SCTR
Ejercicios de aula

1) En una factoría existen 10 depósitos, numerados desde el depósito número 1 hasta el depósito número 10. Cada uno de ellos dispone de un sensor de temperatura. La temperatura medida en los depósitos se almacena en la matriz **`temperaturas`** que se muestra a continuación, desde el depósito número 1 al 10. Codifica un algoritmo en lenguaje C que guarde en otra matriz los números de los depósitos, ordenados por temperatura de menor a mayor, y para los depósitos que estén a la misma temperatura ordenados por número de depósito.

```
float temperaturas[] = {13.5, 18.9, 15.3, 21.6, 19.2, 14.9, 17.4, 15.3, 16.5, 19.2};
```

Una solución:

```
int main() {
    float temperaturas[] = {13.5, 18.9, 15.3, 21.6, 19.2, 14.9, 17.4, 15.3, 16.5, 19.2};
    int maquinas[]       = {1,    2,    3,    4,    5,    6,    7,    8,    9,    10};
    float copia[10];

    for(int i = 0; i < 10; i ++)
        copia[i] = temperaturas[i];
    for (int i = 0; i < 10; i ++)
        for (int j = 0; j < 10; j ++) {
            if (copia[j] > copia[i] || (copia[j] > copia[i] && maquinas[j] > maquinas[i])) {
                float temperatura;
                int maquina;
                temperatura = copia[j];
                copia[j] = copia[i];
                copia[i] = temperatura;
                maquina = maquinas[j];
                maquinas[j] = maquinas[i];
                maquinas[i] = maquina;
            }
        }
    return 0;
}
```

---

2) Codifica la aplicación que se ejecuta en el microcontrolador que gobierna un dispositivo de periferia distribuida que actúa como esclavo en un canal serie asíncrono bajo protocolo Modbus RTU. Este dispositivo utiliza la dirección 14 en el canal, donde un nodo maestro se comunica cíclicamente con varios dispositivos esclavos. Los paquetes enviados por el maestro utilizan los códigos de operación Modbus 0x10 (escritura de varios registros) y 0x03 (lectura de varios registros).

Este dispositivo dispone de 32 entradas digitales manejadas en el microcontrolador mediante los puertos PORTA, PORTB, PORTC, PORTD, cada uno de 8 bits. Cada uno de estos puertos está mapeado en memoria en las direcciones 0xF80, 0xF81, 0xF82 y 0xF83, respectivamente. Cuando se recibe una orden Modbus para la lectura del registro 0, se devuelve un dato de 16 bits con el estado de las señales de los puertos PORTA (en parte baja) y PORTB (en parte alta). Si se recibe una orden para la lectura del registro 1, se devuelve un dato de 16 bits con el estado de las señales de los puertos PORTC (en parte baja) y PORTD (en parte alta).

También dispone de 32 salidas digitales repartidas en los puertos PORTE, PORTF, PORTG, PORTH mapeados en memoria en 0xF84, 0xF85, 0xF86 y 0xF87, respectivamente. Cuando se recibe una orden Modbus para la escritura del registro 0, se recibe un dato de 16 bits con el que se modifica estado de las señales de los puertos PORTE (con la parte baja) y PORTF (con la parte alta). Si se recibe una orden para la escritura del registro 1, se recibe un dato de 16 bits con el estado de las señales para los puertos PORTG y PORTH.

Para indicar si una línea digital de un puerto actúa como entrada o como salida, se utilizan los SFRs denominados TRISA hasta TRISH, mapeados consecutivamente en memoria desde la dirección 0xF92. Un 1 en estos SFRs configuran una señal como entrada y un 0 como salida.

Para programar el envío y recepción en el canal serie asíncrono se dispone de las funciones:

- **`uint8_t recibeByte()`**: espera hasta que se reciba un byte en el canal y lo devuelve.

- **`void enviaByte(uint8_t byte)`**: envía por el canal el byte indicado por parámetro.

En el canal se fija un intervalo de tiempo mínimo de 15 ms de reposo entre ciclos sucesivos de orden/respuesta en el que no se transmite ningún byte. Para medir tiempo podemos utilizar los siguientes recursos del microcontrolador:

- **`void instalaTemporizador(void (*funcion)())`**: instala la función indicada por parámetro para que atienda a las interrupciones producidas por un temporizador configurado previamente para generarlas con una frecuencia de 10 KHz.

- **`void habilita()`** y **`void deshabilita()`**: habilita o deshabilita la interrupción producida por el temporizador.

Una solución:

```
#define MODBUS_ESCRITURA 0x10
#define MODBUS_LECTURA 0x03
#define DIRECCION_MODBUS 14

int nTemporizaciones = 0;


void interrupcionTemporizador() {  // Atiende a la interrupción del temporizador
    nTemporizaciones ++;
}


void configuraSenalesDigitales() {  // Configura los puertos de E/S digital
    uint8_t * p;
    p = (uint8_t *) 0xF92;
    for (int i = 0; i < 8; i++)
        if (i < 4)
            p[i] = 0xFF;
        else p[i] = 0x00;
}


uint8_t recibeDireccionModbus() {  // Detecta período de reposo en el canal y devuelve el primer byte
    uint8_t byte;
    do {
        nTemporizaciones = 0;
        byte = recibeByte();
    } while (nTemporizaciones < 150)
    return byte;
}


int recibeOrden(uint8_t * paquete) {
// Recibe una orden Modbus enviada al nodo DIRECCION_MODBUS y la guarda en el paquete apuntado por
// 'paquete'. Devuelve un buleano falso si entre la recepción de dos bytes hay más de 15ms, o si la orden
// no es de lectura ni de escritura

    paquete[0] = DIRECCION_MODBUS;
    nTemporizaciones = 0;
    if (nTemporizaciones > 150) return 0;
    paquete[1] = recibeByte();
    if (paquete[1] != 0x10 && paquete[1] != 0x03) return 0;
    uint8_t nBytes;
    if (paquete[1] == 0x10)
        nBytes = 13;
    else nBytes = 8;
```

```
    for (int i = 2; i < nBytes; i++) {
        nTemporizaciones = 0;
        paquete[i] = recibeByte();
        if (nTemporizaciones > 150) return 0;
    }
    return 1;
}


uint16_t calculaCRC(uint8_t * paquete, uint8_t nBytes) {
// Calcula y devuelve el CRC16 del paquete de bytes apuntado por 'paquete' que consta de 'nBytes' bytes

    uint16_t CRC = 0xFFFF;
    for(int i = 0; i < nBytes; i++) {
        CRC ^= paquete[i];
        for (int j = 0; j < 8; j++) {
            if (CRC & 0x0001) {
                CRC >>= 1;
                CRC ^= 0xA001;
            } else CRC >>= 1;
        }
    }
    return CRC;
}


int CRCok(uint8_t * paquete) {
// Devuelve un buleano cierto si el CRC16 del paquete es correcto


    uint16_t CRC;
    uint8_t nBytes;
    if (paquete[1] == 0x10) {
        if (paquete[5] == 1)
            nBytes = 11;
        else nBytes = 13;
    } else nBytes = 8;
    CRC = calculaCRC(paquete, nBytes);
    return ((CRC & 0xFF) == paquete[nBytes-2]) && ((CRC >> 8) == paquete[nBytes-1]);
}


void escribeRegistro(uint16_t nRegistro, uint16_t valor) {
// Escribe el registro Modbus indicado en 'nRegistro' con el valor indicado en 'valor'

    uint8_t * p = (uint8_t*) 0xF84;
    p[nRegistro * 2] = valor & 0xFF;
    p[nRegistro * 2 + 1] = valor >> 8;
}


int escribeRegistros(uint8_t * paquete) {
// Escribe los registros Modbus indicados en el paquete. Devuelve un buleano falso
// si en el paquete se indican registros inexistentes.

    uint16_t nRegistros = paquete[4] << 8 | paquete[5];
    uint16_t primerRegistro = paquete[2] << 8 | paquete[3];
    if (primerRegistro + nRegistros <= 2) {
        for (int i = 0; i < nRegistros; i++)
            escribeRegistro(primerRegistro + i, paquete[7 + i*2] << 8 | paquete[7+i*2+1]);
        return 1;
    } else return 0;
}
```

```
void respuestaEscritura(uint8_t * paquete) {
// Envía por el canal la respuesta ante la orden de escritura indicada en 'paquete'

    for(int i = 0; i < 6; i++)
        enviaByte(paquete[i]);
    uint16_t CRC = calculaCRC(paquete, 6);
    enviaByte(CRC & 0xFF);
    enviaByte(CRC >> 8);
}


int leeRegistros(uint8_t * paquete) {
// Devuelve un buleano cierto si en la orden de lectura indicada en el paquete se indican
// sólo registros Modbus 0 y/o 1

    uint16_t nRegistros = paquete[4] << 8 | paquete[5];
    uint16_t primerRegistro = paquete[2] << 8 | paquete[3];
    return primerRegistro + nRegistros <= 2;
}


void respuestaLectura(uint8_t * paquete) {
// Envía la respuesta a una orden de lectura de registros indicada en 'paquete'

    uint16_t nRegistros = paquete[4] << 8 | paquete[5];
    uint16_t primerRegistro = paquete[2] << 8 | paquete[3];
    uint8_t paqueteRespuesta[9];
    paqueteRespuesta[0] = DIRECCION_MODBUS;
    paqueteRespuesta[1] = MODBUS_LECTURA;
    paqueteRespuesta[2] = nRegistros * 2;
    uint8_t * p = (uint8_t*)0xF80;
    for (int i = 0; i < nRegistros; i++) {
        paqueteRespuesta[3+i*2] = p[(i+primerRegistro)*2];
        paqueteRespuesta[3+i*2+1] = p[(i+primerRegistro)*2+1];
    }
    uint16_t CRC = calculaCRC(paqueteRespuesta, 3 + nRegistros * 2);
    paqueteRespuesta[3 + nRegistros*2] = CRC & 0xFF;
    paqueteRespuesta[3 + nRegistros*2 + 1] = CRC >> 8;
    for (int i = 0; i < 5 + nRegistros*2; i++)
        enviaByte(paqueteRespuesta[i]);
}


int esModbusEscritura(uint8_t * paquete) {
// Devuelve un buleano cierto si en el paquete hay una orden Modbus de escritura

    return paquete[1] == MODBUS_ESCRITURA;
}


int esModbusLectura(uint8_t * paquete) {
    // Devuelve un buleano cierto si en el paquete hay una orden Modbus de lectura

    return paquete[1] == MODBUS_LECTURA;
}
```

