp[n]);  // espera como productor
        else while (salida());  // si no, espera a que se pueda depositar pieza en salida
        wait(sTransfer);  // Reserva el transfer como región crítica
        transfiere(n, n + 1);  // Mueve la pieza a la siguiente posición
        signal(sTransfer);  // Libera el transfer como región crítica
        if (n < 5)  // Si no es la posición de salida ...
            signal(sc[n]);  // avisa al siguiente hilo que ya tiene pieza
        if (n > 0)  // Si no es la posición de entrada ...
            signal(sp[n - 1]);  // avisa al productor que ya puede recibir pieza
    }
}

void main() {
    concurrente{
        for (int i = 0; i <= 5; i++)
            hilo(i);  // Un hilo por posición, el sistema transfer es región crítica
    }
}
```



---

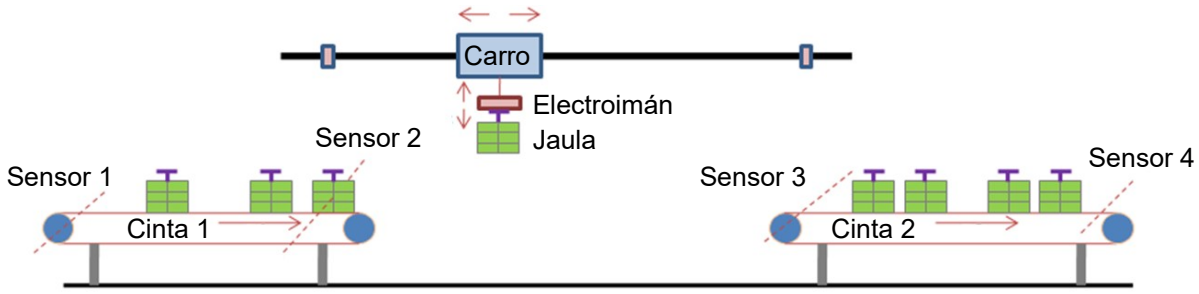
ej64) En una instalación de transporte de jaulas con alimentos se dispone de dos cintas transportadoras 1 y 2 que se pueden poner en marcha (movimiento hacia la derecha) o detener mediante la función **`void cinta(int numCinta, int activar)`**. En la entrada y salida de las cintas hay sensores fotoeléctricos 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 scCarro = 0;  // Semáforo para el carro como consumidor
semaforo spCarro = 1;  // Semáforo para el carro como productor
semaforo spCinta1 = 1; // Semáforo para la cinta 1 como productor
semaforo scCinta2 = 0; // Semáforo para la cinta 2 como consumidor
semaforo sJaulas1 = 0, sJaulas2 = 0;  // Para contar jaulas en las cintas

void hCinta1Entrada() {  // Trata la entrada de jaulas en la cinta 1
    while(1) {
        while(! sensor(1));  // Espera a que se deposite una jaula
        signal(sJaulas1);  // Indica que hay una jaula adicional
        while(sensor(1));  // Espera a que la jaula desaparezca de la entrada
    }
}

void hCinta1Salida() {  // Trata la salida de jaulas en la cinta 1
    while(1) {
        wait(spCinta1);  // Si puede proporcionar una jaula al carro ...
        wait(sJaulas1);  // Si hay alguna jaula en la cinta ...
        cinta(1, 1);  // Pone en marcha la cinta
        while(! sensor(2));  // Espera a que llegue una jaula al final
        cinta(1, 0);  // Para la cinta
        signal(scCarro);  // Avisa al carro que hay jaula disponible
    }
}

void hCarro() {  // Maneja el carro
    sube();  // Inicialmente sube el cable
    izquierda();  // lleva el carro a la izquierda
    electroiman(0);  // y desactiva el electroimán
    while(1) {
        wait(scCarro);  // Espera a que haya jaula disponible en cinta 1
        baja();  // Baja el electroimán
        electroiman(1);  // Activa el electroimán
        sube();  // Lo sube
        signal(spCinta1);  // Avisa a la cinta 1 que ya se extrajo la jaula
        derecha();  // Lleva la jaula hasta la cinta 2
        wait(spCarro);  // Espera a poder bajar la jaula
```

```
        baja();  // La baja
        electroiman(0);  // Deja la jaula desactivando el electroimán
        signal(scCinta2);  // Avisa a la cinta 2 que tiene otra jaula
        sube();  // Sube el electroimán
        izquierda();  // Se mueve hasta la cinta 1
    }
}

void hCinta2Entrada() {  // Trata la entrada de jaulas en la cinta 2
    while(1) {
        wait(scCinta2);  // Espera a que el carro haya dejado una jaula
        signal(sJaulas2);  // Incrementa el contador de jaulas
        while(sensor(3));  // Espera a que la jaula desaparezca de la entrada de la cinta
        signal(spCarro);  // Avisa al carro que ya puede dejar otra jaula
    }
}

void hCinta2Salida() {  // Trata la salida de jaulas de la cinta 2
    while(1) {
        wait(sJaulas2);  // Espera a disponer de alguna jaula
        cinta(2, 1);  // Pone la cinta en movimiento
        while(! sensor(4));  // Espera hasta que alguna jaula llegue al final
        cinta(2, 0);  // Para la cinta
        while(sensor(4));  // Espera a que el operario retire la jaula
    }
}

int main()
{
   concurrente {
       hCinta1Entrada();
       hCinta1Salida();
       hCarro();
       hCinta2Entrada();
       hCinta2Salida();
   }
}
```

