Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería Electrónica Industrial y Automática

Desarrollo de un equipo para la realización de prácticas de Fundamentos de Informática utilizando CODEBLOCKS y la plataforma Arduino.

Autor:

Triana Barreda, Elsa

Tutor:

García Ruiz, Francisco Javier
Ingeniería de Sistemas y Automática

Valladolid, Mayo - 2017.

RESUMEN Y PALABRAS CLAVE

En el presente trabajo se va a desarrollar un equipo de prácticas para la asignatura “Fundamentos de Informática”. Se pretende que el alumno aprenda a programar en C++ mediante la aplicación práctica con diferentes maquetas que utilizan elementos físicos reales tales como motores, diodos LED, etc.
En este punto es necesario plantearse qué material se puede usar y cuáles son los recursos disponibles.
La solución adoptada es el uso de Arduino. Se trata de un micropocesador de software libre muy fácil de manejar y de un precio razonable que permite controlar cualquier dispositivo electrónico.
Dado que la programación requerida por la plataforma Arduino no es C++, se han desarrollado una serie de funciones incluidas en la librería “Led.h” que convertirán el código en C++ introducido por el alumno en instrucciones que el Arduino pueda procesar.

Palabras clave: Arduino, Code::Blocks, Aprendizaje, Programación, Informática.

ABSTRACT AN KEYWORDS

The current document will develop a practice exercise for the subject “Fundamentos de informática”. The student is supposed to learn how to programme with C++ by using the practical application with different models which use real physical items such as motors, LEDs, etc.
At this point it is necessary to consider which material can be used and what the available resources are. The adopted solution is the use of Arduino. It's a free software microprocessor, quite easy to use and reasonably priced, which allows you to control any electronic device.
The programming required by the Arduino platform is not C++, therefore, some functions have been developed and are included in the library "Led.h". These functions will transform the code in C++ introduced by the student, in instructions that the Arduino could process.

Key words: Arduino, Code::Blocks, Learning, Information Technology (IT)

Índice

CAPÍTULO 1. INTRODUCCIÓN

1.1. DESCRIPCIÓN DEL PROBLEMA

Los alumnos de nuevo ingreso en ingeniería se ven forzados a experimentar una serie de cambios muy grandes a nivel académico. Por una parte cambia la metodología en la docencia y por otra parte, el volumen de contenidos tanto teóricos como prácticos es mucho mayor en comparación con lo que están acostumbrados.

Hay asignaturas que son una continuación de los conocimientos adquiridos durante los cursos de bachillerato pero hay otras asignaturas con contenidos totalmente nuevos, en las que los alumnos no tienen los conocimientos básicos para su desarrollo.

En particular, una de estas asignaturas es Fundamentos de Informática. Si bien es cierto que en la actualidad se están empezando a implantar asignaturas de este tipo en colegios e institutos, por norma general los alumnos matriculados por primera vez no tienen conocimiento alguno sobre la materia, dificultando la comprensión de la misma. Una vez adquiridos los conocimientos teóricos necesarios, el alumno debe ponerlos en práctica, lo que, por lo general, les resulta complicado.

El objetivo final, tal como se ve en la guía docente de la asignatura [1], es ofrecer una formación básica y sólida para que el alumno adquiera las destrezas necesarias relacionadas con la informática, tanto en su vida académica como profesional.

La asignatura Fundamentos de Informática forma parte de las asignaturas del módulo básico (y común) de los Grados en Ingeniería. La procedencia de los estudiantes es heterogénea: Bachillerato y PAU y Módulos Superiores, y dependiendo del mismo habrán tenido diferente grado de contacto con los contenidos desarrollados en esta materia. Con ella, se pretende ofrecer una formación básica y sólida al futuro Ingeniero/a. Básica, en el sentido que los diferentes aspectos serán tratados a un nivel introductorio, y sólida, en el sentido de que los conocimientos adquiridos deben sentar las bases para desenvolverse en el resto de su formación académica y desarrollo profesional. Se trata de habilitar a los estudiantes para que adquieran las destrezas necesarias para seguir aprendiendo a lo largo de la vida los aspectos relacionados con la Informática, tanto a nivel de manejo de computadoras como de programación [1].

Considerando que un aspecto fundamental de esta asignatura es la aplicación de los conocimientos adquiridos, el presente trabajo se va a centrar en las sesiones prácticas de la asignatura.

Para tener una buena base en informática es necesario conseguir cierta destreza en la programación. En el primer año de carrera, los estudiantes pueden experimentar una falta de motivación en asignaturas de este tipo, ya que no consiguen encontrar una aplicación práctica real ni son conscientes de su importancia, tanto para asignaturas futuras como para su vida profesional.

En este marco se considera importante, no solo que el alumno supere con éxito la asignatura, sino que sienta que es una base importante con un sinfín de aplicaciones en su futuro académico y profesional.

1.2. OBJETIVOS

Como se ha expresado anteriormente, el objetivo final de este trabajo es despertar una motivación en los estudiantes. Para ello se va a procurar el estudio de la asignatura de una forma más visual y dinámica, donde el alumno pueda ver implementado en tiempo real lo que está programando. En definitiva, se pretende:

- Esta asignatura sienta unas bases muy importantes para otras asignaturas de cursos futuros, por lo que es importante que el alumno entienda Fundamentos de Informática como una herramienta fundamental, tanto para su futuro académico como profesional.
- Motivar a los estudiantes impartiendo las clases prácticas de manera que aprecien los resultados de los problemas que están resolviendo.
- Despertar en el alumno la curiosidad de ver lo que pueden llegar a hacer aplicando los contenidos estudiados.

Para lograr alcanzar estos objetivos se van a desarrollar una serie de prácticas en las que el alumno pueda plasmar los conocimientos adquiridos y vea los resultados por medio de distintos componentes electrónicos.

Se usará la plataforma de prototipado Arduino para la programación. Como soporte se utilizará una placa Arduino Uno y diferentes componentes electrónicos, tales como sensores y diodos LED.

Dado que la plataforma Arduino utiliza un lenguaje de programación un tanto diferente al que deben aprender los alumnos, se han desarrollado una serie

de librerías internas, de manera que el estudiante no debe usar más que los conocimientos obtenidos del lenguaje de programación de alto nivel C.

A lo largo de este trabajo se irán presentando tanto el hardware como el software utilizados, así como las claves del montaje y la programación para, al final, obtener una herramienta con la que el alumno pueda trabajar y donde pueda observar que los conocimientos obtenidos aportan resultados que se aprecian físicamente (mediante un juego de luces, el movimiento de un motor…).

1.3. ESTRUCTURA DE LA MEMORIA

En base al esquema seguido para la realización del presente trabajo, se ha estructurado la memoria de la misma manera:

- PUNTO DE PARTIDA: En primer lugar se ha realizado un análisis del método educativo utilizado hasta el momento en la asignatura Fundamentos de Informática. Se ha revisado la guía docente de la asignatura con el fin de conocer los conocimientos teóricos y prácticos que, se pretende, el alumno adquiera. Se examina, además, el software utilizado hasta el momento en las sesiones prácticas de la asignatura, la plataforma *Code::Blocks.*

- SOLUCIÓN ADOPTADA: Tras realizar el estudio del temario de la asignatura, se obtiene una solución que ayude a incentivar al alumno y facilitar el aprendizaje mediante la visualización directa de los proyectos realizados. Para ello se combinarán el hardware y el software que se desarrollarán a lo largo del presente trabajo.

- SOFTWARE UTILIZADO: Una parte de la solucione del problema es modificar el software utilizado en las sesiones prácticas de la asignatura. En este capítulo se describirá el funcionamiento y se facilitarán los enlaces de descarga de todas las herramientas utilizadas (disponibles en la bibliografía). Los programas a utilizar serán *Code::Blocks Arduino IDE*, el cual presenta la misma interfaz y funcionamiento del que se ha venido usando hasta ahora y la plataforma *Arduino Builder*, para cargar los proyectos en la placa.

- HARDWARE NECESARIO: La otra parte de la solución consiste en crear una caja en la que se situarán distintos dispositivos así como la placa Arduino NANO. Esta caja será el soporte para la realización de las

prácticas y donde el alumno verá cómo se llevan a cabo los proyectos programados en Code::Blocks. En esta parte del trabajo se especifican los componentes utilizados, los conocimientos previos necesarios de cada dispositivo y una descripción del montaje de la caja y conexión de los elementos.

- DESARROLLO DEL PROYECTO: Una parte fundamental del proyecto es la creación de las librerías que servirán para convertir el lenguaje utilizado en C++ en el lenguaje específico de las instrucciones de Arduino. Dado que este trabajo se basa en motivar al alumno y facilitar su aprendizaje se entiende que puede resultar confuso el uso de las instrucciones de Arduino. Es necesario tener en cuenta que se debe incluir ésta librería en cada nuevo proyecto que se realice. En esta parte del trabajo, por tanto, se especificarán las modificaciones a realizar en la librería ya incluida "Servo.h" y se describirá la librería "Led.h" que contendrá las funciones necesarias para el uso de todos los dispositivos.

- CONCLUSIONES Y RESULTADOS: Tras realizar una serie de pruebas al prototipo, se presentarán las conclusiones y se analizarán los resultados así como posibles mejoras a tener en cuenta.

CAPÍTULO 2. PUNTO DE PARTIDA

La metodología docente actual de la asignatura Fundamentos de Informática se basa en la combinación de sesiones teóricas en aula, sesiones prácticas en aula de ordenadores y trabajo personal de estudio y realización de programas. Concretamente se destinan 30 horas a las clases teórico-prácticas y 30 horas a los laboratorios, repartidas en 15 sesiones de 2 horas.

Este trabajo se centra en la parte práctica de la asignatura, es decir, en las 30 horas de laboratorio. El plan de trabajo de esta parte de la asignatura, por bloques temáticos es:

C1 INTRODUCCIÓN AL LENGUAJE C/C++

C2 TIPOS DE DATOS Y ELEMENTOS LÉXICOS DEL C/C++ (1)

PRÁCTICA 0: ACCESO A CUENTAS Y ENTORNO DE CODEBLOCKS

C2 TIPOS DE DATOS Y ELEMÉNTOS LÉXICOS DEL C/C++ (2)

C3 FUNCIONES DE ENTRADA/SALIDA ESTANDAR

PRÁCTICA 1: INTRODUCCIÓN AL C/C++

C4 CONTROL DE FLUJO (1)

PRÁCTICA 2: TIPODE DE DAOTS; ENTRADA - SALIDA

C4 CONTROL DE FLUJO (2)

PRÁCTICA 3: IF, IF-ELSE, SWITCH

C4 CONTROL DE FLUJO (3)

PRÁCTICA 4: FOR, DO, WHILE (1)

C4 CONTROL DE FLUJO (4)

PRÁCTICA 4: FOR, DO, WHILE (2)

C5 FUNCIONES (1)

C5 FUNCIONES (2)

PRÁCTICA 5: FOR, DO, WHILE

C6 PUNTEROS

C7 FICHEROS

PRÁCTICA 6: FUNCIONES

C8 VECTORES Y MATRICES (1)

C8 VECTORES Y MATRICES (2)

PRÁCTICA 7: PUNTEROS Y FICHEROS

C8 VECTORES Y MATRICES (3)

PRÁCTICA 8: VECTORES Y MATRICES (1)

PRÁCTICA 8: VECTORES Y MATRICES (2)

PRÁCTICA 9: ESTRUCTURAS (1)

PRÁCTICA 9: ESTRUCTURAS (2)

Tabla 1: Planificación de la parte práctica de Fundamentos de Informática

A medida que se avanza en los conocimientos teóricos, se realizan las sesiones prácticas. En éstas los alumnos deben familiarizarse con el entorno de desarrollo integrado, Code::Blocks. La realización de los laboratorios así como de los programas que deben realizar los alumnos individualmente, se llevará a cabo en este entorno. Una vez que el estudiante ha redactado correctamente el código, debe compilarlo y el resultado se visualizará a través de una ventana de MSDOS.

En un primer momento, tal como se aprecia en la tabla 1, los alumnos irán aprendiendo conceptos básicos del lenguaje, tales como elementos léxicos, tipos de datos o cómo trabajar con entradas y salidas. A medida que van avanzando las clases teóricas se van introduciendo más conceptos como el uso de estructuras de control o funciones. Dado que se trata de estudiantes nada experimentados, al comienzo de la asignatura se les aportará el código completo del programa, de forma que el alumno solo tenga que compilarlo y ver los resultados. Así, a medida que van avanzando los conocimientos adquiridos, el código proporcionado va siendo cada vez menor hasta que sea el propio alumno quién realice el código completo.

2.1. CODE::BLOCKS

La parte práctica de la asignatura se realizará en el entorno de desarrollo integrado (Integrated Development Environment, IDE). Se trata de una aplicación informática que proporciona servicios integrales para facilitar el desarrollo de software. Consiste en un editor de código fuente, herramientas de construcción y un depurador. En el caso de Fundamentos de Informática, el entorno utilizado es Code::Blocks.

A grandes rasgos se trata de un entorno de desarrollo gratuito compatible con varios compiladores distintos, que puede utilizarse en diversos sistemas operativos. Gracias a los numerosos plugins y opciones, es plenamente configurable. Cabe resaltar que, al prescindir de archivos Make, el proceso de compilación es mucho más rápido.

Para obtener mayor información, manuales de uso o soporte para la descarga, se puede acceder a la página oficial [2]

A continuación se va a realizar una breve descripción del funcionamiento del programa.

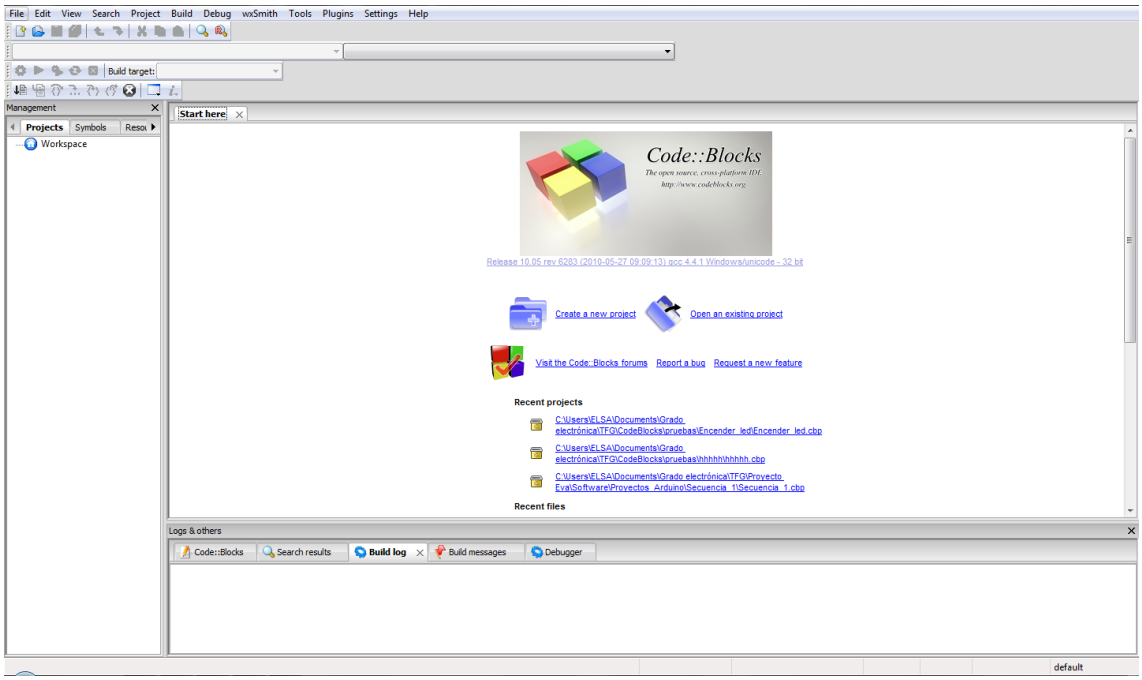

Figura 1: Interfaz de Code::Blocks

Al inicializar el programa aparece la interfaz, desde donde podemos crear un nuevo proyecto o abrir un proyecto ya existente de forma rápida y sencilla. Tal

como se aprecia en la figura 1, existen 3 ventanas, además de las barras de menús y barras de herramientas.

- Ventana de Proyectos (Management): Se accede a la ruta de cada archivo que se haya abierto.
- Ventana de Edición: Inicialmente aparece un menú que da acceso a:
  - Crear un nuevo proyecto (Create e new Project)
  - Abrir un proyecto existente (Open an existing Project)
  - Enlace al foro oficial de Code:Blocks
  - Proyectos recientes (Recent Projects)
  - Archivos recientes (Recent Files)

  Tras crear un nuevo proyecto o abrir uno existente, ésta será la ventana donde se escribe el código en lenguaje C/C++
- Ventana de información (Logs & others): Como se aprecia en la figura 1, existen varias pestañas que aportarán distinta información del proyecto. Cabe destacar las referidas al proceso de compilación del programa, que darán información acerca del estado de la compilación, los posibles errores que se detectan y dónde encontrarlos así como la información correspondiente a la depuración del programa.

El alumno debe familiarizarse y desenvolverse con fluidez en este entorno. Se describe, a continuación, la metodología a seguir para crear un archivo nuevo y los pasos necesarios hasta realizar la compilación y vista de los resultados.

En primer lugar, se creará un archivo nuevo siguiendo la ruta: FILE → NEW → EMPTY FILE (Ctrl-Shift-N)

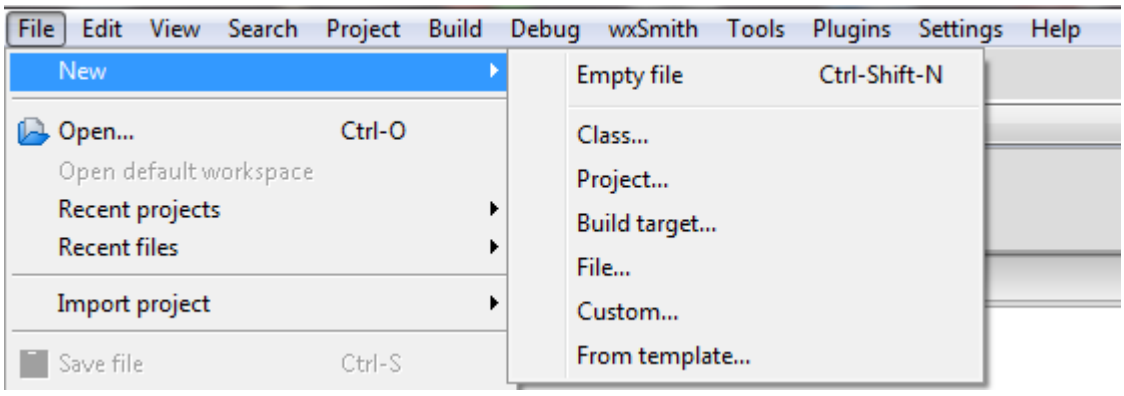

Figura 2: Crear un archivo nuevo en Code::Blocks

Se abre un archivo en blanco llamado Untitled1.c

Figura 3: Archivo vacío en Code::Blocks

En esta ventana es donde se redactará el código del programa. Como ejemplo se va a realizar un programa que imprima por pantalla la frase “Hola mundo”.

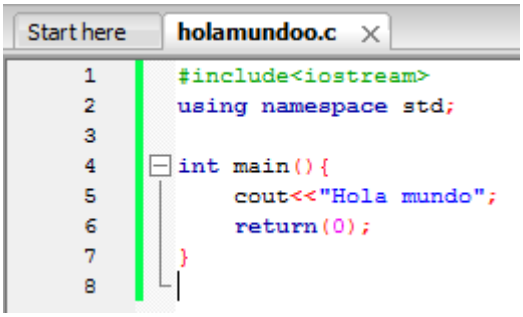

Figura 4: Ejemplo "Hola mundo"

Una vez redactado el código, se compila y ejecuta. Para esto se usará la barra de herramientas:

Figura 5: Barra de herramientas para compilar y ejecutar

De izquierda a derecha, las opciones de esta barra son:

- Build (compilar)
- Run (ejecutar)

- Build and run (compilar y ejecutar)
- Rebuild (volver a compilar)
- Abort (parar)

Pulsando la primera opción, es decir, compilar (build), se obtiene en la ventana de información que no se ha producido ningún error

```
Compiling: C:\Users\ELSA\Desktop\hola_mundo.c
Linking console executable: C:\Users\ELSA\Desktop\hola_mundo.exe
Process terminated with status 0 (0 minute(s), 4 second(s))
0 error(s), 0 warning(s) (0 minute(s), 4 second(s))
```

Figura 6: Compilación ejemplo "Hola mundo"

Puesto que la compilación se ha realizado con éxito, se procede a la ejecución del código. Automáticamente se abre una ventana MSDOS donde se observan los resultados de la ejecución del programa:

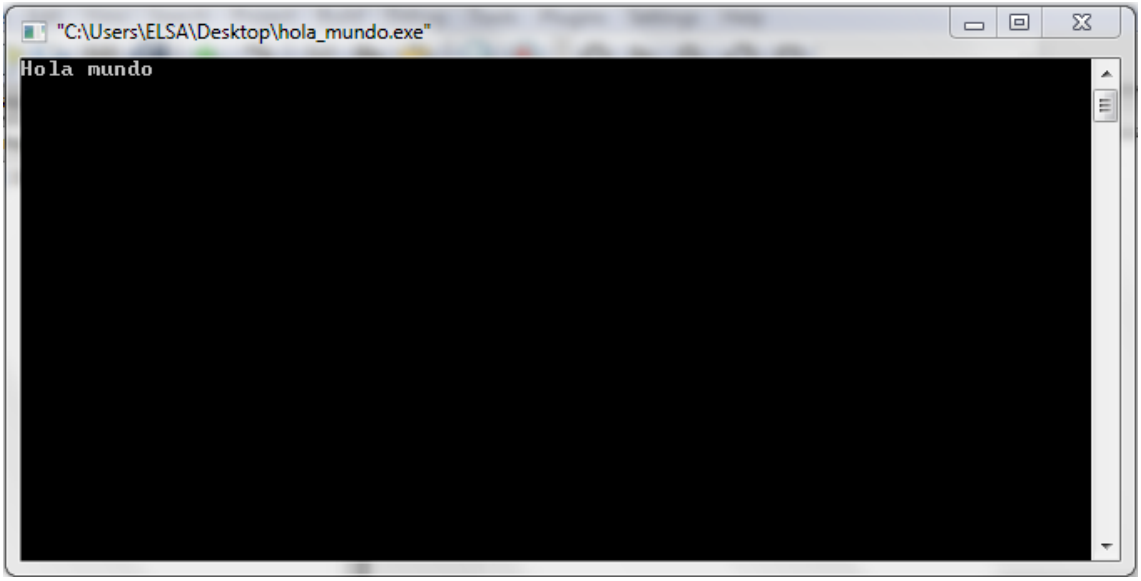

Figura 7: Ejecución ejemplo "hola mundo"

Llegados a este punto el siguiente paso es enseñar al alumno cómo crear un proyecto con Code::Blocks. Un proyecto de este tipo puede estar formado por varios archivos como los realizados en el ejemplo anterior. Para crear un nuevo proyecto basta con seguir la siguiente ruta: FILE → NEW → PROJECT...