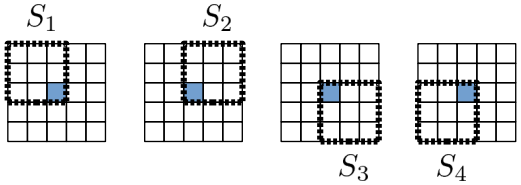
```c
int main() {
    uint8_t paquete[13];
    configuraSenalesDigitales();
    instalaTemporizador(interrupcionTemporizador);
    while(1) {
        if (recibeDireccionModbus() == DIRECCION_MODBUS) {
            if (recibeOrden(paquete)) {
                if (CRCok(paquete)) {
                    if (esModbusEscritura(paquete)) {
                        if (escribeRegistros(paquete))
                            respuestaEscritura(paquete);
                    } else if (esModbusLectura(paquete)) {
                        if (leeRegistros(paquete))
                            respuestaLectura(paquete);
                    }
                }
            }
        }
    }
    return 0;
}
```

---

3) Codifica una función en lenguaje C para poder aplicar un filtro de Kuwahara de cualquier grado sobre una imagen en blanco y negro con el objetivo de reducir el ruido, donde el nivel de gris de cada punto se representa mediante un byte sin signo (desde valor 0 para color negro hasta valor 255 para color blanco). Los bytes que representan a una imagen se guardan en memoria de forma consecutiva, desde la fila superior de la imagen a la inferior y en cada fila de izquierda a derecha.

El filtro de Kuwahara de grado L (siendo $L > 0$ un entero) aplica un algoritmo a cada punto P de la imagen, de coordenadas (p_x, p_y), donde se utiliza una región cuadrada de la imagen de $4L + 1$ puntos de lado, y donde el punto P es el punto central de dicha región cuadrada. Esta región se divide en cuatro sectores cuadrados S_1 a S_4, como se indica en la figura para el caso particular $L = 1$. Las diagonales de estos sectores van desde el punto P central a cada una de las cuatro esquinas de la región cuadrada. Para cada sector S_1 a S_4 se obtiene el valor medio m_1 a m_4 y la desviación típica σ_1 a σ_4 del valor del color de todos sus puntos. En la siguiente expresión se indica cómo se realiza el cálculo de la desviación típica σ en un sector con media de color m, con N puntos y donde el sumatorio se extiende a lo largo de las coordenadas (x, y) de todos sus puntos, cada uno de color $c(x, y)$.

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{x,y}(c(x,y) - m)^2}$$



El filtro de Kuwara se aplica a cada punto P de la imagen original para obtener una imagen filtrada con las mismas dimensiones, donde en la misma posición de P el color es la media de color del sector que presente la desviación típica más pequeña.

Una solución:

```c
#include <stdio.h>
#include <math.h>
#include <stdint.h>
#include <stdlib.h>

int minimo(float lista[]) {
    float m=lista[0];
    int index=0;
    for (int i=0; i < sizeof(lista)/sizeof(lista[0]); i++)
        if (m>lista[i]) {
            m=lista[i];
            index=i;
        }
    return index;
}
```

```
void filtroKuwahara(uint8_t * pMatriz, const unsigned int L, const unsigned int ancho,
    const unsigned int alto, uint8_t * nuevaMatriz) {
    // Se usa una función void porque el resultado de salida es muy complejo (o grande) como para
    // devolverlo en un dato simple de C, como un intPara ello se usan punteros a matrices,
    // (uint8_t * matriz)

    for (unsigned int y=0; y<alto; y++) {
        for (unsigned int x=0; x<ancho; x++) {
            if (y<2*L || y>alto-2*L || x<2*L || x>=ancho-2*L) {
                nuevaMatriz[y*ancho + x]=0; //Todo a negro por ser parte del borde
            }
            else { // Si no es el borde, se aplica el filtro
                float sigma[4]={0,0,0,0};
                float media[4]={0,0,0,0};
                int N=(2*L+1)*(2*L+1);

                // Comenzamos con el análisis de los sectores
                //S1
                for (unsigned int yy=y-2*L; yy<=y; yy++) { //cálculo de la media
                    for (unsigned int xx=x-2*L; xx<=x; xx++) {
                        media[0]+=pMatriz[yy*ancho + xx];
                    }
                }
                media[0]/=N;
                for (unsigned int yy=y-2*L; yy<=y; yy++) { //cálculo de sigma1
                    for (unsigned int xx=x-2*L; xx<=x; xx++) {
                        sigma[0]+= (pMatriz[xx*(yy+1)] - media[0]) * (pMatriz[xx*(yy+1)] - media[0]);
                    }
                }
                sigma[0]=sqrt(1/(N-1)*sigma[0]);

                //S2
                for (unsigned int yy=y-2*L; yy<=y; yy++) { //cálculo de la media
                    for (unsigned int xx=x; xx<=x+2*L; xx++) {
                        media[1] += pMatriz[yy*ancho + xx];
                    }
                }
                media[1]/=N;
                for (unsigned int yy=y-2*L; yy<=y; yy++) { //cálculo de sigma2
                    for (unsigned int xx=x; xx<=x+2*L; xx++) {
                        sigma[1] += (pMatriz[xx*(yy+1)] - media[1]) * (pMatriz[xx*(yy+1)] - media[1]);
                    }
                }
                sigma[1]=sqrt(1/(N-1)*sigma[1]);

                //S3
                for (unsigned int yy=y; yy<=y+2*L; yy++) { //cálculo de la media
                    for (unsigned int xx=x; xx<=x+2*L; xx++) {
                        media[2] += pMatriz[yy*ancho + xx];
                    }
                }
                media[2]/=N;
                for (unsigned int yy=y; yy<=y+2*L; yy++) { //cálculo de sigma3
                    for (unsigned int xx=x; xx<=x+2*L; xx++) {
                        sigma[2] += (pMatriz[xx*(yy+1)] - media[2]) * (pMatriz[xx*(yy+1)] - media[2]);
                    }
                }
                sigma[2]=sqrt(1/(N-1)*sigma[2]);

                //S4
                for (unsigned int yy=y; yy<=y+2*L; yy++) { //cálculo de la media
                    for (unsigned int xx=x-2*L; xx<=x; xx++) {
                        media[3] += pMatriz[yy*ancho + xx];
                    }
                }
                media[3]/=N;
```

```
                for (unsigned int yy=y; yy<=y+2*L; yy++) { //cálculo de sigma4
                    for (unsigned int xx=x-2*L; xx<=x; xx++) {
                        sigma[3] += (pMatriz[xx*(yy+1)] - media[3]) * (pMatriz[xx*(yy+1)] - media[3]);
                    }
                }
                sigma[3]=sqrt(1/(N-1)*sigma[3]);

                // Con la función mínimo se accede al índice de la desviación típica media y se escribe el
                // valor de la media asociada en el punto de la nueva imagen
                nuevaMatriz[y*ancho + x] =(uint8_t) media[minimo(sigma)];

            }
        }
    }

    return;
}

void creaImagen(uint8_t * pMatriz, int tamano) {
    for (int i=0; i<tamano; i++) {
        pMatriz[i]=rand() % 256;
    }
}

int main() {
    const int anchura=100;//1024
    const int altura=100;//768
    const int grado=2;//2
    uint8_t imagen[anchura * altura];
    uint8_t nuevaImagen[anchura * altura];

    srand(123456789);
    creaImagen(imagen,anchura*altura);
    filtroKuwahara(imagen,grado,anchura,altura,nuevaImagen);

    return 0;
}
```