---

ej65) En una instalación industrial existen 50 robots móviles de transporte de mercancías. Hay robots de diferentes modelos. Diferentes modelos tienen diferentes características y diferente batería. Cada robot se identifica mediante un número, desde el robot 1 al robot 50. Los robots 1 a 10 son de un modelo, los robots 11 a 20 son de otro modelo, del 21 al 30 de otro modelo, del 31 al 40 de otro y del 41 al 50 de otro. Cada robot está alimentado mediante una batería extraíble. Cuando la batería de un robot tiene un nivel bajo de carga, el robot tiene que ir a la estación de repostaje donde se cambia su batería por otra batería cargada. En la instalación hay una estación de repostaje que tiene cinco puestos, desde el puesto 1 al puesto 5, en cada puesto puede haber un robot repostando. La estación puede atender hasta cinco robots simultáneamente. Todos los robots que estén al mismo tiempo en la estación de repostaje tienen que ser del mismo modelo. Si hay algún robot repostando, sólo pueden entrar a la estación robots del mismo modelo.

Para el manejo de los robots se dispone de las siguientes funciones:

- `int siguienteTransporte(int robot)`: solicita cuál es el siguiente trabajo de transporte asignado al robot indicado por parámetro. Esta función devuelve el trabajo de transporte asignado, que se identifica mediante un número entero. La llamada a esta función no termina hasta que haya algún trabajo asignado a ese robot.
- `void transporta(int transporte, int robot)`: durante la ejecución de esta función el robot indicado en `robot` realiza el trabajo de transporte indicado en `transporte`.
- `int bateriaBaja(int robot)`: comprueba el estado de la batería del robot indicado por parámetro y devuelve un buleano cierto en el caso de que tenga un nivel de batería bajo, en cuyo caso es necesario cambiar su batería.
- `void robotAEstacion(int robot)`: durante la ejecución de esta función el robot indicado por parámetro se acerca a una zona cercana a la estación de repostaje, pero sin entrar en ningún puesto de la estación.

- `void sustituyeBateria(int robot, int puesto)`: durante la ejecución de esta función el robot indicado en `robot` se introduce en la estación de repostaje en el puesto indicado en `puesto`, la estación de repostaje cambia su batería por otra cargada y el robot sale de la estación de repostaje, dejando ese puesto libre.

Codifica una aplicación para que los robots ejecuten las operaciones de transporte programadas y para que se sustituya su batería cuando su nivel de carga sea bajo.