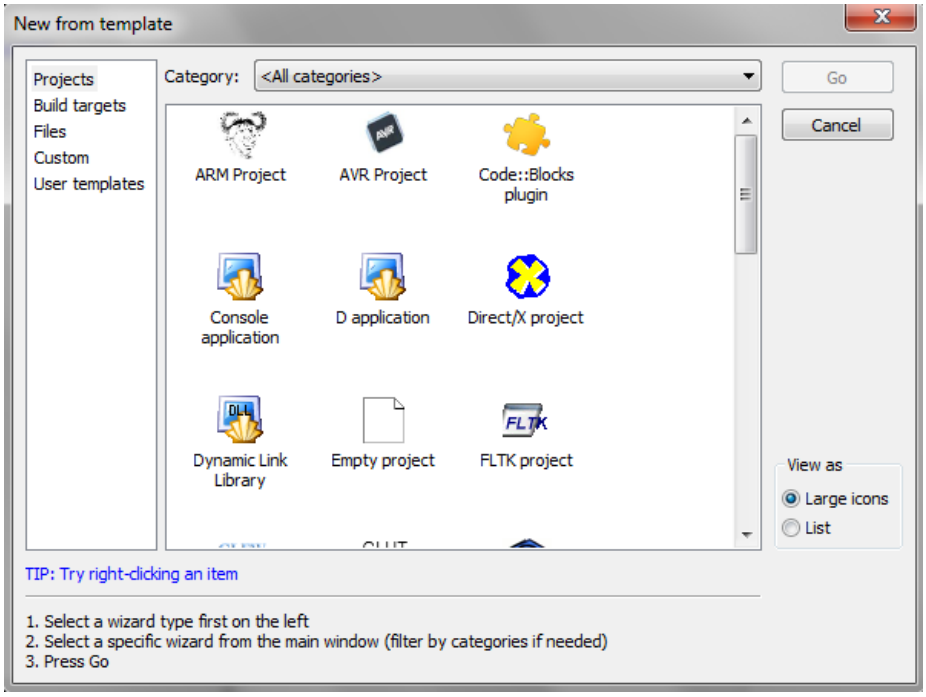

Figura 8: Ventana de creación de un proyecto en Code::Blocks

En ventana mostrada en la figura 8, se selecciona la opción de Empty Project (proyecto vacío). A continuación se abrirán una serie de ventanas:

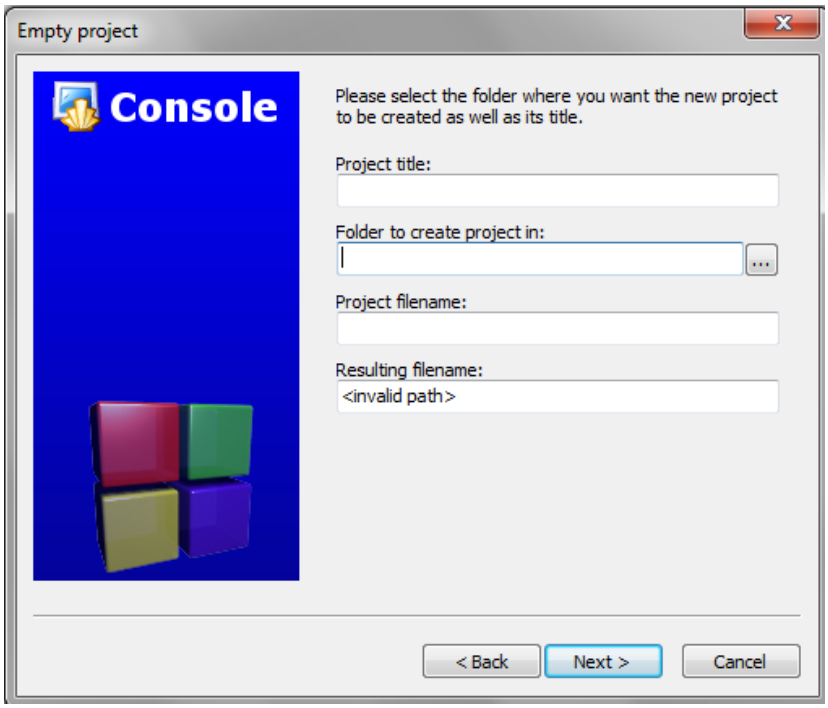

Figura 9: Creación de un proyecto vacío (1)

En esta ventana se dará nombre al proyecto y se buscará la ubicación donde se desea que quede guardado.

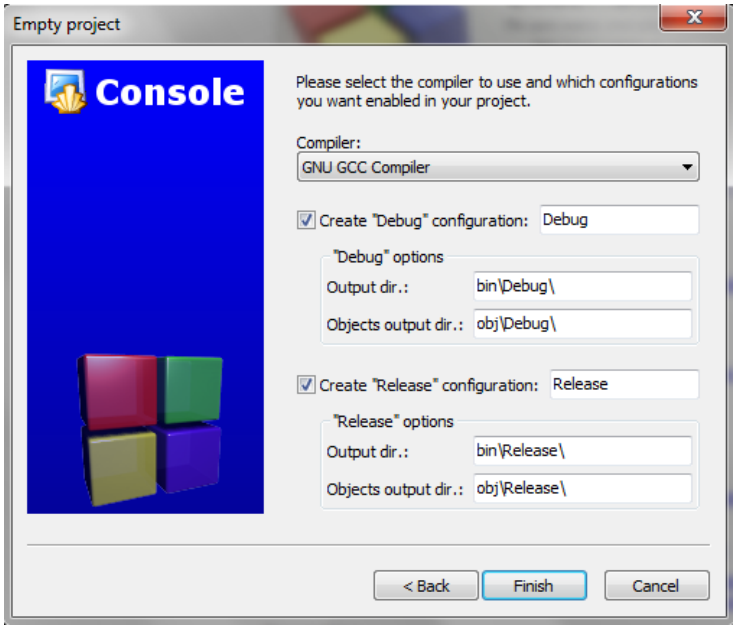

Figura 10: Creación de un proyecto vacío (2)

En esta ventana se selecciona el tipo de compilador. Puesto que el alumno va a trabajar con el lenguaje de programación de alto nivel C++, el compilador seleccionado será GNU GCC Compiler

Una vez terminado este proceso, el alumno debe crear un fichero vacío y guardarle dentro del proyecto creado. De esta manera se pueden implementar varios códigos en distintos archivos e insertarlos todos en el proyecto creado.

CAPÍTULO 3. SOLUCIÓN ADOPTADA

Para los alumnos de nuevo ingreso, la programación puede resultar tediosa y complicada. A la vista de esta falta de motivación se está trabajando en nuevos métodos que aporten un enfoque más práctico y visual. En las sesiones prácticas de la asignatura, el estudiante debe ir implementando unos códigos, cada vez más difíciles y con mayor grado de conocimientos. El problema es que muchas veces estos programas no aportan una respuesta física, la única interacción una vez ejecutado el programa es una ventana de MSDOS en la que, o bien se piden datos como entrada del programa o bien se expresan resultados como salida de éste.

Es por esto que se espera que si el alumno ve resultados más allá de la pantalla del ordenador, su nivel de implicación será mayor. Para obtener estos resultados se va a usar la plataforma Arduino y una serie de sensores y dispositivos. Con el fin de no complicar el aprendizaje, el montaje irá introducido en una caja de manera que el alumno verá únicamente los dispositivos, sin apreciarse las conexiones.

Dado que es la plataforma con la que se ha venido trabajando hasta ahora, se utilizará el entorno de desarrollo Code::Blocks Arduino IDE, cuya interfaz es muy similar a la versión utilizada anteriormente y permite crear, compilar y cargar directamente los proyectos a la plataforma Arduino.

Con esta solución se pretende motivar al alumno y hacerle ver que la programación va mucho más allá de una ventana MSDOS. Al ver resultados físicos, tales como crear movimiento o encender luces, se espera que el estudiante valore esta asignatura como una inversión de futuro, tanto académico como profesional y valore la magnitud de cosas que se pueden hacer con la programación.

CAPÍTULO 4. SOFTWARE UTILIZADO

4.1. ARDUINO

Arduino es una plataforma de prototipos electrónica de código abierto (open-source) basada en hardware y software flexibles y fáciles de usar. El microcontrolador de la placa programa usando el Arduino Programming Language (basado en Wiring) y el Arduino Development Environment (basado en Processing).

Esta plataforma ofrece una serie de ventajas sobre otros sistemas:

- Barato: Las placas Arduino son relativamente baratas comparadas con otras plataformas microcontroladas.
- Multiplataforma: El sistema de Arduino se ejecuta en sistemas operativos Windows, Macintosh OSX y GNU/LINUX.
- Entorno de programación simple y claro: El entorno de programación de Arduino es fácil de usar para principiantes, pero suficientemente flexible para que usuarios más experimentados puedan aprovecharlo también.
- Código abierto y software extensible: El software Arduino está publicado como herramientas de código abierto, disponible para extensión de programadores experimentados. El lenguaje puede ser extendido mediante librerías C++
- Código abierto y hardware extensible: El Arduino está basado en microcontroladores ATMEGA8 y ATMEGA168 de Atmel. Los planos para los módulos están publicados bajo licencia Creative Commons, por lo que diseñadores experimentados de circuitos pueden hacer si propia versión del módulo, extendiéndolo y mejorándolo.

Para más información, se puede acceder a la página oficial de Arduino [3].

En ella, además de la descripción de la placa y su funcionamiento, existen infinidad de códigos, ejemplos y proyectos que pueden resultar interesantes para un principiante en esta plataforma.

Como se ha expresado anteriormente, Arduino tiene su propio lenguaje de programación. Es un tanto contradictorio que el propósito final de este trabajo sea incentivar el aprendizaje del alumno y sin embargo se utilice una plataforma con un lenguaje diferente al estudiado. El estudiante, en realidad, va a utilizar el entorno de desarrollo Code::Blocks Arduino IDE, por lo que una

parte muy importante de este trabajo es la creación de diversas librerías que conviertan el código de programación del alumno (C++) en código Arduino. De esta manera, el estudiante utilizará el código visto en clase y de una manera interna y ajena a él, se procesará para que pueda ser utilizado por la placa Arduino.

Para el desarrollo de este proyecto, como se especificará más adelante, se utilizará la placa Arduino NANO que, entre otras ventajas, resulta más económica y ocupa menos espacio.

4.2. CODE::BLOCKS ARDUINO IDE

Creada por Stanley Huang, Code::Blocks Arduino IDE es una distribución personalizada de Code::Blocks de código abierto. Se trata de una versión mejorada que, conservando la sencillez del entorno, permite crear proyectos con la plataforma Arduino con la incorporación de las funciones necesarias.

Las principales características de esta aplicación son:

- Contiene un asistente específico para la creación de proyectos en Arduino.
- Tiene integrados los archivos principales y las librerías específicas para la creación de proyectos en Arduino.
- Almacena los archivos en la memoria caché para una velocidad de compilación más rápida.
- Contiene el compilador preconfigurado AVR, lo que permite la compilación directa de los proyectos Arduino.
- Es compatible con multitud de placas de desarrollo de la plataforma Arduino.
- Carga los archivos .HEX directamente en la tarjeta Arduino a través de un USB.
- Contiene un monitor serie integrado para la interactuación con Arduino.
- Dado que se trata de un código ejecutable, no de una aplicación propiamente dicha, no requiere ningún tipo de instalación.

A continuación, se dan los pasos a seguir para descargar y configurar el entorno de desarrollo Code::Blocks Arduino IDE.

En primer lugar, se accede a la página oficial de Code::Blocks Arduino IDE [4]:

En esta página se encuentra una descripción detallada del proceso que he seguido el creador del programa hasta obtener la solución final, pasando por los problemas que ha ido encontrando y cómo han sido solucionados. Además de una pequeña guía de funcionamiento basada en capturas de pantalla del programa, se encuentran los enlaces de descarga para Windows y Linux. En este trabajo se va a trabajar con el sistema operativo Windows, por lo que se debe seleccionar ese enlace [5].

En esta página aparecen los enlaces para descargar todas las versiones que el creador ha ido realizando. Dado que cada actualización ofrece una mejora y soluciona problemas encontrados en versiones anteriores, interesa descargar la última versión disponible:

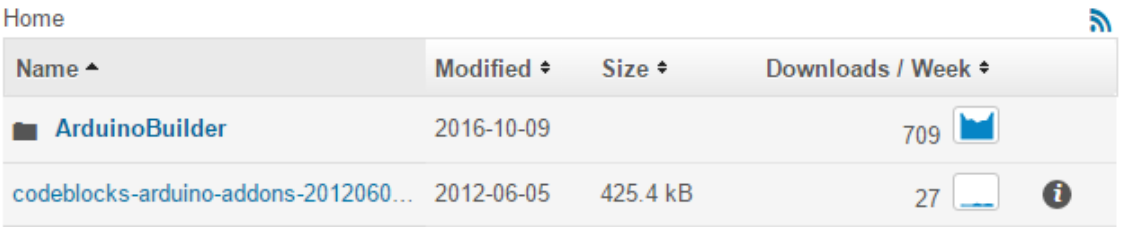

Figura 11: Enlace de descarga de la última versión de Code:Blocks Arduino IDE

Al realizar la descarga, el archivo comprimido aparece en la carpeta de descargas. El archivo descargado se encuentra comprimido en 7z, por lo que es necesario descomprimirlo con el programa 7-Zip para poder usar los archivos [6].

Una vez descomprimido el archivo, se obtiene una carpeta de archivos llamada CodeBlocks en la que se encuentran los siguientes archivos:

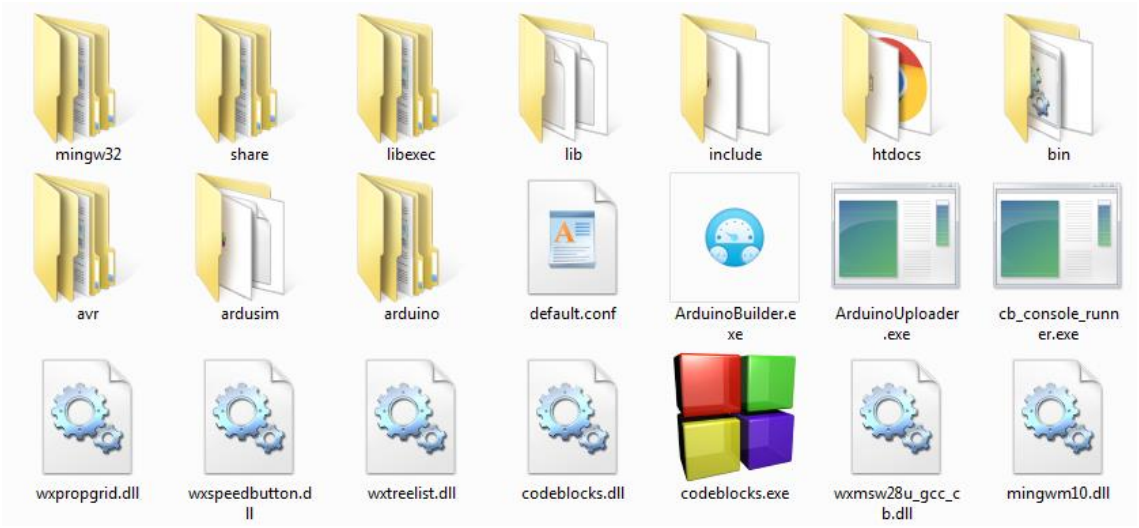

Figura 12: Archivos y carpetas de CodeBlocks Arduino IDE

Cabe destacar que la aplicación Code::Blocks Arduino IDE es un archivo ejecutable, por lo que no se crea un acceso directo. Otro de los archivos importantes es la carpeta arduino:

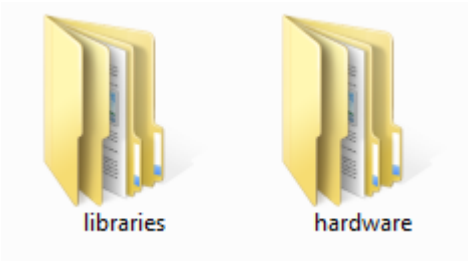

Figura 13: Archivos de la carpeta arduino de Code::Blocks Arduino IDE

En la carpeta libraries se encuentran las librerías propias de la aplicación. En ellas se incluyen las librerías necesarias para el correcto funcionamiento de Arduino. Además, en caso de crear o añadir librerías nuevas, será en esta carpeta donde deben ir situadas.

4.2.1. ESTRUCTURA DE UN PROYECTO EN ARDUINO

Al crear un nuevo proyecto en el programa Code::Blocks Arduino IDE aparece un código de ejemplo. Éste código, denominado por el programa como Sketch, es el proyecto sobre el que se van a sentar las bases de la estructura de un programa en Arduino:

Ejemplo.ino

```
1  #include <Arduino.h>
2
3  /*
4    Turns on an LED on for one second, then off for one second, repeatedly.
5  */
6
7  void setup()
8  {
9      Serial.begin(9600);
10
11     // initialize the digital pin as an output.
12     // Pin 13 has an LED connected on most Arduino boards:
13     pinMode(13, OUTPUT);
14 }
15
16 void loop()
17 {
18     Serial.println("Hello world!");
19
20     delay(1000);              // wait for a second
21     digitalWrite(13, HIGH);   // set the LED on
22     delay(1000);              // wait for a second
23     digitalWrite(13, LOW);    // set the LED off
24 }
25
```

Figura 14: Ejemplo de proyecto Arduino

En primer lugar es necesario incluir la librería. En la primera del código de la figura, aparece la directiva de preprocesador para incluir el archivo de cabecera de la librería Arduino.h, que contiene todas las funciones específicas para el correcto funcionamiento de la placa. La nomenclatura es la misma que en el lenguaje C++, siendo una clara ventaja para los alumnos de Fundamentos de Informática.

La estructura del programa se divide en dos funciones especiales. Estas funciones deben aparecer siempre en el código del proyecto:

- Void detup():

  Se ejecuta una única vez al iniciar el sketch y después de cada arranque o reset de la placa Arduino.

  En esta función debe incluirse la configuración, es decir, la declaración e iniciación de las variables, la definición como entrada o salida de los pines de la placa Arduino y la apertura del puerto serie entre otras.

- Void loop():

  Se ejecuta cíclicamente, una y otra vez mientras la placa se encuentre conectada a la alimentación.

En esta función se especifican las instrucciones que el programa debe llevar a cabo. Controla activamente el comportamiento de la placa Arduino ya que es la parte encargada de cambiar y responder ante los distintos eventos que se produzcan en los pines de la placa.

4.2.2. CREACIÓN DE UN NUEVO PROYECTO EN CODE::BLOCKS ARDUINO IDE

Para crear un nuevo sketch con Code::Blocks Arduino IDE, la ruta a seguir es: File→New→Project...

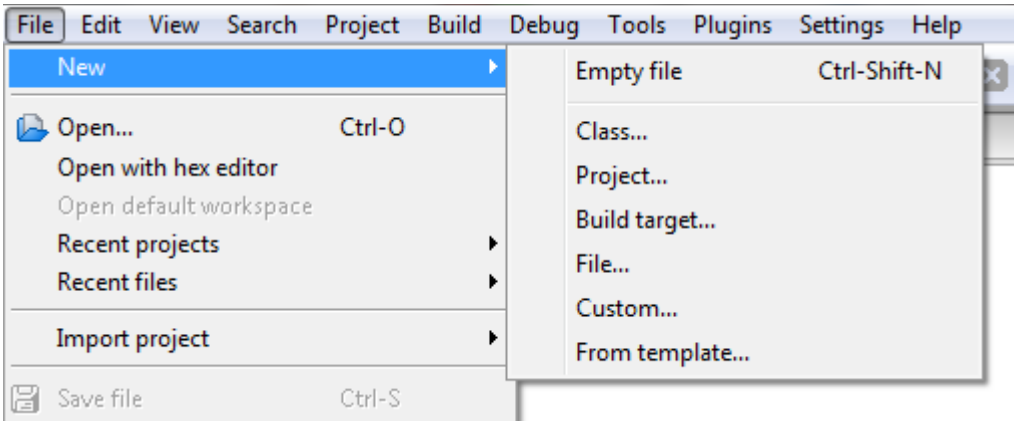

Figura 15: Ruta de creación de un sketch en Code::Blocks Arduino IDE

Se abrirá una ventana con distintas opciones, en la que se debe seleccionar el icono de Arduino Project:

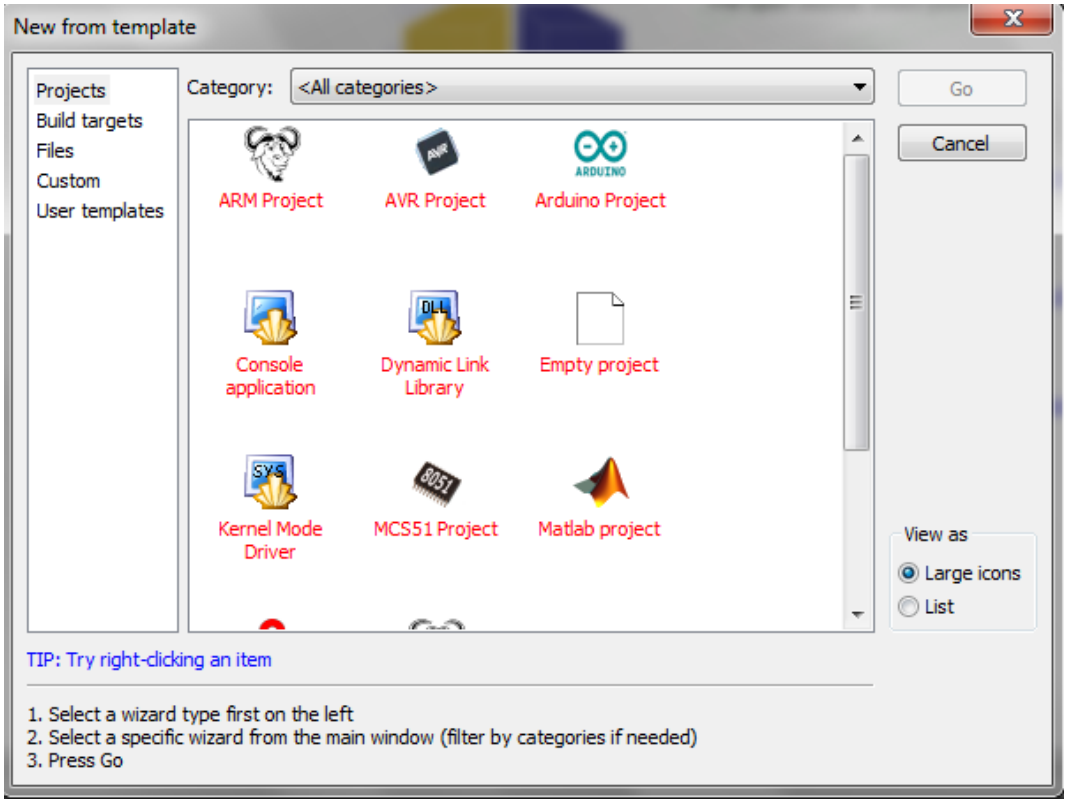

Figura 16: Creación de un proyecto de arduino en Code::Blocks Arduino IDE

Aparecerán las ventanas emergentes en las que se va a configurar el proyecto:

Figura 17: Asistente para la creación de un nuevo proyecto en Code::Blocks Arduino IDE

Al pulsar Next, aparece la siguiente ventana de configuración en la que se selecciona la serie a la que pertenece la placa Arduino a utilizar y la frecuencia de reloj a la que trabaja. En esta parte se dejará los datos que aparecen por defecto. Otra opción a seleccionar es el puerto serie al que se encuentra conectado la placa Arduino y a través del cual se realizará la comunicación.

Para saber el puerto serie al que está conectado la placa es necesario acceder, a través del sistema operativo, a la información de los dispositivos conectados al ordenador. La ruta a seguir para obtener esta información es diferente en función del sistema operativo con el que se trabaje. En el caso de los sistemas más comunes esta ruta es:

- Windows XP:

  Panel de control → Rendimiento y mantenimiento → Sistema → Hardware → Administrador de dispositivos → Puertos COM & LPT

- Windows 7:

  Panel de control → Hardware y sonido → Dispositivos e impresoras

Tras localizar el puerto de conexión, se abre el desplegable y se selecciona el COM. Esta selección es optativa y no es necesario realizarla, pudiéndose seleccionar más adelante.

