CREACIÓN DE LIBRERÍAS

Procedimiento a seguir para crear librerías.

Ejemplo para crear librería Morse.h:

Ejemplo ‘morse.ino’:

```
1  int pin = 13;
2
3  void setup()
4  {
5    pinMode(pin, OUTPUT);
6  }
7
8  void loop()
9  {
10   dot(); dot(); dot();
11   dash(); dash(); dash();
12   dot(); dot(); dot();
13   delay(3000);
14 }
15
16 void dot()
17 {
18   digitalWrite(pin, HIGH);
19   delay(250);
20   digitalWrite(pin, LOW);
21   delay(250);
22 }
23
24 void dash()
25 {
26   digitalWrite(pin, HIGH);
27   delay(1000);
28   digitalWrite(pin, LOW);
29   delay(250);
30 }
```

Para convertir en una librería de código morse, vemos que hay dos funciones dot() y dash() para iluminar un led durante 250 ms y 1 segundo y una variable que es ledPin que determina que pin usar. Este es un estilo de programación clásico usando funciones en lugar de objetos.

Para una librería se necesitan al menos dos ficheros:

- Un fichero de cabecera con la extensión .h. Este fichero tiene las definiciones de la librería, básicamente un listado de todo lo que hay dentro de la librería
- Un fichero fuente con la extensión .cpp. Este fichero el que contiene el código

En este caso la librería se va a llamar morse y generamos un fichero de cabecera llamado morse.h.

Veamos el código de morse.h donde se define la clase Morse donde tiene una línea por cada función o método y también una línea por cada variable o propiedad de la clase.

```
class Morse
{
  public:
    Morse(int pin); //constructor
    void dot();
    void dash();
  private:
    int _pin;
};
```

Una clase es una colección de funciones (métodos) y variables (propiedades) que se guardan todas juntas en un solo lugar. Las funciones pueden ser públicas (public), es decir, pueden llamarse por quien usa la librería o privadas (private), es decir, que solo pueden llamarse desde dentro de la propia clase. Todas las clases tienen una función llamada constructor, que es usada para crear una instancia de la clase. El constructor tiene el mismo nombre que la clase y no tiene tipo de variable de devolución.

En el fichero de cabecera de una librería es necesario la declaración #include que de acceso a los tipos y constantes estándar del lenguaje de Arduino (esto se añade automáticamente en los sketches pero no a las librerías). Esta declaración debe ponerse antes de la definición de la clase. La declaración debe ser:

- Versión IDE 1.x: #include “Arduino.h”
- Versión IDE 0.x: #include “WProgram.h”

También es común poner todo el fichero de cabecera entre estas instrucciones:

```
#ifndef Morse_h
#define Morse_h

// the #include statment and code go here...

#endif
```

Esto evita problemas si alguien accidentalmente incluye dos veces la librería, lo que provocaría un error de compilación. A esto se llama guardián de inclusión múltiple o include guard.

```
// Guardián de inclusión múltiple
#ifndef FICHERO_YA_INCLUIDO
#define FICHERO_YA_INCLUIDO
```

Así se evita que un compilador poco sofisticado abra otra vez el mismo conjunto de ficheros cuando se incluye un fichero de cabecera dos o más veces. Puede darse el caso de no poner las inclusiones en el inicio de un fichero.

La directiva #include existe en dos versiones. En una se pone el nombre de fichero entre comillas, en la otra entre paréntesis angulares (el signo menor y mayor como “comillas”).

```
#include "fichero_con_comillas.h"
#include <fichero_entre_menor_y_mayor.h>
```

La versión con los paréntesis angulares busca los ficheros en todos los directorios que se han especificado en la llamada al compilador – normalmente con la opción “-I”. Estos directorios se suelen rastrear por el fichero incluido en el orden en que aparecen en la línea de comando.

Cuando se incluye un fichero entre comillas, entonces el compilador busca este fichero primero en el mismo directorio que el fichero actualmente compilado y después en los demás directorios. Es decir, la versión con comillas se diferencia de la versión con paréntesis angulares únicamente por buscar primero en el directorio del fichero compilado. Tras no encontrarlo ahí actaa igual.

Cuando se crea una librería se debe documentar poniendo un comentario al comienzo de la librerías con el nombre, breve descripción, quien la ha escrito, fecha, licencia, etc…

El fichero de cabecera 'Morse.h' queda:

```
1  /*
2    Morse.h - Library for flashing Morse code.
3    Created by David A. Mellis, November 2, 2007.
4    Released into the public domain.
5  */
6  #ifndef Morse_h
7  #define Morse_h
8
9  #include "Arduino.h"
10
11 class Morse
12 {
13   public:
14     Morse(int pin);
15     void dot();
16     void dash();
17   private:
18     int _pin;
19 };
20
21 #endif
```

Una vez hecho el fichero de cabecera hay que codificar el fichero fuente 'Morse.cpp'.