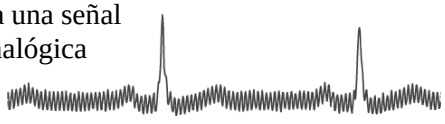
Otra solución:

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


typedef struct {
    float media;
    float sumatorio;
    int valido;
} DatosSector;


DatosSector calculaSector(uint8_t * pImagen, int grado, int nFilas, int nColumnas,
    int x, int y, int nSector) {

    int xInicial, yInicial;
    DatosSector datos;
    switch(nSector) {
    case 1:
        xInicial = x - grado * 2;
        yInicial = y - grado * 2;
        datos.valido = x > grado*2 && y > grado*2;
        break;
    case 2:
        xInicial = x;
        yInicial = y - grado * 2;
        datos.valido = x < nColumnas-grado*2 && y > grado*2;
        break;
```

```
    case 3:
        xInicial = x;
        yInicial = y;
        datos.valido = x < nColumnas-grado*2 && y < nFilas-grado*2;
        break;
    case 4:
        xInicial = x - grado * 2;
        yInicial = y;
        datos.valido = x > grado*2 && y < nFilas-grado*2;
        break;
    }

    if (datos.valido) {
        datos.media = 0;
        for (int y1 = yInicial; y1 < yInicial + grado * 2; y1++)
            for (int x1 = xInicial; x1 <= xInicial + grado * 2; x1++)
                datos.media = datos.media + pImagen[y1 * nColumnas + x1];
        datos.media = datos.media / ((2*grado+1)*(2*grado+1));
        datos.sumatorio = 0;
        for (int y1 = yInicial; y1 < yInicial + grado * 2; y1++)
            for (int x1 = xInicial; x1 <= xInicial + grado * 2; x1++)
                datos.sumatorio = datos.sumatorio +
                    (pImagen[y1 * nColumnas + x1] - datos.media) *
                    (pImagen[y1 * nColumnas + x1] - datos.media);
    }
    return datos;
}


void filtroKuwahara(uint8_t * pImagen, uint8_t * pImagenProcesada,
     int grado, int nFilas, int nColumnas) {

    DatosSector datos[4];

    for(int y = 0; y < nFilas; y++)
        for(int x = 0; x < nColumnas; x++) {
            for(int sector = 1; sector <= 4; sector++)
                datos[sector-1] = calculaSector(pImagen, grado, nFilas, nColumnas,
                                                x, y, sector);
            int sector = 0;
            while(!datos[sector].valido)
                sector++;
            for (int i = sector+1; i < 4; i++)
               if (datos[sector].sumatorio > datos[i].sumatorio && datos[i].valido)
                   sector = i;
            pImagenProcesada[y * nColumnas + x] = datos[sector].media;
         }
}


uint8_t imagen[1024 * 1024];
uint8_t imagenProcesada[1024 * 1024];

int main()
{
    filtroKuwahara(imagen, imagenProcesada, 1, 1024, 1024);
    return 0;
}
```

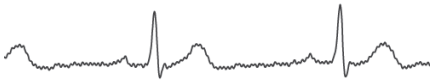
---

4) 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:



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

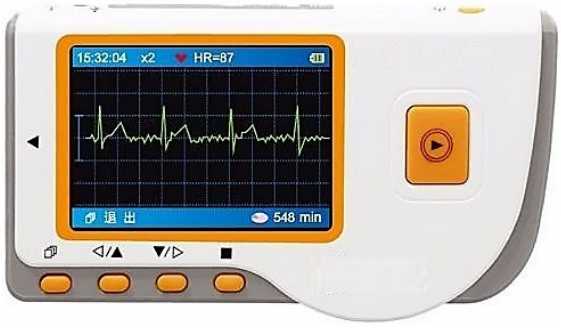
El filtro parte de condiciones iniciales nulas.

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 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
        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 * 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
            }
        }
    }
}
```

---

5) 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.

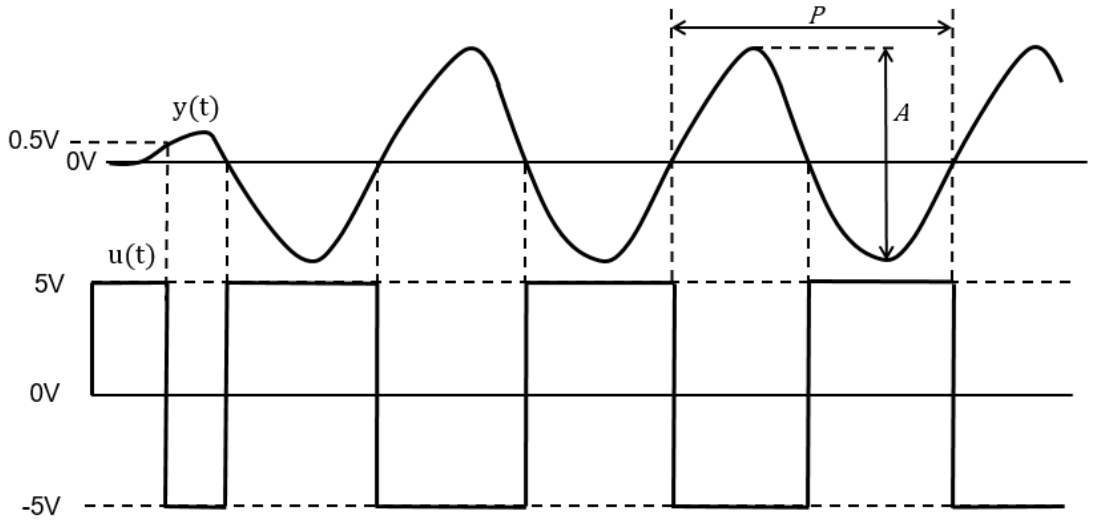
- Primero se aplicará una actuación de 5V hasta que la salida de la planta alcance 0.5 V.