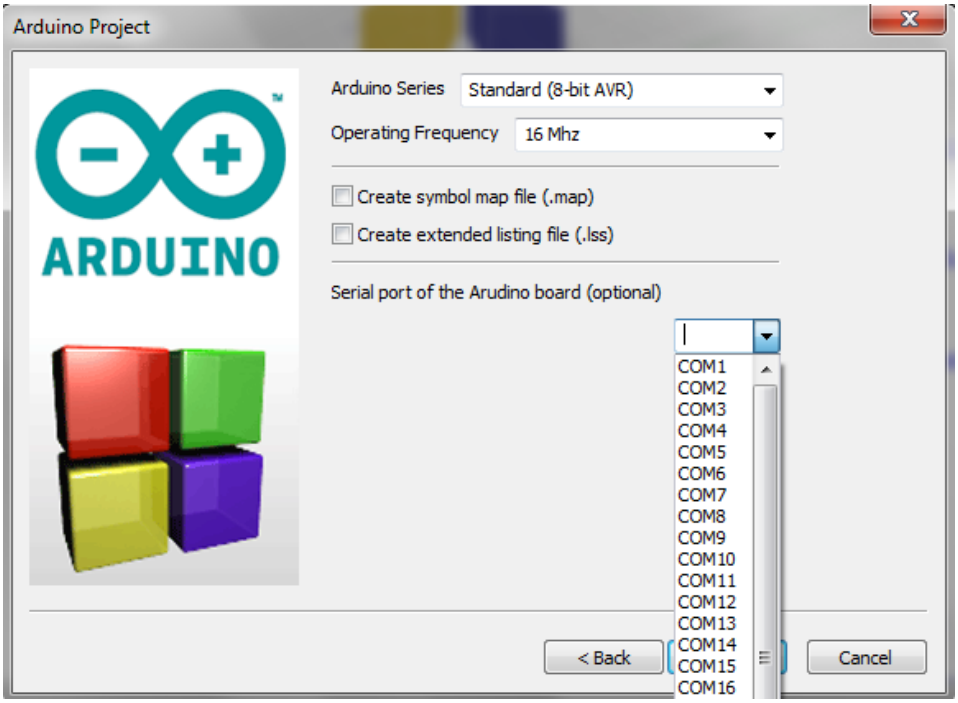

Figura 18: Configuración del tipo de placa, frecuencia de funcionamiento y COM

Tras realizar las selecciones pertinentes, al pulsar Next aparecerá una última ventana, en la que se dará nombre al proyecto y y se especificará la ruta donde va a crearse y guardarse dicho sketch:

Figura 19: Configuración del nombre y ruta de acceso de un nuevo proyecto

En esta parte es importante tener en cuenta que ni el título del proyecto ni ninguna de las carpetas de la ruta de accesos puede llevar un espacio en el

nombre ya que generará errores más adelante en la creación del archivo de extensión .hex

Tras realizar la configuración y pulsar Finish, el asistente se cerrará y en la ventana de edición de la interfaz de Code::Blocks Arduino IDE aparecerá el nuevo sketch. Siempre que se crea un nuevo proyecto se abrirá el código de ejemplo estudiado anteriormente. Es una clara ventaja ya que borrando las instrucciones del programa quedará la estructura que debe tener el proyecto, facilitando el trabajo a los estudiantes. La estructura que debe dejar el alumno para empezar a trabajar en un nuevo proyecto es:

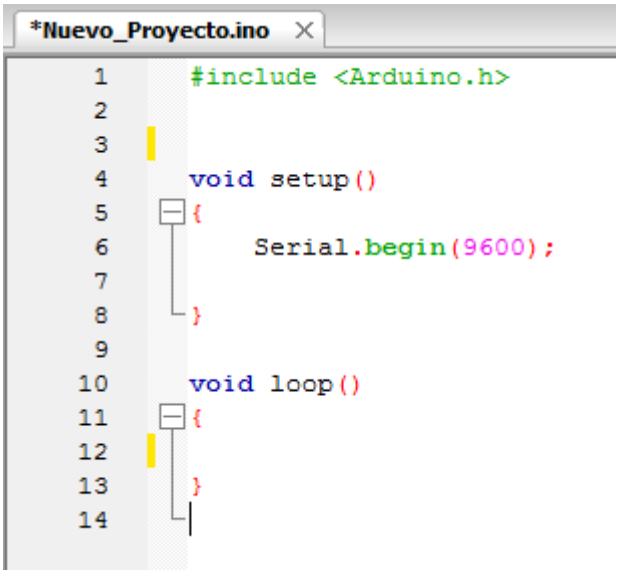

Figura 20: Estructura básica de un nuevo proyecto

4.2.3. ESTRUCTURA DE UNA LIBRERÍA DE FUNCIONES DE ARDUINO

Una de las grandes ventajas que proporciona la plataforma Arduino es la versatilidad obtenida mediante la creación de librerías que permiten el correcto funcionamiento de los diversos elementos que quieran conectarse a la placa. El entorno Arduino, así como la aplicación Code::Blocks Arduino IDE, proporcionan una serie de librerías incorporadas por defecto que facilitan el uso desde los dispositivos más simples hasta los más complejos. Dado que el planteamiento de Arduino es el software libre y gratuito, muchas de éstas librerías se encuentran al alcance de cualquier usuario en la red. Pero además, gracias a la gran versatilidad del entorno, es posible realizar una librería propia con las características e instrucciones que el usuario demande. Ésta es una parte fundamental del presente trabajo, que se basa en la creación de la librería “Led.h”.

Las librerías de funciones para Arduino se escriben en lenguaje C orientado a objetos y deben tener siempre la misma estructura. Para realizar un estudio más detallado del funcionamiento de las librerías, se usará la librería Morse.h que implementará un código morse mediante el parpadeo de un LED conectado a la placa Arduino.

- Archivo de cabecera (Morse.h)

Morse.h

```
1  #ifndef MORSE_H_INCLUDED
2  #define MORSE_H_INCLUDED
3
4  //Declaracion de la clase Morse
5  class Morse {
6      //prototipo de las funciones de la clase
7      public: morse (int pin);
8              void punto ();
9              void raya();
10     //variables privadas
11     private: int _pin;
12 };
13
14 #endif // MORSE_H_INCLUDED
```

Figura 21: Archivo de cabecera: Morse.h

En primer lugar aparecen las directivas de preprocesador que definen el archivo de cabecera Morse.h e incluyen el archivo de cabecera Arduino.h.

El siguiente paso es la declaración de la clase Morse. En esta parte se realiza una agrupación de un conjunto de funciones y variables, tanto públicas como privadas, bajo un mismo nombre.

En primer lugar se definen los prototipos de las funciones públicas:

- Morse (int pin): Función especial denominada constructir de la clase que permite crear una instancia de la clase, es decir, una llamada a un entero de la clase.
- Void punto(): Declaración de la función punto.
- Void raya(): Declaración de la función raya.

En última instancia se encuentra la definición de las variables privadas pertenecientes a la clase. Estas variables únicamente serán usadas dentro de las funciones pertenecientes a esta clase

- Archivo fuente (Morse.cpp)

Morse.cpp

```
1  #include "Arduino.h"
2  #include "Morse.h"
3
4  //Definicion del constructor de la clase
5  Morse :: Morse (int pin)
6  {
7      pinMode (pin, OUTPUT);
8      _pin = pin;
9  }
10
11 //Definicion de la funcion punto
12 void Morse :: punto()
13 {
14     digitalWrite (_pin, HIGH);
15     delay (250);
16     digitalWrite (_pin, LOW);
17     delay(250);
18 }
19
20 //Definicion de la funcion raya
21 void Morse :: raya()
22 {
23     digitalWrite (_pin, HIGH);
24     delay(1000);
25     digitaloWrite (_pin, LOW);
26     delay(250);
27 }
```

Figura 22: Archivo fuente: Morse.cpp

Nuevamente, en primer lugar aparecen las directivas de preprocesador en las que se incluyen los archivos de cabecera “Arduino.h” y “Morse.h”.

El siguiente paso es la definición del constructor de la clase, donde se indican las acciones que deben realizarse cuando se crea una instancia a la clase. En este caso, se pasa como parámetro de la instancia el pin en el que se va a conectar el LED, asignando éste valor a la variable privada _pin y además configura dicho pin como salida.

Para finalizar, se definen las funciones que forman parte de la librería, indicando las instrucciones que debe realizar cada una.

- Archivo keywords.txt (opcional)

keywords.txt

```
/* Las clases son de tipo KEYWORD1*/

Morse KEYWORD1

/* Las funciones son de tipo KEYWORD2*/

punto KEYWORD2
raya  KEYWORD2
```

Figura 23: Archivo keywords.txt

Mediante este código se indica al entorno las palabras que deben aparecer resaltadas al emplear las funciones de la librería en los sketch de Arduino. En cada línea se incluye la palabra clave seguida de un tabulador sin espacios y la referencia KEYWORD1 cuando se trata de una clase y KEYWORD2 cuando se trata de una función

A continuación se detallan las claves para el uso de la librería definida:

En primer lugar es necesario incluir las directivas de preprocesador Arduino.h y Morse.h.

Se creará una instancia a la clase Morse, denominada morse, y se indicará como parámetro el pin en le que se va a situar el LED.

En la función loop () se realizará la llamada a las funciones mediante morse.punto(); y morse.raya();

prueba_morse.ino

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

Morse morse(1);

void setup()
{

}

void loop()
{
    morse.punto();
    morse.raya();
    delay(1000);
}
```

Figura 24: Sketch que usa la libreria Morse.h

El código representará un punto y una raya del código morse mediante el LED conectado en el pin 1 de la placa Arduino.

El objetivo principal del presente proyecto es la realización de una librería que convierta el lenguaje C++ usado por el alumno en un lenguaje específico con el que trabaja Arduino, pudiendo realizar el uso de todos los dispositivos sin necesidad de aprender nuevas instrucciones que compliquen el aprendizaje de la asignatura.

4.2.4. CREACIÓN DE UNA LIBRERÍA NUEVA EN CODE::BLOCKS ARDUINO IDE

Una vez definida la estructura de una librería compatible con Arduino, es necesario definir la metodología a seguir para incluir ésta en la plataforma Code::Blocks Arduino IDE. De esta manera la librería quedará guardada en el programa y el alumno únicamente debe conocer su nombre e incluirla al principio del código, como ha venido haciendo en la programación en C++.

A lo largo de este proyecto se ha desarrollado una librería propia que incluye todas las funciones necesarias para el correcto funcionamiento de los distintos dispositivos. Esta librería se ha denominado “Led.h” y a continuación se detallan los pasos a seguir para incluirla en el software del programa.

En primer lugar es necesario conocer la ruta de acceso al directorio donde se guardan las carpetas correspondientes a las librerías para Arduino que vienen configuradas por defecto en el entorno de desarrollo.

Al descargar la aplicación Code::Blocks Arduino IDE se obtiene el siguiente árbol de archivos:

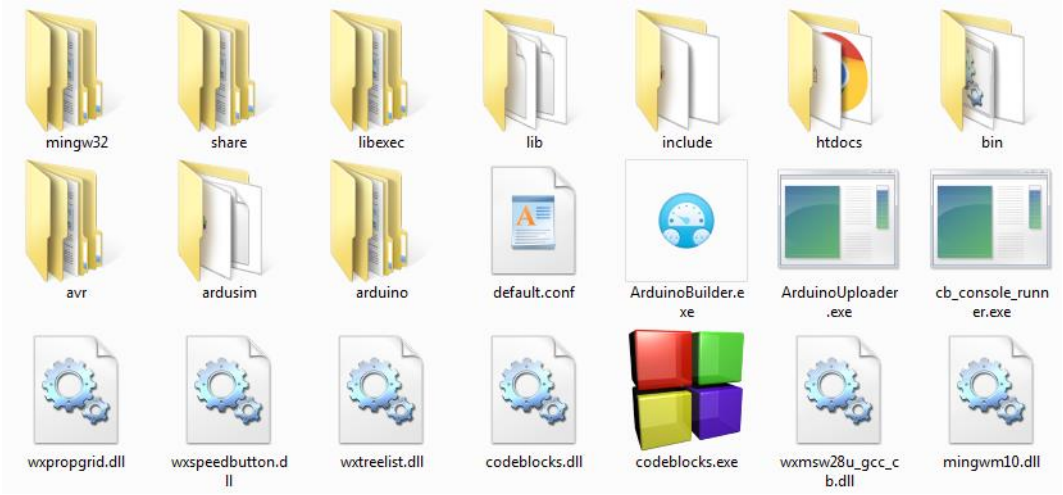

Figura 25: Árbol de archivos de Code::Blocks Arduino IDE

Accediendo a la carpeta Arduino aparecen dos nuevas carpetas, libraries y hardware. Al abrir la carpeta libraries se obtienen las carpetas correspondientes a las librerías para Arduino incluidas por defecto en el entorno de desarrollo. En este punto se crea la nueva carpeta correspondiente a la librería propia que va a desarrollarse. En este caso la librería tendrá el nombre Led:

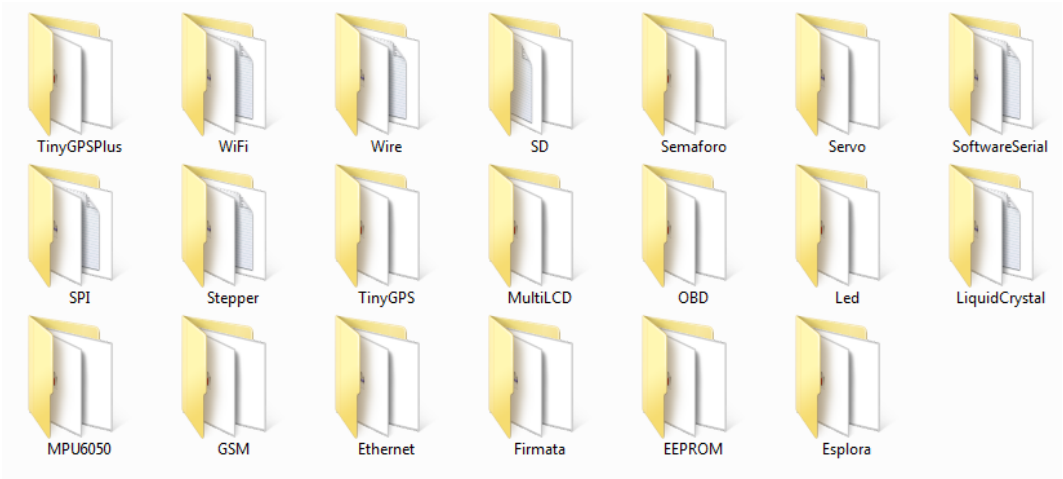

Figura 26: Creación de la carpeta Led dentro de libraries

Una vez creada la carpeta donde se guardará la librería, se procede a crear los archivos y desarrollar los códigos.

- Archivo de cabecera "Led.h"

En el interfaz de Code::Blocks Arduino IDE, mediante la siguiente ruta: File → New → File

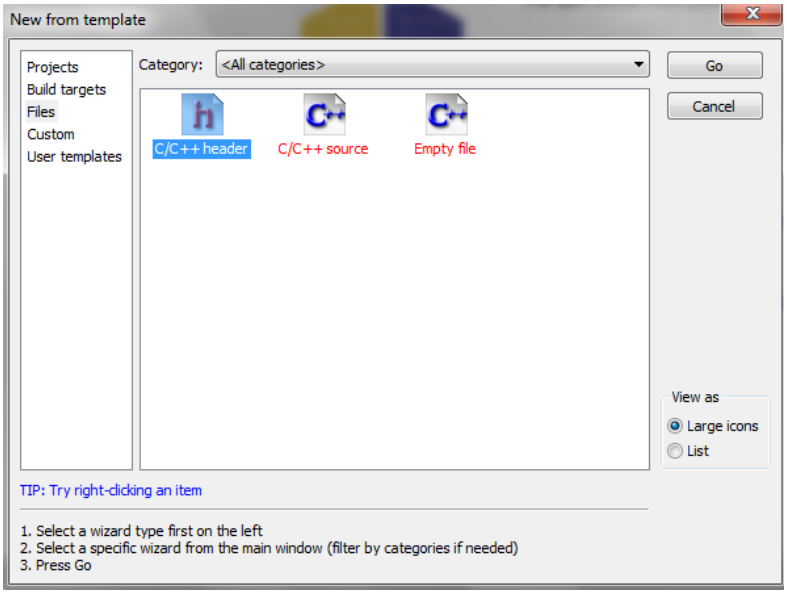

Figura 27: Nuevo archivo de cabecera en Code::Blocks Arduino IDE

Al seleccionar C/C++header, aparecerán una serie de ventanas emergentes para realizar la configuración del archivo de cabecera:

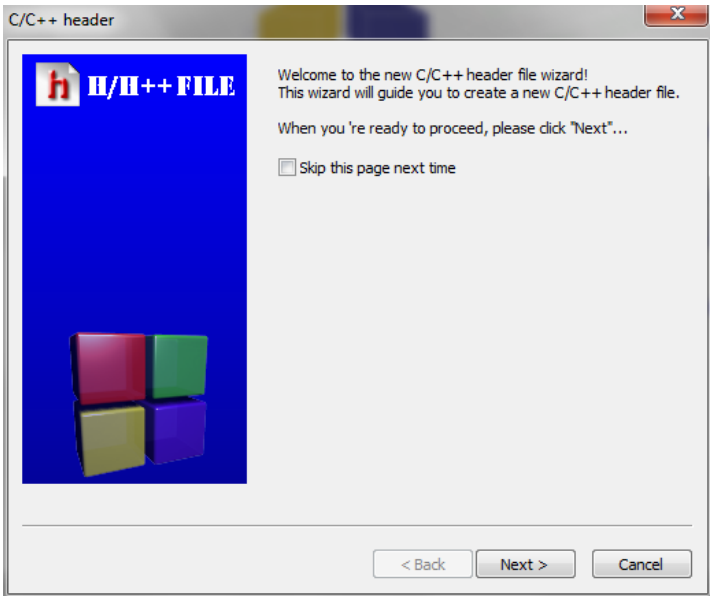

Figura 28: Asistente para la creación de un archivo de cabecera

Al pulsar Next comienza la configuración del archivo de cabecera:

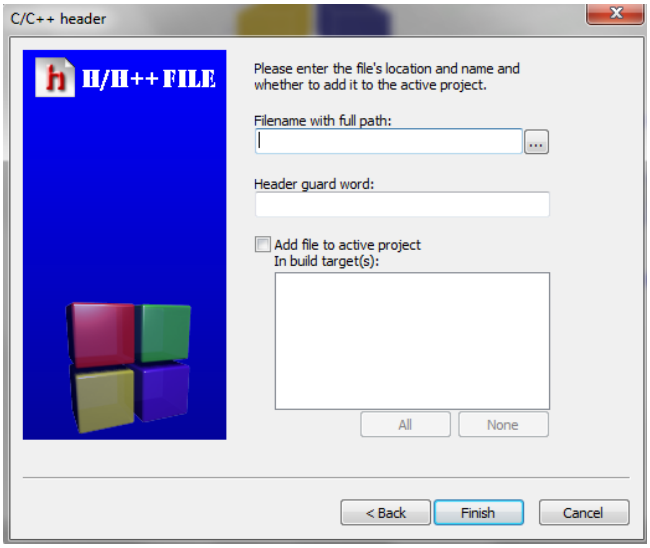

Figura 29: Configuración del nombre y ruta del archivo de cabecera

Se abre la ventana para buscar la ruta en la que se encuentra la carpeta Led que se ha creado anteriormente y además se le da nombre al archivo de cabecera, que también se llamará Led:

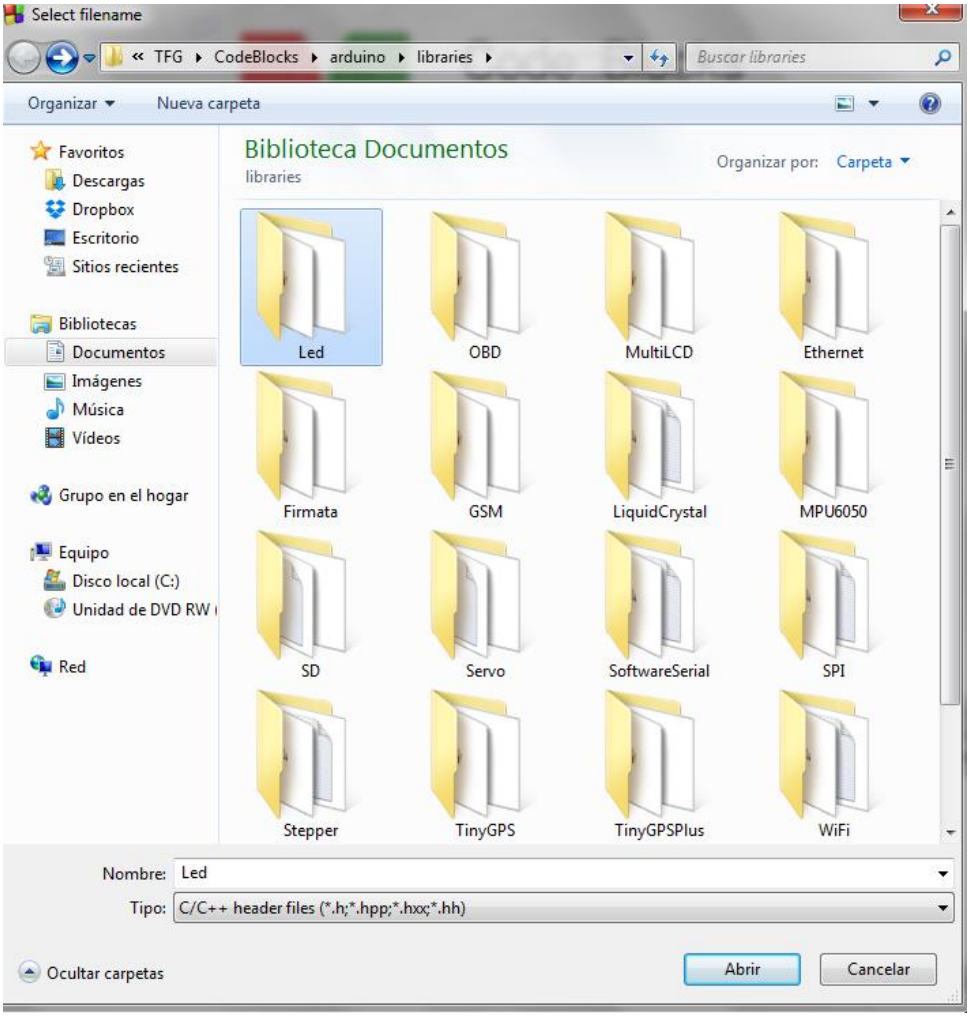

Figura 30: Selección de la ruta y el nombre del archivo de cabecera

Tras guardar los cambios se pulsa Finish. Automáticamente se cierra el asistente y en la ventana de edición del interfaz de Code::Blocks Arduino IDE aparece el código correspondiente al archivo Led.h:

```
1  #ifndef LED_H_INCLUDED
2  #define LED_H_INCLUDED
3
4
5
6  #endif // LED_H_INCLUDED
```

Figura 31: Código base del archivo de cabecera Led.h

A partir de este código base, se desarrolla el archivo de cabecera, en el que deben ir incluidas las declaraciones de las funciones que formarán la librería. Tras redactar el código se debe guardar pero no compilar:

Start here | Led.h

