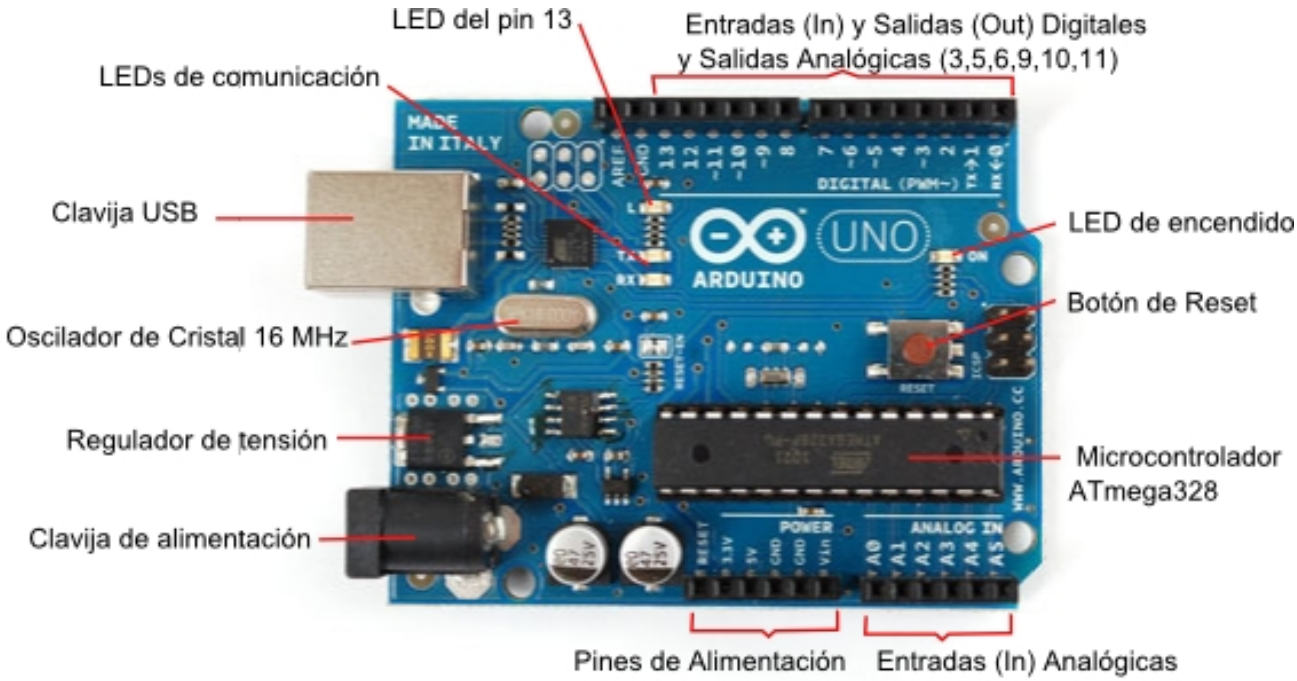
Apuntes de
ARDUINO
Nivel Enteraillo

Daniel Gallardo García
Profesor de Tecnología del IES Laguna de Tollón

1. CARACTERÍSTICAS DE ARDUINO



Las **características técnicas** de la Arduino UNO son:

Microcontrolador:	ATmega328
Tensión de funcionamiento:	5 V
Tensión de entrada (recomendada):	7 – 12 V
Tensión de entrada (límite):	6 – 20 V
Pines de Entradas/Salidas Digitales:	14 (6 proporcionan PWM)
Pines de Entradas Analógicas:	6
Intensidad C.C. por pines de E/S:	40 mA
Intensidad C.C. por el pin de 3,3 V:	50 mA
Intensidad C.C. por el pin de 5 V:	300 mA
Memoria Flash:	32 KB (0,5 KB para bootloader)
SRAM:	2 KB
EEPROM:	1 KB
Frecuencia señal de reloj:	16 MHz

Veamos algunos **pines** que pasamos por alto en el Nivel Pardillo:

- **pin0 RX**: se usa para recibir datos del PC (*serial data*). En caso de establecer una comunicación Serial en nuestro programa, no conectaremos nada a este pin.

- **pin1 TX**: se usa para transmitir datos al PC (*serial data*). En caso de establecer una comunicación Serial en nuestro programa, no conectaremos nada a este pin.

- **RESET**: funciona igual que el botón de reset. Cuando a este pin le llega un pulso de tensión de valor de 0 V (es decir, poner 0 V y luego volver a quitar el 0 V), la Arduino se resetea, comenzando a ejecutar desde el principio el programa que esté funcionando.

- **AREF**: en el caso de utilizar sensores que generen un rango de tensión por debajo de 5 V, podemos incrementar la precisión de la lectura haciendo que los 1024 valores posible no vayan desde los 0 V a los 5 V, sino a un rango menor de tensión (de 0 a 1,1 V; o de 0 a 3,3 V). Para ello empleamos la función **analogReference( )**, que presenta tres variantes:

  ```
  analogReference(INTERNAL);      // toma como tensión de referencia 1,1 V

  analogReference(EXTERNAL);      /* toma como referencia la tensión que haya en el pin AREF. Si quiero que esa tensión sea 3,3 V, lo único que tendré que hacer será conectar el pin 3,3V con el pin AREF */

  analogReference(DEFAULT);       // toma el valor por defecto: 5 V
  ```

  Para que surja efecto los nuevos valores de referencia de la tensión en las entradas analógicas, debemos llamar a la función `analogReference()` antes de utilizar la función `analogRead()`.

- Los **pines A0…A5** también pueden utilizarse como entradas digitales o como salidas analógicas y digitales. No tenemos más que incluirlas en el `void setup()` como los siguientes pines despúes del 13, es decir: el 14, 15, 16, 17, 18, 19 (que corresponden al A0, A1, A2, A3, A4, A5):

  ```
  pinMode(16,OUTPUT);  //utilizaré el pin A2 como salida
  ```

- Asimismo, no es estrictamente necesario configurar los pines como `INPUT` u `OUTPUT` dentro de **`void setup()`**. Simplemente debemos interpretar al bloque `void setup()` como una parte del código que Arduino solamente la corre una vez (al principio), y en la que podemos ejecutar cualquier función.

2. TIPOS DE VARIABLES

Los **tipos de variables** son:

- `boolean`: almacena un valor con dos posibilidades: 0 o 1, o verdadero o falso.
- `char`: almacena un caracter, como una letra o símbolo. También se puede emplear para un número entero entre -128 a 127 (1 byte).
- `byte`: almacena un número natural entre 0 y 255 (1 byte).
- `int`: almacena un número entero entre -32769 y 32767 (2 bytes).
- `unsigned int`: almacena un número natural entre 0 y 65536 (2 bytes).

- `long`: almacena un número entero entre -2147483648 y 2147483647 (4 bytes).
- `unsigned long`: almacena un número entero entre 0 y 4294967295 (4 bytes).
- `float`: almacena un número decimal con un rango entre $-3.4028235 \cdot 10^{38}$ y $3.4028235 \cdot 10^{38}$ (4 bytes).
- `double`: en el lenguaje C, almacenaría un número decimal con muchísima precisión, con un valor máximo de $1,7976931348623157 \cdot 10^{308}$. Sin embargo, en Arduino es lo mismo que float (4 bytes).
- `const`: especifica que la variable definida no podrá ser cambiada durante el programa, siendo siempre un valor constante:

```
const float pi=3.1415;
```

Una variable ya declarada se puede **cambiar de tipo de variable** durante el programa:

```
float pi=3.1415;
int x;
…
x=int(pi);        /*x pasará a valer 3, pues es el resultado de eliminar la parte
                  decimal. Es igualmente válida la siguiente nomenclatura:
                        x=(int)pi;              */
```

En el caso de que vayamos a **declarar varias variables del mismo tipo**, podemos acortar nuestro código separando las variables por coma:

```
int a=6,b=9,c=11;
```

3. ARRAYS

ARRAY: almacena una lista de variables, accediendo a cada una de ellas a través de su índice (el primero siempre es el cero):

```
int datos[] = {4,7,9,12,18};
```

siendo `datos[0]=4; datos[1]=7; … datos[4]=18;`

En este caso no ha sido necesario especificar el tamaño del array porque hemos incluido el valor de todas las variables, pero si no declaro los valores iniciales, debo especificar el tamaño del array:

```
int datos[5];
```

Existe también otra forma de llamar los distintos elementos del array, a través de un **puntero**. El primer elemento corresponde a `*datos`, y para llamar a los siguientes no tenemos más que sumar su posición relativa:

```
*datos      ==    datos[0]
*datos+1    ==    datos[1]
*datos+2    ==    datos[2]
  …         ==       …
*datos+n    ==    datos[n]
```

También es posible almacenar datos ordenados en forma de **matriz** o **array multidimensional** donde cada dato estará localizado por un doble índice (o triple, o cuádruple, etc…). En los arrays multidimensionales sí **es necesario especificar el tamaño de las respectivas dimensiones** (excepto la primera) aunque declares todos sus elementos.

```
int matriz[3][3] = {{2,4,8},{3,9,27},{5,25,125}};
```

siendo `matriz[0][0]=2; matriz[0][1]=4; … matriz[2][2]=125;`

Array de CARACTERES o STRING: almacena una cadena de caracteres, y se emplea para almacenar textos. Existen varias opciones:

```
char letra='a';
```

Almacena un carácter.

```
char texto[]="Me gusta Arduino!";
```

Los 18 caracteres que componen la cadena de texto (17 caracteres + 1 para indicar el fin de la cadena, que aunque no aparezca, está ahí: `'\0'`) son almacenados como elementos separados en el array. Por ejemplo:

```
char texto[0]='M';      char texto[2]=' ';      char texto[16]='!';
```