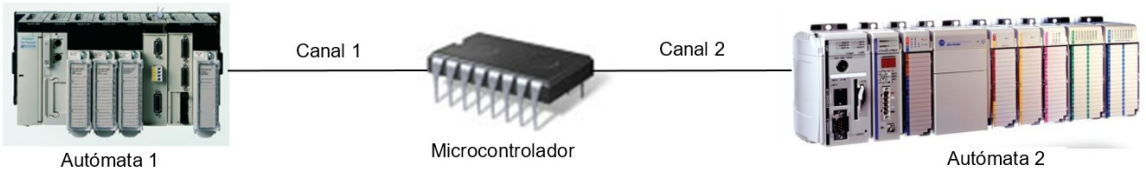
Una solución:

```
semaforo sReservaEstacion = 1;  // Para reservar la estación, es recurso compartido entre modelos
semaforo sModelo[5] = {1, 1, 1, 1, 1};  // Cada modelo tiene su región crítica
int nRobots[5] = {0, 0, 0, 0, 0};  // Número de robots por modelo en la estación o intentando el repostaje
int puestoLibre[5] = {1, 1, 1, 1, 1};  // Buleanos indicando qué posiciones de la estación están libres
semaforo sPuestoLibre = 1;  // Para región crítica de acceso a la matriz puestoLibre
semaforo sReservaPuesto = 5;  // Para reservar un puesto. Hay 5 disponibles en el cargador.

void hRobot(int robot) {  // Función que define el código del hilo de cada robot

    int modelo = (robot - 1) / 10;  // Modelo de robot, de 0 a 4

    while(1) {  // Repite indefinidamente ...

        do {
            int transporte = siguienteTransporte(robot);  // Obtiene el siguiente trabajo de transporte
            transporta(transporte, robot);  // Realiza el transporte
        } while(!bateriaBaja(robot));  // mientras haya carga en la batería del robot

        robotAEstacion(robot);  // Acerca el robot a la estación de repostaje

        wait(sModelo[modelo]);  // Región crítica para los robots de ese modelo
        nRobots[modelo] ++;  // Un robot más de ese modelo va a repostaje
        if (nRobots[modelo] == 1)  // Si es el primero de ese modelo ...
            wait(sReservaEstacion);  // La estación es recurso compartido entre los modelos
        signal(sModelo[modelo]);  // Sale de la región crítica del modelo

        wait(sReservaPuesto); // Sólo hay 5 puestos en la estación, reserva uno

        wait(sPuestoLibre);  // Región crítica para manejar la matriz puestoLibre
        int puesto = 1;
        while (! puestoLibre[puesto-1])  // Busca un puesto libre
            puesto ++;
        puestoLibre[puesto-1] = 0;  // Lo marca como ocupado
        signal(sPuestoLibre);  // Libera el acceso a la matriz

        sustituyeBateria(robot, puesto);  // Sustituye la batería del robot en el puesto

        wait(sPuestoLibre);  // Región crítica para manejar la matriz puestoLibre
        puestoLibre[puesto-1] = 1;  // Marca el puesto como libre
        signal(sPuestoLibre);  // Libera el acceso a la matriz

        signal(sReservaPuesto);  // Libera el puesto en la estación

        wait(sModelo[modelo]);  // Entra en la región crítica del modelo
        nRobots[modelo] --;  // Hay un robot menos en la estación
        if (nRobots[modelo] == 0)  // Si es el último que sale ...
            signal(sReservaEstacion);  // Libera la estación para robots de otros modelos
        signal(sModelo[modelo]);  // Sale de la región crítica del modelo
    }
}
```

```
void main() {
    concurrente {
        for(int robot=1; robot<=50; robot++)
            hRobot(robot);  // Lanza un hilo por cada robot
    }
}
```

---

ej66) En una instalación industrial existen dos autómatas programables de diferentes fabricantes, cada uno con su canal serie asíncrono, para intercambiar información entre ellos. Sin embargo, utilizan diferentes protocolos de comunicación y no pueden entenderse. Insertamos enmedio un microcontrolador con dos canales serie asíncronos para que realice una labor de traductor.



El protocolo que utiliza el autómata 1 consiste en el envío y recepción de paquetes en formato texto con el siguiente contenido:
- Comienzan con el caracter `'$'`.
- Siguen dos caracteres que expresan en hexadecimal el número de bytes de información que van en el paquete. Como máximo puede haber 16 bytes de información en un paquete. Por ejemplo, `"0B"` indica que hay 11 bytes de información.
- A continuación vienen todos los bytes de información, cada byte expresado en dos caracteres en hexadecimal. Por ejemplo, `"FF"` indica un byte con todos sus bits a 1.
- Sigue un checksum de todos los caracteres, desde el `'$'` inicial hasta el último carácter del último byte de información. Son los 8 bits bajos de la suma del código ASCII de todos los caracteres enviados, desde el `'$'` inical hasta el último carácter del último byte de información. Este checksum es un byte que también se envía en texto, expresado en dos caracteres en hexadecimal.
- Termina con un carácter de salto de línea `'\n'`.

El autómata 2 utiliza un protocolo diferente, basado en el envío y recepción de paquetes en binario.
- Comienza con un byte que indica el número de bytes de información en el paquete (como máximo, 256 bytes).
- Sigue con todos los bytes de información.
- Finaliza con un byte que es igual a la XOR de todos los bytes del paquete, salvo esta XOR.

En el mismo canal es posible el envío y recepción simultáneos. Se dispone de las siguientes funciones para manejar los canales 1 y 2:
- `uint8_t recibe(int canal)`: espera hasta que se reciba un byte por el canal `canal` y devuelve el byte recibido.
- `void envia(int canal, uint8_t byte)`: envía por el canal `canal` el byte `byte`.

Codifica un programa en lenguaje C para el microcontrolador de forma que la información que envía un autómata la reciba el otro y viceversa, traduciendo convenientemente los protocolos para que ambos autómatas se entiendan. Sólo se retransmitirán los paquetes con checksum o XOR correctos.

Para la conversión de texto a binario y viceversa se puede utilizar lo siguiente. Mediante `sprintf(cadena, "%02X", 14)` se guarda `"0E"` en la cadena de caracteres `cadena`, es la representación del valor `14` en dos caracteres en hexadecimal. Mediante `sscanf("0E", "%x", &valor)` se guarda el valor `14` en la variable `int valor`.