Primero deben ponerse las declaraciones, esto da al resto de código acceso a las funciones estándar de Arduino y a las definiciones del fichero de cabecera:

```
1  #include "Arduino.h"
2  #include "Morse.h"
```

Lo siguiente es poner el constructor de la clase. Esto define que ocurre cuando se crea una instancia de la clase. En este caso el usuario debe especificar cual es el pin que se va a usar. Configuramos el pin como salida y los guardamos en una variable privada para usarlo desde otras funciones.

```
1  Morse::Morse(int pin)
2  {
3    pinMode(pin, OUTPUT);
4    _pin = pin;
5  }
```

El código “Morse::” antes del nombre de la función indica que la función es parte de la clase Morse. Esto se ve en todas las funciones de la clase. La variable llamada “_pin” es una variable privada tal y como se ha definido en el fichero de cabecera y se pone el simbolo “_” delante por convención para indicar que es privada y para diferenciarlo del argumento de la función, pero puede llamarse de cualquier forma mientras coincida con la definición en el fichero de cabecera.

Después de definir el constructor, se deben definir las funciones o métodos de la clase. Son las funciones que se habían definido anteriormente en el sketch:

```
1  void Morse::dot()
2  {
3    digitalWrite(_pin, HIGH);
4    delay(250);
5    digitalWrite(_pin, LOW);
6    delay(250);
7  }
8
9  void Morse::dash()
10 {
11   digitalWrite(_pin, HIGH);
12   delay(1000);
13   digitalWrite(_pin, LOW);
14   delay(250);
15 }
```

También es habitual añadir el comentario del fichero al principio del fichero. El fichero ‘Morse.cpp’ queda de la siguiente forma:

```
1  /*
2    Morse.cpp - Library for flashing Morse code.
3    Created by David A. Mellis, November 2, 2007.
4    Released into the public domain.
5  */
6
7  #include "Arduino.h"
8  #include "Morse.h"
9
10 Morse::Morse(int pin)
11 {
12   pinMode(pin, OUTPUT);
13   _pin = pin;
14 }
15
16 void Morse::dot()
17 {
18   digitalWrite(_pin, HIGH);
19   delay(250);
20   digitalWrite(_pin, LOW);
21   delay(250);
22 }
23
24 void Morse::dash()
25 {
26   digitalWrite(_pin, HIGH);
27   delay(1000);
28   digitalWrite(_pin, LOW);
```

```
  delay(250);
}
```

De esta forma ya tenemos una librería completa. Ahora para incluirla en nuestro IDE debemos crear un directorio Morse dentro del subdirectorio “libraries” del directorio de nuestro entorno de trabajo definido en las propiedades del IDE. Copiar Morse.h y Morse.cpp dentro del directorio Morse y abrir o reiniciar el IDE de Arduino. A partir de este momento veremos nuestra librería disponible en el IDE y podremos incluirla en los sketches con la declaración #include <Morse.h>. La librería será compilada por los sketches que la usen.

El anterior sketch quedaría ahora sustituido por:

```
#include <Morse.h>

Morse morse(13);

void setup()
{
}

void loop()
{
  morse.dot(); morse.dot(); morse.dot();
  morse.dash(); morse.dash(); morse.dash();
  morse.dot(); morse.dot(); morse.dot();
  delay(3000);
}
```

Podemos ver que primero se llama a la declaración de la librería Morse. Esto hace que la librería esté disponible en el sketch y lo incluye en el código enviado a la placa Arduino, lo que hace que si la librería es muy pesada, ocupe mucha más memoria nuestro sketch y si no voy a usar una librería es mejor no incluirla para ahorrar espacio.

También observamos que creamos una instancia de la clase Morse llamada “morse”. Al ejecutar esta línea el constructor de la clase es llamado pasando un argumento, creando así el objeto “morse” en nuestro sketch. Luego podemos llamar a los métodos dot() y dash() precedidos del prefijo morse del nombre del objeto.

Es posible tener múltiples instancias de la clase Morse, cada una un pin diferente guardado en la variable privada “_pin”.

Si creamos una librería es conveniente crear el fichero keywords.txt dentro del directorio Morse. De esta forma conseguiremos resaltar las palabras clave que definamos en el fichero keywords. En cada línea del fichero keywords.txt se indica el nombre de la palabra clave y seguido por un tabulador, el tipo de keyword.

- Las clases deben ser del tipo KEYWORD1 que se resaltan en naranja.
- Las funciones deben ser del tipo KEYWORD2 que se resaltan en marrón

Para que el fichero keywords.txt se aplique al IDE es necesario reiniciar el IDE.
El fichero keywords.txt quedaría:

Morse KEYWORD1
dash KEYWORD2
dot KEYWORD2

También es aconsejable ofrecer ejemplos de uso de la librería para que los posibles usuarios sepan usarla. Esto se hace creando un directorio “examples” dentro del directorio Morse y añadir en el subdirectorio los sketches de ejemplos que serán visibles desde el IDE.

IMPORTANTE: Como se ve en el código de una librería usamos código de de Arduino y por lo tanto podría la librería funcionará con cualquier placa compatible. Pero generalmente las librerías tienen código a bajo nivel que puede que solo valga para un tipo de procesadores y en ese caso habrá que adaptar la librería a las instrucciones del microprocesador con el que queramos usar la librería.