Nótese que se emplean comillas dobles (`" "`) para cadenas y comillas simples (`' '`) para caracteres individuales.

```
char texto[]={'M','e',' ','g','u','s','t','a',' ',
              'A','r','d','u','i','n','o','!'};
```

De esta otra forma, válida pero más laboriosa, declaro individualmente todos los caracteres que componen el array.

```
char* palabra="Arduino";
char* frase="Me gusta Arduino";
char* color[]={"Rojo","Azul","Verde limón"};
```

Esta otra variante, con el asterisco (`*`), se utiliza para almacenar palabras, frases, o un array de palabras y frases, donde cada elemento del array es un grupo de caracteres:

```
char* color[0]="Rojo";        char* color[2]="Verde limón";
```

La función **sizeof ( )** permite reconocer el tamaño del array, en número de bytes:

```
sizeof(datos);
```

Por ejemplo:

```
int edades[]={36,5,68,15,22,41};
char texto[]= "Me llamo Daniel";
int x=sizeof(edades);     /* x tomará un valor de 12 (cada variable definida como int ocupa 2 bytes */
int y=sizeof(texto);      /* y tomará un valor de 16 (cada caracter ocupa 1 bytes + 1 que indica el final de la cadena */
```

Esta función sólo se utiliza en los casos en los que se emplee el tamaño del array y sea probable cambiar dicho array sin tener que cambiar el resto de código del programa. ***Ejemplo**: imprimir un mensaje letra a letra*:

```
char mensaje[]="Leeme despacito, pisha"; //ocupa 22+1 bytes
int i;

void setup(){
  Serial.begin(9600);
}

void loop() {
  for (i=0; i < sizeof(mensaje)-1; i++){ //imprimiré los 22 caracteres
    Serial.print(i,DEC);
    Serial.print(" = ");
    Serial.println(mensaje[i],BYTE);
    delay(500);
  }
}
```

Si por ejemplo quisiera hacer algo parecido con un array del tipo `int`, debería poner:

```
  for (i=0; i < sizeof(datos)/2; i++){    //cada número ocupa 2 bytes
```

4. DEFINIR FUNCIONES `void función(){… int función(){…`

Según las características de un programa, se puede volver muy útil la utilización de funciones definidas por nosotros mismos. Todo lenguaje de programación debe prestarse a esta posibilidad, y Arduino no es una excepción. Existen varias formas de definir una función:

- Si de la función que queremos definir no se espera ningún valor de retorno, y se limita a realizar una serie de órdenes que dependan (o no) de ciertas variables, entonces la función (que sería una mera **subrutina**) se define de la siguiente manera:

```
void funcion (a,b) {…}
```

donde `a` y `b` serían variables que la función debería utilizar (por supuesto, habrá funciones que no necesiten de ninguna variable). Si dichas variables no estaban declaradas con anterioridad, habrá que declararlas en ese mismo momento:

```
void funcion (int a,int b) {…}
```

- Si la función que vamos a definir quiero que **me devuelva un valor**, debo declararla como si de una variable se tratara, especificando qué tipo de variable es dicho valor (`int`, `long`, `float`,...). Para que la función tome un valor, debemos utilizar la función **`return`** acompañada de la variable cuyo valor quiero asignar a la función.

```
int funcion (a,b) { …      //también puede manejar variables
     int val=0;
     … ;
     val= … ;
     return val;
}
```

Si deseo en algún momento del código de la función interrumpirla y salir de ella (de manera análoga al `break` en las estructuras), y seguir el programa por la siguiente línea después de la llamada de la función, utilizaré el comando `return;` (sin acompañarlo de ninguna variable o dato, con lo cual devuelve un valor vacío).

Proyecto 1. *Emisor de señal de emergencia luminoso: S.O.S. en código Morse*

Emplearemos un simple LED para enviar nuestra señal de socorro en código Morse S.O.S. (Save Our Souls), es decir: • • • (S) — — — (O) • • • (S).
Tendríamos dos opciones a la hora de realizar el programa:
Hacerlo verdaderamente sencillo, con mucho *copiar* y *pegar* líneas como:

```
digitalWrite(ledPin,HIGH); delay(200); digitalWrite(ledPin,LOW); delay(200)
```

para cada punto o raya (la raya sería con un `delay(500)`).
O acortar mucho (y por qué no, embellecer) el código de nuestro programa empleando dos cosas: definiendo una función que hará el pulso de luz (flash) y un array para la duración de dichos pulsos:

```
int ledPin=13;
int tiempo[]={200,200,200,500,500,500,200,200,200};

void setup(){
  pinMode(ledPin,OUTPUT);
}

void loop(){
  for(int i=0;i<9;i++) flash(tiempo[i]);
  delay(800);
}

void flash(int duracion){
  digitalWrite(ledPin,HIGH); delay(duracion);
  digitalWrite(ledPin,LOW); delay(duracion);
```

```
}
```

Observamos que podemos definir la función al final del código. Asimismo, debemos tener cierto cuidado con los arrays, pues en este caso, `tiempo` no es ninguna variable que haya sido definida, sino que las variables son:

`tiempo[0], tiempo[1], … tiempo[8].`

5. FUNCIONES DE TIEMPO `delay(); millis();`

Ya vimos anteriormente la función **delay ( );**

```
delay(1000);          //realiza una pausa en el programa de 1000 ms
```

Veamos ahora otras funciones de tiempo:

delayMicroseconds ( ); millis ( ); micros ( );

```
delayMicroseconds(350);   //realiza una pausa en el programa de 350 µs

x = millis();   /*asigna a x el número de milisegundos que han pasado desde que
                la placa Arduino fue reseteada (es una variable del tipo unsigned
                long). Se produciría un desbordamiento a los 50 días */

x = micros();   /*asigna a x el número de microsegundos que han pasado desde
                que la placa Arduino fue reseteada (es una variable del tipo
                unsigned long). Se produciría un desbordamiento a los 70 minutos */
```

Debemos tener presente **algunas consideraciones** sobre estas funciones de tiempo:

- Puede que sea necesario declarar los argumentos para `delay()` y `delayMicroseconds()` como variables del tipo `long` o `unsigned long` si lo que queremos es emplear pausas largas (del orden del minuto, por ejemplo). Si consideramos `delay(60*1000);` no hará una pausa de 1 minuto, pues al utilizar los números 60 y 1000, Arduino los interpreta como enteros (int) y el resultado sobrepasa el límite de 32767, y por el contrario daría un valor de -5536. Es posible indicar a Arduino que interprete un número como `long`, como `unsigned` o como ambos colocando tras él las letras **`l`** o **`L`**, y/o **`u`** o **`U`**:

  `255u` considera a la constante 255 como una variable del tipo `unsigned int`
  `1000L` considera a la constante 1000 como una variable del tipo `long`
  `32767ul` considera a la constante 32767 como una variable del tipo `unsiged long`

  De esta forma, `delay(60*1000L);` hará una pausa de 1 minuto.

- En el caso de no querer detener el programa, por ejemplo al hacer parpadear un LED (como ocurre con `delay()`), y que pueda realizar otras tareas mientras parpadea, debemos hacer uso de la función `millis()`. Un ejemplo sería:

```
int estado = LOW;
unsigned long tiempo = 0;
unsigned long intervalo = 500;
```

```
void setup(){pinMode(13,OUTPUT);}

void loop(){
  if(tiempo+intervalo < millis()){
    estado = !estado;
    digitalWrite(13,estado);
    tiempo = millis();
  }
}
```

Otra función que se utiliza en las entradas digitales es: **pulseIn(**pin,val**);**
que determina el tiempo transcurrido hasta que se vuelva a repetir el mismo valor de lectura en la entrada digital nº pin. Es una función muy utilizada, por ejemplo, para leer sensores de infrarrojo (IR), de ultrasonido (donde Tigger y Echo sean el mismo pin) o acelerómetros, que producen como salida pulsos de duración variable.

```
x = pulseIn(7,HIGH);
```
/*asigna a `x` el tiempo transcurrido hasta el próximo pulso en estado `HIGH` */

6. FUNCIONES MATEMÁTICAS `x = abs(val);`

Arduino incluye en su lenguaje muchas **funciones matemáticas**. Veámoslas:

```
x = min(a,b);
```
//asigna a `x` el valor más pequeño entre `a` y `b`

```
x = max(a,b);
```
//asigna a `x` el valor más grande entre `a` y `b`

```
x = abs(a);
```
//asigna a `x` el valor absoluto de `a`

```
x = constrain(val,a,b);
```
/*asigna a `x` el valor de `val` siempre y cuando `val` esté comprendido entre `a` y `b`. En caso de `val` salga de dicho intervalo, tomará los valores extremos de este:

```
x = sq(a);
```
//asigna a `x` el valor a^2

```
x = pow(a,b);
```
//asigna a `x` el valor a^b

```
x = sqrt(a);
```
//asigna a `x` el valor raíz cuadrada de `a`

```
x = sin(a);
```
//asigna a `x` el seno(`a`), estando a en radianes

```
x = cos(a);
```
//asigna a `x` el coseno(`a`), estando a en radianes

```
x = tan(a);
```
//asigna a `x` la tangente(`a`), estando a en radianes

7. OPERADORES BINARIOS `x = 1 & 0;`

Arduino posee una serie de funciones con los que **operar dos bits (bit a bit)**, que permiten resolver muchos problemas comunes en programación.