```
1  #ifndef LED_H_INCLUDED
2  #define LED_H_INCLUDED
3
4  #include "Arduino.h"
5
6  /* Declaración de las cabeceras de las funciones
7  que forman parte de la librería*/
8
9
10 //Función para definir los pines de los LED
11
12 void leds();
13
14 //Función para encender un LED concreto
15
16 void LedOn(int led);
17
18 //Función para apagar un LED concreto
19
20 void LedOff(int led);
21
22 //Función de incremento y decremento del brillo de un led.
23
24 void brillo(char color);
25
26 //Función que indica si el interruptor está pulsado
27
28 int interruptor();
29
30 //Función para modificar los colores del RGB
31
32 void RGB(int R, int G, int B);
33
34 //Función que devuelve el valor leido por el sensor LDR
35
36 int LDR ();
```

Figura 32: Archivo de cabecera "Led.h"

Una vez guardado el archivo de cabecera y sin compilar, se procede a realizar el archivo fuente, donde irán incluidas las instrucciones de las funciones que se han declarado en el archivo de cabecera.

- **Archivo fuente Led.cpp**

Siguiendo la ruta mostrada anteriormente: File →New → File

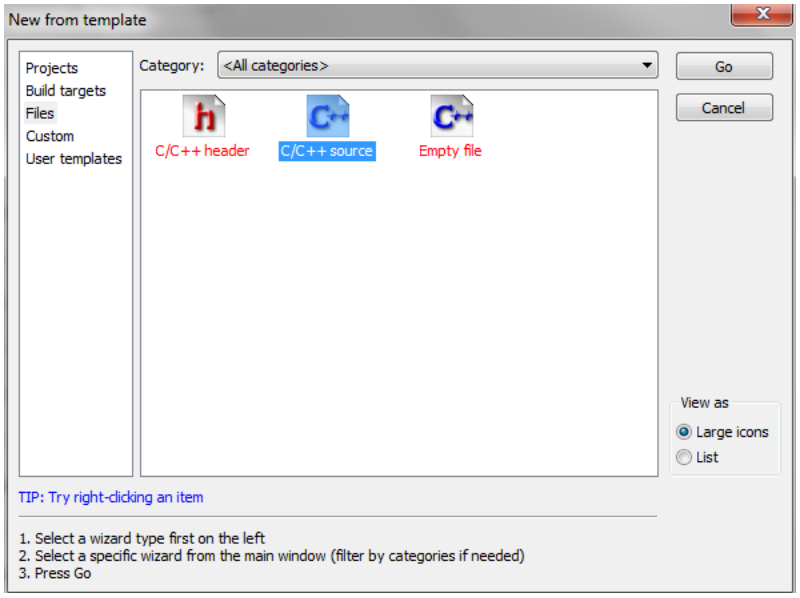

Figura 33:Selección de nuevo archivo fuente

Al seleccinar C/C++source, se abrirá el asistente para la configuración del archivo fuente:

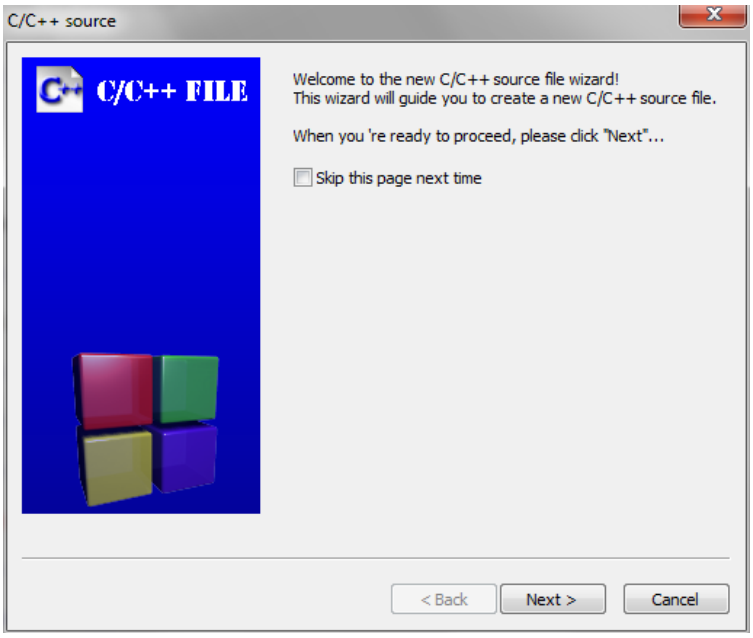

Figura 34: Asistente para la configuración del archivo fuente

Al pulsar Next se pasa a la configuración del tipo de lenguaje en que se va a realizar el archivo:

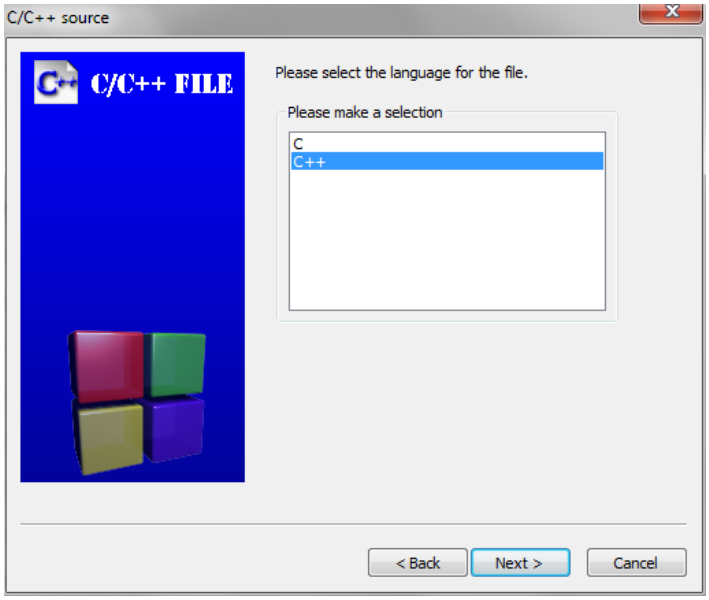

Figura 35: Selección del lenguaje del archivo fuente

Se selecciona el lenguaje C++ y se continúa con el asistente pulsando Next:

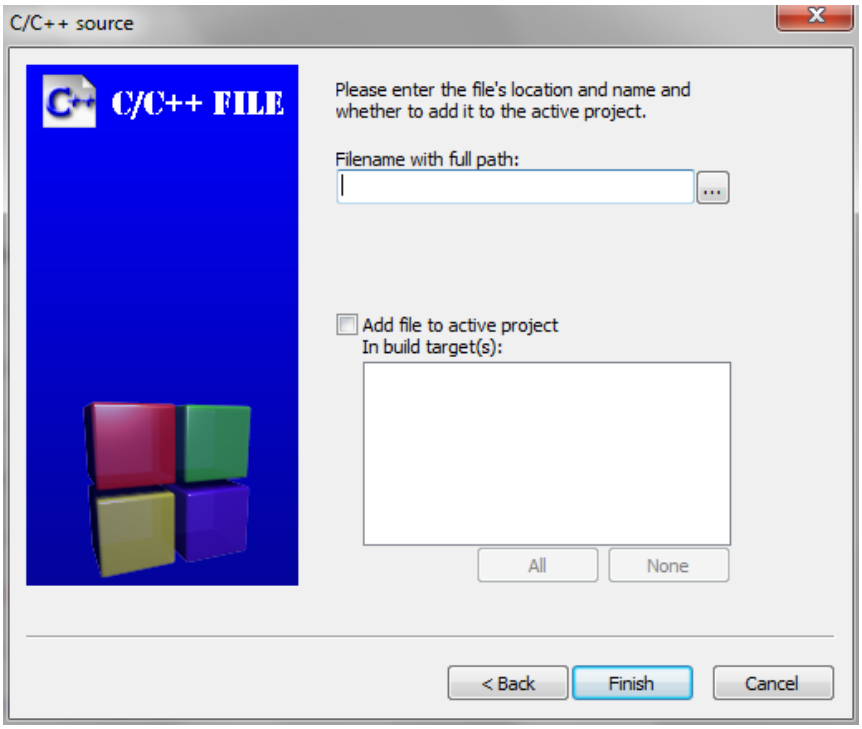

Figura 36: Configuración del nombre y la ruta del archivo fuente

Como con el archivo de cabecera, se debe buscar la ruta donde se ha creado la carpeta Led dentro de la carpeta Libraries. En este caso el archivo también será llamado Led:

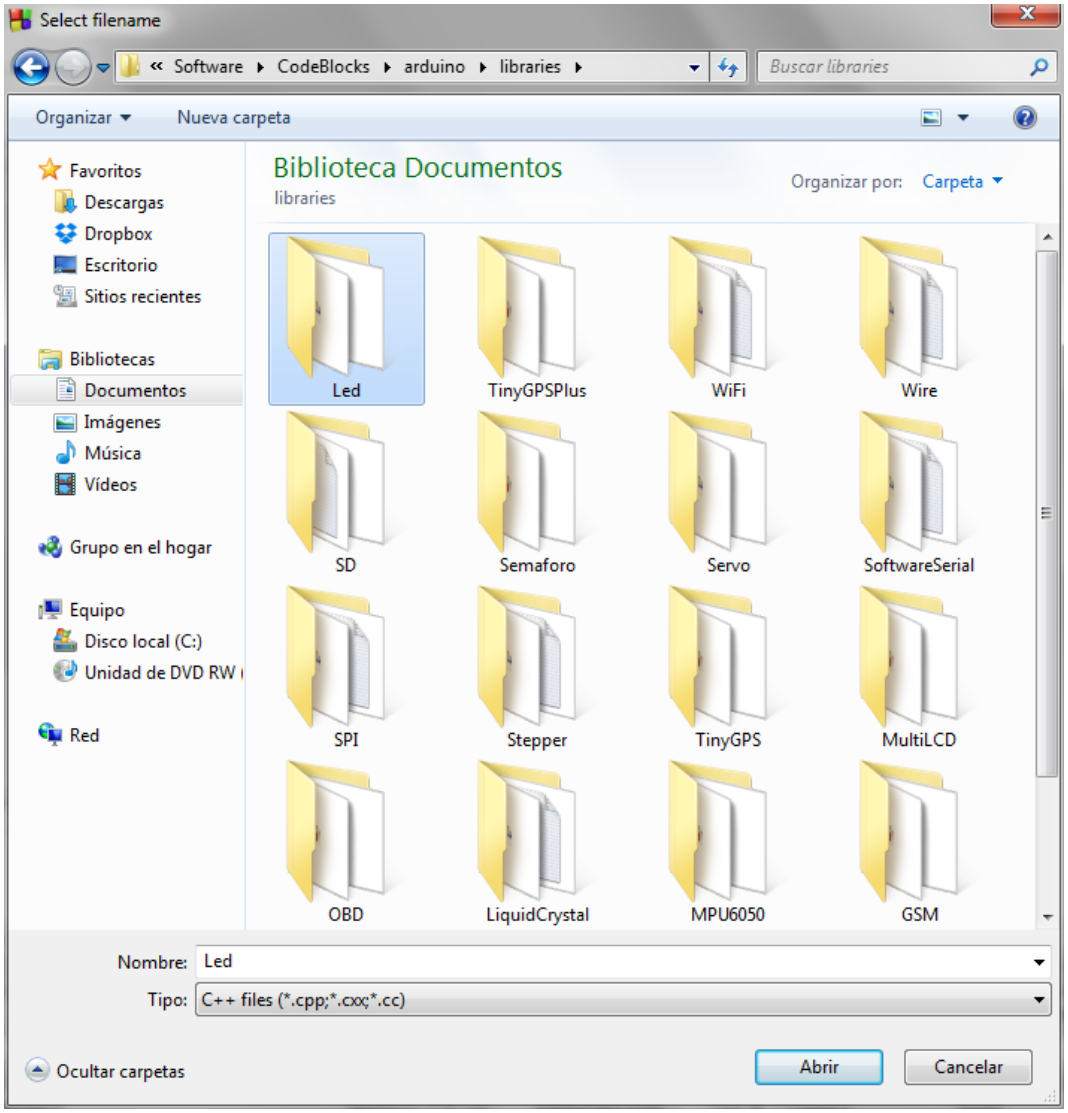

Figura 37: Selección de la ruta y el nombre del archivo fuente

Tras guardar los cambios y pulsar en Finish, el asistente se cerrará y en la ventana de edición de Code::Blocks Arduino IDE se habrá creado una nueva pestaña correspondiente a Led.cpp. Al contrario que en el archivo de cabecera, este archivo está inicialmente vacío.

El primer paso es escribir las directivas de preprocesador para incluir los archivos de cabecera de las librerías Arduino y Led. A partir de aquí se irán desarrollando las instrucciones de las funciones definidas en el archivo de cabecera Led.h:

Start here | Led.h | Led.cpp

```
1  #include "Arduino.h"
2  #include "Led.h"
3
4
5  int leer_serial()
6  {
7          int dato=0;
8          if (Serial.available()>0)
9          {
10             dato=Serial.read();
11         }
12         return(dato);
13         }
14
15 //Función que define los pines de los LED
16 void leds()
17 {
18     for(int j=2; j<15; j++)
19         pinMode(j, OUTPUT);
20 }
21
22 //Función que enciende un LED específico
23 void LedOn (int pin)
24 {
25         leds();
26         digitalWrite(pin, HIGH);
27 }
28
29 //Función que apaga un LED específico
30
31 void LedOff (int pin)
32 {
33     leds();
34     digitalWrite(pin, LOW);
35 }
36
```

Figura 38: Archivo fuente Led.cpp

Tras redactar todo el código se guarda el archivo sin compilarlo.

4.2.5. INCORPORACIÓN DE UNA LIBRERÍA PROPIA EN CODE::BLOCKS ARDUINO IDE

Hasta ahora se ha creado una librería nueva que se ha guardado en la carpeta donde se encuentran las librerías incluidas por defecto en el entorno de desarrollo. El hecho de guardar en esa ruta la nueva librería no implica que

pueda utilizarse en un nuevo proyecto. Para poder hacer uso de la librería es necesario pasar por un proceso un tanto complejo en el que se debe copiar el archivo de cabecera .h y el archivo fuente .cpp. Para realizar esta copia en un nuevo proyecto es necesario colocarse sobre el nombre del proyecto que estará en la parte izquierda del interfaz de Code__Blocks Arduino IDE, pulsar el botón derecho del ratón y abrir la opción Add files… Se seleccionarán los archivos de cabecera .h y fuente .cpp correspondientes a la librería que se quiere utilizar. Este procedimiento debería repetirse en cada proyecto nuevo que se quiera usar esta librería.

Este método resulta tedioso y un tanto arcaico en el caso de realizar varios proyectos nuevos, por lo que se concluye que la mejor opción es modificar el archivo default.conf de Code::Blocks Arudino IDE. Esta modificación es algo más compleja que añadir archivos como en el caso anterior, pero es una solución definitiva que sólo habrá de realizarse una vez.

Se basa en modificar el software del programa y hacer que la librería Led.h forme parte de la configuración, pudiéndose usar en cada nuevo proyecto sin necesidad de realizar ninguna modificación.

En primer lugar se busca el archivo default.conf en la carpeta de Code::Blocks Arduino IDE

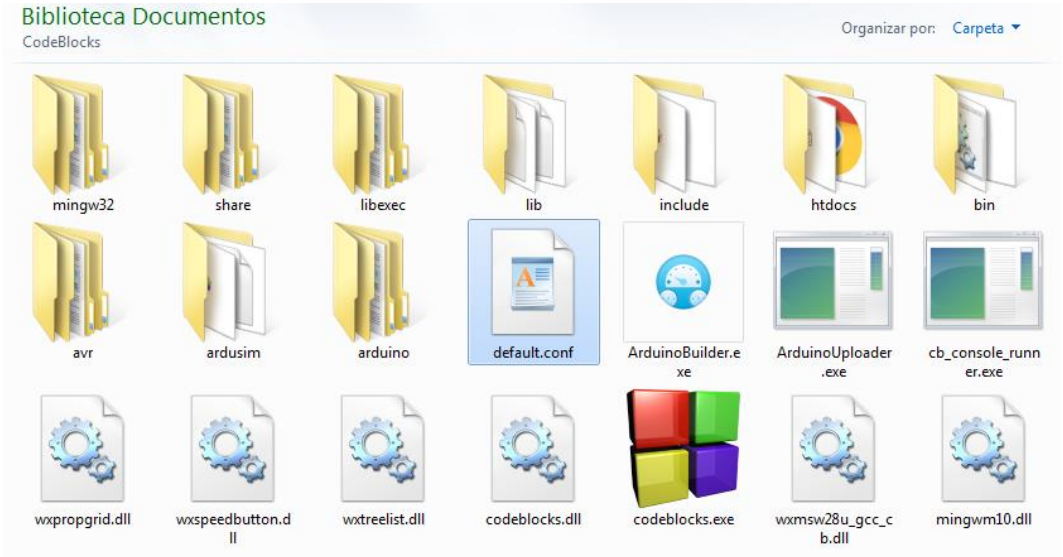

Figura 39: Localización del archivo default.conf en la carpeta de archivos de Code::Blocks Arduino IDE

Dado que el archivo tiene extensión .conf, el sistema operativo no va a reconocerlo al intentar abrirlo, por lo que se selecciona el programa WordPad:

Se abrirá un archivo de texto con un código bastante largo. Interesa encontrar las variables de aplicación en las que se encuentran definidas las rutas a las

librerías. Para facilitar la localización se usará la opción de búsqueda de WordPad y se tratará de encontrar la palabra Ethernet

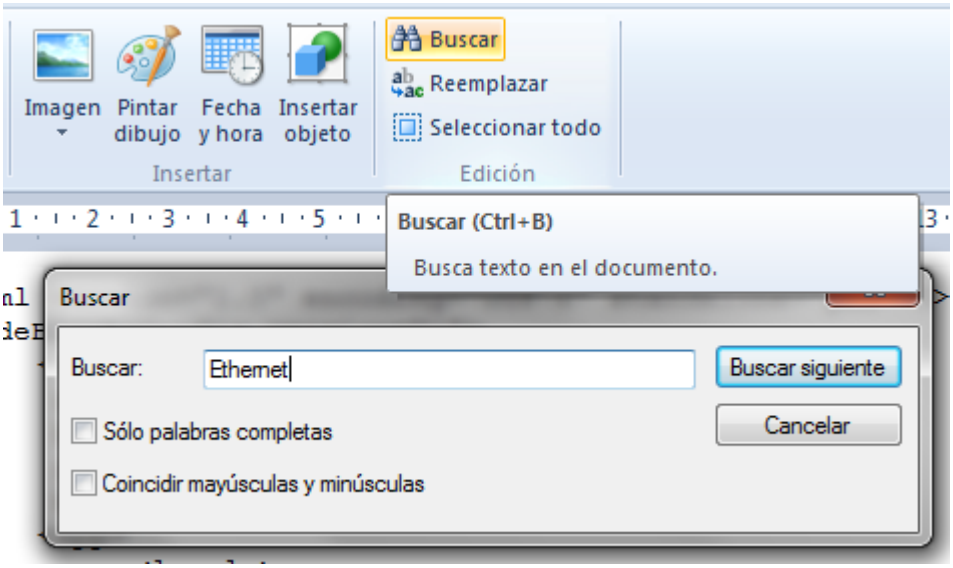

Figura 40: Búsqueda de la palabra Ethernet en WordPad

Una vez localizado el punto donde se encuentra la palabra Ethernet hay que desplazarse por el código hasta localizar todas las variables APP_PATH correspondientes a las distintas librerías para Arduino que forman parte por defecto del entorno de desarrollo.

```
                    <INCLUDE_DIRS>
                        <str>
                            <![CDATA[.;$(APP_PATH)\arduino
\libraries\EEPROM;$(APP_PATH)\arduino\libraries\Esplora;
$(APP_PATH)\arduino\libraries\Ethernet;$(APP_PATH)\arduino
\libraries\Ethernet\utility;$(APP_PATH)\arduino\libraries
\Firmata;$(APP_PATH)\arduino\libraries\LiquidCrystal;
$(APP_PATH)\arduino\libraries\OBD;$(APP_PATH)\arduino
\libraries\SD;$(APP_PATH)\arduino\libraries\SD\utility;
$(APP_PATH)\arduino\libraries\Servo;$(APP_PATH)\arduino
\libraries\SoftwareSerial;$(APP_PATH)\arduino\libraries\SPI;
$(APP_PATH)\arduino\libraries\Stepper;$(APP_PATH)\arduino
\libraries\TinyGPS;$(APP_PATH)\arduino\libraries\WiFi;
$(APP_PATH)\arduino\libraries\WiFi\utility;$(APP_PATH)\arduino
\libraries\Wire;$(APP_PATH)\arduino\libraries\Wire\utility;
$(APP_PATH)\arduino\libraries\MultiLCD;$(APP_PATH)\arduino
\libraries\MPU6050;$(APP_PATH)\arduino\libraries\UTFT;
$(APP_PATH)\arduino\libraries\GSMC:\WinAVR\avr\include;]]>
                        </str>
                    </INCLUDE_DIRS>
                    <LIBRARY_DIRS>
                        <str>
                            <![CDATA[C:\WinAVR\avr\lib;]]>
                        </str>
                    </LIBRARY_DIRS>
                    <LIBRARIES>
```

Figura 41: Localización de las variables APP_PATH

Observando el código se aprecia que las variables APP_PATH vienen definidas por la ruta de acceso a la librería corrspondiente:

$(APP_PATH)\arduino\libraries\nombre_librería

Para incluir una librería propia simplemente se añade la variable APP_PATH correspondiente a la misma:

```
                    </str>
                </LINKER_OPTIONS>
                <INCLUDE_DIRS>
                    <str>
                        <![CDATA[.;$(APP_PATH)\arduino
\libraries\EEPROM;$(APP_PATH)\arduino\libraries\Esplora;
$(APP_PATH)\arduino\libraries\Ethernet;$(APP_PATH)\arduino
\libraries\Ethernet\utility;$(APP_PATH)\arduino\libraries
\Firmata;$(APP_PATH)\arduino\libraries\LiquidCrystal;
$(APP_PATH)\arduino\libraries\OBD;$(APP_PATH)\arduino
\libraries\SD;$(APP_PATH)\arduino\libraries\SD\utility;
$(APP_PATH)\arduino\libraries\Servo;$(APP_PATH)\arduino
\libraries\SoftwareSerial;$(APP_PATH)\arduino\libraries\SPI;
$(APP_PATH)\arduino\libraries\Stepper;$(APP_PATH)\arduino
\libraries\TinyGPS;$(APP_PATH)\arduino\libraries\WiFi;
$(APP_PATH)\arduino\libraries\WiFi\utility;$(APP_PATH)\arduino
\libraries\Wire;$(APP_PATH)\arduino\libraries\Wire\utility;
$(APP_PATH)\arduino\libraries\MultiLCD;$(APP_PATH)\arduino
\libraries\MPU6050;$(APP_PATH)\arduino\libraries\UTFT;
$(APP_PATH)\arduino\libraries\GSM;$(APP_PATH)\arduino
\libraries\Led;C:\WinAVR\avr\include;]]>
                    </str>
```

Figura 42: Inclusión de la librería Led en el archivo default.conf

Se guardan los cambios del archivos y se comprueba que el archivo default ahora aparece sin la extensión .conf. Éste es el proceso a seguir para añadir cada nueva librería. Es un tanto complejo pero no está pensado para que lo realice el alumno. Se espera que el software sea preparado antes que el estudiante comience a utilizarlo y puesto que este proceso solo es necesario realizarlo una única vez se concluye que es la mejor opción para añadir una nueva librería.

4.3. ARDUINO BUILDER

Del creador del entorno Code::Blocks Arduino IDE, Stanley Huang, se trata de una aplicación mediante la que se puede compilar el código fuente y cargar el código compilado (archivo .HEX) en la placa Arduino.

Esta aplicación permite compilar, cargar y ejecutar un programa en tres sencillos pasos:

- Selección del archivo.
- Selección del tipo de tarjeta Arduino que se va a usar.
- Selección en el puerto serie en el que se ha conectado la tarjeta.

Al ejecutar el Arduino Builder, se encuentra la siguiente interfaz:

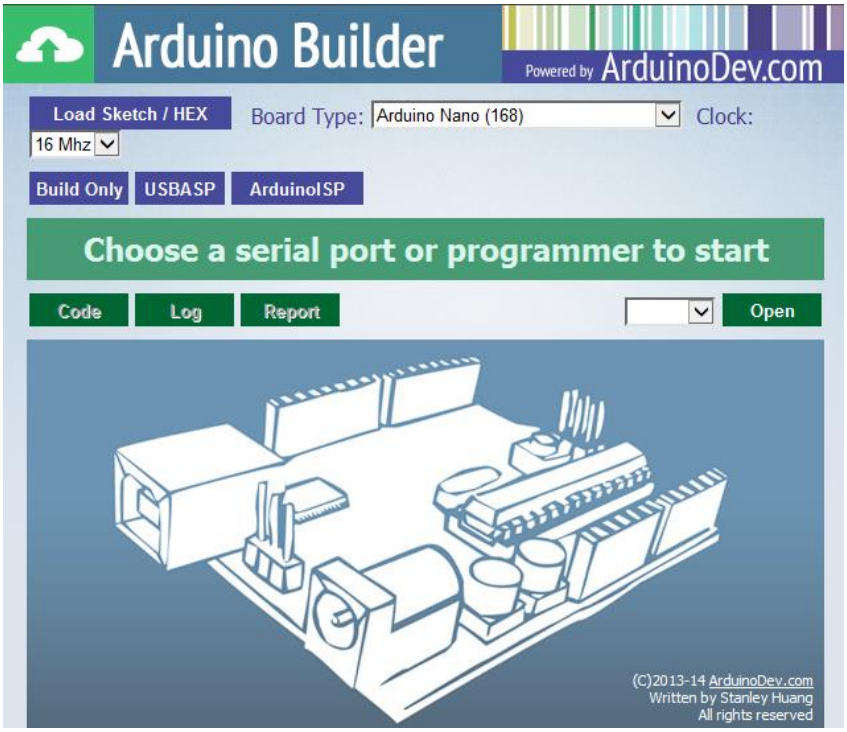

Figura 43: Interfaz Arduino Builder

Las opciones que se pueden modificar son:

- Frecuencia de funcionamiento o velocidad de reloj. En este caso, este valor es el de la propia placa de Arduino, es decir, 16MHz.
- Tipo de placa con que se va a trabajar. En este caso se usará Arduino NANO.
- Load Sketch/HEX: Permite abrir un archivo, sin compilar o compilado. En caso de que el archivo no esté compilado, el propio programa lo compilará generando el archivo HEX. Es importante tener en cuenta que en toda la ruta de carpetas donde se encuentra el archivo, no puede existir ninguna carpeta en cuyo nombre exista un espacio, dado que creará un error a la hora de generar el archivo .HEX
- Permite seleccionar la velocidad de transmisión a la que se comunica con el puerto serie. Dado que en Arduino se trabaja con la velocidad 9600, se elegirá está misma y al pulsar open se abrirá la ventana que permite la comunicación con la placa.

Una vez seleccionados el tipo de placa a usar y la frecuencia de trabajo, se procede a cargar un código compilado. Para ello se pincha en la pestaña Sketch/HEX. Aparecerá una ventana emergente en la que se buscará la ruta de acceso al archivo que deseamos cargar. Este archivo debe tener la extensión .ino o en su defecto .hex.

Es necesario trabajar con la placa conectada al ordenador. Si el ordenador no puede instalar el controlador del dispositivo, es necesario descargar los drivers. El enlace para la descarga se puede consultar en la bibliografía del trabajo [7]. En la página aparecen dos enlaces, es necesario seleccionar el del CHIP-CH340, ya que es el que tiene la tarjeta de Arduino con la que se va a trabajar. Una vez descargado, se descomprime y se abre el administrador de dispositivos. En el dispositivo llamado USB2.0-Serial se pulsa el botón derecho y se selecciona la opción Actualizar software de controlador… y se selecciona buscar software de controlador en el equipo. Se abrirá una ventana emergente en la que hay que seleccionar la ruta donde se ha descomprimido el archivo. Es importante dejar seleccionado el archivo CH341SER. Cuando el proceso haya finalizado, se abrirá una ventana con el mensaje de que el controlador del dispositivo se ha instalado correctamente. En este punto ya se puede usar la tarjeta Arduino NANO con total normalidad.

Al cargar el archivo aparecerá el código del programa en la interfaz de Arduino Builder:

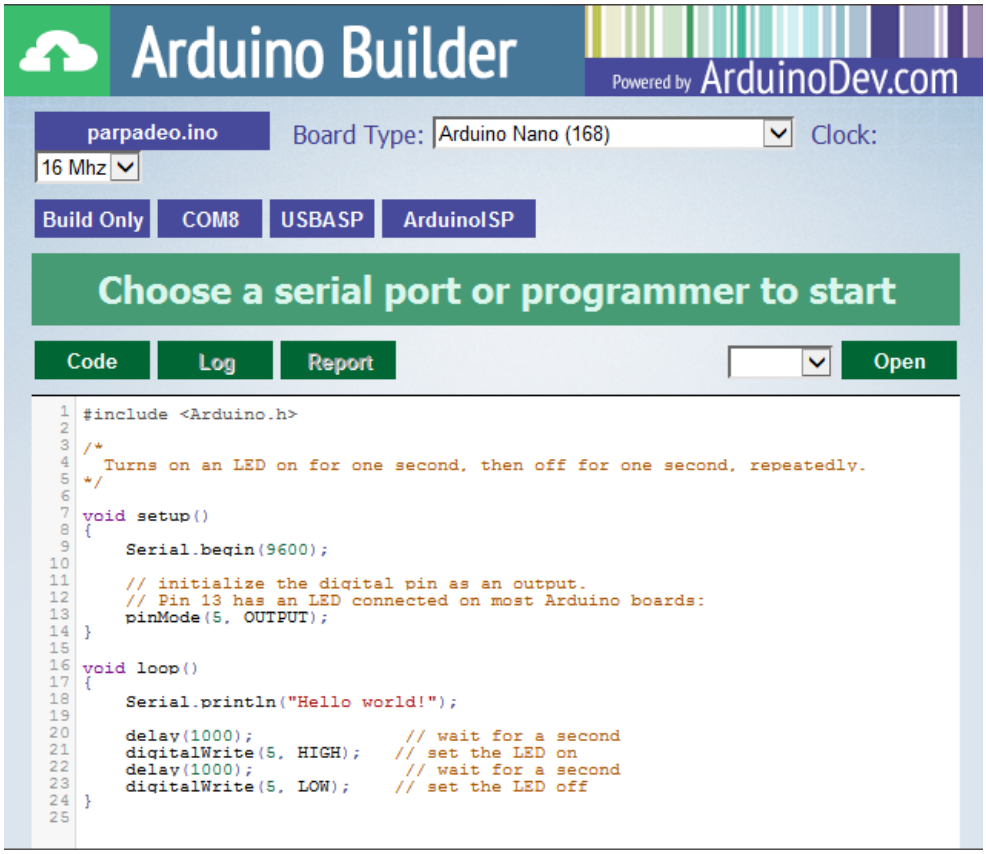

Figura 44: Interfaz de Arduino Builder al cargar un proyecto

Una vez se ha cargado con éxito el proyecto, se procede a cargarle a la placa. Para ello se pincha en la pestaña COM8, que es el puerto serie donde se ha conectado la placa. En primer lugar se realizará un proceso de compilación del código cargado y más tarde se cargará el código en la placa. Para ver la evolución de este proceso, se accede a la pestaña Log:

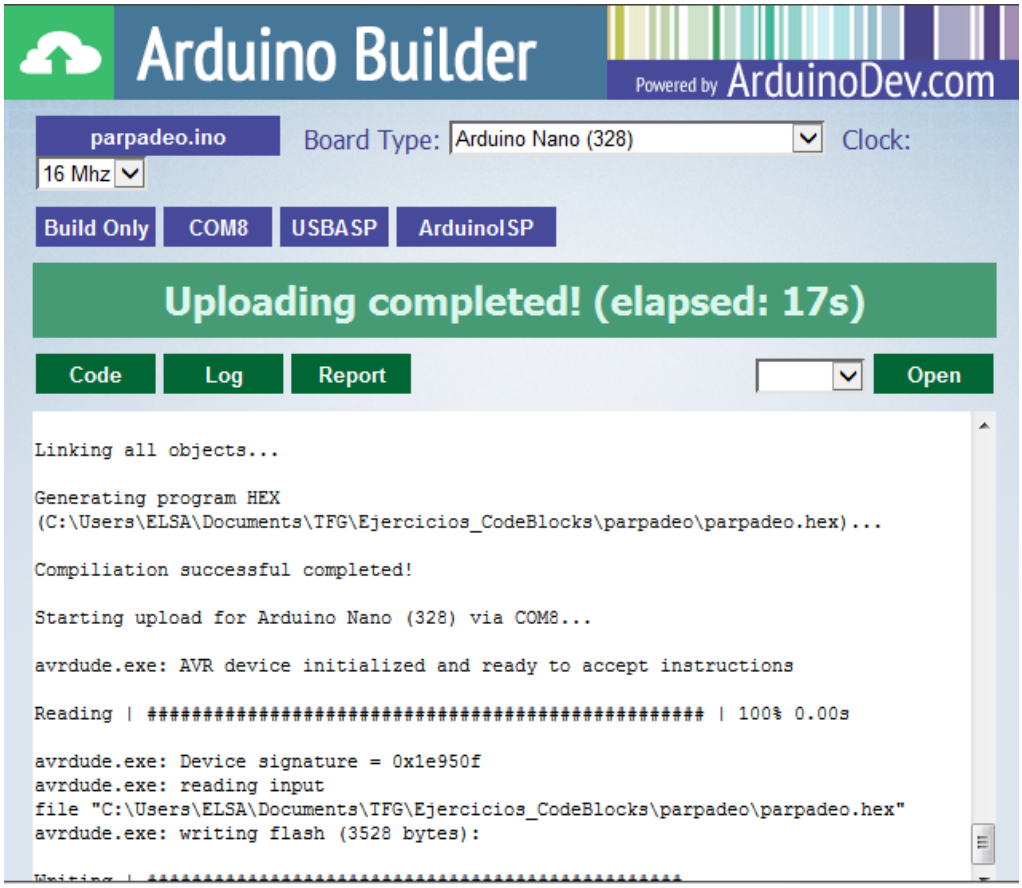

Figura 45: Vista de la pestaña Log de Arduino Builder

En esta vista se aprecia que el archivo se ha compilado y cargado correctamente, además del tiempo que se ha tardado en realizar el proceso.

Para obtener una visión general del uso de las memorias FLASH, SRMA y EEPROM, se accede a la pestaña Report:

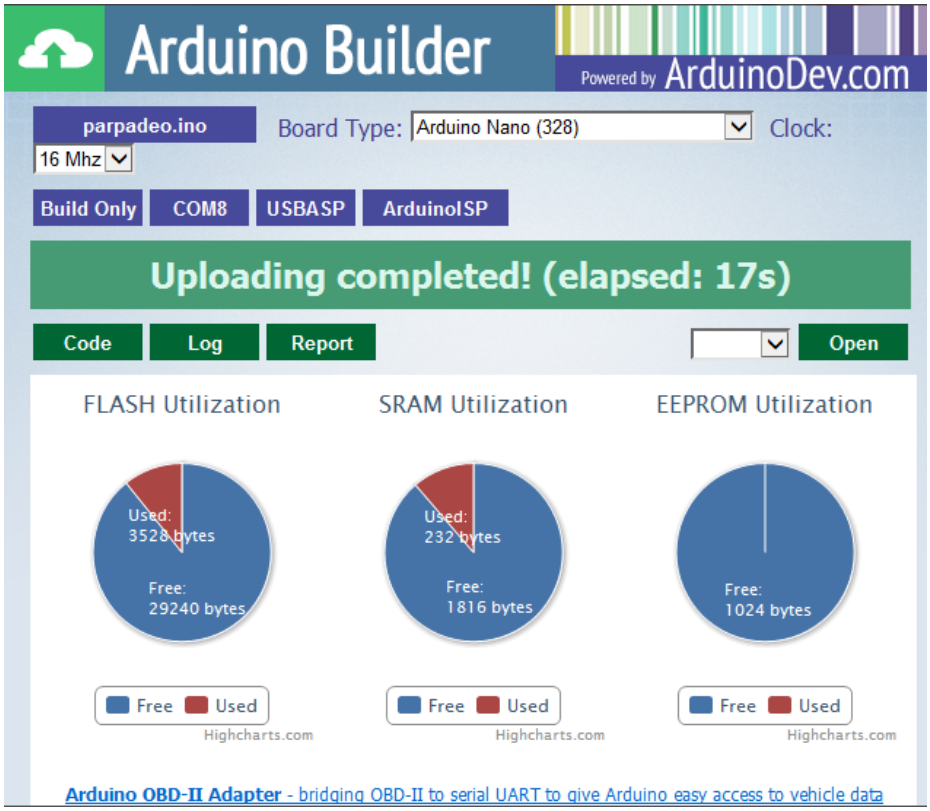

Figura 46: Vista de la pestaña Report de Arduino Builder

Es posible acceder al puerto serie seleccionando la velocidad de transmisión 9600:

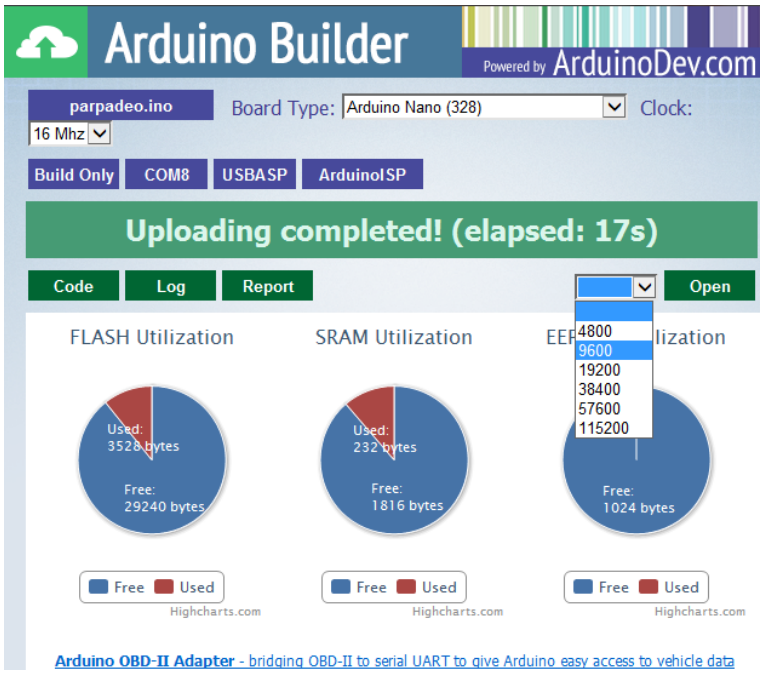

Figura 47: Selección de la velocidad de transmisión de Ardiuno Builder

En el código que se ha cargado, se solicita que saque por el puerto serie la frase Hello World. Al pinchar en open con esa velocidad de transmisión de obtiene:

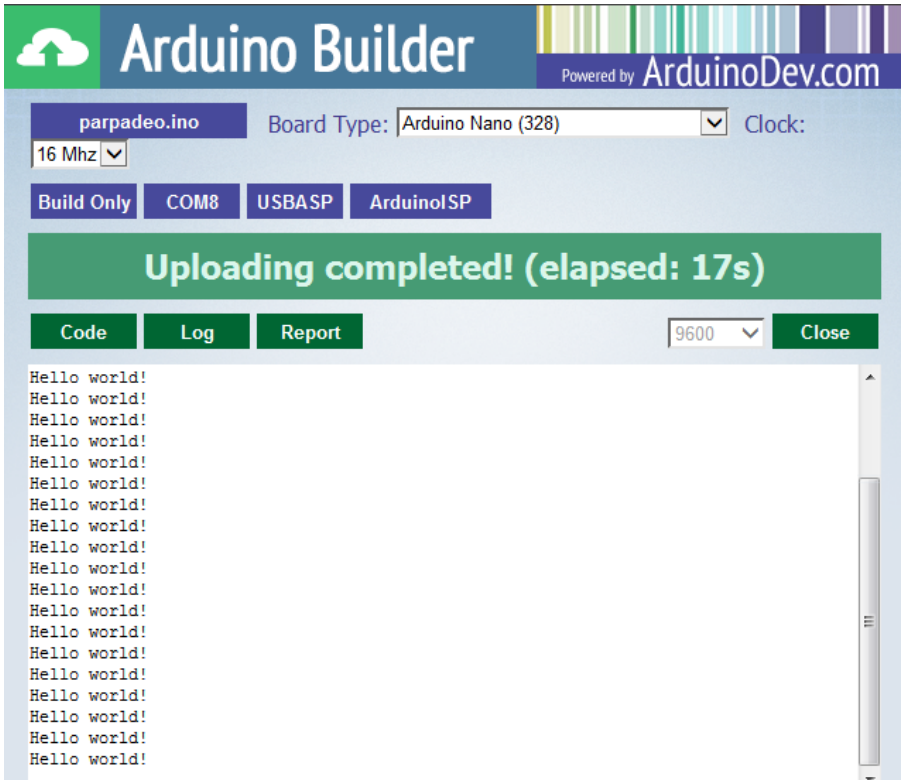

Figura 48: Monitor serie de Arduino Builder

4.3.1. Uso de Arduino Builder con Code::Blocks Arduino IDE

Hasta este momento se ha estudiado el funcionamiento de Code::Blocks Arduino IDE, se han definido los pasos a seguir para crear una nueva librería de funciones y para incluirla en el software del entorno de desarrollo y se ha visto el funcionamiento de la aplicación Arduino Builder. En este punto se estudiará el uso conjunto de ambos programas y cómo debe proceder el alumno para cargar un nuevo proyecto desarrollado en Code::Blocsk Arduino IDE a la placa Arduino mediante el entorno Arduino Builder.

En primer lugar es necesario conectar la placa al ordenador, esto se realiza mediante un cable USB. El ordenador reconoce el hardware y al abrir la aplicación Arduino Builder aparece el COM en el que se ha conectado la placa. En el caso de que el ordenador no reconozca el dispositivo, es necesario descargar el controlador del dispositivo siguiendo los pasos especificados en el apartado 4.3 Arduino Builder del presente trabajo.

El alumno debe crear una nueva carpeta donde almacenar sus nuevos proyectos en el equipo de trabajo. De esta manera, al crear un nuevo proyecto en Code::Blocks Arduino IDE, debe especificar la ruta de esa nueva carpeta para guardar el nuevo sketch.

Para ilustrar los pasos a seguir en este punto, se va a desarrollar un programa sencillo en el que se encenderá un LED a través de las instrucciones creadas en la librería de funciones Led.h

Encender_LED.ino

```
1   #include <Arduino.h>
2   #include <Led.h>
3
4   void setup()
5   {
6       Serial.begin(9600);
7
8   }
9
10  void loop()
11  {
12      LedOn(1);
13  }
14
```

Figura 49: Código del programa Encender_LED

Con la placa conectada al equipo se compila el programa para asegurarse de que no hay ningún error en el código. Para compilar el programa es necesario pulsar la tecla Build  . En la ventana inferior seleccionando la pestaña Build log, se aprecia el progreso de la compilación y, una vez ha finalizado, indicará si hay errores o si el programa se ha compilado satisfactoriamente.

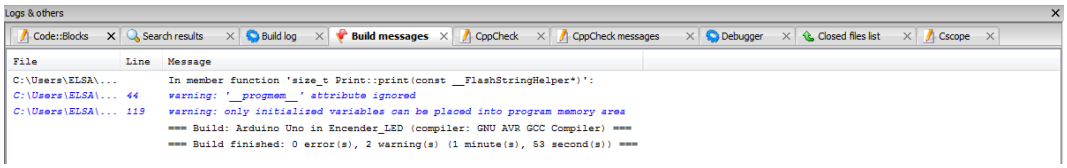

Figura 50: Compilación del programa Encender_LED

Tras comprobar que la compilación ha finalizado con éxito, se procede a ejecutar el programa. Antes de ejecutar el programa, es necesario seleccionar el tipo de placa usada ya que una vez abierto Arduino Builder no es posible modificarla. En este caso el tipo de placa es Arduino NANO (328) y se selecciona en la barra de herramientas de Code::Blocks Arduino IDE. Para ejecutar el sketch se pulsará la tecla Run  . Al pulsar esta tecla se abrirá

automáticamente el entorno Arduino Builder. Debe tenerse en cuenta que sólo es posible tener abierta una única instancia de Arduino Builder, por lo que antes de ejecutar un sketch es necesario revisar que Arduino Builder esté cerrado.

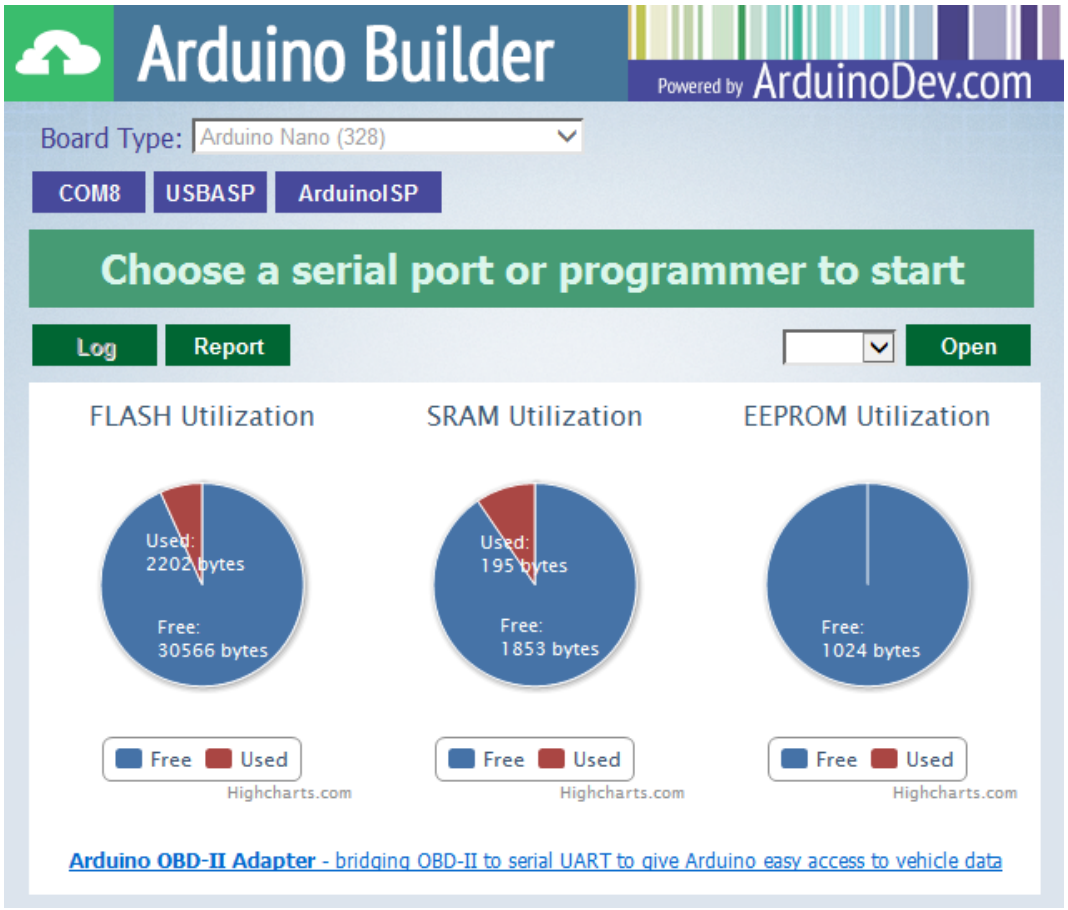

Figura 51: Arduino Builder desde Code::Blocks Arduino IDE

En este caso, como se aprecia en la figura 51, el puerto al que está conectado la placa Arduino se corresponde con el COM8. Seleccionando la pestaña del COM, el programa procederá a compilar el código y a generar el archivo de extensión HEX.

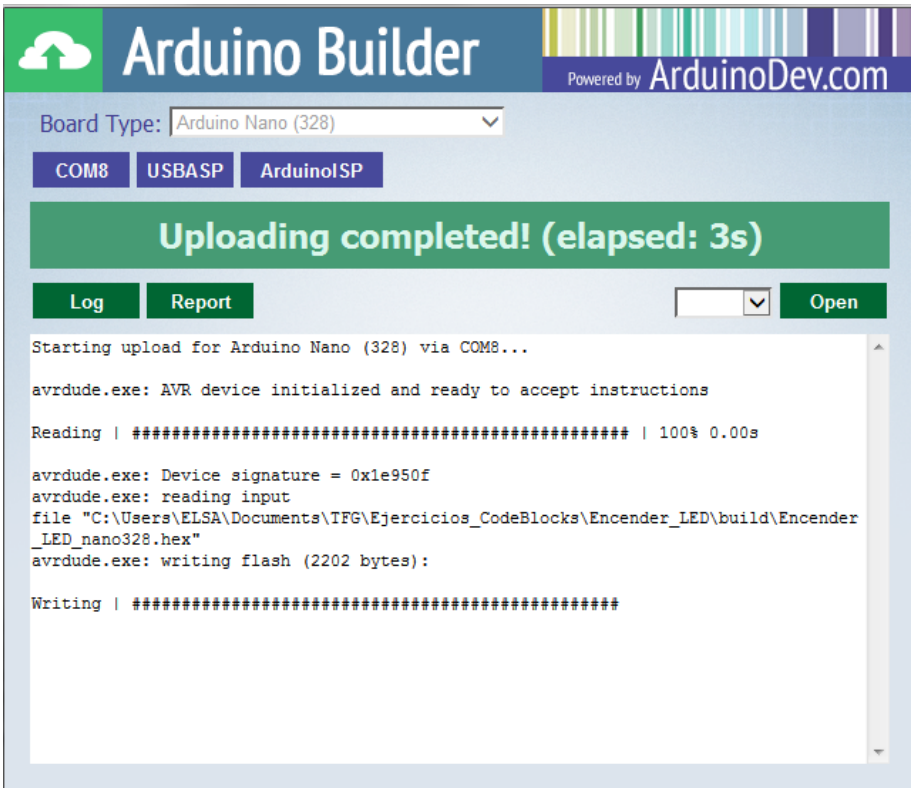

Figura 52: Resultado de cargar el código a la placa

La carga del código a la placa se ha realizado satisfactoriamente, se comprueba que el LED correspondiente al pin 1 se ha encendido. Este LED se mantendrá encendido siempre que la placa esté conectada a la alimentación.

Figura 53: Resultado del código Encender_LED

.

CAPÍTULO 5. HARDWARE NECESARIO

5.1. MATERIAL NECESARIO

Teniendo en cuenta las limitaciones presentes, se ha procurado usar el mayor número de elementos para ofrecer mayor flexibilidad a la hora de realizar problemas de programación. La placa Arduino NANO tiene 14 pines digitales y 8 analógicos. Una importante ventaja de esta placa es que los pines analógicos se pueden utilizar como digitales si se les nombra como pin 14 en lugar de A0, 15 en lugar de A1…

El número de dispositivos usados se ha elegido en función del número de pines disponibles, y éstos son:

- 13 LED (pin digital)
- 1 LED RGB (3 pines digitales PWM)
- 1 sensor LDR (pin analógico)
- 1 zumbador (pin digital)
- 1 botonera (pin analógico)
- 1 Motor SG90 (pin digital PWM)
- 1 placa Arduino NANO
- 1 placa Shield para Arduino NANO

5.2. CONOCIMIENTOS PREVIOS

5.2.1. DATOS ANALÓGICOS Y DIGITALES

Los datos utilizados en programación pueden ser constantes o variables. Las constantes mantienen siempre el mismo valor mientras que las variables cambian, bien por ser una secuencia de números o bien porque es un dato que modifica el usuario.

Dentro de este campo se trabaja con datos analógicos y satos digitales. Los datos analógicos pueden tomar cualquier valor, es decir, de cero a infinito. Sin embargo, los datos binarios sólo pueden tomar dos valores: verdadero o falso, es decir, uno o cero.

La plataforma Arduino cuenta con señales PWM, es decir, señales digitales que se modulan en un número finito de valores. Dado que trabaja con 8 bits, trabaja con 256 valores, es decir, de cero a 255.

5.2.2. ENTRADAS Y SALIDAS

En primer lugar se va a diferenciar las entradas y salidas de funciones propias de programación y las entradas y salidas de Arduino.

Cuando se habla de funciones, una entrada es un valor que se introduce en una función de manera que, al ejecutar esa función, se utilizará ése valor. Las funciones pueden tener una o varias entradas. Por ejemplo, si se crea una función que realiza la suma de dos números, las entradas de esa función serán los dos números que se desean sumar. Cuando se llame a la función introduciendo como parámetro de entrada los dos números que queremos sumar, la función realizará los cálculos y generará la solución:

int suma (int x, int y)

Donde x e y son las entradas, es decir, los números que se quieren sumar.

La salida de una función es el resultado de la misma, y que se devuelve al programa principal mediante la función return():

