Apéndice: Introducción a la programación en C++

Veamos una breve introducción a la programación, aplicada a Arduino pero que es común con muchos lenguajes en concreto C++

Comentarios

Un comentario es una línea que no influye en nuestro programa pero que nos sirve para documentar nuestro código. Es muy importante comentar nuestro código, puesto que nos ayudará a entender lo que estamos haciendo y si pensamos compartirlo con más gente, les facilitará enormemente la labor de comprender lo que pretendemos hacer.

Existen 2 tipos de comentarios:

- El comentario de bloque, todo el contenido entre los símbolos / *y* / será tomado como contenido del comentario

```
/* Hola esto es un comentario tan largo y extenso que no
una sola línea. */
```

- Línea de comentario: a partir de las dos barras // se ignorará el contenido hasta la siguiente línea

```
// Este es un comentario de línea
```

Bloques

Un bloque es un conjunto de líneas de código que se encuentran encerradas entre dos llaves\

```
    { … }
```

Más adelante veremos que determinadas sentencias de control incluyen bloques de instrucciones, como por ejemplo las instrucciones de control o las de decisión\

Variables

Una variables es una forma de etiquetar y guardar un valor. El valor que contiene puede ser de diferentes tipos, como puede ser un número entero, un número decimal, o un carácter.

Las variables conservan su valor hasta que lo modificamos. Podemos modificar su valor y a partir de ese momento contendrá ese nuevo valor.

El tipo de la variable será definido cuando declaramos la variable por primera vez y a partir de ahí no se modificará. Definiremos la asignación como el momento en que le damos valor a la variable. Podemos declarar y asignar un valor en la misma sentencia. Veamos ejemplos de declaraciones y asignaciones:

```
int a=10;
float c=10.2;
int b;
b=15;
```

Las variables tiene un ámbito (scope en inglés) que define la zona del código donde existen. Fuera de esta zona no podemos acceder a su

valor.

```
int a=10;
void prueba()  {
    float b=10.2*a;
}
```

En este ejemplo la variable *b* sólo está definida dentro de la función prueba, es decir, no podremos usarla fuera de la misma. En cambio *a*, al estar definida fuera, también puede ser usada dentro.

Puedes encontrar más ejemplos en la página de Arduino.

Tipos de variables

Veamos algunos de los tipos de variables existentes. Cada tipo determinado, tiene un rango y una precisión:

```
byte: 8bits 0 a 255
int: 16bits -32768 a 32767
word: 16bits 0 a 65535
long: 32bits -2x10\^6 a 2x10\^6
float: guarda decimales -3.4x10\^34 a 3.4x10\^34
```

¡¡¡Cuidado con los números negativos!!! ¡¡Cuidado con pasarnos!!

Cuando una variable llega a su límite, tanto superior, como inferior se produce un desbordamiento y continua por el otro límite. Por ejemplo si tenemos una variable de tipo word con valor 0 y le restamos 1 ¡¡¡pasará a

tener el valor 65535!!!

Operadores aritméticos

Representa la operación que se realizará entre los valores de la expresión:

```
+, -, \*, /, % (módulo, el resto de la división entre los va
```

Ejemplos:

```
a=a+3;
b=a/3;
c=b%3;
```

Operadores

- Operadores Booleanos: && (AND / Y), || (OR / O), !(NOT / NO) Representan las operaciones lógicas que podemos aplicar a las condiciones
- Operadores de acceso a punteros: *, & Los usaremos para acceder a memoria
- Operadores de bits: &, |, \^, ~, \<\<, >> Permiten manipular los bits individuales de las variables representadas en lenguaje binario
- Operadores compuestos:
- Incremento/decremento de variables: ++, -- Incrementan/decrementan el valor de la variable
- Asignación y operación: +=, -=, *=, /=, &=, |= Realizan la operación indicada y luego la asignación del resultado a la variable

Array

Un array es una agrupación de variables con un tipo y un tamaño determinados. Podemos acceder a cada elemento individualmente por medio del operador [], indicando la posición. Empiezan en la posición 0 y la última será la que indica su longitud menos 1.

```
int miArray[5]; //Declaramos un array de 5 elementos enteros
int miOtroArray[]={1,23,2}; // Declaramos un array de 3 eleme
```

Constantes

Son variables a las que no podemos modificar al valor. Algunos ejemplos:

```
true/false
HIGH/LOW
INPUT/OUTPUT
```

Conviene usarlas para evitar confusiones.

Funciones

Una función es un conjunto de instrucciones agrupadas para un nombre, al que le pasaremos unos argumentos y devolverá un valor.

¿Por qué usamos funciones?

- Cuando repetimos un fragmento de código

- Ahorro de espacio
- Estructuración

El formato siempre es:

```
tipo funcion(tipo argumento 1, tipo argumento 2)  {
  // Codigo
}
```

Librerías

Cuando vamos a reutilizar varias veces un código puede ser buena idea que creemos una librería, que no es más que una colección de código que empaquetamos de una manera concreta y cuyas funciones podemos utilizar sin más que incluirlas en nuestro código. En el ejemplo se incluye la librería serie que se usa para comunicar el pc con Arduino.

```
#include <Serial.h>
```

Existen muchas librerías cuyo uso iremos viendo a lo largo del curso

Estructuras de control

Son aquellas estructuras que nos permiten modificar el flujo de ejecución de nuestro programa, es decir el orden en el que se ejecutan las sentencias.

