Curso Programación Intermedia de Arduino

(Máquinas de Estado)

Tomás de Camino Beck, Ph.D.

tomasdecamino.com

Maquinas de Estado

Tomás de Camino Beck, Ph.D.

¿Qué es?

Un autómata de estado finito (AEF) o máquina de estados, es un modelo matemático simple de computación. A pesar de ser simple es extremadamente útil, no solamente para entender cómo las computadoras funcionan, sino como una manera de programar sistemas automáticos que hacen tareas repetitivas determinísticas, si se que suena pretencioso, pero básicamente permite entender las máquinas que hacen tareas repetitivas, y poder construir programas de computadora que lo hagan.

Un AEF tiene los siguientes elementos:

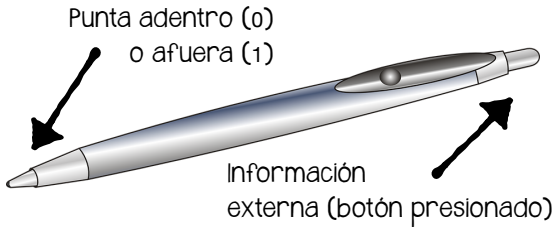
- Tiene unos estados internos que gobiernan cómo funciona la máquina
- Recibe información de entrada
- Dependiendo de la información de entrada y sus estados internos, genera un salida.

El Lapicero como un AEF

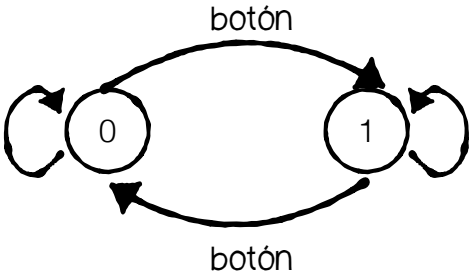
Pensemos en un lapicero con punta retráctil como una máquina. Sabemos bien que en un lapicero para meter o sacar la punta debemos apretar un botón.

Así entonces, si tuviéramos que describir un poco más formalmente este proceso, podríamos decir,

- Hay dos estados, es decir, punta adentro, digamos estado 0, y punta afuera digamos estado 1.
- Presionar el botón es la información de entrada. Cada vez que se aprieta el botón, algo ocurre



Si construimos un esquema de lo que sucede, básicamente tenemos una máquina de dos estados que recibe información de un botón (presionado o no),
Si la máquina esta en estado 0, se quedará así hasta que el botón sea presionado, y pasa al estado 1. De igual manera se queda



en estado 1 hasta que el botón sea presionado. En este caso entendemos el "loop" como "quedarse" sin cambio, pero se puede omitir.

Comentario final

Un AEF es una abstracción que permite representar procesos como cambios de estado en una máquina basada en información. Los diagramas de "grafos" (bolas y flechas), nos permiten representar gráficamente a los AEF

Datos Estructurados

Tomás de Camino Beck, Ph.D.

¿Qué es?

Todos conocemos los tipos de variable que se pueden declarar en C/C++, estos incluyen **`int`**, **`float`**, **`boolean`**, etc. Estos tipos funcionan muy bien cuando las variables no están relacionadas entre si. Pero imaginemos ahora la situación que queremos una variable que contiene dos coordenadas x,y, ¿cómo las puedo mantener como una sola? es allí donde utilizamos datos estructurados con el comando **`struct`** de C/C++.

Código en C++

digamos ue queremos definir un tipo de dato para puntos en dos dimensiones. construimos el dato de la siguiente manera.

```
1 struct punto2D{
2  float x,y;
3 };
```

Noten como usamos el comando **`struct`**, y le asignamos un nombre a la variable. En este ejemplo la variable punto contiene dos variables de punto flotante que las llamamos `x` y `y`. Eso es todo!