```
Int suma (int x, int y);
{
        suma=x+y;
        return (suma);
}
```

Cuando se trata de Arduino, las entradas son valores que se reciben mediante un sensor externo. Estos valores pueden ser analógicos o digitales y se pueden leer tantas veces como se quiera. Por ejemplo, usando un sensor LDR se puede obtener el nivel de luz que incide en el sensor. Le entrada será, por tanto, el valor leído por el sensor y que se puede ver directamente en la pantalla. Como se ha mencionado se puede leer éste valor tantas veces como se quiera, en los intervalos que se desee: cada segundo, cada minuto…

La salida en Arduino es el valor que se desea que tome el dispositivo. Por ejemplo, en el caso de un LED, éste estará encendido o apagado, es decir, valdrá uno o cero. Si se pretende que el diodo esté encendido, la salida será 1, mientras que si se pretende que esté apagado, la salida será cero.

5.2.3. DIODO LED

Un diodo LED (Light Emitting Diode) solo permite el paso de corriente en un sentido, mientras que si pasa por el sentido contrario funciona como un interruptor abierto (no deja pasar la corriente). Cuando la corriente pasa en el sentido correcto, se polariza directamente. Cuando está polarizado en directa, el diodo emite luz.

Dado que el diodo está apagado o está encendido, se considera como variable digital, pudiendo adoptar el valor uno o cero (true o false). Sin embargo es posible, además de encender y apagar, modificar la intensidad de un LED. Gracias a los pines PWM de los que dispone Arduino, se puede regular la intensidad de la luz que emite el diodo en 256 valores, siendo el 0 el valor en el que el LED se encuentra apagado y el 255 el valor en el que tiene máxima intensidad (está totalmente encendido).

Dado que la corriente sólo pasa en un sentido, es importante tener en cuenta la polaridad a la hora de conectar el diodo. Por lo general tienen una patilla más larga, la cual se corresponde al positivo, mientras que la corta se conectará a tierra (GND)

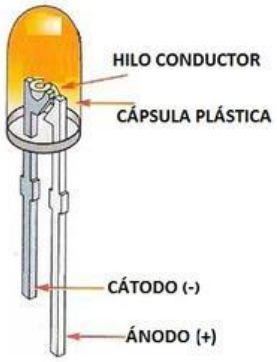

Figura 54: Diodo LED

5.2.4. LED RGB

El LED RGB (Red Green Blue) es un tipo especial de LED que consta de tres colores. El RGB tiene cuatro patillas, una de ellas se conecta a tierra y las otras tres, correspondientes a cada uno de los colores, se conectan a pines PWM. En principio se puede configurar cada una de las patillas como true o false, de manera que cuando una patilla esté activa, el RGB lucirá de ese color. El motivo por el que se conecta a pines PWM es el de poder conseguir

mezclas de colores, ya que con la combinación de los colores primarios se pueden obtener todos los colores:

Figura 55: Espectro de colores

Como se ha expresado anteriormente, las salidas PWM ofrecen una modulación de la señal de 256 valores, por lo que se configura cada color de 0 a 255. Si consideramos que el primer valor es para el color rojo, el segundo para verde y el tercero para el azul, las configuraciones serán:

COLOR	*CONFIGURACIÓN*
ROJO	*RGB (255, 0, 0)*
VERDE	*RGB(0, 255, 0)*
AZUL	*RGB (0, 0, 255)*
AMARILLO	*(255, 255, 0)*
BLANCO	*(255, 255, 255)*

Tabla 2: Configuración colores RGB

A continuación se muestra la imagen del RGB utilizado y el color correspondiente a cada patilla, así como la patilla que debe conectarse a tierra:

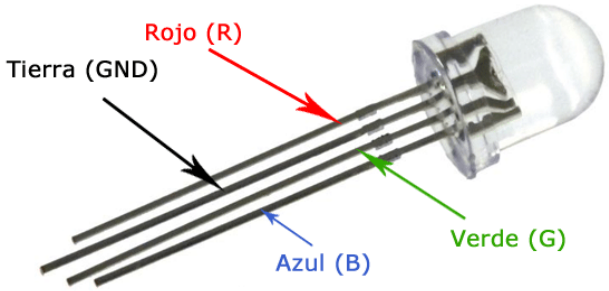

Figura 56: Diodo LED RGB

5.2.5. BOTONERA

En primer lugar se va a describir el funcionamiento de un interruptor simple. Cuando el interruptor no está pulsado, se comporta como un circuito abierto, es decir, no permite pasar la corriente. Cuando se pulsa el interruptor el circuito se cierra por lo que la corriente pasa permitiendo que el resto del circuito funcione correctamente. Por lo tanto, el estado del pulsador será abierto o cerrado (uno o cero). En base a esto se puede trabajar de dos maneras con un interruptor. Por un lado se puede ejecutar una función mientras el interruptor esté pulsado. Por ejemplo, si se pretende que un LED se encienda mientras el interruptor esté pulsado, se lee el estado del mismo y mientras sea 1 (interruptor cerrado) el LED brillará y cuando deje de pulsarse, éste se apagará. La otra manera de trabajar con un pulsador es hacer que se cumpla una función cuando se pulse el interruptor y deje de cumplirse cuando se vuelva a pulsar, es decir, sin necesidad de estar apretando el pulsador todo el tiempo. En este caso se va a trabajar con esta segunda opción.

Una botonera es un circuito con varios pulsadores conectados con resistencias de distintos valores. En estado de reposo todos los botones están sin pulsar, por lo que no se permite pasar corriente. Cuando uno de los interruptores se pulsa, permitirá que pase corriente por el mismo, y dado que cada uno está conectado a una resistencia de diferente valor, la tensión obtenida será diferente en cada caso.

Dado que se va a leer el valor de la tensión obtenido en cada caso, la botonera se conectará en un pin analógico, capaz de leer distintos valores.

En este caso se va a utilizar una botonera de la marca ElecFreaks. Se trata de un módulo con 5 botones con las resistencias internas, que permiten ahorrar en espacio y recursos, ya que no es necesario realizar el circuito.

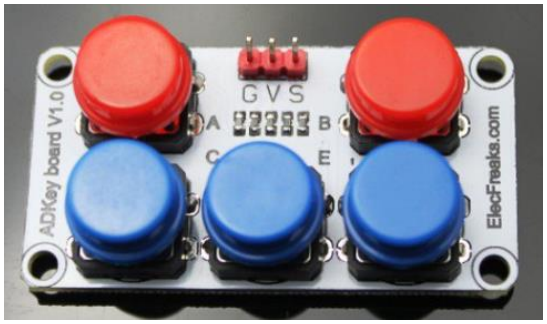

Figura 57: Botonera ElecFreaks

La botonera, como el resto de dispositivos, se va a situar en la tapa de una caja de manera que sean invisibles para los ojos y se vean solo los pulsadores. Esta botonera presenta el inconveniente de tener las conexiones en el mismo lado de los botones, impidiendo que se pueda ajustar bien a la tapa. La mejor opción para solucionar esto es doblar las patillas de conexión 90 grados en sentido exterior a la placa, de esta manera los botones saldrán de la tapa completamente mientras que los cables quedarán por dentro.

5.2.6. FOTORRESISTENCIA LDR

Una fotorresistencia es un componente electrónico cuya resistencia varía con la diferencia de intensidad de luz incidente. El valor de la resistencia eléctrica de un LDR es alto cuando hay luz incidiendo en él y bajo cuando está a oscuras. Al tratarse de una resistencia variable, el valor de la tensión leído en función de la luz incidente es diferente en cada caso. Dado que se van a leer valores diferentes, es necesario conectarlo en uno de los pines analógicos. La conexión en este caso es un poco más compleja ya que es necesario conectar una resistencia de Ω. El modo de conexión es una de las patillas del LDR conectada directamente a alimentación (5V) y la otra patilla conectada a uno de los extremos de la resistencia. El otro extremo de la resistencia irá conectado a tierra.

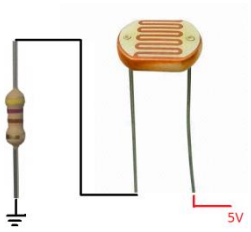

Figura 58: Conexión del LDR

5.2.7. SERVO MOTOR SG90

Los servos son motores de corriente continua que en lugar de generar un giro continuo están preparados para moverse en ángulo fijo en respuesta a una señal de control y mantenerse en dicha posición. Es por esto que son muy frecuentes en aeromodelismo y robótica, dado que su funcionamiento y control son muy precisos.

Un servo está formado por un conjunto reductor (engranajes), un motor de CC y un circuito de control. En la práctica se comporta como un bloque funcional que posiciona su eje en un ángulo preciso en función de la señal de control.

El servo debe conectarse a un pin PWM, correspondiendo los 256 valores en los que se puede modular a los 360 grados de giro que tiene el eje.

Se ha elegido el servo SG90 debido a su reducido tamaño, bajo precio, gran precisión y facilidad de conexión con la placa arduino.

Figura 59: Servo SG90

5.2.8. BUZZER (ZUMBADOR)

El funcionamiento de un buzzer se basa en la piezoelectricidad (presión-electricidad), es decir, en determinados materiales, al ejercer una fuerza mecánica, responden creando una corriente eléctrica y viceversa. Normalmente estos efectos son reversibles y cesan cuando se deja de ejercer la fuerza externa. Existen dos tipos de zumbadores, activos y pasivos. Los zumbadores activos funcionan siempre a la misma frecuencia, por lo que al

recibir tensión emitirán siempre el mismo sonido, independientemente del nivel de tensión aplicado.

El otro tipo de zumbadores, que serán los que se usen en este trabajo, son los pasivos. Estos funcionan con diferentes frecuencias en función de la tensión aplicada. Al funcionar con diferentes frecuencias se obtendrá un sonido diferente con cada una de ellas. Dado que se va a someter a una tensión eléctrica variable, el buzzer debe estar conectado a pin PWM. Dado que se pueden obtener diferentes sonidos, es posible reproducir las notas de la escala musical en función de la frecuencia.

Nota	Frecuencia (Hz)
Do	261.63
Re	293.66
Mi	329.63
Fa	349.23
Sol	392.00
La	440.000
Si	493.88

Tabla 3: Frecuencia de las notas musicales

En este trabajo se va a utilizar el buzzer pasivo de la marca ElecFreaks.

Figura 60: Buzzer Pasivo ElecFreaks

5.2.9. ARDUINO NANO

Para la realización de este proyecto se va a utilizar la placa Arduino nano. La elección de esta placa se ha realizado en base a que es cinco veces más

barata que lo placa Arduino UNO con las mismas prestaciones. Además tiene dos pines analógicos más (A6 y A7) permitiendo la conexión de más dispositivos.

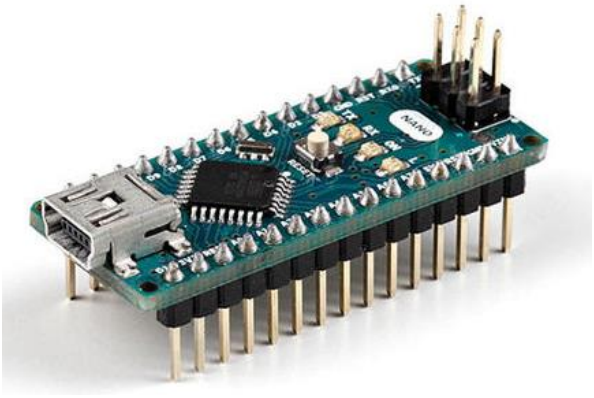

Figura 61: Arduino NANO

5.2.10. SHIELD PARA ARDUINO NANO

Para facilitar las conexiones de la placa con los dispositivos, se va a utilizar un shield que se conecta directamente a la placa Arduino NANO y que tiene en cada pin la conexión de tensión, señal y tierra, evitando así utilizar circuitos externos.

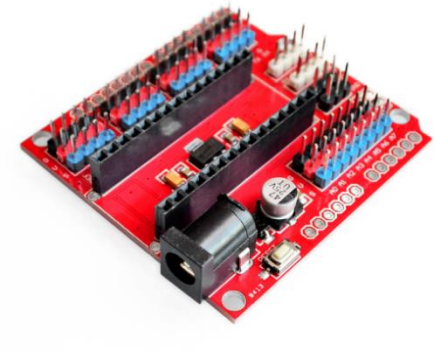

Figura 62: Shield para Arduino NANO

La placa se va a introducir en una caja con una apertura para la conexión USB del Arduino NANO, de manera que se pueda conectar directamente sin necesidad de abrirla. Dado que la conexión para la alimentación sale ligeramente del perfil de la placa, se concluye que la mejor opción es quitarla. La alimentación se va a realizar directamente desde el puerto USB por lo que no supone ningún inconveniente retirar la clavija de alimentación.

CAPÍTULO 6. DESARROLLO DEL PROYECTO

6.1. CREACIÓN DEL SOFTWARE

6.1.1. LIBRERÍA Servo.h

La librería Servo.h se incluye en las librerías de Arduino del programa Code::Blocks. En esta librería se encuentran todas las instrucciones para mover un servo por lo que no es necesario realizar una nueva librería para generar los movimientos del motor. Es necesario, sin embargo definir el objeto del servo e indicar e qué pin se encuentra conectado. Resulta imposible definir una nueva función en la que se cree el objeto del motor y se inicialice el pin sin que el alumno se vea obligado a escribir esa parte del código. Esto es debido a que el programa en Arduino se va a ejecutar en bucle repetidamente, generando un error en el comportamiento del motor al crear el objeto cada vez que se ejecuta. Es necesario, por tanto, que sea el alumno quién inicialice al servo cuando vaya a utilizarlo. Éste código le será proporcionado de antemano y sólo deberá copiarlo en la función setup() de Arduino. Es imprescindible, así mismo, incluir la librería “Servo.h” para realizar los movimientos del motor. El código que debe introducir el alumno es:

```
#include "Servo.h"
```

En primer lugar hay que definir el objeto del servo, para ello se debe poner la siguiente línea de código después de incluir la librería y antes del setup():

```
Servo miservo;
```

Lo siguiente es asociar dicho servo al pin conectado. Para ello se debe copiar la siguiente línea de código dentro del setup():

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

    miservo.attach(6);

}
```

6.1.2. LIBRERÍA Led.h

La parte fundamental del presente trabajo es la creación de una librería que convierta las instrucciones usadas por el alumno en lenguaje C++ en instrucciones específicas de Arduino, haciendo que todos los dispositivos usados funcionen correctamente. En este punto se van a mostrar las funciones usadas en la librería Led.h y qué realiza exactamente cada función:

- **Leds():** Función que define los pines donde van conectados los LED como pines de salida. Dado que es necesario dejar los pines con PWM para ciertos dispositivos, los LED no irán conectados de los pines 1 al 13, por lo que además se realizarán los cambios pertinentes para que la numeración de los LED sea del 1 al 13 independientemente de en qué pin se encuentran conectados. Siempre que el alumno vaya a usar cualquiera de los LED, debe llamar a esta función.
- **LedOn()**: Función que enciende el LED correspondiente al número que el alumno introduzca en el paréntesis.
- **LedOff():**Función que apaga el LED correspondiente al número que el alumno introduzca en el paréntesis.
- **retardo():** Función que convierte el número de segundos, introducidos en el paréntesis, que el alumno quiere hacer de retardo en milisegundos y realiza el delay.
- **parpadeo(led, Ton, Toff):** Función a la que se deben pasar tres parámetros: el número correspondiente al LED que se quiere hacer parpadear, el tiempo en segundos que debe estar encendido dicho LED y el tiempo en segundos que debe estar apagado.
- **RGB(R, G, B):** Función a la que se deben pasar tres parámetros, que son el nivel de 0 a 255 que se desea del color rojo (R), del color verde (G) y del color azul(B). inicializa los pines con PWM 9, 10 y 11 como salidas correspondientes a las patillas R, G y B del dispositivo.
- **LDR():** Obtiene una lectura analógica del valor tomado por el sensor en cada momento y lo almacena en una variable. Inicializa el pin analógico A5 como entrada del sensor.
- **zumbador(frecuencia, segundos):** Función que tiene como parámetro de entrada la frecuencia a la que el alumno desea que suene el zumbador y el tiempo que debe estar sonando. Inicializa el pin A5 como salida correspondiente al zumbador.
- **zumbadorapagado():** Apaga el buzzer.
- **botonera:** Función que almacena en una variable qué botón de la botonera ha sido pulsado. Al llamar a la función, ésta devuelve la variable con el valor del botón pulsado. Inicializa el pin analógico A6 como entrada correspondiente a la botonera

- **estado_boton():** Indica si el botón está pulsado en un momento dado o no lo está.
- **hora(dh, uh, dm, um):** Función a la que se introduce como parámetros las decenas de las horas (dh), unidades de las horas (uh), decenas de los minutos (um) y unidades de los minutos um). La función representará en números binarios mediante los 13 LED, la hora introducida.
- **reloj (dh[], uh[], dm[], um[]):** Función con 4 parámetros de entrada. Cada parámetro es un vector. El número de elementos de los vectores, se corresponde al número de LED de cada columna del reloj, y son 2 para las decenas de las horas (dh[]), 4 para las unidades de las horas (uh[]), 3 para las decenas de los minutos (dm[]) y 4 para las unidades de los minutos (um[]). El alumno introducirá, de manera vectorial qué LED debe estar apagado (introduciendo un 0) y qué LED debe estar encendido (introduciendo un 1).

6.2. REALIZACIÓN DEL HARDWARE

6.2.1. MONTAJE DE LA CAJA

Para realizar el montaje de la caja se ha diseñado la situación de los elementos. Y se han tomado las medidas oportunas de los diámetros y distancias. Partiendo de una caja de plástico de dimensiones 19x11.5x7.5 cm, como se presenta a continuación:

Figura 63: Caja protoripo

En la tapa de la caja se han señalado los puntos donde debe ir cada elemento:

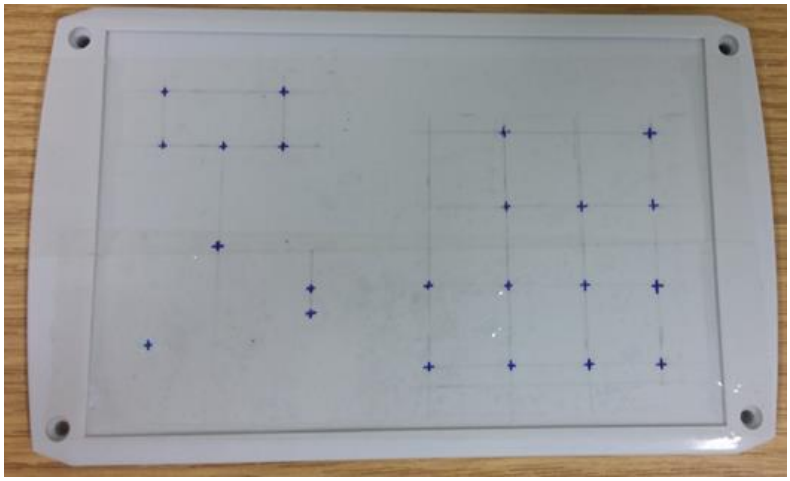

Figura 64: Señalización de la colocación de los elementos

En el CD adjunto en el presente trabajo, se han incluido los archivos en AUTOCAD con los planos de la tapa, la base y el USB para facilitar el montaje de la misma.

El plano de la tapa contiene todos los agujeros que se deben hacer y las medidas de los diámetros para seleccionar las brocas.

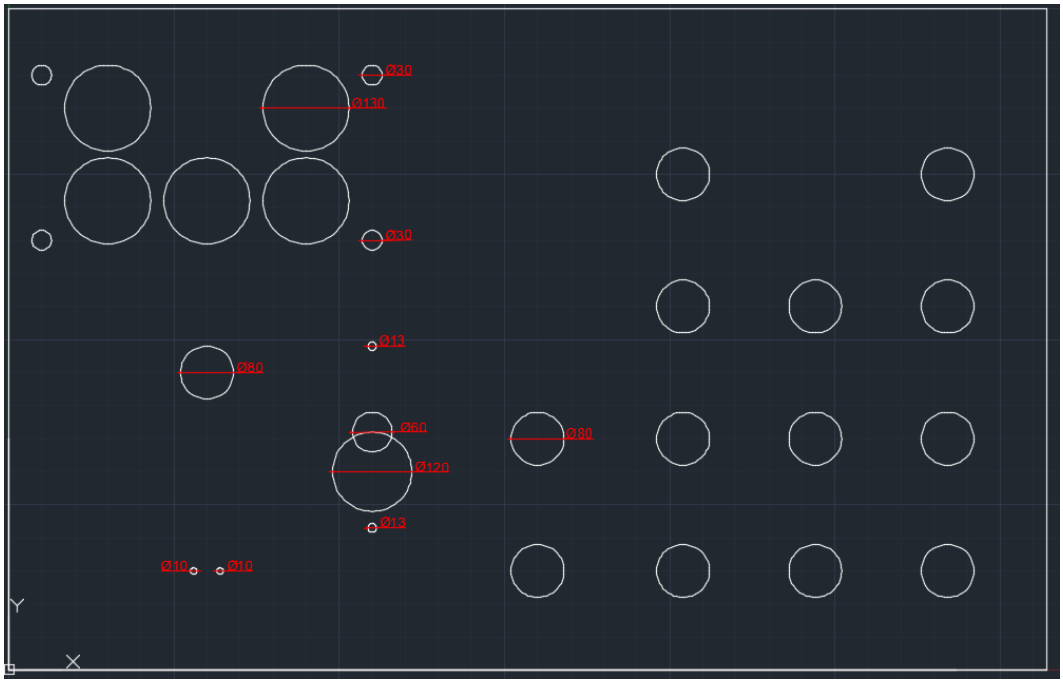

Figura 65: Plano de la tapa de la caja

El plano de la base contiene los agujeros para sujetar el shield del Arduino.