- **&** **AND:** multiplicación booleana de dos bits:

```
x = bit1 & bit2
```

&	0	1
0	0	0
1	0	1

Si por ejemplo aplicamos este operador a dos variables tipo `int`, que recordemos que eran número de 16 bits, los bits se multiplicaran uno a uno, sin tener en cuenta los acarreos. Ejemplo:

```
int a=92;   //en binario es   0000000001011100
int b=101;  //en binario es   0000000001100101
int x=a&b;  //el resultado es 0000000001000100, que corresponde al 68
```

- **|** **OR:** sumador booleano de dos bits:

```
x = bit1 | bit2
```

|	0	1
0	0	0
1	0	1

Si por ejemplo aplicamos este operador a los dos `int` del ejemplo anterior:

```
int a=92;   //en binario es   0000000001011100
int b=101;  //en binario es   0000000001100101
int x=a|b;  //el resultado es 0000000001111111, que corresponde al 125
```

- **^** **XOR:** sumador exclusivo booleano de dos bits:

```
x = bit1 ^ bit2
```

^	0	1
0	0	0
1	0	1

- **~** **NOT:** inversor booleano:

```
x = ~ bit1
```

~	
0	1
1	0

Si por ejemplo aplicamos este operador a una variable tipo `int`:

```
int a=103;  //en binario es   0000000001100111
int x=~a;   //el resultado es 1111111110011000, que corresponde al -104
```

Arduino también posee dos **funciones de desplazamiento de bits**:

- **<<** **Bitshift left:** desplaza los bits que componen una variable hacia la izquierda, rellenando los huecos que se van generando a la derecha con ceros, y perdiéndose todos aquellos bits que han "rebosado" por la izquierda.

  Veamos algún ejemplo :

```
int a=5;    //en binario es   0000000000000101
int b=a<<3; //en binario es   0000000000101000, que corresponde al 40
int c=a<<14; //en binario es  0100000000000000, y se pierde el primer 1
```

  Es muy útil como multiplicador de potencias de 2:

```
1<<0 == 1        1<<1 == 2        1<<2 == 4        1<<3 == 8
1<<4 == 16       1<<5 == 32       1<<6 == 64       1<<7 == 128 ...
```

- **>>** **Bitshift right:** desplaza los bits que componen una variable hacia la derecha, rellenando los huecos que se van generando a la derecha con ceros o unos dependiendo de si el signo del valor decimal de la variable es positivo o negativo, respectivamente, y perdiéndose todos aquellos bits que han "rebosado" por la derecha.

  Veamos algún ejemplo :

```
int a=-16;  //en binario es   1111111111110000
int b=a>>3; //en binario es   1111111111111110, se rellena con 1s
int c=(unsigned int)a>>3; //  0001111111111110, se rellena con 0s
```

  Es muy útil como divisor de potencias de 2:

```
int a=1000;
int x=a>>3; //es como dividir entre 8, siendo x=125
```

8. ESTRUCTURAS `while(){}; do{}while();`

Hasta ahora ya hemos visto tres tipos de estructura para los programas de Arduino:

- **for**

```
for (incio; condición; incremento) {…}
```

- **if**

```
if() {…}
if() {…} else {…}
if() {…} else if() {…} … else if() {…} else {…}
```

- **switch**

```
switch(val){
case 3:…; break; case 12: … ; break; … default:…; }
```

Veamos ahora otras estructuras: while y do…while.

- **while**

`while(condición){…}` /*mientras se cumpla la condición expresada en el paréntesis, Arduino realizará las órdenes incluidas entre las llaves. En caso de no cumplirse, se saltará dicho bloque */

- **do … while**

`do{…}while(condición);` /*básicamente es similar a la estructura `while`, con la diferencia de que al menos una vez hace lo que está entre llaves, y luego lo seguirá haciendo mientras se cumpla la condición especificada en `while` */

Existen dos órdenes que pueden incluirse dentro de cualquier bucle (instrucciones comprendidas entre las llaves) de las estructuras anteriores: **break** y **continue**.

`break;` /*rompe el bucle y el programa continúa por los códigos que aparecen detrás del bloque */

`continue;` /*salta el resto de código del bloque para comenzarlo nuevamente desde el principio. Se utiliza en las estructuras **for, while** y **do** */
Veamos un ejemplo: si tenemos conectados 5 LEDs en los pines del 9 al 13, y quiero que parpadeen en secuencia solamente los que ocupan posición impar, puedo utilizar el siguiente trozo de código:

```
for(int i=9; i<=13; i++) {
      if(i%2==0) continue;    //como i es par, me lo salto
      digitalWrite(i,HIGH); delay(500);
      digitalWrite(i,LOW); delay (500);
}
```

En caso de ser `i` un número par, al dividirlo entre 2 dará un módulo igual a cero, y el comando `continue` hará que Arduino se salte todo el resto del código dentro del bucle for para comenzarlo de nuevo, pero incrementando `i` en 1.

Proyecto 2. Lectura en cm de la distancia de un objeto

Para este proyecto emplearemos un **sensor de ultrasonido** como el de la imagen. En este caso posee 4 patas: dos para alimentación (5 V y GND), una para el disparador de ultrasonido (Trigger) y otro para el receptor del eco del ultrasonido (Echo). También existe otro modelo con solo tres patas: dos de alimentación y una tercera que habrá que declararla como OUPUT cuando hagamos el disparo, y luego declararla como INPUT para recibir el rebote, y medir el tiempo a través de `pulseIn(pinSensor,HIGH)`.

La idea es sencilla: con el disparador enviamos un pulso en `HIGH`, y calculamos el tiempo que tarda dicha señal en llegar rebotada al receptor. Luego hacemos una conversión entre los milisegundos que emplea la señal para llegar al receptor y los centímetros que habrá desde el sensor al objeto que produce el rebote.

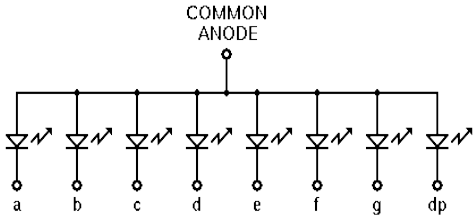
El programa podría ser el siguiente:

```
int disparo=8, eco=7;     //pines para el sensor de distancia
long tiempoInicial,tiempoFinal,duracion;  /*variables que utilizaremos para
                                          obtener el tiempo hasta el rebote */
int cm,senalEco;     //variables para los centímetros y para detectar el rebote

void setup(){
  Serial.begin(9600);
  pinMode(disparo,OUTPUT);
  pinMode(eco,INPUT);
}

void loop(){
  //Hago un disparo: lanzo un pulso de 5 us de duración
  digitalWrite(disparo,LOW);
  delayMicroseconds(2);
  digitalWrite(disparo,HIGH);
  delayMicroseconds(5);
  tiempoInicial=micros();     //pongo el "cronómetro" a cero
  digitalWrite(disparo,LOW);
  //Detectaré el tiempo que tarda en llegar el rebote
  senalEco=digitalRead(eco);
  while(senalEco==LOW){
    senalEco=digitalRead(eco); //para saber cuándo salgo del valor LOW
  }
  while(senalEco==HIGH){
    senalEco=digitalRead(eco);
    tiempoFinal=micros();
  }
  //Calculo la duración del recorrido sensor-obstáculo-sensor
  duracion=tiempoFinal-tiempoInicial;
  cm=int(duracion/58);  /*el sonido se desplaza a 340m/s o 29ms/cm
          y como tiene que recorrer dos veces la distancia hasta el objeto,
          deberemos dividir entre 2*29 los microsegundos transcurridos */
```

```
  Serial.print("tiempo empleado: "); Serial.print(duracion);
  Serial.print(" ms"); Serial.print('\t');
  Serial.print("distancia: "); Serial.print(cm); Serial.print(" cm");
            /*expresaremos la distancia a través del Serial Monitor, pero se
            podría hacer con un display de 2 dígitos */
  delay(1000);    /*una pausa es necesaria para poder leer con comodidad los
                  datos en el Serial Monitor */
}
```

9. DISPLAY 7-SEGMENTOS

Un dispositivo de salida muy empleado es el display 7-segmentos, empleado normalmente para mostrar en él un número del 0 a 9.



Este display posee 10 patillas:

- Los 7 segmentos que forman el número (a,b,c,d,e,f,g).
- El punto (dp).
- Dos pines para el ánodo común.

Debemos conectar nuestro display de la siguiente manera: Conectaremos una de las dos patas de ánodo común a la tensión de 5 V, y cada una de las restantes patas (segmentos y el punto) a un pin de salida digital a través de una resistencia (de 470 Ω, por ejemplo).
De esta manera se encenderán los segmentos conectados a una salida digital de `LOW`.

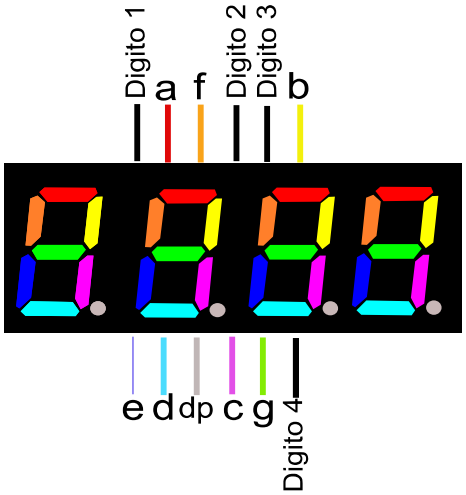
Proyecto 3. La cuenta atrás

```
int pinSegmentos[]={9,8,18,17,16,10,11,19}; //(a,b,c,d,e,f,g,.)
            //utilizaré los pines A2,A3,A4,A5 como salidas digitales
byte segmentosNumero[10][8]={
  {1,1,1,1,1,1,0,0},  //número 0
  {0,1,1,0,0,0,0,0},  //número 1
  {1,1,0,1,1,0,1,0},  //número 2
  {1,1,1,1,0,0,1,0},  //número 3
  {0,1,1,0,0,1,1,0},  //número 4
  {1,0,1,1,0,1,1,0},  //número 5
  {1,0,1,1,1,1,1,0},  //número 6
  {1,1,1,0,0,0,0,0},  //número 7
  {1,1,1,1,1,1,1,0},  //número 8
  {1,1,1,1,0,1,1,0}};  //número 9
```

```
void setup(){
  for(int i=0;i<8;i++) pinMode(pinSegmentos[i],OUTPUT);
}

void loop(){
  for(int i=9;i>=0;i--) {
    for(int j=0;j<8;j++) digitalWrite(pinSegmentos[j],!segmentosNumero[i][j]);
    /*recordemos que a la hora de interpretar una señal digital, 1,HIGH y TRUE
      es lo mismo. Asimismo, lo mismo ocurre con 0,LOW y FALSE.
      Como cada segmento encenderá cuando el pin esté en LOW, tal y como hemos
      definido nuestros número, debemos indicar que en el pin j escriba lo
      contrario de los que indica: si es un 1 pues que ponga un LOW, y si es
      un 0 pues que ponga un HIGH */
    delay(1000);
  }
}
```

10. DISPLAY 4 DÍGITOS 7-SEGMENTOS

Otro dispositivo de salida muy práctico es el display de 4 dígitos 7-segmentos, empleado normalmente para mostrar en él cuatro números del 0 a 9.