Ahora, para declarar la variable lo hacemos al igual que declaramos otra variables,

```
6 punto2D mypunto;
```

Ahora para acceder a los datos de la estructura, utilizamos el punto de la siguiente manera,

```
13 void loop() {
14   mypunto.x = (float)analogRead(A0);
15   mypunto.y = (float)analogRead(A1);
16 }
```

así de simple! Veamos el código completo. En este ejemplo asumimos que los pines analógicos leen alguna información de posición con algún sensor

```
1  struct punto2D{
2   float x,y;
3  };
4
5
6  punto2D mypunto;
7
8  void setup() {
9    pinMode(A0,INPUT);
10   pinMode(A1,INPUT);
11 }
12
13 void loop() {
14   mypunto.x = (float)analogRead(A0);
15   mypunto.y = (float)analogRead(A1);
16 }
```

Comentario final

Hay varias ventajas de hacer esto. Por ejemplo con el ejemplo anterior, podemos definir funciones que tomen datos de tipo punto2D, en lugar de tener que estar pasando los valores x y y de forma separada. Asimismo hace que el código sea más legible, pues al definirlos es claro que nos estamos refiriendo a un punto de dos dimensiones. Adicionalmente puede ser muy práctico para declarar un arreglo de puntos en dos dimensiones

Punteros

Tomás de Camino Beck, Ph.D.

¿Qué es?

Un de las características más poderosas y novedosas cuando apareció C como lenguaje de programación eran los punteros (o apuntadores).

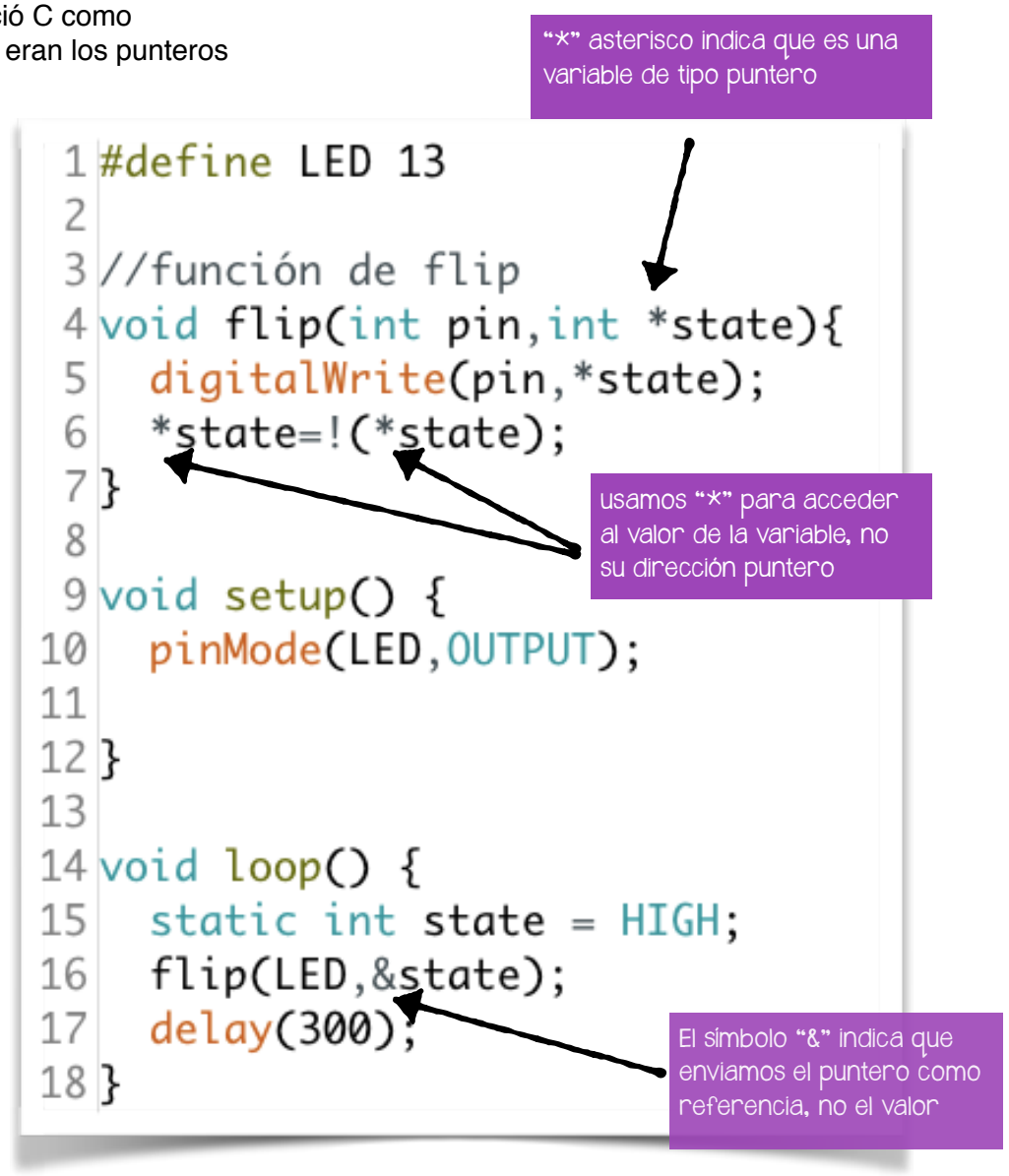
Los punteros son variables que en lugar de contener un valor directamente, contiene la dirección de memoria donde fue almacenado. Veámoslo como un sistema de casilleros. Digamos que un amigo tiene guardado en un casillero unos zapatos que son míos. Mi amigo me puede pasar directamente los zapatos, o en su lugar pasarme el número de casillero.