Sentencias condicionales

Permiten decidir si se ejecutan o no, unas sentencias en fnción de que se cumplan determinadas condiciones

Sentencia if

Ejecutaremos un código si se cumple la condición y otro distinto si no se cumple

```
if (condición){
// código si se cumple
}
else{
// código si no se cumple
}
```

Ejemplo

```
if ((valorEntrada < 500) && (valorEntrada>100)){
    // código A
}
else{
  // código B
}
```

Sentencia switch

Dependiendo del valor de una variable ejecutaremos un código distinto

```
switch (var) {
    case 1:
        //hacemos algo si var es 1
        break;
    case 2:
```

```
        //hacemos algo si var es 2
        break;
    default:
        // si nada concuerda, default
        // default es opcional
    }
```

Bucles

Realizamos un bucle siempre que queremos que se repita un determinado fragmento de código. Todo bucle tiene una condición que determinará las veces que se repetirá el código.

Existen 3 tipos de bucles:

Bucle for

Es el más natural para usar cuando la iteración tiene un número claro de repeticiones:

```
for (inicializacion; condicion; incremento) {
    //sentencia(s);
}
```

por ejemplo

```
for (int i=0;i\<20;i=i+1) {
    //sentencia(s);
}
```

que se realizará 20 veces

Bucle while

El bucle while ejecutará las sentencias de su bloque mientras que la condición inicial se cumpla

```
while(expresion){
    // sentencia(s);
}
```

Si la condición no se cumple inicialmente no se realizará ninguna iteración.

Bucle do

En este bucle la comprobación se hace después de la iteración, por lo que tenemos garantizado que al menos se ejecutará una vez la iteración

```
do    {
//sentencia(s)
} while (condicion);
```

¿Como salir de los bucles?

De un bucle saldremos cuando no se verifique la condición, pero también podemos formar la salida usando las siguientes instrucciones

- break // sale del bucle
- continue // salta el paso actual del bucle
- return // sale de la función

- goto label //salta a la etiqueta *label*

```
    for(int i=0;i\<10;i++){
        if (bsalto\>0)
            continue; // Se salta el código del resto de la i
        else
            break; //sale del bucle
    }
```

String: o cadenas de caracteres

Para facilitar el trabajo con cadenas de caracteres, existe la clase String. Tiene (entre otros muchos) los siguientes métodos:

- compareTo() //compara dos cadenas
- concat() // concatena
- endsWith() // comprueba si termina por...
- equals() //comprueba si son iguales
- equalsIgnoreCase() // comprueba igualdad ignorando mayúsculas y minúsculas
- indexOf() // devuelve la primera posición del carácter que le pasamos
- lastIndexOf() // devuelve la última posición del carácter que le pasamos
- length() // longitud de la cadena

- replace() // reemplaza
- startsWith() // comprueba si empieza por
- substring() // devuelve un fragmento de la cadena original
- toLowerCase() // convierte a minúsculas
- toUpperCase() // convierte a mayúsculas
- trim() // elimina los espacios iniciales y finales

Conversiones de tipos

Muchas veces es necesario convertir valores de variables de tipos diferentes. Para eso existen en Arduino un conjunto de funciones que nos permiten hacer estas transformaciones con garantías:

```
char() // Convierte a un carácterer
byte() // Convierte a un byte
int() // Convierte a un int
word() // Convierte a un word
long() // Convierte a un long
float() // Convierte a un número decimal
```

Por ejemplo, veamos como convierte a entero. Obviamente perdemos la parte decimal.

```
float a=2.4;
int b=int(a);
```

Funciones matemáticas

Aunque el procesador de Arduino no excesivamente potente en lo que a cálculo matemático se refiere, existen librerías que nos permiten hacer la mayoría de las funciones matemáticas usuales:\

Matemáticas

- min(a,b) // mínimo de a y b
- max(a,b) // máximo de a y b
- abs(a) // valor absoluto de a
- pow(a,b) // devuelve a elevado a b
- sqrt(a) // devuelve la raiz cuadrada de a

Trigonometría

- sin(a) // calcula el seno de a
- cos(a) // calcula el coseno de a
- tan(a) // calcula la tangente de a

Números aleatorios

- randomSeed(a) // inicializa los número aleatorios con a
- random() // devuelve un número aleatorio

Obviamente, dada la poca capacidad de cálculo de Arduino, el usar estas funciones matemáticas complejas es muy costoso computacionalmente, por lo que se debería de evitar u optimizar su uso.

Introducción a la programación de objetos

Un objeto es un conjunto de funciones (métodos) y variables (atributos) que trataremos de una forma global.

El trabajar con objetos nos permite encapsular (esconder las interioridades), mostrando un interface (una vista) con lo que permitimos que otros vean (atributos públicos) y usen (métodos públicos) de nosotros.

A los que los usamos nos facilitan el trabajo al agrupar las funcionalidades.

Por ejemplo el objeto Serial, permite trabajar con todo lo relacionado con el manejo de los puertos serie:

- Serial.print enviará por el puerto serie
- Serial.read leerá por el puerto serie
- Serial.write escribirá...

La mayoría de las librerías de Arduino están formadas por objetos.

¿Donde encontrar más información?

- Arduino
- Arduino programing notebook (inglés)
- Guía de programación de Arduino (español)
- Freeduino page