Este display posee 10 patillas:

- Los 7 segmentos que forman el número (a,b,c,d,e,f,g).
- El punto (dp).
- Un pin para el ánodo común para cada uno de los cuatro dígitos.

Debemos conectar nuestro display de la siguiente manera:
Conectaremos cada uno de los 4 pines de ánodo común a una salida digital (solo se activará el dígito cuyo pin esté en `HIGH`), y cada una de las restantes patas (segmentos y el punto) a un pin de salida digital a través de una resistencia (de 470 Ω, por ejemplo). De esta manera se encenderán los segmentos conectados una salida digital de `LOW`.

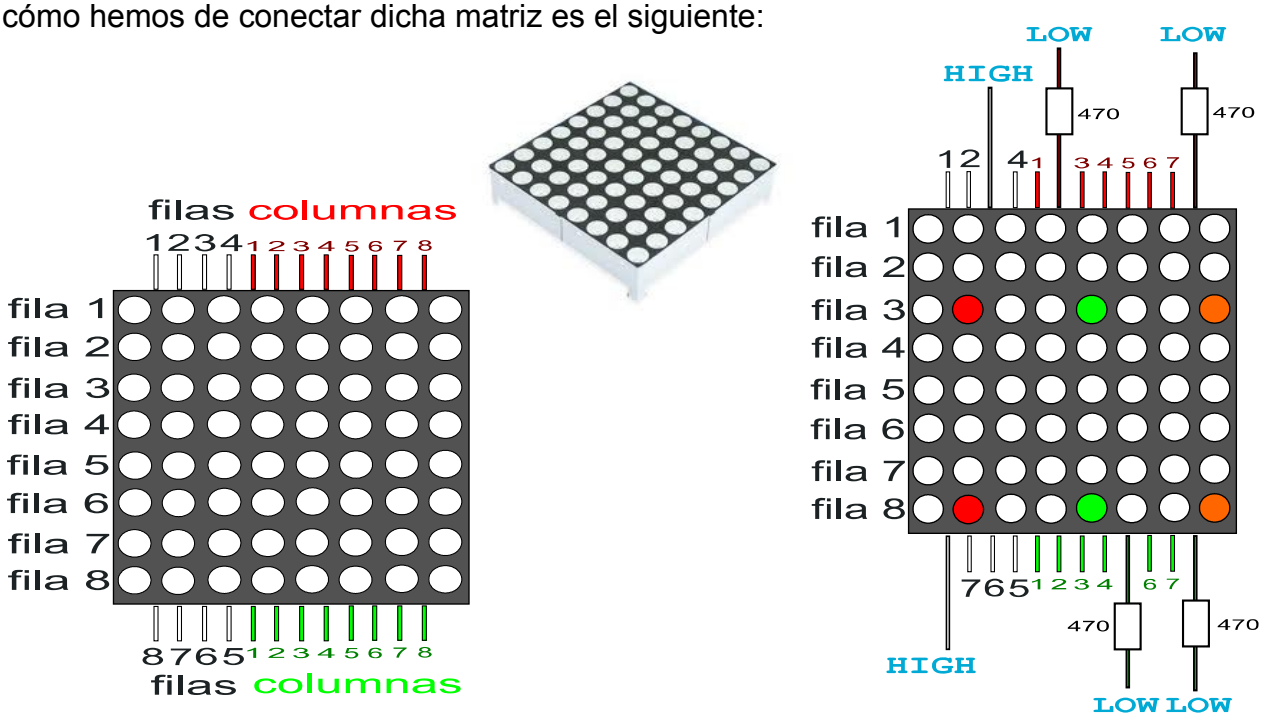
Proyecto 4. ***Reloj con minutos y segundos***

```
int segmentosPines[]={7,5,17,15,14,6,18,16};     //(a,b,c,d,e,f,g,.)
int digitosPines[]={10,9,8,19};           //(dígito1,dígito2,dígito3,dígito4)
```

```
byte segmentosNumeros[10][8]={
  {1,1,1,1,1,1,0,0},  //número 0
  {0,1,1,0,0,0,0,0},  //número 1
  {1,1,0,1,1,0,1,0},  //número 2
  {1,1,1,1,0,0,1,0},  //número 3
  {0,1,1,0,0,1,1,0},  //número 4
  {1,0,1,1,0,1,1,0},  //número 5
  {1,0,1,1,1,1,1,0},  //número 6
  {1,1,1,0,0,0,0,0},  //número 7
  {1,1,1,1,1,1,1,0},  //número 8
  {1,1,1,1,0,1,1,0}  //número 9
};
int unidadesSeg=0,decenasSeg=0,unidadesMin=0,decenasMin=0;
long tiempo=0,intervalo=1000; /*si pusiera intervalo=10 tendría un cronómetro
                              con una precisión de centésimas de segundo*/
void setup(){
  for(int i=0;i<8;i++) pinMode(segmentosPines[i],OUTPUT);
  for(int j=0;j<4;j++) pinMode(digitosPines[j],OUTPUT);
  }

void loop(){
  ponHora(unidadesSeg,decenasSeg,unidadesMin,decenasMin);
  if(tiempo+intervalo<millis()){
    unidadesSeg++;
    if(unidadesSeg==10){unidadesSeg=0; decenasSeg++;}
    if(decenasSeg==6){decenasSeg=0; unidadesMin++;}
    if(unidadesMin==10){unidadesMin=0; decenasMin++;}
    if(decenasMin==6) decenasMin=0;
    tiempo=millis();
  }
}
void ponHora(int x,int y,int z, int t){
  int valores[]={t,z,y,x};
  for(int pin=0;pin<4;pin++) { /*haré todo dentro de un for: iré llamando a
                              cada uno de los dígitos e iré colocando en el
                              display su valor correspondiente*/
    for(int i=0;i<4;i++) digitalWrite(digitosPines[i],LOW);
                  //pongo a LOW todos los pines de dígitos y solo activo uno
    digitalWrite(digitosPines[pin],HIGH);
    for(int j=0;j<8;j++)
      digitalWrite(segmentosPines[j],!segmentosNumeros[valores[pin]][j]);
    int punto=x%2;      //separaré minutos y segundos con un punto parpadeante
    if(punto==0 && pin==1) digitalWrite(segmentosPines[7],LOW);
    delay(5);           /*entre dígito y dígito debemos hacer una pequeña pausa
                        de 3-5 ms sin pausa la lectura es confusa, y con más
                        pausa, molesta la lentitud de refresco*/
  }
}
```

11. MATRIZ DE LEDS 8 x 8

Un último dispositivo de salida muy usado es la matriz de LEDs, en este caso una 8x8 bicolor (64 LEDs rojos y 64 LEDs verdes), con 24 pines de control. Un esquema de cómo hemos de conectar dicha matriz es el siguiente:



La forma de poder representar una imagen formada por estos 64 puntos es hacer barridos por las distintas filas o columnas, de manera análoga a como se trabajaba con el display de 4 dígitos. Para evitar variaciones en la intensidad de brillo de los LEDs, si el barrido lo hacemos por filas deberemos colocar las resistencias limitadoras de corriente (120 – 470 Ω) en los pines reservados a las columnas (como aparece en la figura), pero si el barrido lo hacemos por columnas deberá ser en los pines reservados para las filas donde intercalemos las resistencias.