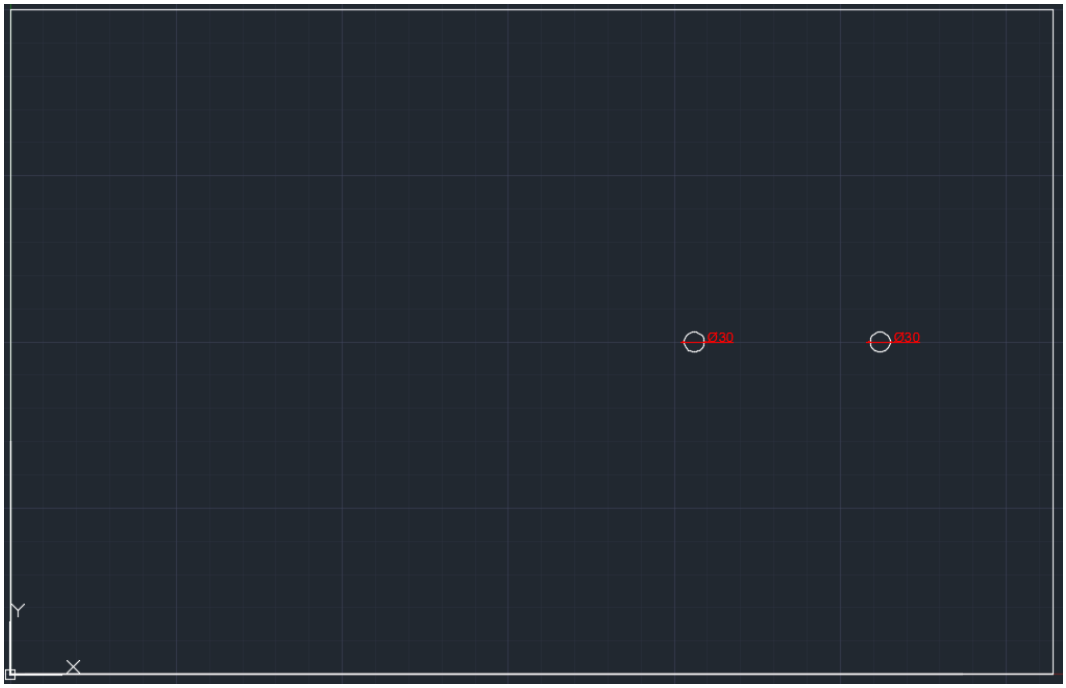

Figura 66: Plano de la base de la caja

Por último, el plano del USB presenta el rectángulo que se debe dejar en uno de los laterales de la caja para poder conectar el cable con la placa. Si se modifica la situación de la placa, también debe modificarse este rectángulo.

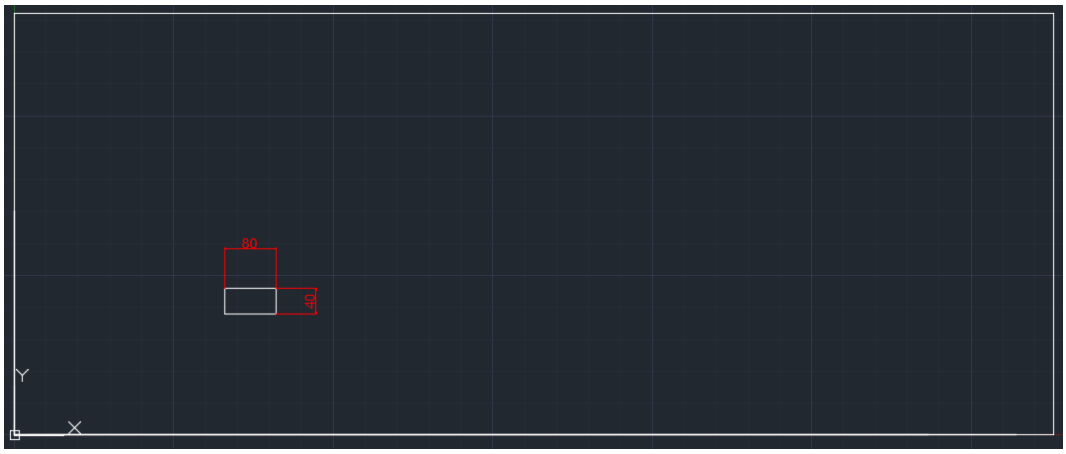

Figura 67: Plano del lateral de la caja

Con un taladro de banco, se han realizado los agujeros, en función del diámetro de cada dispositivo. Además se han realizado los agujeros en la base de la caja para sujetar la placa Arduino así como el hueco necesario para conectar el cable USB por el lateral de la caja. En los taladros destinados a los LED, se han colocado unos soportes para LED que además de fijarles a la tapa de la caja dan mejor presencia al resultado final:

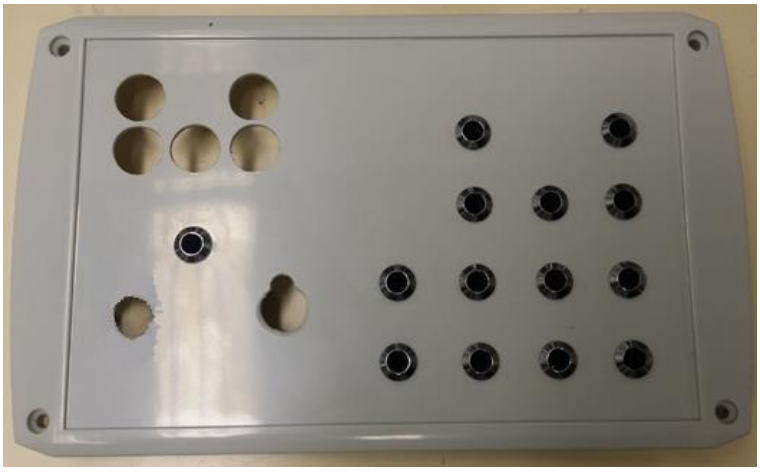

Figura 68: Agujeros de la tapa para los dispositivos

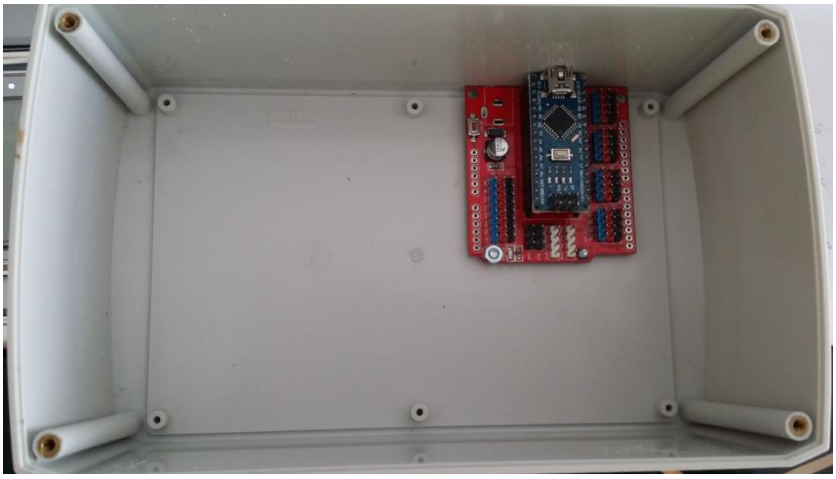

Figura 69: Sujeción de la placa a la base de la caja

Figura 70: Agujero para conectar la placa al cable USB

Una vez creada la caja se procede a situar y conectar los distintos dispositivos. Una vez está todo correctamente situado y conectado, se cerrará la caja con los tornillos de manera que el alumno no la podrá abrir en ningún momento.

6.2.2. CONEXIÓN DE LOS DISPOSITIVOS

Como se ha expresado anteriormente, la placa tiene pines analógicos y digitales, pudiéndose usar los analógicos como digitales. Dentro de los pines digitales, existen 6 con salida PWM, es decir, pueden generar una salida analógica. Dado que el generador PWM de Arduino trabaja con 8 bits, existen 256 valores distintos de codificación de la señal. Por esto, se puede realizar una escritura analógica de 0 a 255 en cualquier pin con WM. En al caso de este trabajo, esto es útil para el uso del LED RGB o para el control del motor SG90. Teniendo en cuenta todo esto, la conexión de los elementos en cada pin será:

PIN	ELEMENTO
2	LED
3	LED
4	LED
5	LED
6	SERVO SG90
7	LED
8	LED
9	RGB
10	RGB
11	RGB
12	LED
13	LED
A0	LED
A1	LED
A2	LED
A3	LED
A4	LED
A5	BUZZER
A6	BOTONERA
A7	LDR

Tabla 4: conexión de los elementos en la placa

A la hora de conectar los distintos elementos, es muy importante tener en cuenta las polaridades de los dispositivos, dónde se debe conectar la señal y dónde la tierra. En el capítulo 5 del presente trabajo se han especificado los conocimientos previos a tener en cuenta de cada dispositivo incluyendo, en cada caso, el modo correcto de conectar las patillas a la placa Arduino.

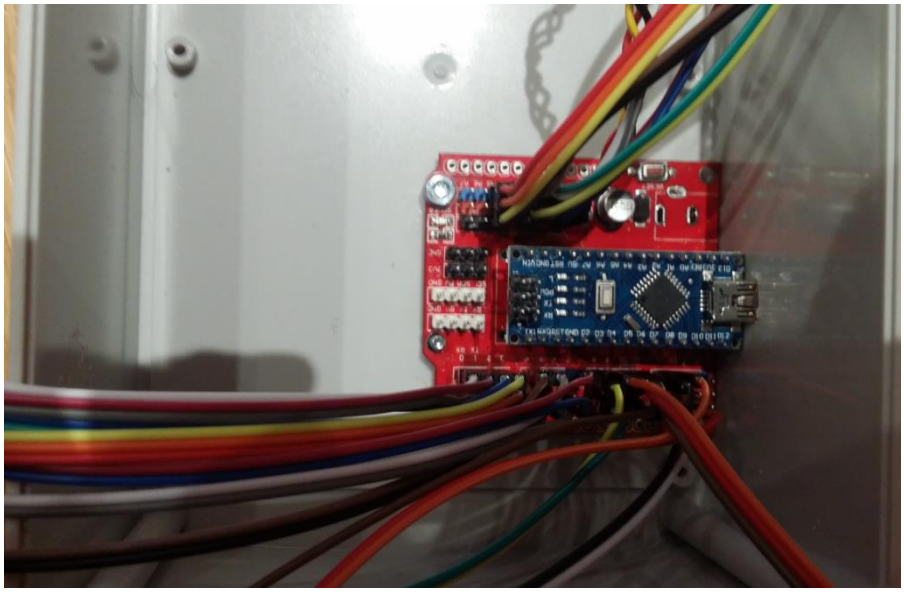

Figura 71: Conexiones Arduino

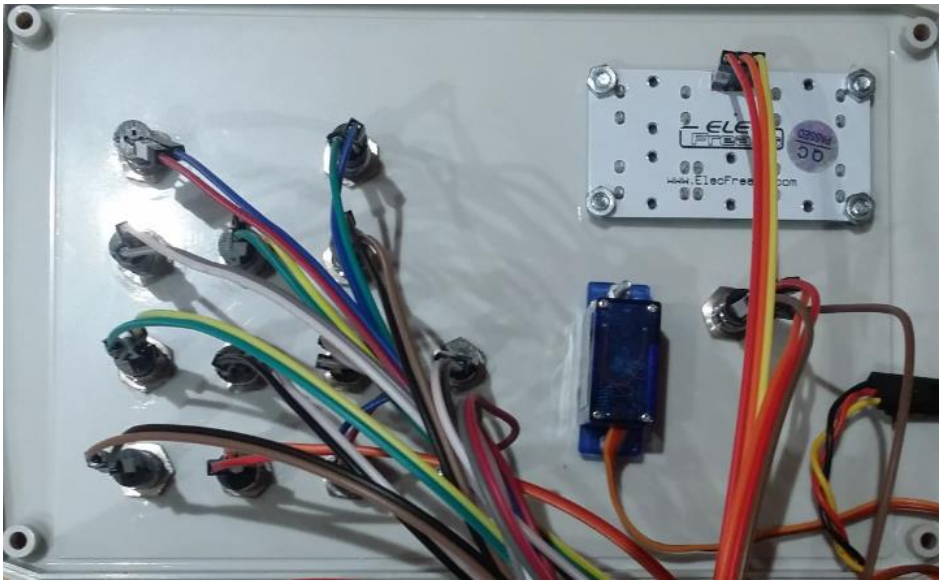

Figura 72: Conexiones de los dispositivos

Una vez se han conectado todos los dispositivos con la placa Arduino, se procede a cerrar la caja y poner los tornillos y los embellecedores.

Figura 73: Resultado final

CAPÍTULO 7. CONCLUSIONES

A la hora de realizar un proyecto en ingeniería, además de obtener los resultados óptimos, se busca hacer un uso lo más eficiente posible de los recursos y se estudia a quién va dirigido dicho proyecto ya que tan importante es obtener la solución al problema como que la persona a la que va dirigido el proyecto pueda hacer uso del mismo.

En este caso los recursos utilizados han sido tanto software como hardware. En el caso de los primeros es totalmente gratuito, fácil de instalar y de usar. Además, en la asignatura Fundamentos de Informática se ha venido usando en los últimos años la plataforma Code::Blocks, con un interfaz prácticamente igual a la plataforma usada como solución a este ejercicio Code:Blocks Arduino IDE. Por otra parte el hardware empleado es muy fácil de manejar y se ha buscado que el coste económico sea el mínimo (el precio de todos los dispositivos utilizados para la realización del proyecto está entorno a los 20 euros).

Dado que el proyecto está orientado a alumnos de nuevo ingreso en ingeniería, se ha procurado que el resultado sea lo más sencillo de usar posible, en cuanto a las plataformas utilizadas para realizar la programación como a los propios programas que deben desarrollar como ejercicios complementarios del proyecto.

Con la realización de este trabajo se espera motivar a los estudiantes, ya que, al principio, la programación en C++ puede resultar difícil y pesada. Se ha pretendido que el alumno entienda que lo que está programando en el ordenador tiene repercusiones reales, que en este caso observará a través de la caja con distintos dispositivos, y tomará conciencia de la magnitud de cosas que puede realizar con la programación.

La asignatura Fundamentos de Informática es básica para alumnos de ingeniería en general y un pilar fundamental para algunas especialidades como la de Electrónica Industrial y Automática, por lo que se considera primordial que el alumno tenga ganas de aprender y vea la asignatura como una herramienta que le permita solucionar problemas futuros.

Por otra parte y como ampliación, este proyecto puede tener utilidad en otras asignaturas tales como Fundamentos de Automática o Fundamentos de Electrónica mediante el estudio de los distintos dispositivos empleados como solución del hardware.

7.1. SOLUCIÓN A POSIBLES PROBLEMAS

A lo largo de este documento se ha detallado la solución a adoptar ante diferentes problemas que pueden surgir a la hora de configurar el software. Dada la importancia de éstos para el correcto funcionamiento del proyecto, se resumen brevemente en este punto y se indican las referencias para obtener un estudio más detallado de los mismos.

1. Es crucial que tanto en Code::Blocks Arduino IDE como en Arduino Builder se seleccione la placa Arduino que se va a utilizar. En este caso se debe seleccionar la placa Arduino NANO 328.
2. Ni el título del proyecto ni ninguna de las carpetas de la ruta de accesos puede llevar un espacio en el nombre ya que generará errores más adelante en la creación del archivo de extensión .hex. Véase el apartado 4.2.1 Estructura de un programa en Arduino.
3. Es posible que el ordenador no pueda instalar el controlador del dispositivo, siendo necesario descargar los drivers. El enlace para la descarga se puede consultar en la bibliografía del trabajo [7]. Los pasos a seguir se especifican en el apartado 4.3 Arduino Builder.

CAPÍTULO 8. BIBLIOGRAFÍA

- [1]. Guía docente de la asignatura “Fundamentos de Informatica”

https://alojamientos.uva.es/guia_docente/uploads/2016/452/42357/1/Documento.pdf

- [2]. Página oficial de Code::Blocks

http://www.codeblocks.org/

- [3]. Página oficial de Arduino

www.arduino.cc

- [4]. Página oficial de Code::Blocks Arduino IDE

http://arduinodev.com/codeblocks/

- [5]. Página de descarga de Code::Blocks Arduino IDE para Windows

https://sourceforge.net/projects/arduinodev/files/

- [6]. Página de descarga del compresor 7-Zip

https://7-zip.softonic.com/

- [7]. Página de descarga de los drivers para arduino NANO

http://dinastiatecnologica.com/producto/arduino-nano-cable-usb-compatible/

- Proyecto fin de carrera “Seminario práctico de fundamentos de informática con Arduino: de la teoría a la práctica” del autor Eva María Domínguez Sanz.

ANEXO 1: ÍNDICE DE TABLAS

ANEXO 2: ÍNDICE DE FIGURAS

ANEXO 3: GUÍA PARA LOS ALUMNOS

ÍNDICE

1. CAJA DE DISPOSITIVOS

Para la realización de las siguientes prácticas se usará una caja con distintos dispositivos. La caja debe permanecer conectada al ordenador en todo momento mediante un cable USB. A continuación se especifican los distintos elementos. Más adelante se presentarán las funciones y su modo de uso para el correcto funcionamiento de cada dispositivo.

Figura 74: Caja de dispositivos

- Botonera: Consta de cinco pulsadores:

Figura 75: Botonera

- RGB: LED multicolor. Puede programarse para que luzca rojo, verde, azul o cualquier combinación de estos tres colores:

Figura 76: RGB

- LDR: Fotorresistencia. Se realiza una lectura de la luz que incide en el dispositivo, tomando valores próximos a cero cuando se encuentra tapado (ausencia de luz) y valores elevados cuando hay mucha luz.

Figura 77: LDR

- SERVOMOTOR: Motor que gira una posición dada. A diferencia de un servo de rotación continua, este motor no da vueltas completas, sino que se sitúa en la posición especificada en grados.

Figura 78: Servo

- Conjunto de LED (Reloj binario): Diodos LED que pueden apagarse o encenderse. La disposición en columnas se ha realizado para formar un reloj binario donde las columnas, de izquierda a derecha, se corresponden a las decenas de las horas, las unidades de las horas, las decenas de los minutos y las unidades de los minutos respectivamente.

Figura 79: LEDS

2. PROYECTO ARDUINO

Para la realización de estas prácticas se usará la plataforma Code::Blocks Arduino IDE. En el interfaz de la aplicación se pulsa File ⟶ new ⟶ Project…

Y se selecciona Arduino Project:

Al crear un nuevo proyecto en Arduino aparecerán una serie de ventanas emergentes:

Figura 80: Creación de un proyecto en arduino

Pulsando en Next, se obtiene:

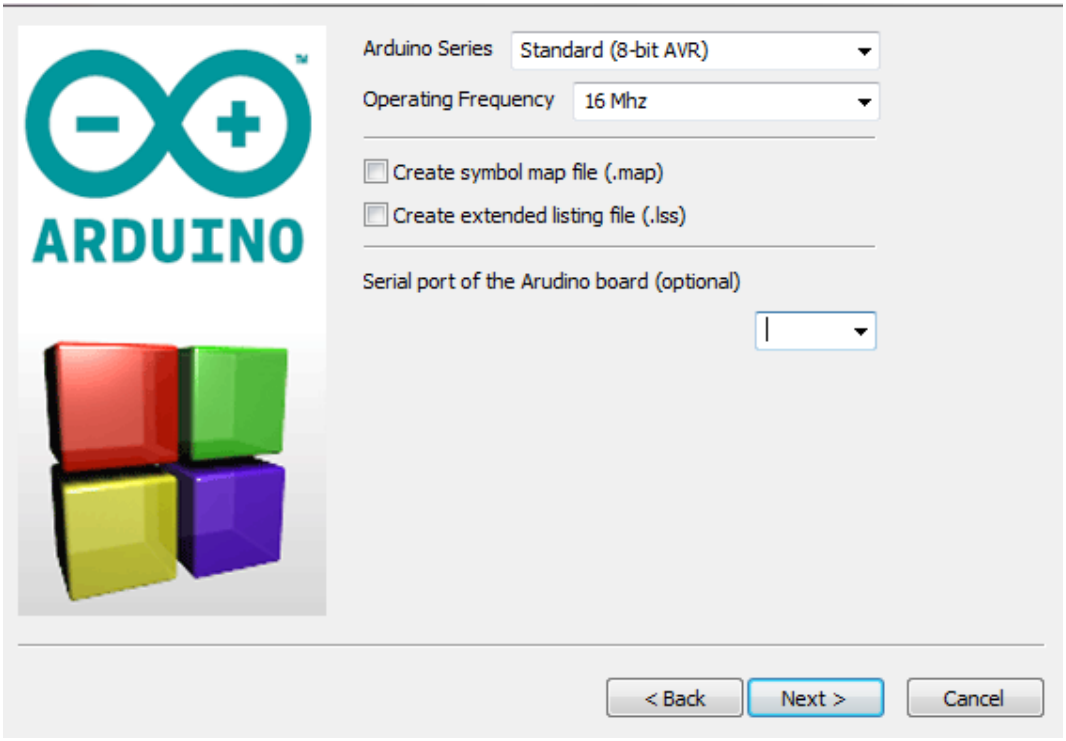

Figura 81: Configuración de parámetros

Se dejan las opciones que vienen por defecto y se pulsa Next.

Figura 82: Título y ruta de acceso

En esta ventana se dará título al proyecto en la pestaña Project title. Es muy importante no poner espacios en el nombre del proyecto, ya que creará un

conflicto interno que hará imposible cargar el programa, en su lugar se puede sustituir el espacio por una barra baja. Se definirá la ruta donde desea guardarse el nuevo proyecto en la pestaña Folder to create Project in. Una vez dado nombre al proyecto y definido el lugar donde se debe crear, se pulsa Finish.