- 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 <stdio.h>
#define D 5
#define PI 3.14159265359

void salidaAnalogica(float salida);

double entradaAnalogica(int entrada);

unsigned int MS;

float mediaDe5(float* numeros) {
    float med=0;
    for (int i=0; i<5; i++) med+=numeros[i];
    return med/5;
}

void desplazaYAnade5(float* numeros,float add) {
    for (int i=5-1; i>=0; i--) {
        if (i!=0) numeros[i]=numeros[i-1];
        else numeros[i]=add;
    }
}

void ajustePID(float* pKp, float* pKi, float* pKd) {
    float P[5]={0,0,0,0,0}, A[5]={0,0,0,0,0};

    do
        salidaAnalogica(5);
    while (entradaAnalogica(1)<0.5);

    do
        salidaAnalogica(-5);
    while (entradaAnalogica(1)>=0);

    float yPrev=0;
    float yAct=entradaAnalogica(1);
    int condicion=0;

    do {
        salidaAnalogica(5);
        MS=0;
        float min=0, max=0;
        do {
            if (yAct<min) min=yAct;
            if (yAct>max) max=yAct;
            yPrev=yAct;
            yAct = entradaAnalogica(1);
            if (yPrev<0 && yAct>=0) salidaAnalogica(-5);
        } while (!(yPrev>0 && yAct<=0));
        desplazaYAnade5(P,MS);
        desplazaYAnade5(A,max-min);
        float medP=mediaDe5(P);
        float medA=mediaDe5(A);
        for (int i=0;i<5;i++) {
            if (!(P[i]>medP*(1-0.02) && P[i]<medP*(1+0.02) && A[i]>medA*(1-0.02) && A[i]<medA*(1+0.02))){
                condicion=0;
                break;}
            else condicion=1;
        }
    } while (!condicion);
    float Ku=4*D/PI/mediaDe5(A);
    *pKp=0.6*Ku;
    *pKi=1.2*Ku/mediaDe5(P);
    *pKd=0.075*Ku;

}
```

```
int main()
{
    float Kp,Ki,Kd;
    ajustePID(&Kp,&Ki,&Kd);

    return 0;
}
```

---

6) 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 CLR está conectada a 5V, no se utiliza. La transferencia de información desde el microcontrolador al convertidor DA se realizará según se indica en la figura 5 del manual del convertidor. En el microcontrolador se utilizan los puertos de señales digitales PORTA, PORTB y PORTC, cada uno de 8 bits, accesibles en direcciones de memoria consecutivas desde 0xF80 y configurables en direcciones consecutivas desde 0xF92 donde residen los registros TRISA, TRISB y TRISC donde un bit a 0 hace que la correspondiente señal actúe como salida y un 1 como entrada.
Este convertidor AD5725 se conecta al microcontrolador de la siguiente forma:
- el bus de datos de 12 bits D0 a D11 se conecta a las 8 señales digitales del puerto PORTA (los 8 bits menos significativos D0 a D7) y a las 4 señales menos significativas del puerto PORTB (los 4 bits más significativos D8 a D11).
- las señales A0, A1, RW, CS y LDAC del convertidor se conectan a las 5 señales menos significativas del puerto PORTC y en ese orden.

En la clase a desarrollar tiene que existir al menos un método que permita modificar todas las salidas analógicas y un método que permita modificar una única salida analógica.

Una solución:

```
#include <stdint.h>

class AD5725 {

public:
    AD5725();
    void salidas(float s0, float s1, float s2, float s3);
};

AD5725::AD5725() {
    uint8_t * p = (uint8_t*) 0xF92;  // Para acceder a los registros de configuración de los puertos

    p[0] = 0;
    p[1] = p[1] & 0xF0;
    p[2] = p[2] & 0xE0;
    // Configura las salidas en los puertos PORTA, PORTB y PORTC

    p = (uint8_t*) 0xF80;
    p[2] = p[2] | 0x1C;
    // Pone a 1 las salidas 2, 3 y 4 de PORTC, conectadas a RW, CS y LDAC
}

void AD5725::salidas(float s0, float s1, float s2, float s3) {
    // Modifica las cuatro salidas analógicas del convertidor a los valores de
    // tensión indicados por parámetro

    uint8_t * p = (uint8_t*) 0xF80;  // Apunta a los SFR de los puertos

    p[2] &= ~(1 << 2);  // RW = 0

    p[2] &= 0xFC;  // A0 = 0, A1 = 0
    int n = (int)(s0 * 4095 / 5);
    p[0] = n & 0xFF;  // PORTA = D0 a D7
    p[1] &= 0xF0;
```

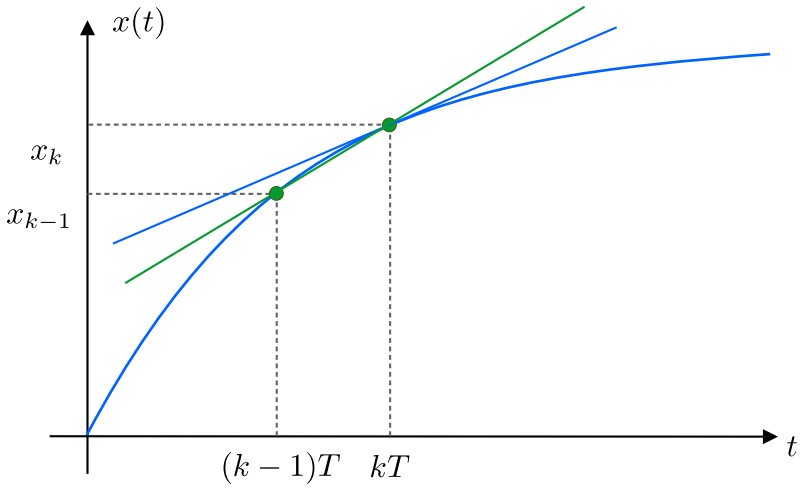
```
    p[1] |= (n & 0xF00) >> 8;  // PORT = D8 a D11
    p[2] &= ~(1 << 3);  // CS = 0
    p[2] |= 1 << 3;  // CS = 1

    p[2] &= 0xFC;
    p[2] |= 0x01; // A0 = 1, A1 = 0
    n = (int)(s1 * 4095 / 5);
    p[0] = n & 0xFF;  // PORTA = D0 a D7
    p[1] &= 0xF0;
    p[1] |= (n & 0xF00) >> 8;  // PORT = D8 a D11
    p[2] &= ~(1 << 3);  // CS = 0
    p[2] |= 1 << 3;  // CS = 1

    p[2] &= 0xFC;
    p[2] |= 0x02; // A0 = 0, A1 = 1
    n = (int)(s2 * 4095 / 5);
    p[0] = n & 0xFF;  // PORTA = D0 a D7
    p[1] &= 0xF0;
    p[1] |= (n & 0xF00) >> 8;  // PORT = D8 a D11
    p[2] &= ~(1 << 3);  // CS = 0
    p[2] |= 1 << 3;  // CS = 1

    p[2] &= 0xFC;
    p[2] |= 0x03; // A0 = 1, A1 = 1
    n = (int)(s3 * 4095 / 5);
    p[0] = n & 0xFF;  // PORTA = D0 a D7
    p[1] &= 0xF0;
    p[1] |= (n & 0xF00) >> 8;  // PORT = D8 a D11
    p[2] &= ~(1 << 3);  // CS = 0
    p[2] |= 1 << 3;  // CS = 1

    p[2] &= ~(1 << 4);  // LDAC = 0
    p[2] |= 0x1C;  // RW = CS = LDAC = 1
}

int main()
{
    AD5725 convertidor;
    // Construye un objeto para manejar un convertidor AD5725

    convertidor.salidas(1.5, 2.4, 0.3, 2.7);
    // Modifica las salidas analógicas del convertidor