Proyecto 5. Mensaje dinámico a través de una matriz 8x8

En este caso, pondremos las resistencias en los pines de control de filas. El programa podría ser el siguiente:

```
int numeroPantallas=20;       /*numero de veces que repite una "pantalla" antes de desplazarla un
                               lugar. Evidentemente este parámetro decidirá también la velocidad de
                               avance del texto por la matriz */
int tiempoBarridoColumna=500; /*microseconds en el que está activa cada una de las columnas.
                               Tardará 8*500 us en barrer toda la "pantalla" */
/* AQUÍ ES DONDE DEBEMOS ESCRIBIR LA FRASE QUE QUEREMOS QUE "CIRCULE" POR NUESTRA MATRIZ 8X8
     Se debe empezar por un doble espacio, y se sustituyen la ñ por "`" y la Ñ por "^"  */
char fraseDinamica[]="  Esto parece una FARMACIA DE GUARDIA + + + ";
int index = 1; //será la posición del caracter de la cadena que está siendo mostrado
int offset = 0; //marcará el deslizamiento de las columnas

int filaMatriz[] = {13,12,11,10,19,18,17,16}; //pines para el control de las filas de la matriz
int columnaMatriz[] = {9,8,7,6,5,4,3,2}; //pines para el control de las columnas de LED rojos

//Necesitaremos especificar la posición de cada caracter en letras[][] declarado más abajo
//comenzamos por los números
```

```
const int cero=0,uno=1,dos=2,tres=3,cuatro=4,cinco=5,seis=6,siete=7,ocho=8,nueve=9;
//las letras minúsculas
const int a=10,b=11,c=12,d=13,e=14,f=15,g=16,h=17,i=18,j=19,k=20,l=21,m=22,n=23,
          enhe=24,o=25,p=26,q=27,r=28,s=29,t=30,u=31,v=32,w=33,x=34,y=35,z=36;
//las letras mayúsculas
const int A=37,B=38,C=39,D=40,E=41,F=42,G=43,H=44,I=45,J=46,K=47,L=48,M=49,N=50,
          ENHE=51,O=52,P=53,Q=54,R=55,S=56,T=57,U=58,V=59,W=60,X=61,Y=62,Z=63;
//y algunos simbolos
const int espacio=64,exclamacion=65,comillas=66,almohadilla=67,dollar=68,porcentaje=69,
          ampersand=70,parentesisAbierto=71,parentesisCerrado=72,asterisco=73,mas=74,
          coma=75,menos=76,barra=77,dosPuntos=78,puntoComa=79,menor=80,igual=81,mayor=82,
          interroganteAbierto=83,interroganteCerrado=84,arroba=85,guionBajo=86,punto=87;

byte datos[] = {0,0,0,0,0,0,0,0};  /*este array almacenará los 8 números (de 8 cifras cada uno)
                                     que configuran las 8 columnas mostradas en cada "pantallazo" */

//construiremos los bitmap (8x7) de cada caracter, siendo 1=ON y 0=OFF
//primero los números (el _ será para distinguir a los bitmaps de los "posicionadores"
const int _cero[]= {B01110, B10001, B10011, B10101, B11001, B10001, B01110, B00000};
const int _uno[]= {B00100, B01100, B00100, B00100, B00100, B00100, B01110, B00000};
const int _dos[]= {B01110, B10001, B00001, B00010, B00100, B01000, B11111, B00000};
const int _tres[]= {B11111, B00010, B00100, B00010, B00001, B10001, B01110, B00000};
const int _cuatro[]= {B00010, B00110, B01010, B10010, B11111, B00010, B00010, B00000};
const int _cinco[]= {B11111, B10000, B11110, B00001, B00001, B10001, B01110, B00000};
const int _seis[]= {B00110, B01000, B10000, B11110, B10001, B10001, B01110, B00000};
const int _siete[]= {B11111, B10001, B00010, B00100, B01000, B01000, B01000, B00000};
const int _ocho[]= {B01110, B10001, B10001, B01110, B10001, B10001, B01110, B00000};
const int _nueve[]= {B01110, B10001, B10001, B01111, B00001, B00010, B01100, B00000};
//las letras minúsculas
const int _a[]= {B00000, B00000, B01110, B00001, B01111, B10001, B01111, B00000};
const int _b[]= {B10000, B10000, B10110, B11001, B10001, B10001, B11110, B00000};
const int _c[]= {B00000, B00000, B01110, B10000, B10000, B10001, B01110, B00000};
const int _d[]= {B00001, B00001, B01101, B10011, B10001, B10001, B01111, B00000};
const int _e[]= {B00000, B00000, B01110, B10001, B11111, B10000, B01110, B00000};
const int _f[]= {B00110, B01001, B01000, B11100, B01000, B01000, B01000, B01000};
const int _g[]= {B00000, B00000, B01111, B10001, B01111, B00001, B00001, B01110};
const int _h[]= {B10000, B10000, B10110, B11001, B10001, B10001, B10001, B00000};
const int _i[]= {B00100, B00000, B00100, B01100, B00100, B00100, B01110, B00000};
const int _j[]= {B00010, B00000, B00110, B00010, B00010, B00010, B10010, B01100};
const int _k[]= {B01000, B01000, B01001, B01010, B01100, B01010, B01001, B00000};
const int _l[]= {B01100, B00100, B00100, B00100, B00100, B00100, B01110, B00000};
const int _m[]= {B00000, B00000, B11010, B10101, B10101, B10101, B10101, B00000};
const int _n[]= {B00000, B00000, B10110, B11001, B10001, B10001, B10001, B00000};
const int _enhe[] = {B01110, B00000, B10110, B11001, B10001, B10001, B10001, B00000};
const int _o[]= {B00000, B00000, B01110, B10001, B10001, B10001, B01110, B00000};
const int _p[]= {B00000, B00000, B11110, B10001, B11110, B10000, B10000, B10000};
const int _q[]= {B00000, B00000, B01101, B10011, B01111, B00001, B00011, B00001};
const int _r[]= {B00000, B00000, B10110, B11001, B10000, B10000, B10000, B00000};
const int _s[]= {B00000, B00000, B01110, B10000, B01110, B00001, B11110, B00000};
const int _t[]= {B01000, B01000, B11100, B01000, B01000, B01001, B00110, B00000};
const int _u[]= {B00000, B00000, B10001, B10001, B10001, B10011, B01101, B00000};
const int _v[]= {B00000, B00000, B10001, B10001, B10001, B01010, B00100, B00000};
const int _w[]= {B00000, B00000, B10001, B10001, B10101, B10101, B01010, B00000};
const int _x[]= {B00000, B00000, B10001, B01010, B00100, B01010, B10001, B00000};
const int _y[]= {B00000, B00000, B10001, B10001, B01111, B00001, B00001, B01110};
const int _z[]= {B00000, B00000, B11111, B00010, B00100, B01000, B11111, B00000};
//las letras mayúsculas
const int _A[] = {B01110, B10001, B10001, B10001, B11111, B10001, B10001, B00000};
const int _B[] = {B11110, B10001, B10001, B11110, B10001, B10001, B11110, B00000};
const int _C[] = {B01110, B10001, B10000, B10000, B10000, B10001, B01110, B00000};
const int _D[] = {B11110, B10001, B10001, B10001, B10001, B10001, B11110, B00000};
const int _E[] = {B11111, B10000, B10000, B11110, B10000, B10000, B11111, B00000};
const int _F[] = {B11111, B10000, B10000, B11110, B10000, B10000, B10000, B00000};
const int _G[] = {B01110, B10001, B10000, B10011, B10001, B10001, B01111, B00000};
const int _H[] = {B10001, B10001, B10001, B11111, B10001, B10001, B10001, B00000};
const int _I[] = {B01110, B00100, B00100, B00100, B00100, B00100, B01110, B00000};
const int _J[] = {B00111, B00010, B00010, B00010, B00010, B10010, B01100, B00000};
const int _K[] = {B10001, B10010, B10100, B11000, B10100, B10010, B10001, B00000};
const int _L[] = {B10000, B10000, B10000, B10000, B10000, B10000, B11111, B00000};
const int _M[] = {B10001, B11011, B10101, B10101, B10001, B10001, B10001, B00000};
const int _N[] = {B10001, B10001, B11001, B10101, B10011, B10001, B10001, B00000};
const int _ENHE[] = {B01110, B00000, B10001, B11001, B10101, B10011, B10001, B00000};
const int _O[] = {B01110, B10001, B10001, B10001, B10001, B10001, B01110, B00000};
const int _P[] = {B11110, B10001, B10001, B11110, B10000, B10000, B10000, B00000};
const int _Q[] = {B01110, B10001, B10001, B10001, B10101, B10010, B01101, B00000};
const int _R[] = {B11110, B10001, B10001, B11110, B10100, B10010, B10001, B00000};
const int _S[] = {B01110, B10001, B10000, B01110, B00001, B10001, B01110, B00000};
```