Una solución:

```
void hAutomata1() {  // Trata los paquetes enviados por el autómata 1
    while(1) {
        uint8_t bytes[50];  // Matriz donde se acumulan los caracteres recibidos
        char cadena[3];  // Cadena de caracteres auxiliar
        int nRecibidos = 0;  // Número de caracteres recibidos
        do
            bytes[nRecibidos++] = recibe(1);  // Recibe el siguiente byte del autómata 1 y lo almacena
        while(bytes[nRecibidos-1] != '\n');  // Mientras no se detecte el final de un paquete
        uint8_t checksum = 0;  // Variable para calcular el checksum de los caracteres recibidos
        for (int i = 0; i < nRecibidos-3; i++)  // Recorre todos los caracteres, menos checksum y el '\n'
            checksum += bytes[i];  // Suma código ASCII de todos los caracteres para obtener el checksum
        cadena[0] = bytes[nRecibidos-3];  // Recoge el checksum que viene en el paquete
        cadena[1] = bytes[nRecibidos-2];  // en dos caracteres
        cadena[2] = 0;  // Finaliza la cadena
        int checksumRecibido;
        sscanf(cadena, "%x", &checksumRecibido);  // Lo convierte de texto a binario
        if (checksum == checksumRecibido) {  // Si el checksum calculado y el del paquete coinciden ...
            cadena[0] = bytes[1];  // Extrae en texto el número de bytes de información
            cadena[1] = bytes[2];  // en dos caracteres
            cadena[2] = 0;  // Finaliza la cadena
            int nBytes;
            sscanf(cadena, "%x", &nBytes);  // Obtiene el número de bytes de información en binario
            envia(2, nBytes);  // Envía el número de bytes al autómata 2
            uint8_t xor = nBytes;  // Inicializa el cálculo de la XOR
            for(int i = 0; i < nBytes; i++) {  // Recorriendo el número de bytes de información ...
                cadena[0] = bytes[3+i*2];  // Primer carácter del byte i-ésimo
                cadena[1] = bytes[3+i*2+1];  // Segundo carácter del byte i-ésimo
                cadena[2] = 0;  // Finaliza la cadena
                int dato;
                sscanf(cadena, "%x", &dato);  // Lo convierte a binario
                envia(2, dato);  // y lo envía al autómata 2
                xor ^= dato;  // Actualiza la XOR del paquete
            }
            envia(2, xor);  // Envía la XOR al autómata 2
        }
    }
}

void hAutomata2() {  // Trata los paquetes enviados por el autómata 2
    while(1) {
        uint8_t bytes[256];  // Para guardar bytes de información del paquete enviado por el autómata 2
        char cadena[3];  // Cadena de caracteres auxiliar
        int nBytesInformacion = recibe(2);  // Recibe el número de bytes de información en el paquete
        uint8_t xor = nBytesInformacion;  // Para calcular la XOR del paquete
        for(int i = 0; i < nBytesInformacion; i++) {
            bytes[i] = recibe(2);  // Recibe un nuevo byte de información y lo guarda
            xor ^= bytes[i];  // Actualiza la XOR
        }
        uint8_t xorRecibida = recibe(2);  // Recibe la XOR enviada por el autómata 2
        if (xor == xorRecibida) {  // Si la XOR calculada y recibida son iguales ...
            envia(1, '$');  // Envía el comienzo del paquete al autómata 1
            uint8_t checksum = '$';  // Para calcular el checksum
            for(int i = 0; i < nBytesInformacion; i++) {  // Para cada bytes de información ...
                sprintf(cadena, "%02X", bytes[i]);  // Convierte a texto en hexadecimal en dos caracteres
                envia(1, cadena[0]);  // Envía al autómata 1 el primer carácter
                envia(1, cadena[1]);  // y el segundo carácter
                checksum += cadena[0];  // Actualiza el checksum
                checksum += cadena[1];  // Actualiza el checksum
            }
            sprintf(cadena, "%02X", checksum);  // Prepara el checksum en dos caracteres en hexadecimal
            envia(1, cadena[0]);  // Envía al autómata 1 el primer carácter
            envia(1, cadena[1]);  // y el segundo carácter
            envia(1, '\n');  // Envía al autómata 1 el carácter de final de paquete
        }
    }
}
```

```
void main() {
    concurrente {
        hAutomata1();  // Trata los paquetes enviados por el autómata 1
        hAutomata2();  // Trata los paquetes enviados por el autómata 2
    }
}
```

En esta solución se está suponiendo que el autómata 2 envía al autómata 1 paquetes que éste puede recibir, es decir, de hasta 16 bytes de información. Si quisiéramos tratar también un paquete enviado por el autómata 2 con más bytes de información, habría que añadir unas pocas instrucciones adicionales para partir el paquete en varios paquetes de hasta 16 bytes de información para reenviarlos al autómata 1.