Se abrirá un código de ejemplo sobre el que se va a trabajar:

```
1   #include <Arduino.h>
2
3   /*
4     Turns on an LED on for one second, then off for one second, repeatedly.
5   */
6
7   void setup()
8   {
9       Serial.begin(9600);
10
11      // initialize the digital pin as an output.
12      // Pin 13 has an LED connected on most Arduino boards:
13      pinMode(13, OUTPUT);
14  }
15
16  void loop()
17  {
18      Serial.println("Hello world!");
19
20      delay(1000);              // wait for a second
21      digitalWrite(13, HIGH);   // set the LED on
22      delay(1000);              // wait for a second
23      digitalWrite(13, LOW);    // set the LED off
24  }
25
```

Figura 83: Ejemplo de proyecto en arduino

En primer lugar el alumno debe eliminar todas las instrucciones del código, dejando únicamente el setup() y el loop(). Se trabajará a partir de este código:

```
#include <Arduino.h>


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

}

void loop()
{

}
```

Figura 84: Código base para proyecto en arduino

En primer lugar se debe incluir la librería “Led.h” ya que es donde se encuentran todas las funciones que harán que el alumno pueda usar los distintos dispositivos de la caja.

```
#include <Arduino.h>

#include "Led.h"

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

}

void loop()
{

}
```

Como se puede apreciar, un proyecto con Arduino consta de dos partes. Por un lado está la función setup(). Las instrucciones que se den en esta función se realizarán una única vez al principio del programa. Por esto es la parte donde se inicializan variables o donde, por ejemplo, se abre la comunicación con el puerto serie:

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

}
```

La segunda parte del proyecto, el loop(), podría definirse como el main() que se ha venido usando hasta ahora. La principal diferencia es que esta función se ejecuta repetidamente en un bucle del que no se sale nunca. El programa está constantemente repitiendo las mismas instrucciones. Si por ejemplo, se desea que el programa haga una cosa diferente en función del botón pulsado, es necesario que se esté leyendo continuamente el valor del botón activo. Es por esto que en los presentes ejercicios, las instrucciones se darán siempre en el loop().

Excepciones: Como se ha mencionado anteriormente, los objetos deben definirse en el setup() ya que sólo se ejecuta una vez. Es por esto que al usar el servomotor, deben introducirse algunas instrucciones en esta parte. Esto se definirá más adelante, cuando se estudien cada una de las funciones de la librería “Led.h”.

3. MANUAL DE FUNCIONES DE LA LIBRERÍA "Led.h"

A lo largo de los próximos ejercicios se van a usar una serie de funciones que se encuentran en la librería "Led.h". Para poder hacer uso de las mismas es imprescindible que en cada nuevo proyecto se incluya la librería:

```
#include "Led.h"
```

Las funciones disponibles son:

escribir()

Esta función escribirá en el puerto serie la variable introducida por el alumno. Para abrir el puerto serie en Arduino Builder, una vez se ha cargado el código en la placa, se selecciona la velocidad de 9600 en la pestaña de la derecha y se pulsa open.

Ejemplo de uso:

```
escribir("Hola mundo!");
```

Al abrir el puerto serie se imprimirá la frase Hola mundo! repetidamente.

retardo()

Esta función realizará una espera (delay) del tiempo introducido por el alumno. El tiempo debe introducirse en segundos.

LedOn() y LedOff()

LedOn enciende y LedOff apaga el número de LED que se haya introducido respectivamente. La numeración de los LED es la siguiente:

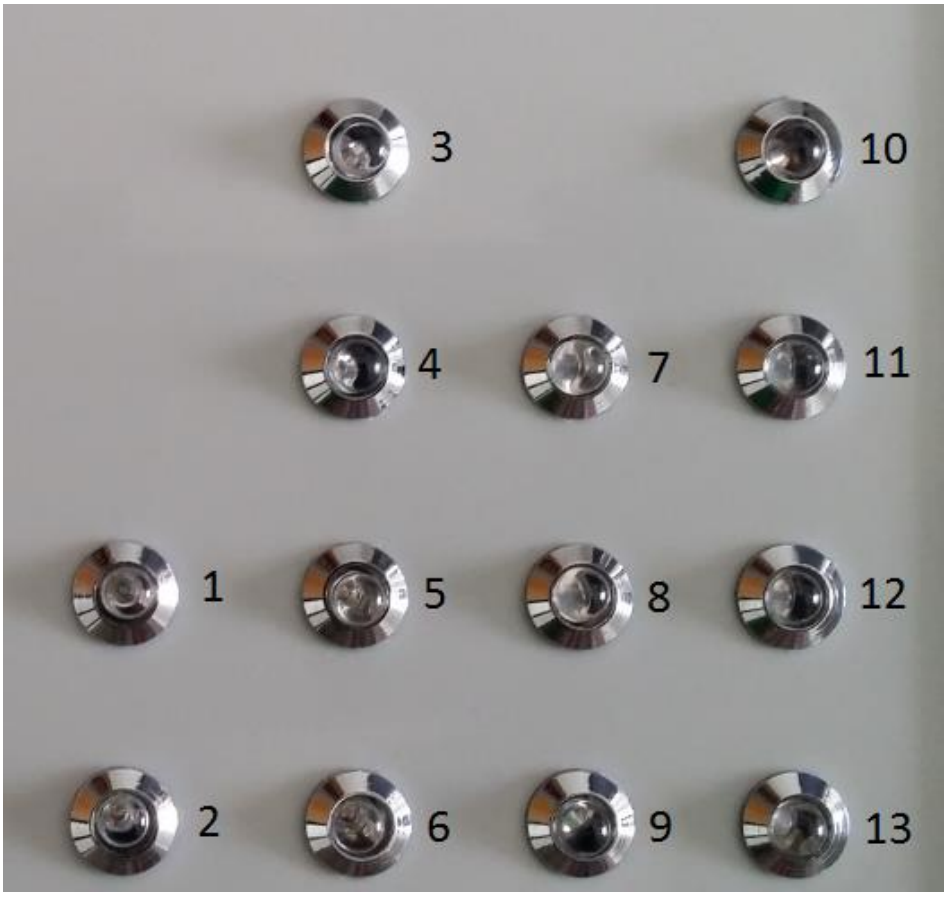

Figura 85: Numeración de los LEDS

Ejemplo de uso:

```
LedOn(5);
retardo(10);
LedOff(5);
retardo(8);
```

En este caso se encenderá el LED 5 durante 10 segundos y se apagará durante 8 segundos.

botonera()

Dado que existen 5 botones, esta función devolverá el valor del botón que se haya pulsado (del 1 al 5). Se creará una variable en la que se almacenará el valor devuelto por la función. La numeración de los botones es la siguiente:

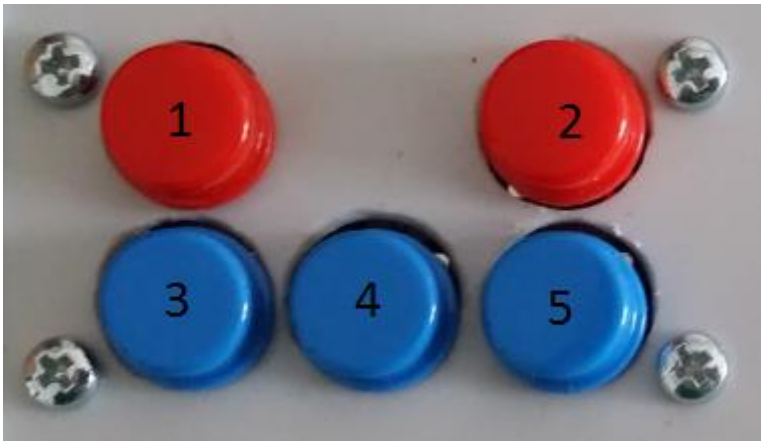

Figura 86: Numeración de los pulsadores

Ejemplo de uso:

```
int boton=botonera();
if(boton==1)
    LedOn(1);
```

En este caso cuando se pulse el botón 1 se encenderá el LED 1.

LDR()

Esta función devuelve el valor analógico leído por el sensor LDR. Se almacenará en una variable para poder trabajar con él.

Ejemplo de uso:

```
int valor=LDR();
escribir(valor);
```

En este caso, al abrir el puerto serie en Arduino Builder (a la velocidad de 9600) aparecerán los valores leídos por el sensor LDR.

RGB()

Esta función controla el LED RGB mediante 3 valores. El rango de operación es de 0 a 255, siendo el 0 nada de ese color y el 255 todo de ese color. Se

deberán introducir los valores que se quieran para el rojo, el verde y el azul en ese orden.

Ejemplo de uso:

```
RGB(0, 255, 0);
```

De esta manera el RGB se iluminará en color verde.

zumbador()

Esta función hace que suene el zumbador con la frecuencia introducida durante un tiempo determinado en segundos.

La frecuencia de las notas de la escala musical es:

Nota	Frecuencia (Hz)
Do	261.63
Re	293.66
Mi	329.63
Fa	349.23
Sol	392.00
La	440.000
Si	493.88

Tabla 2: Frecuencia de las notas musicales

Ejemplo de uso:

```
zumbador(350, 5);
```

En este caso sonará la nota FA durante 5 segundos.

SERVO

La caja de dispositivos cuenta con un servo motor SG90. Éste no es un motor de rotación continua, sino que se mueve por posiciones. Es decir, el servo no puede realizar vueltas completas, en su lugar se sitúa en una posición definida en grados y que va de 0 a 180.

Para poder usar esta función es imprescindible incluir la librería “Servo.h”

```
#include "Servo.h"
```

En primer lugar hay que definir el objeto del servo, para ello se debe poner la siguiente línea de código después de incluir la librería y antes del setup:

```
Servo miservo;
```

Lo siguiente es asociar dicho servo al pin conectado. Para ello se debe copiar la siguiente línea de código dentro del setup:

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

    miservo.attach(6);

}
```

Para escribir la posición en grados que se desea tomar el comando es miservo.write(grados) y para especificar el tiempo requerido para llegar a esa posición se usará la función retardo con el tiempo en segundos. De esta manera se puede variar la velocidad del servo.

Es muy importante tener en cuenta que la posición máxima del servo es de 180º y la mínima de 0º.

Ejemplo de uso:

```
#include <Arduino.h>
#include "Servo.h"
#include "Led.h"
Servo miservo;

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

    miservo.attach(6);

}

void loop()
{
    miservo.write(0);
    retardo(0.5);
    miservo.write(179);
    retardo(0.5);
}
```

En este ejemplo el servo se moverá entre sus posiciones máximas y mínimas tardando medio segundo en realizar cada recorrido. El movimiento se repetirá en un bubcle. Si se modifica el retardo a 0.25 se consigue que la velocidad aumente realizando el recorrido en la mitad de tiempo.

hora (, , ,)

Esta función sirve para manejar el reloj binario disponible en la caja de dispositivos. Como argumento de entrada a la función debe introducirse la hora en decimales que se desea representar en binario separando cada dígito por comas (ya que se trata de un vector). El modo de uso es:

hora(decenas_horas , unidades_horas , decenas_minutos , unidades_minutos);

reloj (, , ,)

Esta es otra de las funciones que maneja el reloj binario. En este caso cada uno de los 4 argumentos de entrada es un vector de diferentes tamaños. Cada vector se corresponde con cada una de las columnas del reloj. . El tamaño se corresponde al número de LED de cada columna. En primer lugar se deben crear los vectores con los valores 0 (LED apagado) o 1 (LED encendido). Al pasar a la función los valores de los vectores, se encenderán los LED indicados representando la hora introducida en números binarios.

Ejemplo de uso:

```
#include <Arduino.h>
#include "Led.h"


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

}

void loop()
{
    int a[2]={1,1};
    int b[4]={1,1,0,1};
    int c[3]={1,1,1};
    int d[4]={1,0,1,0};
    reloj(a,b,c,d);

}
```

4. FUNCIONES IF, IF-ELSE

Para el estudio de estas estructuras se realizará el problema de una luz de escalera. Se irá resolviendo en distintas fases añadiendo, cada vez, un grado de complejidad. Se utilizará uno de los pulsadores de la botonera y uno de los diodos LED a elegir, ambos, por el alumno.

1. *La luz se enciende cuando se pulsa un botón. y se apaga cuando se vuelve a pulsar.*
   En este punto el alumno decidirá cuál de los botones será el que accione la luz y qué led se encenderá.

```
#include <Arduino.h>
#include <Led.h>

/*Enciende un LED al pulsar un boton*/

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

void loop()
{
    int boton=botonera();
    if(boton==1)
        LedOn(1);
}
```

2. *La luz se enciende al pulsar un botón, espera un tiempo definido (por ejemplo 10 segundos) y se apaga. Si cuando la luz está encendida se vuelve a pulsar el botón no sucede nada.*

3. *La luz se enciende cuando se pulsa un botón y se apaga al volver a pulsarle.*
   En este punto será necesario crear una variable de estado que indique si el valor del botón ha cambiado, de manera que si se detecta algún cambio, el led se encenderá o se apagará.
   La variable será de tipo bool, de manera que su valor cambiará entre true o false. En primer lugar se inicializa como false y cada vez que se

pulse el botón cambiará su valor, de manera que al ser true se encenderá el led y al ser false se apagará.

```
bool estado = false;
    if(botón==1)
estado=!estado;
```

4. *La luz se enciende automáticamente si se detecta que es de noche.* Para esto se usará un sensor LDR. En primer lugar el alumno leerá los valores que alcanza dicho sensor. Para ello utilizará la función escribir() en la que introducirá la variable en la que ha guardado el valor leído por el sensor LDR. Una vez abierto el puerto serie en Arduino Builder (velocidad de 9600), debe estudiar el rango de funcionamiento del sensor, es decir, qué valor toma cuando hay mucha luz incidente y cuál cuando está totalmente tapado. Una vez estudiados estos valores el alumno debe escoger un valor por debajo del cual debe encenderse el LED.
5. *La luz se enciende cuando se pulsa el botón sólo cuando es de noche.*

5. FUNCIÓN SWITCH

Para el estudio de esta función se utilizará la botonera.

1. *Encender el diodo RGB de un color diferente en función del botón pulsado.*
   Por ejemplo, al pulsar el botón 1 se pondrá azul, el 2 verde, el 3 rojo…
2. *Hacer sonar el zumbador con una nota diferente en función del botón pulsado.*
   Por ejemplo, al pulsar 1 hacer sonar la nota DO, 2 RE, 3 MI…
3. *Movimiento del servo mediante la botonera. Al pulsar el botón 1, el servo se situará en la posición de 0º e irá moviéndose un número definido de grados en función del botón pulsado. Además se especificará el tiempo para realizar cada movimiento*
   Por ejemplo, al pulsar el botón 1 se moverá a la posición de 30º en medio segundo, el botón 2 a 90 en un cuarto de segundo…

6. FUNCIONES DO, DO-WHILE

1. *Encender el LED cuando se detecta que es de noche.*
   Para resolver este problema se utilizará el sensor LDR y un LED a elegir por el alumno.
2. *Recorrer todos los colores del RGB.*
3. *Hacer que se enciendan uno a uno todos los LEDS con una espera entre encendidos a elegir por el alumno.*

7. FUNCIÓN FOR

Para el estudio de este bucle se usará una fila de LED. La disposición de los LED no va a ser lineal ya que para un ejercicio que se presentará más adelante es necesario que sean 4 filas diferentes pero para realizar estos ejercicios se considerará que es una misma línea y se contarán los LED del 1 al 13.

1. *Ir encendiendo uno a uno, en orden, cada LED.*
2. *Ir encendiendo uno a uno, en orden, cada LED y una vez que están todos encendidos ir apagándolos uno a uno en orden inverso.*
3. *Ir encendiendo uno a uno, en orden, cada LED pero una vez que se enciende uno se apaga el anterior, de manera que en cada momento solo habrá un LED encendido.*
4. *Ampliación del ejercicio anterior, se irá encendiendo cada vez un LED del 1 al 13 y luego cada vez un LED del 13 al 1.*
5. *Recorrer todos los colores del LED RGB.*

8. VECTORES

Para el estudio de vectores se va a realizar un reloj binario. Para ello se dispondrán cuatro columnas de LED. La primera con dos, la segunda con cuatro, la tercera con tres y la cuarta con cuatro. Las dos primeras columnas representan las horas y las dos segundas los minutos.

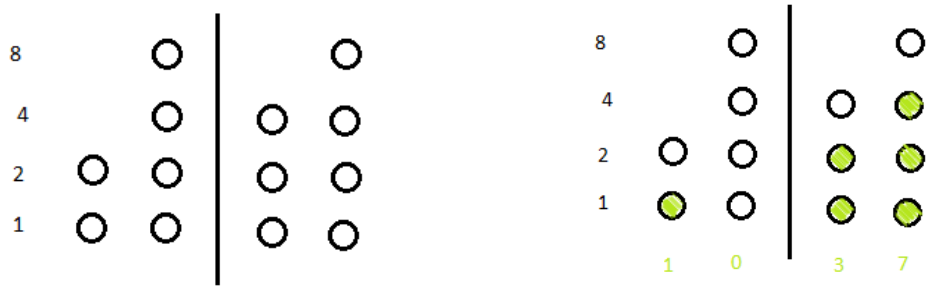

Figura 87: Reloj binario

Tal como se ve en la imagen de ejemplo, las columnas de la izquierda corresponden a las horas, en el ejemplo está el primer LED encendido, que corresponde al 1 y en la segunda columna no hay ninguno encendido, por lo que es un 0 (la hora se da en formato 24 h). Las columnas de la derecha corresponden a los minutos y, sumando los LED encendidos, se ve que son 3 y 7. Por lo que serán las 10:37

El objetivo final de este ejercicio es que el alumno comprenda y practique el uso de números binarios. Las actividades que se plantean se basan únicamente en el manejo del número binario. Para ir más allá en la parte de programación es posible que sea el alumno quién programe las funciones reloj() y hora().

1. *Representar una hora específica (por ejemplo, las 13:30).*
   Para este punto se utilizará la función reloj(1,3,0,0);
2. *Representar una hora binaria.*
   Para ello se configurarán los vectores mediante unos y ceros. El alumno debe entender la hora decimal que se ha representado mediante vectores en sistema binario.

ANEXO 4: SOLUCIÓN DE LOS EJERCICIOS

ÍNDICE

1. FUNCIONES IF, IF-ELSE

1.1. Encender un LED al pulsar un botón.

```
#include <Arduino.h>
#include <Led.h>

/*Enciende un LED al pulsar un boton*/

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

void loop()
{
    int boton=botonera();
    if(boton==1)
        LedOn(1);
}
```

1.2. Encender un LED al pulsar un botón, esperar un tiempo determinado y apagarlo

```
#include <Arduino.h>
#include "Led.h"
/*Al pulsar un boton, enciende un LED un
tiempo determinado y lo apaga */
void setup()
{
    Serial.begin(9600);

}

void loop()
{
    int boton=botonera();
    if(boton==1){
        LedOn (1);
        retardo(10);
        LedOff(1);
    }

}
```

1.3. Encender un LED al pulsar un botón y apagar el LED al volver a pulsar el botón

```
#include <Arduino.h>
#include "Led.h"

/*Enciende el LED al pulsar un boton
y lo apaga al volver a pulsarle*/

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

}

void loop()
{
    bool estado=true;
    int boton=botonera();
    if((boton==1)&&(estado==true))
        LedOn(1);
    if(boton==-1);
        estado=!estado;
    if((boton==1)&&(estado==false))
        LedOff(1);
    Serial.println(estado);
    Serial.println(boton);
    delay(5);


}
```

1.4. Encender un LED cuando la lectura del sensor LDR sea menor que un valor determinado.

```
#include <Arduino.h>
#include "Led.h"

/*Enciende el LED cuando el sensor LDR detecete
un valor menor que el indicado*/

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

}

void loop()
{
    int valor=LDR();
    escribir(valor);
    if(valor<100)
        LedOn(1);
    else
        LedOff(1);

}
```

1.5. Encender un LED cuando se pulsa un botón y además la lectura del sensor LDR sea menor que un valor determinado.

```
#include <Arduino.h>
#include "Led.h"

/*Enciende un LED al pulsar un botón solo cuando es de noche*/

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

}

void loop()
{
    int boton=botonera();
    int valor=LDR();
    if((boton==1)&&(valor<100))
        LedOn(1);
    else
        LedOff(1);
}
```

2. FUNCIÓN SWITCH

2.1. Cambiar el color del LED RGB en función del botón pulsado

```
#include <Arduino.h>
#include "Led.h"

/*Cambia el color del RGB en función
del botón pulsado*/

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

void loop()
{
    int boton=botonera();
    switch(boton)
    {
    case 1:
        RGB(255, 0, 0);
        break;
    case 2:
        RGB(0, 255, 0);
        break;
    case 3:
        RGB(0, 0, 255);
        break;
    case 4:
        RGB (50, 150, 20);
        break;
    case 5:
        RGB(255, 255, 255);

    }

}
```

2.2. Cambiar la frecuencia del zumbador (distintas notas de la escala musical) en función del botón pulsado

```
#include <Arduino.h>
#include "Led.h"

/* Cambiar la nota del zumbador en
Función del botón pulsado */

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

void loop()
{
    int boton=botonera();
    switch(boton)
    {
    case 1:
        zumbador(260, 2);
        break;
    case 2:
        zumbador(294, 2);
        break;
    case 3:
        zumbador(330, 2);
        break;
    case 4:
        zumbador(350, 2);
        break;
    case 5:
        zumbador(392, 2);
    }
}
```

2.3. Situar al servo en una posición diferente (distinto ángulo de giro) en función del botón pulsado en cada momento.

```
#include <Arduino.h>
#include "Servo.h"
#include "Led.h"
Servo miservo;

/* Sitúa al servo en una posición
diferente en fución del botón pulsado */

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

    miservo.attach(6);

}

void loop()
{
int boton=botonera();
    switch(boton)
    {
    case 1:
        miservo.write(0);
        retardo(0.5);
        break;
    case 2:
        miservo.write(30);
        retardo(0.25);
        break;
    case 3:
         miservo.write(50);
        retardo(0.75);
        break;
    case 4:
        miservo.write(100);
        retardo(0.1);
        break;
    case 5:
        miservo.write(179);
        retardo(0.3);
        break;
    }

}
```

3. FUNCIONES DO, DO-WHILE

3.1. Encender un LED cuando se detecta que la lectura del sensor LDR es menor que un valor determinado (es de noche).

```
#include <Arduino.h>
#include "Led.h"

//Enciende un LED cuando es de noche


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

void loop()
{
        int valor=LDR();
        Serial.println(valor);
        do
        {
            valor=LDR();
            Serial.println(valor);
            LedOn(2);
        }
        while(valor<280);
        do
        {
            valor=LDR();
            Serial.println(valor);
            LedOff(2);
        }
        while(valor>281);
}
```

3.2. Recorre, una a una, todas las combinaciones posibles de colores del LED RGB

```
#include <Arduino.h>
#include "Led.h"

//Recorreer todos los colores del RGB

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


}

void loop()
{
    int n=0;
    do
    {
        int color=0;
        color=color+n;
        retardo(2);
        n++;
        if(color==1)
            RGB(255,0,0);
        if(color==2)
            RGB(0,255,0);
        if(color==3)
            RGB(0,0,255);
    }
    while(n<3);
}
```

3.3. Ir encendiendo, uno a uno, cada uno de los 13 LED

```
#include <Arduino.h>
#include "Led.h"

//Hacer que se enciendan los LED uno a uno

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

}

void loop()
{
   int num=0;
    do
    {
        int LED=0;
        LED=LED+num;
        retardo(2);
        num++;
        LedOn(LED);

    }
    while(num<=13);
}
```

4. FUNCIÓN FOR

4.1. Ir encendiendo, uno a uno, cada uno de los 13 LED (esta vez con una función for)

```
#include <Arduino.h>
#include "Led.h"

//Enciende uno a uno cada LED

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

void loop()
{
     for (int j=1; j<14; j++)
  {
    LedOn (j);
    retardo(1);
  }
}
```

4.2. Encender uno a uno cada LED. Tras estar encendidos todos, ir apagando uno a uno cada LED en orden inverso.

```
#include <Arduino.h>
#include "Led.h"

//Enciende uno a uno cada LED. Tras encender
//todos se apagan uno a uno en orden inverso


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

void loop()
{
     for (int j=1; j<14; j++)
  {
    LedOn (j);
    retardo(1);
  }
  for (int i=14; i>0; i--)
  {
    LedOff (i);
    retardo(1);
  }
}
```

4.3. Ir recorriendo, uno a uno, cada LED pero en cada momento solo uno permanecerá encendido. (Al encender uno se debe apagar el anterior).

```
#include <Arduino.h>
#include "Led.h"

//Enciende uno a uno cada LED
//y va apagando el anterior

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

void loop()
{
 int j, k;
 for (j=1; j<14; j++)
  {
    LedOn (j);
    retardo(2);
    k=j-1;
    for ( k=1; k<14; k++)
  {
    LedOff (k);
  }
  }
}
```

4.4. Ir recorriendo los LED de manera que en cada momento solo haya uno encendido. Una vez se hayan encendido todos realizar la misma operación en sentido inverso.

```
#include <Arduino.h>
#include "Led.h"

//Enciende cada vez un LED en un sentido
//y despues en sentido contrario

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

void loop()
{
 int j, k, i, h;
 for (j=1; j<14; j++)
  {
    LedOn (j);
    retardo(2);
    k=j-1;
    for ( k=1; k<14; k++)
  {
    LedOff (k);
  }
  }
 for (i=14; i>1; --i)
  {
    LedOn (i);
    retardo(2);
    h=i-1;
    for ( h=14; h>1; --h)
  {
    LedOff (h);
  }
  }
}
```

4.5. Recorrer todos los colores del LED RGB

```
#include <Arduino.h>
#include "Led.h"

//Recorre todos los colores del RGB

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

void loop()
{
  int i;
  for(i=0; i<256; i++)
  {
  RGB (i, 0, 0);
  retardo(3);
  RGB (0, i, 0);
  retardo(3);
  RGB (0, 0, i);
  retardo(3);
  RGB(i, i, 0);
  retardo(3);
  RGB(i, 0, i);
  retardo(3);
  RGB(0, i, i);
  retardo(3);
  RGB(i, i, i);
  retardo(3);
  }

}
```

5. VECTORES

5.1. Transformar la hora introducida en sistema decimal, a sistema binario, encendiéndose los LED correspondientes a dicha hora.

```
#include <Arduino.h>
#include "Led.h"
//Transforma la hora introducida en sistema decimal
//en la hora en sistema binario.


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

void loop()
{
    hora (1,4,0,0);

}
```

5.2. Introducir la hora en números binarios, de manera que se encenderán los LED correspondientes a dicha hora.

```
#include <Arduino.h>
#include "Led.h"

//Representa la hora en numeros binarios

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

}

void loop()
{
    int a[2]={1,1};
    int b[4]={1,1,1,1};
    int c[3]={1,1,1};
    int d[4]={1,1,1,1};
    reloj(a,b,c,d);

}
```

5.3. Ampliación

Los ejercicios planteados en el apartado de vectores están diseñados para que el alumno, además de comprender el mecanismo de paso de vectores a funciones, se familiarice con el sistema binario. Para alumnos más avanzados en programación puede ser interesante que sean ellos mismos quienes programen las funciones hora y reloj.