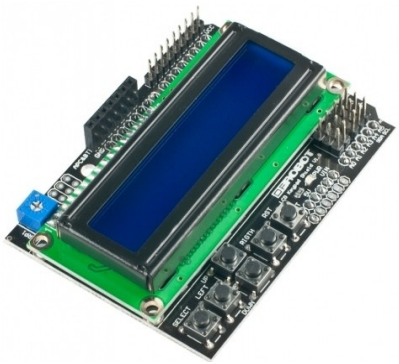
```
const int _T[] = {B11111, B00100, B00100, B00100, B00100, B00100, B00100, B00000};
const int _U[] = {B10001, B10001, B10001, B10001, B10001, B10001, B01110, B00000};
const int _V[] = {B10001, B10001, B10001, B10001, B10001, B01010, B00100, B00000};
const int _W[] = {B10001, B10001, B10001, B10101, B10101, B10101, B01010, B00000};
const int _X[] = {B10001, B10001, B01010, B00100, B01010, B10001, B10001, B00000};
const int _Y[] = {B10001, B10001, B10001, B01010, B00100, B00100, B00100, B00000};
const int _Z[] = {B11111, B00001, B00010, B00100, B01000, B10000, B11111, B00000};
//caracteres especiales
const int _espacio[] = {B00000, B00000, B00000, B00000, B00000, B00000, B00000, B00000};
const int _exclamacion[] = {B00100, B00100, B00100, B00100, B00100, B00000, B00100, B00000};
const int _comillas[] = {B01010, B01010, B01010, B00000, B00000, B00000, B00000, B00000};
const int _almohadilla[] = {B01010, B01010, B11111, B01010, B11111, B01010, B01010, B00000};
const int _dollar[] = {B00100, B01111, B10100, B01110, B00101, B11110, B00100, B00000};
const int _porcentaje[] = {B11000, B11001, B00010, B00100, B01000, B10011, B00011, B00000};
const int _ampersand[] = {B01100, B10010, B10100, B01000, B10101, B10010, B01101, B00000};
const int _parentesisAbierto[] = {B00010, B00100, B01000, B01000, B01000, B00100, B00010, B00000};
const int _parentesisCerrado[] = {B01000, B00100, B00010, B00010, B00010, B00100, B01000, B00000};
const int _asterisco[] = {B00000, B00000, B00100, B10101, B01110, B10101, B00100, B00000};
const int _mas[] = {B00000, B00000, B00100, B00100, B11111, B00100, B00100, B00000};
const int _coma[] = {B00000, B00000, B00000, B00000, B00000, B01100, B00100, B01000};
const int _menos[] = {B00000, B00000, B00000, B00000, B11111, B00000, B00000, B00000};
const int _barra[] = {B00000, B00000, B00001, B00010, B00100, B01000, B10000, B00000};
const int _dosPuntos[] = {B00000, B01100, B01100, B00000, B01100, B01100, B00000, B00000};
const int _puntoComa[] = {B00000, B00000, B01100, B01100, B00000, B01100, B00100, B01000};
const int _menor[] = {B00010, B00100, B01000, B10000, B01000, B00100, B00010, B00000};
const int _igual[] = {B00000, B00000, B11111, B00000, B11111, B00000, B00000, B00000};
const int _mayor[] = {B01000, B00100, B00010, B00001, B00010, B00100, B01000, B00000};
const int _interroganteAbierto[]={B00100, B00000, B00100, B01000, B10000, B10001, B01110, B00000};
const int _interroganteCerrado[]={B01110, B10001, B00001, B00010, B00100, B00000, B00100, B00000};
const int _arroba[] = {B01110, B10001, B00001, B01101, B10101, B10101, B01110, B00000};
const int _guionBajo[] = {B00000, B00000, B00000, B00000, B00000, B00000, B11111, B00000};
const int _punto[] = {B00000, B00000, B00000, B00000, B00000, B01100, B01100, B00000};

/*declaro el abecedario: un array con los arrays de los bitmap de los distintos carateres que voy
a mostrar en la matriz. Como los elementos del array con cadenas de caracteres utilizo int*   */
const int* letras[]={ _cero,_uno,_dos,_tres,_cuatro,_cinco,_seis,_siete,_ocho,_nueve, //del 0 al 9
          _a,_b,_c,_d,_e,_f,_g,_h,_i,_j,                                  //del 10 al 19
          _k,_l,_m,_n,_enhe,_o,_p,_q,_r,_s,                               //del 20 al 29
          _t,_u,_v,_w,_x,_y,_z,_A,_B,_C,                                  //del 30 al 39
          _D,_E,_F,_G,_H,_I,_J,_K,_L,_M,                                  //del 40 al 49
          _N,_ENHE,_O,_P,_Q,_R,_S,_T,_U,_V,                               //del 50 al 59
         _W,_X,_Y,_Z,_espacio,_exclamacion,_comillas,_almohadilla,_dollar,_porcentaje,// 60 al 69
         _ampersand,_parentesisAbierto,_parentesisCerrado,_asterisco,_mas,
         _coma,_menos,_barra,_dosPuntos,_puntoComa,                                  //del 70 al 79
         _menor,_igual,_mayor,_interroganteAbierto,_interroganteCerrado,
          _arroba,_guionBajo,_punto };                                               //del 80 al 86

void setup() {
  for(int i=0; i<8; i++){            //configuro los pines de control
    pinMode(filaMatriz[i], OUTPUT);
    pinMode(columnaMatriz[i], OUTPUT);
  }
}

void loop() {
  actualizaMatriz();
}

void actualizaMatriz(){
  construyePantalla();
  muestraPantalla(numeroPantallas);
}

//Utilizaré el siguiente array con potencias de 2 para manejar los bits de cada bitmap
const int potencias[] = {1,2,4,8,16,32,64,128};
         //cada número corresponde, en binario, a 00000001, 00000010, 00000100,..., 10000000

void construyePantalla(){     //carga el actual estado de desplazamiento al array datos[]
  int caracterAnterior = cogeCaracter(fraseDinamica[index-1]); //por esto empezamos en index=1
  int caracterActual = cogeCaracter(fraseDinamica[index]);
  int caracterPosterior = cogeCaracter(fraseDinamica[index+1]);
  for(int fila=0; fila<8; fila++){ //hago un barrido por la fila
    datos[fila]=0;                 //reseteo a 0 la fila que estoy considerando
    for(int columna=0; columna<8; columna++){ //hago un barrido por la columna
    /*construiré un número decimal a base de sumas de tal forma que tenga unos en los lugares
    donde quiero que se encienda el LED */
      datos[fila] = datos[fila] +
```

```
        (potencias[columna] & (letras[caracterAnterior][fila] << (offset+7) ));
    /*cuando muestro un caracter en el borde derecho de la matriz (offset==0), se deberían mostrar
     en las dos columnas del borde izquierdo las dos últimas columnas del caracter anterior */
      datos[fila] = datos[fila] +
        (potencias[columna] & (letras[caracterActual][fila] << offset ));
    //muestro el caracter actual, y lo voy desplazando a través del offset hacia la izquierda
      datos[fila] = datos[fila] +
        (potencias[columna] & (letras[caracterPosterior][fila] >> (7-offset) ));
    /*cuando voy desplazando el caracter actual, irá apareciendo por la derecha las primeras
    columnas del caracter posterior. Se ha puesto (offset+7) y (7-offset) porque los caracteres
    tienen una anchura de 5 columnas, y quiero que entre caracter y caracter haya 2 columnas
    de separación */
     }
   }
   offset++;            //he terminado una fila, pasaría a construir la siguiente
   if(offset==6){       //significaría que he desplazado una letra entera
     offset=0; index++;        //cargo un caracter nuevo y pongo offset a cero
     if(index==sizeof(fraseDinamica)-2) index=1; //si termina la frase, la reinicio
   }
}

void muestraPantalla(int repeticiones){
  for(int n=0; n<repeticiones; n++){ //muestra la pantalla acutal un número de repeticones
    for(int columna=0; columna<8; columna++){ //haré el barrido de la matriz columna por columna
      for(int i=0; i<8; i++) digitalWrite(filaMatriz[i], LOW); /*pone a LOW todos los pines de
                                                                 control de filas */
      for(int i=0; i<8; i++){ //solo pondré en LOW el pin de la columna que estamos considerando
        if(i==columna) digitalWrite(columnaMatriz[i], LOW);
        else digitalWrite(columnaMatriz[i], HIGH); /*el resto de pines los pongo a HIGH
                                                     (apago la columna de LEDs) */
      }
      for(int fila=0; fila<8; fila++){ //haré el barrido, dentro de la columna activa, por la fila
        int Bit=(datos[columna] >> fila) & 1;       /*detectaré los "1" de las fila si
                                                    al multiplicarlo por 1 da 1 */
        if(Bit==1) digitalWrite(filaMatriz[fila], HIGH); /*si el bit en el array datos[] es un 1,
                                                           encenderá el LED  */
      }
      delayMicroseconds(tiempoBarridoColumna);      //es el tiempo que cada columna está encendida
    }
  }
}
       // veamos como asociamos cada caracter de la cadena con su bitmap
int cogeCaracter(char caracter){
  int returnValue;
  switch(caracter){
    case '0': returnValue = cero; break;      case '1': returnValue = uno; break;
    case '2': returnValue = dos; break;       case '3': returnValue = tres; break;
    case '4': returnValue = cuatro; break;    case '5': returnValue = cinco; break;
    case '6': returnValue = seis; break;      case '7': returnValue = siete; break;
    case '8': returnValue = ocho; break;      case '9': returnValue = nueve; break;
    case 'A': returnValue = A; break;         case 'a': returnValue = a; break;
    case 'B': returnValue = B; break;         case 'b': returnValue = b; break;
    case 'C': returnValue = C; break;         case 'c': returnValue = c; break;
    case 'D': returnValue = D; break;         case 'd': returnValue = d; break;
    case 'E': returnValue = E; break;         case 'e': returnValue = e; break;
    case 'F': returnValue = F; break;         case 'f': returnValue = f; break;
    case 'G': returnValue = G; break;         case 'g': returnValue = g; break;
    case 'H': returnValue = H; break;         case 'h': returnValue = h; break;
    case 'I': returnValue = I; break;         case 'i': returnValue = i; break;
    case 'J': returnValue = J; break;         case 'j': returnValue = j; break;
    case 'K': returnValue = K; break;         case 'k': returnValue = k; break;
    case 'L': returnValue = L; break;         case 'l': returnValue = l; break;
    case 'M': returnValue = M; break;         case 'm': returnValue = m; break;
    case 'N': returnValue = N; break;         case 'n': returnValue = n; break;
    case '^': returnValue = ENHE; break;      case '`': returnValue = enhe; break;
    case 'O': returnValue = O; break;         case 'o': returnValue = o; break;
    case 'P': returnValue = P; break;         case 'p': returnValue = p; break;
    case 'Q': returnValue = Q; break;         case 'q': returnValue = q; break;
    case 'R': returnValue = R; break;         case 'r': returnValue = r; break;
    case 'S': returnValue = S; break;         case 's': returnValue = s; break;
    case 'T': returnValue = T; break;         case 't': returnValue = t; break;
    case 'U': returnValue = U; break;         case 'u': returnValue = u; break;
    case 'V': returnValue = V; break;         case 'v': returnValue = v; break;
    case 'W': returnValue = W; break;         case 'w': returnValue = w; break;
    case 'X': returnValue = X; break;         case 'x': returnValue = x; break;
    case 'Y': returnValue = Y; break;         case 'y': returnValue = y; break;
    case 'Z': returnValue = Z; break;         case 'z': returnValue = z; break;
```

```
    case ' ': returnValue = espacio; break;    case '!': returnValue = exclamacion; break;
    case '"': returnValue = comillas; break;   case '#': returnValue = almohadilla; break;
    case '$': returnValue = dollar; break;     case '%': returnValue = porcentaje; break;
    case '&': returnValue = ampersand; break;  case '(': returnValue = parentesisAbierto; break;
    case ')': returnValue = parentesisCerrado; break;    case '*': returnValue = asterisco; break;
    case '+': returnValue = mas; break;        case ',': returnValue = coma; break;
    case '-': returnValue = menos; break;      case '/': returnValue = barra; break;
    case ':': returnValue = dosPuntos; break;  case ';': returnValue = puntoComa; break;
    case '<': returnValue = menor; break;      case '=': returnValue = igual; break;
    case '>': returnValue = mayor; break;      case '¿': returnValue = interroganteAbierto; break;
    case '?': returnValue = interroganteCerrado; break;  case '@': returnValue = arroba; break;
    case '_': returnValue = guionBajo; break;  case '.': returnValue = punto; break;
  }
  return returnValue;
}
```