    return 0;
}
```

---

7) La derivada temporal de una señal contínua es

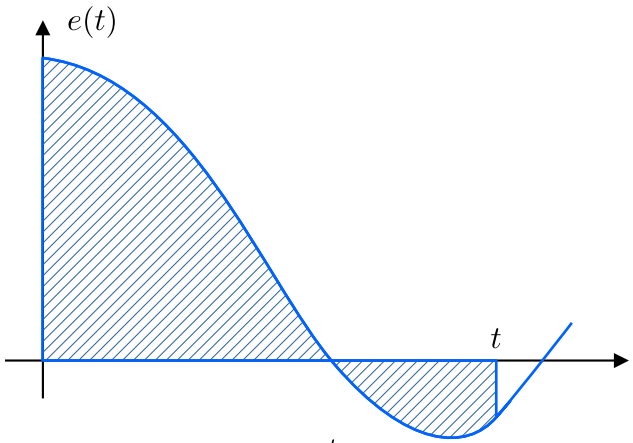
$$\frac{dx}{dt}(t) = \lim_{\tau \to 0} \frac{x(t) - x(t-\tau)}{\tau}$$

Si se utiliza un período de muestreo T relativamente pequeño con respecto a la dinámica de la señal, se puede utilizar la aproximación de Euler hacia atrás en $t = kT$.



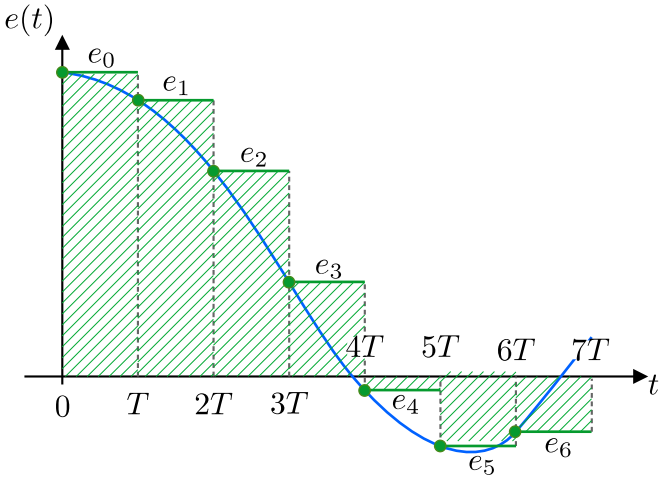
$$\frac{dx}{dt}(kT) \approx \frac{x(kT) - x((k-1)T)}{T} = \frac{x_k - x_{k-1}}{T} = \frac{\Delta x}{T}$$

La integral $u(t)$ de esa señal $e(t)$ desde $t = 0$ hasta $t = kT$ se puede aproximar de tres formas diferentes:



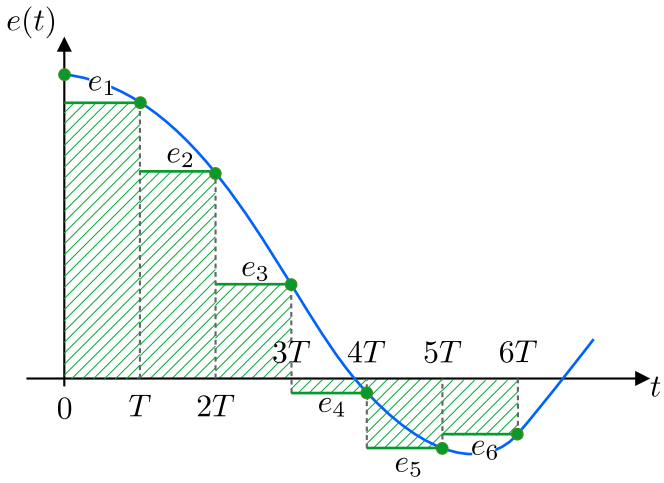
$$u(t) = \int_0^t e(\tau)d\tau$$

a) método rectangular hacia adelante



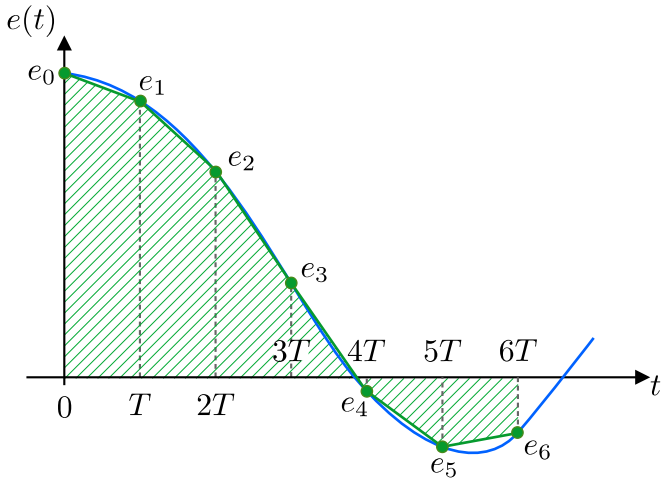
$$u_k = u(kT) = \int_0^{kT} e(\tau)d\tau \cong \sum_{i=1}^{k} Te_{i-1} = T\sum_{i=1}^{k} e_{i-1} = u_{k-1} + Te_{k-1}$$

b) método rectangular hacia atrás



$$u_k = u(kT) = \int_0^{kT} e(\tau)d\tau \cong \sum_{i=1}^{k} Te_i = T\sum_{i=1}^{k} e_i = u_{k-1} + Te_k$$

c) método trapezoidal



$$u_k = u(kT) = \int_0^{kT} e(\tau)d\tau \cong \sum_{i=1}^{k} T\frac{e_i + e_{i-1}}{2} = \frac{T}{2}\sum_{i=1}^{k}(e_i + e_{i-1})$$
$$= u_{k-1} + \frac{T}{2}(e_k + e_{k-1})$$

Si el período de muestreo T es lo suficientemente pequeño, no hay una diferencia significativa entre estas aproximaciones de la integral.

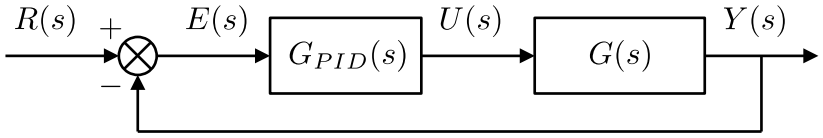
Para para la aproximación del término integral se utiliza más frecuentemente la aproximación rectangular hacia atrás.

La ecuación diferencial de un regulador PID contínuo en su forma ideal es:

$$u(t) = K_p\left[e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\frac{de(t)}{dt}\right] = K_p e(t) + K_i\int_0^t e(\tau)d\tau + K_d\frac{de(t)}{dt}$$

$$K_i = \frac{K_p}{T_i} \qquad K_d = K_p T_d$$

$$G_{PID}(s) = \frac{U(s)}{E(s)} = K_p\left(1 + \frac{1}{T_i s} + sT_d\right) = K_p + K_i\frac{1}{s} + K_d s$$



Discretización del término derivativo con un período de muestreo T lo suficientemente rápido mediante el método de Euler hacia atrás:

$$\frac{de}{dt}(kT) \cong \frac{e_k - e_{k-1}}{T}$$

Discretización de la integral del error con el método rectangular hacia adelante:

$$\int_0^{kT} e(\tau)d\tau \cong T\sum_{i=1}^{k} e_{i-1}$$

El PID discretizado queda:

$$u_k = K_p\left[e_k + \frac{T}{T_i}\sum_{i=1}^{k} e_{i-1} + \frac{T_d}{T}(e_k - e_{k-1})\right]$$

Si la integral del error se aproxima con el método rectangular hacia atrás:

$$\int_0^{kT} e(\tau)d\tau \cong T\sum_{i=1}^{k} e_i$$

$$u_k = K_p\left[e_k + \frac{T}{T_i}\sum_{i=1}^{k} e_i + \frac{T_d}{T}(e_k - e_{k-1})\right]$$

Si la integral del error se aproxima por el método trapezoidal:

$$\int_0^{kT} e(\tau)d\tau \cong T\sum_{i=1}^{k}\frac{e_i + e_{i-1}}{2} = T\left(\frac{e_0 + e_k}{2} + \sum_{i=0}^{k-1} e_i\right)$$

$$u_k = K_p\left[e_k + \frac{T}{T_i}\left(\frac{e_0 + e_k}{2} + \sum_{i=0}^{k-1} e_i\right) + \frac{T_d}{T}(e_k - e_{k-1})\right]$$

El PID discreto se suele implantar utilizando una formulación recursiva en vez de utilizar los sumatorios que figuran en las expresiones anteriores, ya que un reajuste de K_p o T_i en un instante determinado se aplica de forma retroactiva a la suma de errores anteriores en la integral, lo cual provoca cambios bruscos en la actuación.

Con la aproximación integral hacia atrás:

$$u_k = K_p\left[e_k + \frac{T}{T_i}\sum_{i=1}^{k} e_{i-1} + \frac{T_d}{T}(e_k - e_{k-1})\right]$$

$$u_{k-1} = K_p \left[e_{k-1} + \frac{T}{T_i} \sum_{i=1}^{k-1} e_{i-1} + \frac{T_d}{T}(e_{k-1} - e_{k-2}) \right]$$

$$u_k - u_{k-1} = K_p \left[e_k - e_{k-1} + \frac{T}{T_i} e_k + \frac{T_d}{T}(e_k - 2e_{k-1} + e_{k-2}) \right]$$

$$u_k = u_{k-1} + K_p \left(1 + \frac{T}{T_i} + \frac{T_d}{T} \right) e_k - K_p \left(1 + 2\frac{T_d}{T} \right) e_{k-1} + K_p \frac{T_d}{T} e_{k-2}$$

$$u_k = u_{k-1} + q_0 e_k + q_1 e_{k-1} + q_2 e_{k-2}$$

$$q_0 = K_p \left(1 + \frac{T}{T_i} + \frac{T_d}{T} \right), \quad q_1 = K_p \left(1 + 2\frac{T_d}{T} \right), \quad q_2 = K_p \frac{T_d}{T}$$

En función de la aproximación de la integral:

Parámetros	Hacia delante	Hacia atrás	Trapezoidal
q_0	$K_p \left(1 + \frac{T_d}{T} \right)$	$K_p \left(1 + \frac{T}{T_i} + \frac{T_d}{T} \right)$	$K_p \left(1 + \frac{T}{2T_i} + \frac{T_d}{T} \right)$
q_1	$-K_p \left(1 - \frac{T}{T_i} + 2\frac{T_d}{T} \right)$	$-K_p \left(1 + 2\frac{T_d}{T} \right)$	$-K_p \left(1 - \frac{T}{2T_i} + 2\frac{T_d}{T} \right)$
q_2	$K_p \frac{T_d}{T}$	$K_p \frac{T_d}{T}$	$K_p \frac{T_d}{T}$

Codifica una clase que implante un regulador PID digital utilizando una formulación recursiva y pudiendo utilizar diferentes aproximaciones del término integral.

Una solución:

```
#include <iostream>
using namespace std;


class PID {
public:
    enum Aproximacion {Delante, Atras, Trapezoidal};
private:
    Aproximacion aproximacion;
    double Kp, Ti, Td, T;
    double e1, e2, u1;
    double q0, q1, q2;
    void calculaCoeficientes();
public:
    PID(double, double, double, double, Aproximacion);
    void setKp(double);
    void setTi(double);
    void setTd(double);
    void setT(double);
    void setParametros(double, double, double);
    void setAproximacion(Aproximacion);
    double actuacion(double);
};
```

```
PID::PID(double Kp, double Ti, double Td, double T, Aproximacion aproximacion) {
    setParametros(Kp, Ti, Td);
    this->T = T;
    this->aproximacion = aproximacion;
    calculaCoeficientes();
    u1 = 0;
    e1 = 0;
    e2 = 0;
}


void PID::setKp(double valor) {
    Kp = valor;
    calculaCoeficientes();
}


void PID::setTi(double valor) {
    Ti = valor;
    calculaCoeficientes();
}


void PID::setTd(double valor) {
    Td = valor;
    calculaCoeficientes();
}

void PID::setT(double valor) {
    T = valor;
    calculaCoeficientes();
}


void PID::setParametros(double Kp, double Ti, double Td) {
    this->Kp = Kp;
    this->Ti = Ti;
    this->Td = Td;
    calculaCoeficientes();
}


void PID::setAproximacion(Aproximacion aproximacion) {
    this->aproximacion = aproximacion;
    calculaCoeficientes();
}


double PID::actuacion(double e) {
    double u = u1 + q0 * e + q1 * e1 + q2 * e2;
    u1 = u;
    e2 = e1;
    e1 = e;
    return u;
}


void PID::calculaCoeficientes() {
    switch(aproximacion) {
    case Delante:
        q0 = Kp * (1 + Td / T);
        q1 = -Kp * (1 - T / Ti + 2 * Td / T);
        q2 = Kp * Td / T;
        break;
        // Definir casos Atras, Trapezoidal
    }
}
```

```
int main() {

    PID pid(3.5, 10.3, 5.3, 0.01, PID::Atras);
    arrancaTemporizador(0.01);
    double r = 800;
    while(1) {
        double y = mideSalida();
        double e = r - y;
        double u = pid.actuacion(e);
        actua(u);
        esperaTemporizacion();
    }

    return 0;
}
```

---

8) El controlador PID digital se puede formular generando la actuación u_k separando en paralelo la componente proporcional p_k, integral i_k y derivativa d_k.

Esto puede ser interesante para poder tratar y ajustar cada componente por separado.

$$u_k = p_k + i_k + d_k$$

$$p_k = K_p e_k \;\Rightarrow\; P(z) = K_p E(z)$$

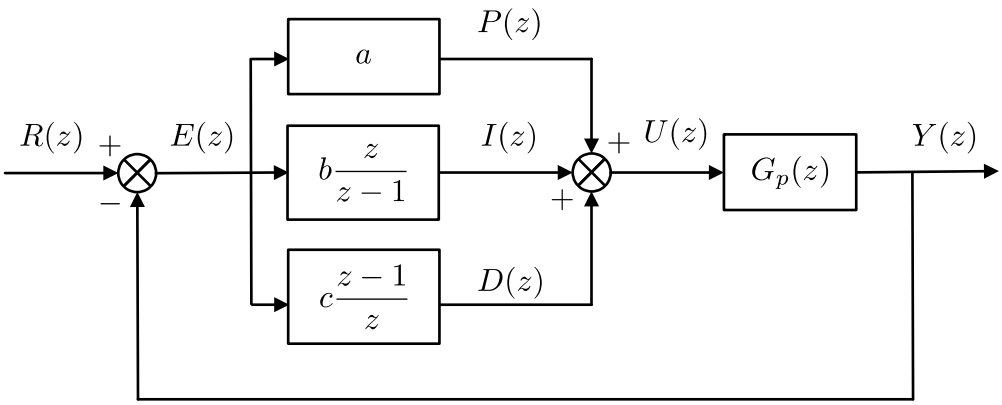
$$i_k = K_p \frac{T}{T_i} \sum_{i=1}^{k} e_i = i_{k-1} + K_p \frac{T}{T_i} e_k \;\Rightarrow\; I(z) = z^{-1} I(z) + \frac{K_p T}{T_i} E(z)$$

$$\Rightarrow\; I(z) = \frac{K_p T}{T_i (1 - z^{-1})} E(z) = \frac{K_p T}{T_i} \frac{z}{z-1} E(z)$$

$$d_k = K_p T_d \frac{e_k - e_{k-1}}{T} \;\Rightarrow\; D(z) = \frac{K_p T_d}{T} (1 - z^{-1}) E(z) = \frac{K_p T_d}{T} \frac{z-1}{z} E(z)$$

$$U(z) = P(z) + I(z) + D(z) = \left(a + b \frac{z}{z-1} + c \frac{z-1}{z} \right) E(z)$$

$$a = K_p, \qquad b = K_p \frac{T}{T_i}, \qquad c = K_p \frac{T_d}{T}$$



$$P(z) = aE(z) \Rightarrow p_k = ae_k$$
$$I(z) = b\frac{1}{1-z^{-1}}E(z) \Rightarrow i_k = i_{k-1} + be_k$$
$$D(z) = c(1-z^{-1})E(z) \Rightarrow d_k = c(e_k - e_{k-1})$$
$$U(z) = P(z) + I(z) + D(z) \Rightarrow u_k = p_k + i_k + d_k$$

Codifica una clase que implante un regulador PID digital utilizando una formulación paralela.

---

9) Codifica un programa para la carga automática de objetos mediante un robot de tipo Scara en una cinta transportadora que está en movimiento contínuo, y utilizando la información que nos proporciona un sistema de visión artificial.

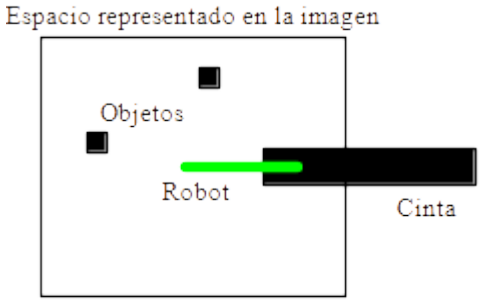


Figura 1

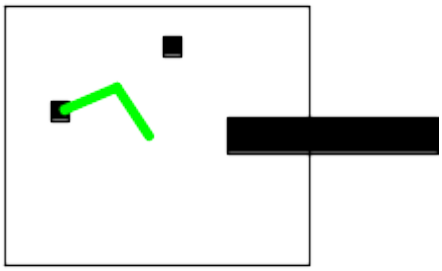


Figura 2

La imagen recogida por el sistema de visión es una vista en planta en la que la articulación fija del robot ocupa su centro, figura parte de la cinta y los objetos (como máximo 5 simultáneos) se encuentran diseminados y de forma que no se encuentran próximos entre sí, ni al robot, ni a la cinta, ni a los bordes de la imagen, de forma que se pueda simplificar su detección.

La imagen se compone de 256*256 puntos, en la que cada objeto sólo ocupa uno.
Cada punto de la imagen representa un cuadrado de 10 mm. de lado en la realidad.
Para obtener una imagen hay que realizar una llamada a la subrutina **`void recoge_imagen(unsigned char *)`**, pasándole la dirección de memoria en la que queremos almacenarla.
Esta función nos devuelve la imagen como un conjunto de bytes. Cada byte representa un punto (valor<100 negro, >200 blanco). El color del robot en la imagen, de la cinta y de los objetos es negro y el del suelo es blanco.
Los bytes se almacenan de forma que recorren la imagen de izquierda a derecha y de abajo hacia arriba por filas.

Para manejar el robot podemos utilizar las siguientes funciones:
`void sube()` sube el brazo vertical del robot,
`void baja()` baja el brazo vertical del robot,
`void abre()` abre la garra del robot,
`void cierra()` cierra la garra del robot y
`void muevete(double, double)` mueve el robot hasta que la garra se sitúa sobre el punto cuyas coordenadas X e Y (tomando como origen la esquina inferior izquierda de la imagen y expresándolas en mm.) se pasan como primer y segundo parámetro, respectivamente.

El programa tiene que utilizar el sistema de visión para detectar objetos colocados en su entorno por otros robots y debe mover este robot para que los vaya recogiendo automáticamente (figura 2) y los deposite en la cinta transportadora en la posición (2500, 1280).
Hay que manejar todo este sistema de forma que se puedan relizar capturas y procesado de imágenes mientras se está realizando cualquier movimiento con el robot.

Una solución:

```
int mx[5] = {-1, -1, -1, -1, -1}, my[5] = {-1, -1, -1, -1, -1};
semaforo sc = 0, sp = 0;
int iCarga = 0;
```

```
int iDescarga = 0;

void hVision() {
    unsigned char imagen[256*256];

    while(1) {
        recoge_imagen(imagen);
        for (int x = 1; x < 255; x++)
            for(int y = 1; y < 255; y++) {
                if (imagen[y*256+x] < 100 &&
                    imagen[y*256+x+1] > 200 &&
                    imagen[y*256+x-1] > 200 &&
                    imagen[(y+1)*256+x] > 200 &&
                    imagen[(y+1)*256+x-1] > 200 &&
                    imagen[(y+1)*256+x+1] > 200 &&
                    imagen[(y-1)*256+x] > 200 &&
                    imagen[(y-1)*256+x-1] > 200 &&
                    imagen[(y-1)*256+x+1] > 200) {
                    int encontrado = 0;
                    for (int i = 0; i < 5; i++) {
                        if (mx[i] == x && my[y] == y)
                            encontrado = 1;
                    }
                    if (! encontrado) {
                        mx[iCarga] = x;
                        my[iCarga] = y;
                        iCarga++;
                        if (iCarga == 5)
                            iCarga = 0;
                        signal(sc);
                        if (iCarga == iDescarga)
                            wait(sp);
                    }
                }
            }
    }
}

void hRobot() {
    while(1) {
        abre();
        sube();
        wait(sc);
        int x = mx[iDescarga] * 10;
        int y = my[iDescarga] * 10;
        mx[iDescarga] = -1;
        my[iDescarga] = -1;
        if (iCarga == iDescarga)
            signal(sp);
        muevete(x, y);
        iDescarga++;
        if (iDescarga == 5)
            iDescarga = 0;
        baja();
        cierra();
        sube();
        muevete(2500, 1280);
        baja();
    }
}

int main()
{
    concurrente {
        hVision();
        hRobot();
    }
    return 0;
}
```

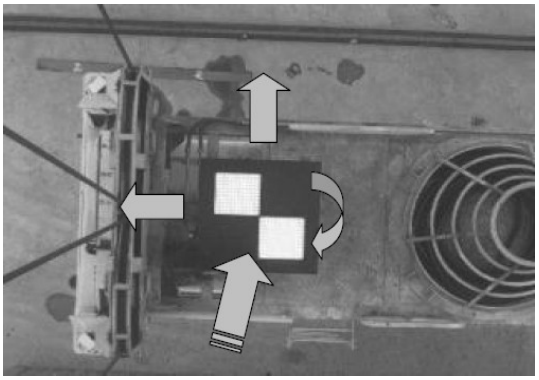
10) En un puente grúa existe un sistema embebido que dispone de una cámara en el carro superior apuntando hacia abajo para tomar imágenes de dos elementos reflectores cuadrados dispuestos en la plataforma de las garras tal y como se muestra en la figura. La cámara tiene una resolución de 1280x1024 puntos.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

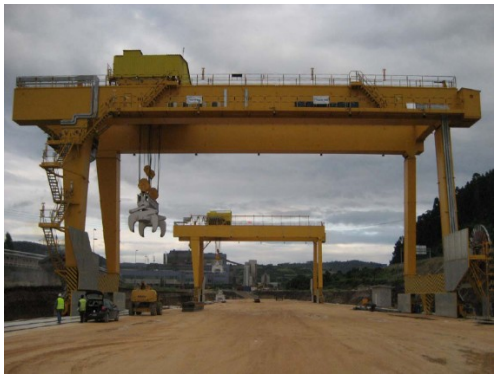
Se dispone de la función `void capturaImagen(uint8_t* p)` que acciona la cámara, espera a que tome una imagen y a que se almacene en la dirección indicada por parámetro. Cada punto de la imagen se guarda en un byte en blanco y negro expresando su nivel de gris (desde 0=negro hasta 255=blanco), recorriendo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

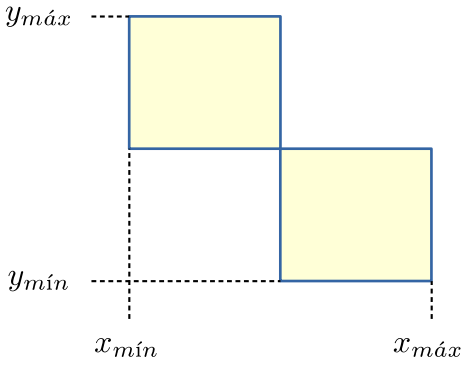
Codifica una aplicación para este sistema embebido que tiene que estar procesando continuamente las imágenes para determinar la la desviación de la carga en m. (como máximo, 1 m.) adelante-atrás e izquierda-derecha, así como el ángulo de giro de la carga (como máximo, ± 20 grados).

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.

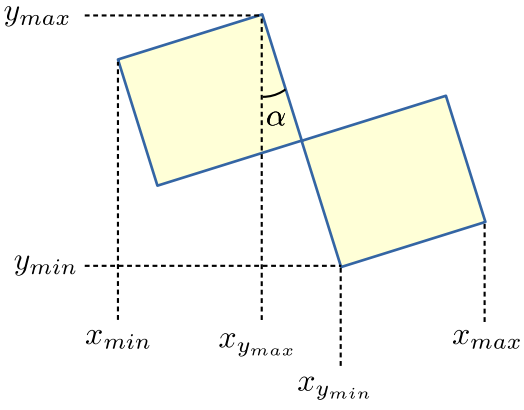
Existe la clase `Canal` para manejar las comunicaciones con un computador. Dispone de un constructor sin parámetros, un método `void envia(void* p, int n)` que transmite un paquete de `n` bytes guardados a partir de `p`, y un método `int recibe(void* p)` que espera a que se reciba un paquete de bytes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



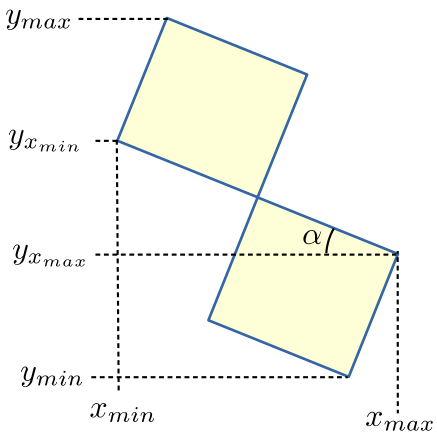




Ángulo negativo



Ángulo positivo



$$\alpha = -\arctan\left(\frac{x_{y_{min}} - x_{y_{max}}}{y_{max} - y_{min}}\right) \qquad \alpha = \arctan\left(\frac{y_{x_{min}} - y_{x_{max}}}{x_{max} - x_{min}}\right)$$

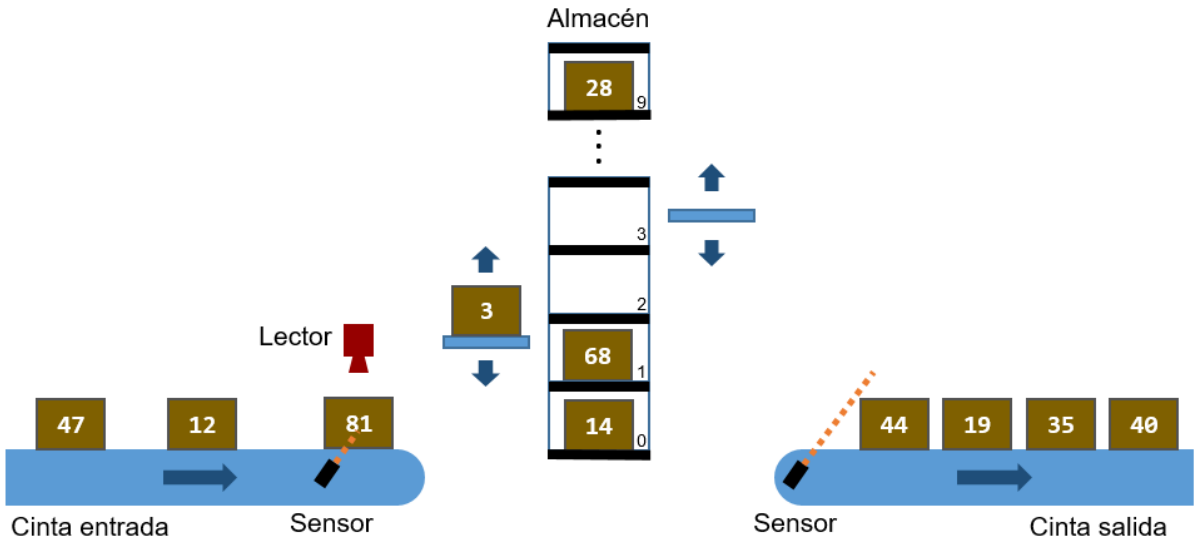
Una solución:

```
#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
    }
}
```

---

11) 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 y utiliza herramientas de sincronización para optimizar el uso de la CPU.

Una solución:

```
semaforo spCintaEntrada = 1, scElevadorEntrada = 0;
semaforo spElevadorEntrada = 0, scElevadorSalida = 0;
semaforo s = 1;
semaforo scCintaSalida = 0, spElevadorSalida = 1;
uint32_t codigos[10];
int libres[10] = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
int nPaquetes = 0;

void hCintaEntrada() {
    while(1) {
        wait(spCintaEntrada);
        cintaEntrada(1);
        while(! sensorEntrada());
        cintaEntrada(0);
        signal(scElevadorEntrada);
    }
}

void hElevadorEntrada() {
    int i;

    while(1) {
        elevadorEntrada(0);
        wait(scElevadorEntrada);
        uint32_t codigo = leeCodigo();
        cogeEntrada();
        signal(spCintaEntrada);

        int hayLibre = 0;
        wait(s);
        for (i = 0; i < 10; i++)
            if (libres[i]) {
                hayLibre = 1;
                break;
            }
        signal(s);
        if (hayLibre) {
            elevadorEntrada(i);
        } else {
            int iMin = 0;
            for (i = 1; i < 10; i++) {
                if (codigos[i] < codigos[iMin])
                    iMin = i;
            }
            elevadorEntrada(iMin);
            i = iMin;
            wait(spElevadorEntrada);
        }
        dejaEntrada();
        wait(s);
        nPaquetes ++;
        codigos[i] = codigo;
        libres[i] = 0;
        signal(s);
        signal(scElevadorSalida);
    }
}
```

```
void hElevadorSalida() {
    int i, codigoMinimo, primeroEncontrado, iMin;

    elevadorSalida(0);
    while(1) {
        wait(scElevadorSalida);
        primeroEncontrado = 0;
        wait(s);
        for (i = 0; i < 10; i++)
            if (!libres[i])
                if (primeroEncontrado) {
                    if (codigos[i] < codigoMinimo) {
                        codigoMinimo = codigos[i];
                        iMin = i;
                    } else {
                        codigoMinimo = codigos[i];
                        iMin = i;
                        primeroEncontrado = 1;
                    }
            }
            signal(s);
            elevadorSalida(iMin);
            cogeSalida();
            wait(s);
            libres[iMin] = 1;
            if (nPaquetes == 10)
                signal(spElevadorEntrada);
            nPaquetes--;
            signal(s);
            elevadorSalida(0);
            wait(spElevadorSalida);
            dejaSalida();
            signal(scCintaSalida);
    }
}

void hCintaSalida() {
    while(1) {
        wait(scCintaSalida);
        cintaSalida(1);
        while(sensorSalida());
        cintaSalida(0);
        signal(spElevadorSalida);
    }
}

int main() {
    concurrente {
        hCintaEntrada();
        hElevadorEntrada();
        hElevadorSalida();
        hCintaSalida();
    }
    return 0;
}
```

---

12) 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.

Para el controlador PI:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau$$

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

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

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

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

$$a = K_p + K_i T$$

Una solución para S1:

```
float consigna;
semaforo s = 0, sConsigna = 1;

void hConsola() {
    Consola consola;
    char texto[20];
    bool primeraVez = true;
    while(1) {
        consola.introduce(texto);
        wait(sConsigna);
        consigna = atof(texto);
        signal(sConsigna);
        if (primeraVez) {
            signal(s);
            primeraVez = false;
        }
    }
}
```

```
void hComunicaciones() {
    Canal canal;
    wait(s);
    Temporizador temporizador(1000);
    while(1) {
        temporizador.espera();
        float consignaLocal;
        wait(sConsigna);
        consignaLocal = consigna;
        signal(sConsigna);
        canal.envia(& consignaLocal, sizeof(float));

    }
}

int main() {
    concurrente {
        hComunicacion();
        hConsola();
    }
}
```

Para S2:

```
float consigna;
semaforo s = 1, sPrimeraConsigna = 0;
int nPeriodosEntreRecepciones = 0;

void hPI() {
    float uk, uk1 = 0, ek, ek1 = 0;
    float Kp = 0.1;
    float Ki = 0.1;
    float T = 0.1;
    float a = Kp + Ki * T;
    wait(sPrimeraConsigna);
    Temporizador temporizador(T * 1000);
    while(1) {
        temporizador.espera();
        wait(s);
        if (nPeridosEntreRecepciones > 1 / T)
            consigna = 0;
        ek = consigna - entrada();
        signal(s);
        uk = uk1 + a * ek - Kp * ek1;
        salida(uk);
        uk1 = uk;
        ek1 = ek;
    }
}

void hComunicaciones() {
    Canal canal;
    float consignaLocal;
    bool primeraVez = true;
    while(1) {
        canal.recibe(& consignaLocal);
        wait(s);
        consigna = consignaLocal;
        nPeriodosEntreRecepciones = 0;
        signal(s);
        if (primeraVez) {
            signal(sPrimeraConsigna);
            primeraVez = false;
        }
    }
}
```

```
int main() {
    concurrente {
        hPI();
        hComunicaciones();
    }

}
```

---

13) 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 * p)` para inicializar un semáforo binario apuntado por `p`.

- Las funciones `void bsemWait(bsem * p)` y `void bsemSignal(bsem * p)` para realizar las operaciones wait y signal sobre un semáforo binario apuntado por `p`.

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 * p, int valor)` para inicializar el semáforo apuntado por `p` a un determinado valor

- `void wait(semaforo * p)` y `void signal(semaforo * p)` 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));
}
```

---

14) 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();
    }
}
```