12. PANTALLA LCD 16x2 `#include <LiquidCrystal.h>`

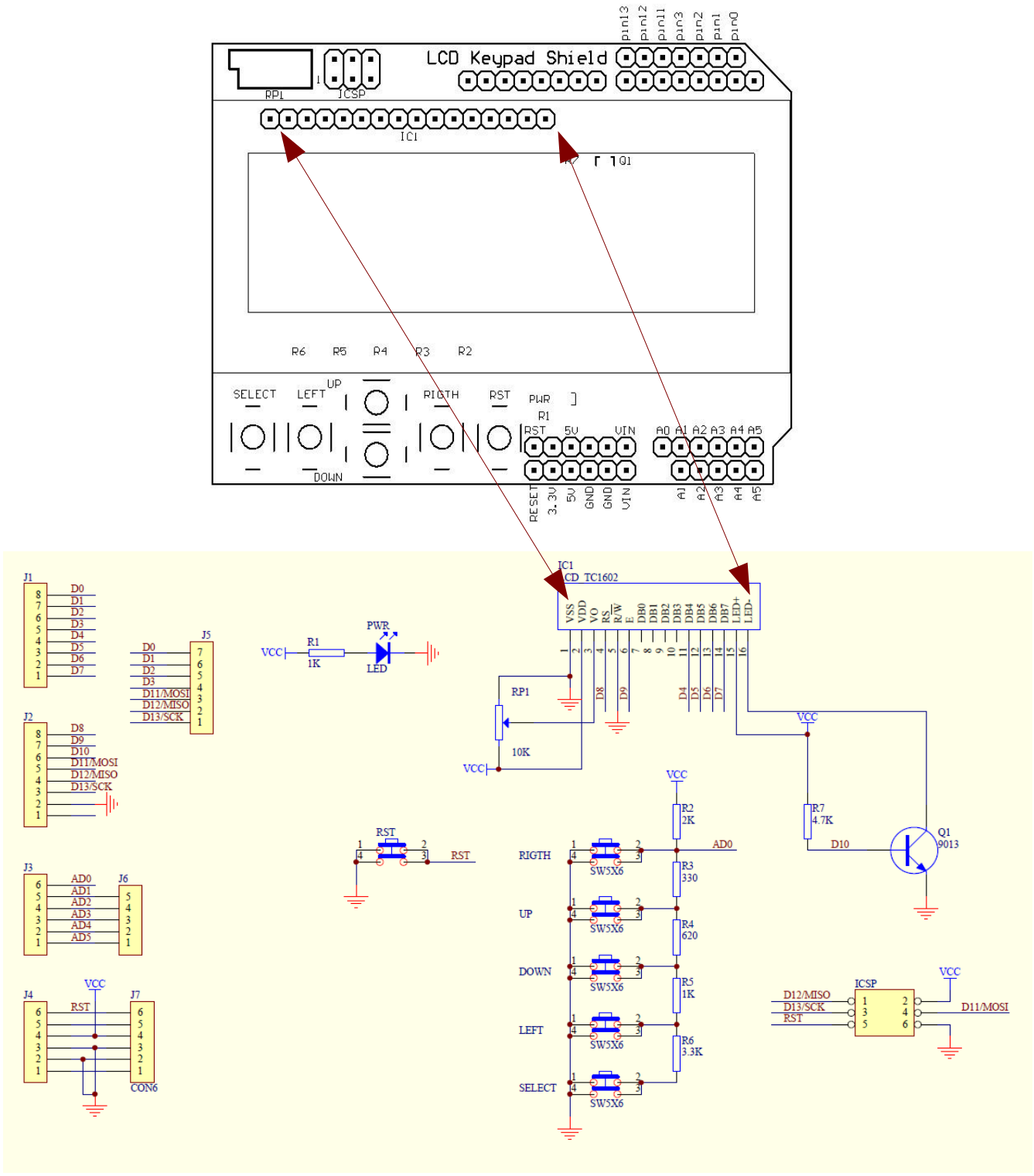
Uno de los dispositivos más espectaculares para nuestra Arduino es sin duda la pantalla LCD. En el mercado podemos encontrar gran variedad, pero el modelo HD44780 es el más extendido, incluso se encuentran montadas sobre un shield para Arduino UNO:



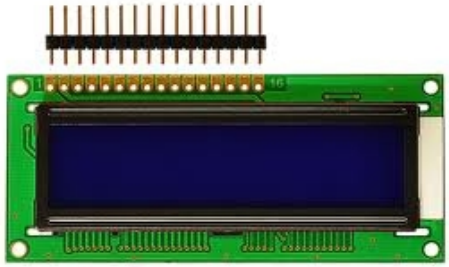
PIN CONNECTIONS			
PIN	Symbol	Level	Function
1	VSS	—	GND(0V)
2	VDD	—	Supply Voltage for Logic(+5V)
3	V0	—	Power supply for LCD
4	RS	H/L	H: Data; L: Instruction Code
5	R/W	H/L	H: Read; L: Write
6	E	H/L	Enable Signal
7	DB0	H/L	Data Bus Line
8	DB1	H/L	
9	DB2	H/L	
10	DB3	H/L	
11	DB4	H/L	
12	DB5	H/L	
13	DB6	H/L	
14	DB7	H/L	
15	LEDA	H/L	Backlight Power(+5V)
16	LEDK	—	Backlight Power(0V)

¿Qué pines de nuestra Arduino se utilizan para controlar la pantalla LCD?

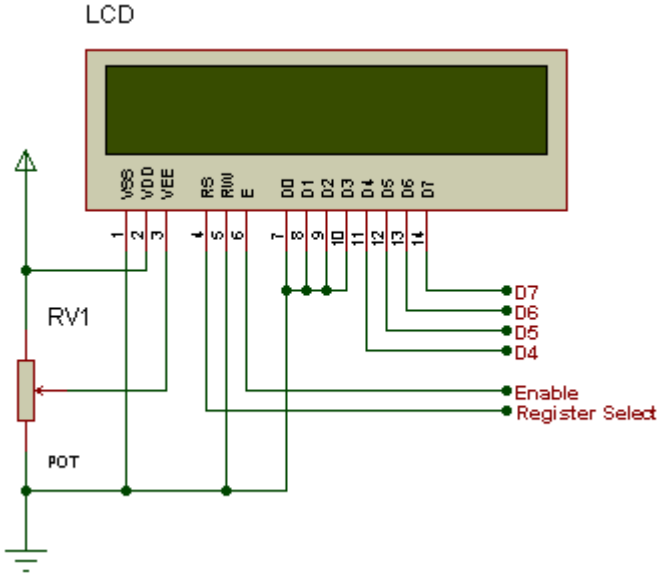
Pin Arduino	Pin LCD		FUNCIÓN
A0		Botones (select, up, right, down and left)	Definida por el programador
4	11	DB4	Bus de 4 bits
5	12	DB5	
6	13	DB6	
7	14	DB7	
8	4	RS (register select)	Manda datos e instrucciones
9	6	E (enable)	Habilita la conexión
10	15	Backlight	Controla la luminosidad
	3	Potenciómetro	Controla el contraste



Una opción más barata es adquirir la LCD y soldarle una hilera de 16 pines:



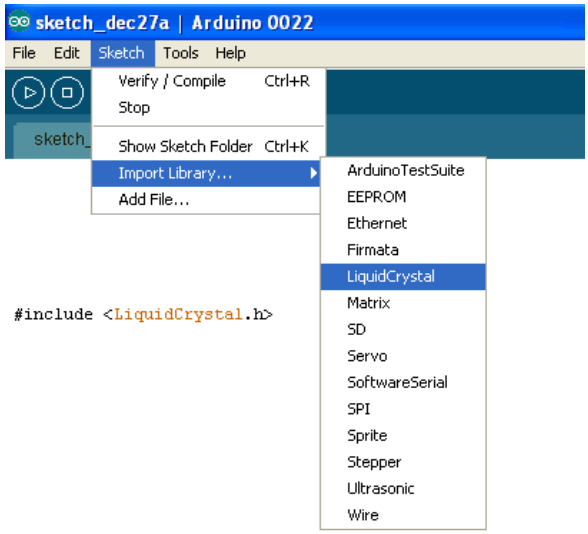
En este caso, los pines corresponden a:



Pin LCD		
1	VSS	0 V
2	VDD	5 V
3	V0	Potenciómetro (de 0 V a 5 V) para controlar el contraste de la pantalla, o una resistencia de 2,2K a 0 V.
4	RS	Register select
5	R/W	0 V (solo vamos a escribir, no a leer datos)
6	E	enable
7	DB0	(sólo para transmitir datos a 8 bits, cosa que no será necesaria nunca)
8	DB1	
9	DB2	
10	DB3	
11	DB4	BUS de transmisión de datos a 4 bits
12	DB5	
13	DB6	
14	DB7	
15	LED+	5 V con una resistencia de 220 Ω (para LCD con retroiluminación)
16	LED-	0 V (para LCD con retroiluminación)

Veamos cómo controlar nuestro dispositivo LCD a través de Arduino:

- En primer lugar, debemos abrir la librería `LiquidCrystal.h`, a través de:
  Sketch / Import Library / LiquidCrystal



- Luego debemos declarar, como si de variables fuera, nuestra pantalla LCD y debemos asignarle un nombre. Además debemos especificar los pines que vamos a usar para comunicar Arduino con la LCD
  - En el caso de la Shield LDR, tal y como está construida, son los pines: 8, 9, 4, 5, 6, 7; que corresponderán a: RS (register select), E (enable), DB4, DB5, DB6, DB7 (bus de datos).
  - En el caso de una LDR simple, podemos escoger los pines que queramos de nuestra Arduino.

  Esta declaración se hace con el siguiente comando:

```
LiquidCrystal nombreLcd(rs,e,db4,db5,db6,db7);
```

  Con la configuración de la shield LCD quedaría así:

```
LiquidCrystal miLcd(8,9,4,5,6,7);
```

  Con la configuración de la pantalla LCD simple, podría quedar así:

```
LiquidCrystal miLcd(5,6,7,8,9,10);
```

- La patilla LED+ de nuestra LCD controla la luminosidad de la pantalla, y en nuestro dispositivo LCD está conectada a la salida analógica pin 10, y se controla la luminosidad a través de un `analogWrite(10,brillo);`

- Dentro del void setup() debemos iniciar la librería para nuestra LCD, utilizando la función `begin()` y especificando el tamaño, que en nuestro caso es de 16x2 caracteres:

```
nombreLcd.begin(16,2);
```

Las funciones para controlar nuestra LCD son:

- `nombreLcd.setCursor(0,7);`

Sitúa el cursor de nuestra pantalla de 16x2 en la primera línea, en el sexto carácter.

- `nombreLcd.print("Hola!");`

Imprime en la posición fijada con anterioridad lo que esté entre comillas, y sitúa el cursor tras el último caracter. Actúa de manera análoga al `print` en **`Serial`**`.print();` Por ejemplo: `lcd.print(val,DEC);` // imprime el valor `val` en base 10

- `nombreLcd.clear();`

Borra todo lo que hubiera en la pantalla hasta ese momento y sitúa el cursor en la posición (0,0).

- `nombreLcd.write(symbol[i]);`

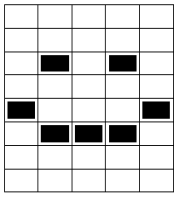
Se emplea para mandar un caracter a la LCD. Puedo mandar el caracter '+' (cuyo código ASCII es el 43) de varias formas:

```
byte suma=B00101011;        //la B es para especificar que está en binario
nombeLcd.write(suma);           //lo puede mandar como variable
nombreLcd.write(B00101011);     //lo puede mandar en código binario
nombreLcd.write(43);            //lo puede mandar en base decimal
nombreLcd.write(0x2B);          //lo puede mandar en base hexadecimal
```

- `nombreLcd.createChar(número,dato);`

Esta función permite imprimir un símbolo creado por nosotros mismos, puesto que cada caracter posee una matriz de 5x8 píxeles. Para ello debemos generar dicho símbolo a través de 8 números binarios de 5 bits, correspondientes a cada una de las filas que componen dicho símbolo. Nuestro display LCD puede almacenar 8 símbolos diseñados por nosotros, y debemos indicar dónde lo almacena a través de dicho número (de 0 a 7). Se suele incluir dentro del `void setup`, y luego, para imprimir dicho símbolo en la pantalla utilizamos la función `nombreLcd.write(número)`.
Veamos un ejemplo: imprimiremos una carita sonriendo:

```
byte sonrisa[]={B00000, B00000, B01010, B00000, B10001, B01110, B00000,
                B00000};
void setup(){
nombreLcd.createChar(0,sonrisa);
}
void loop(){
nombreLcd.write(0);
}
```



- `nombreLcd.scrollDisplayRight();`
  `nombreLcd.scrollDisplayLeft();`

Esta función permite desplazar lo que esté en la pantalla una posición hacia la derecha o a la izquierda, dependiendo de qué función estemos utilizando.

Proyecto 6. El pez nadador en su pecera LCD

Consistirá en mostrar en nuestra pantalla LCD un pez que de desplaza por la pantalla, y va haciendo pequeñas pompitas. El programa puede ser el siguiente:

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(8,9,4,5,6,7);     //configurado para la shield LCD

byte pescadoIzquierda[]={B00000,B00000,B00000,B00000,B01101,B11110,B01101,
                         B00000};
byte pescadoDerecha[]={B00000,B00000,B00000,B00000,B10110,B01111,B10110,
                         B00000};
byte pescadoCentro[]={B00000,B00000,B00000,B00000,B00100,B01110,B00100,
                         B00000};
byte burbuja1[]={B00010,B00000,B00100,B00010,B00100,B01110,B00100,B00000};
byte burbuja2[]={B00000,B00100,B00010,B00000,B00100,B01110,B00100,B00000};
byte burbuja3[]={B00100,B00000,B00000,B00000,B00100,B01110,B00100,B00000};
byte x=0;                //x e y son números muy pequeños,
byte y=0;                //así que los defino como byte para optimizar memoria
int tiempo=600;

void setup(){
  lcd.begin(16,2);
  lcd.createChar(0,burbuja1);       //creo los caracteres de los peces
  lcd.createChar(1,burbuja2);
  lcd.createChar(2,burbuja3);
  lcd.createChar(3,pescadoIzquierda);
  lcd.createChar(4,pescadoDerecha);
  lcd.createChar(5,pescadoCentro);
}

void loop(){
  desplazarDerecha(9);  //el pez nadará hacia la derecha 10 posiciones
  pararCentro();        //se parará mirando al frente
  pompas();             //respirará echando pompas
  y=1;                  //bajará a la fila de abajo
  desplazarIzquierda(5);//ahora nadará hacia la izquierda 6 posiciones
  pararCentro();        //se parará otra vez
  pompas();             //hará pompas otra vez
  y=0;                  //subirá a la fila de arriba
  desplazarDerecha(11); //nadará hacia la derecha 12 posiciones y se perderá
  delay(tiempo*10);     //tras un tiempo considerable...
  x=0;
  y=0;                  //...aparecerá de nuevo en la posición 0,0
}

void desplazarDerecha(int posiciones) {
  lcd.setCursor(x,y);
  lcd.write(4);
  delay(tiempo);
  for(int i=0;i<posiciones;i++) {
    lcd.scrollDisplayRight();
    delay(tiempo);
    x++;
```

```
  }
  lcd.clear();
}

void desplazarIzquierda(int posiciones) {
  lcd.setCursor(x,y);
  lcd.write(3);
  delay(tiempo);
  for(int i=0;i<posiciones;i++) {
    lcd.scrollDisplayLeft();
    delay(tiempo);
    x--;
  }
  lcd.clear();
}

void pararCentro(){
  lcd.setCursor(x,y);
  lcd.write(5);
  delay(tiempo);
  lcd.clear();
}

void pompas(){
  for(int i=0;i<3;i++) {
    lcd.setCursor(x,y);
    lcd.write(i);
    delay(tiempo);
  }
  lcd.clear();
}
```

Proyecto 7. Botones de la Shield LCD

Este otro programa permitirá testear los botones que incorpora la shield LCD

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(8, 9, 4, 5, 6, 7);     //configurado para la shield LCD

// definimos algunas variables que usaremos en este programa:
int botonApretado = 0;
int lecturaAnalogica  = 0;
#define botonRIGHT  0
#define botonUP     1
#define botonDOWN   2
#define botonLEFT   3
#define botonSELECT 4
#define botonNONE   5

// veamos cómo leer los botones
int leerBotonesLCD(){
  lecturaAnalogica = analogRead(0);  //lee el valor de la entrada analógica A0
  /* los botones están configurados de tal forma que el valor de entrada en A0
  valen, respectivamente: 0, 144, 329, 504, 741, 1023 */
  if (lecturaAnalogica > 1000) return botonNONE; /* el valor más habitual
  será el de no estar accionando ningún botón, por eso lo pongo en primer
  lugar, para que el programa gane en velocidad (aunque no creo que se note)*/
```

```
 if (lecturaAnalogica < 50)   return botonRIGHT;
 if (lecturaAnalogica < 195)  return botonUP;
 if (lecturaAnalogica < 380)  return botonDOWN;
 if (lecturaAnalogica < 555)  return botonLEFT;
 if (lecturaAnalogica < 790)  return botonSELECT;
 return botonNONE;  // si no se cumple ninguna, que devuelva botonNONE
}

void setup(){
 lcd.begin(16, 2);              // iniciamos la librería
 lcd.setCursor(0,0);
 lcd.print("Aprieta los botones"); // imprime este mensaje
}

void loop(){
 lcd.setCursor(9,1); //sitúa al cursor en la 10ª columna, 2ª fila
 lcd.print(millis()/1000);     // muestra los segundos transcurridos
 lcd.setCursor(0,1);           // cursor al principio de la 2ª fila
 botonApretado = leerBotonesLCD(); // hace lectura de los botones
 switch (botonApretado) {      /* dependiendo de qué botón pulsemos, hará
                                  una acción u otra */
   case botonRIGHT: {
     lcd.print("DERECHA");
     break;
     }
   case btnLEFT: {
     lcd.print("IZQUIERDA");
     break;
     }
   case botonUP: {
     lcd.print("ARRIBA");
     break;
     }
   case botonDOWN: {
     lcd.print("ABAJO");
     break;
     }
   case botonSELECT: {
     lcd.print("SELECT");
     break;
     }
   case botonNONE: {
     lcd.print("NINGUNO");
     break;
     }
 }
}
```

Bibliografía:

- *Curso de Arduino de Francis Perea.*
- *Getting Started With Arduino (Massimo Banzi).*
- *Beginning Arduino Programming (Brian Evans).*
- *30 Arduino Projects for he Evil Genios (Simon Monk).*