En la mayoría de textos verán explicaciones completas sobre direcciones de memoria y cosas de esas, acá esto no me interesa, solo me voy a concentrar en la flexibilidad que dan sobre todo para construir estructuras en memoria en tiempo de ejecución.

Código en C++

La mejor forma de ilustrarlo es directamente en código. Hay varios símbolos que se utilizan para punteros, y esto es a lo que muchos temen. Acá en este ejemplo voy a usar "*" y "&"

En las explicaciones verán que usamos variables por valor, es decir nos interesa el valor almacenado de la variable, y variables por referencia, es decir nos interesa la dirección donde está almacenado




```
1  #define LED 13
2
3  //función de flip
4  void flip(int pin,int *state){
5    digitalWrite(pin,*state);
6    *state=!(*state);
7  }
8
9  void setup() {
10   pinMode(LED,OUTPUT);
11
12 }
13
14 void loop() {
15   static int state = HIGH;
16   flip(LED,&state);
17   delay(300);
18 }
```

Comentario final

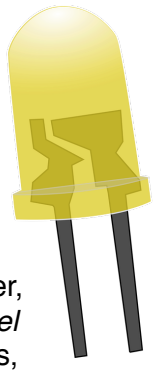
Hay muchas circunstancias en programación avanzada por las cuales utilizamos punteros, pero no se desesperen si no los están utilizando, creo que serán claras las situaciones donde las ocupan.

Objetos C++

Tomás de Camino Beck, Ph.D.

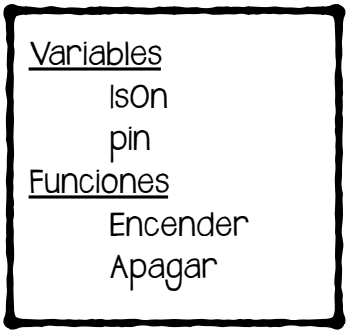
¿Qué es?

En C++ es posible pensar en porciones de códigos como objetos. Un objeto es una estructura de datos que contiene atributos y comportamientos (funciones y procedimiento).



Pensemos en un LED. Si lo vemos como un objeto, un LED, como un objeto, tiene como atributo estar encendido o apagado, que lo podemos llamar *isOn*, y también conocemos dos acciones o comportamientos que puede tener, que es *prender el LED* o *apagar el LED*. En caso de Arduino además, un led está asociado a un número de *pin,*

Objeto LED



Código en C++

Veamos un código simple construido para abstraer un LED como un objeto en C++

Noten como el “constructor” (linea 6 a la 9) que es e que crea el objeto en memoria cuando se instancia en la computadora, es un procedimiento que lleva el mismo nombre asignado de la clase creada (linea 1).

Define un objeto

asignamos nombre

```
1  class TLed{
2    bool isOn= false;
3    int p = 13;
4
5    public:
6    TLed(int pin){
7      p  = pin;
8      pinMode(p, OUTPUT);
9    }
10
11   void turnOn(){
12     isOn=true;
13     digitalWrite(p,HIGH);
14   }
15
16   void turnOff(){
17     isOn=false;
18     digitalWrite(p,LOW);
19   }
20 };
```

Variables que son atributos del objeto

Constructor

operaciones del objeto

Cuando se instancia, digamos con el nombre *myled* el objeto se declara de la siguiente manera

```
TLed myled(13);
```

Comentario final

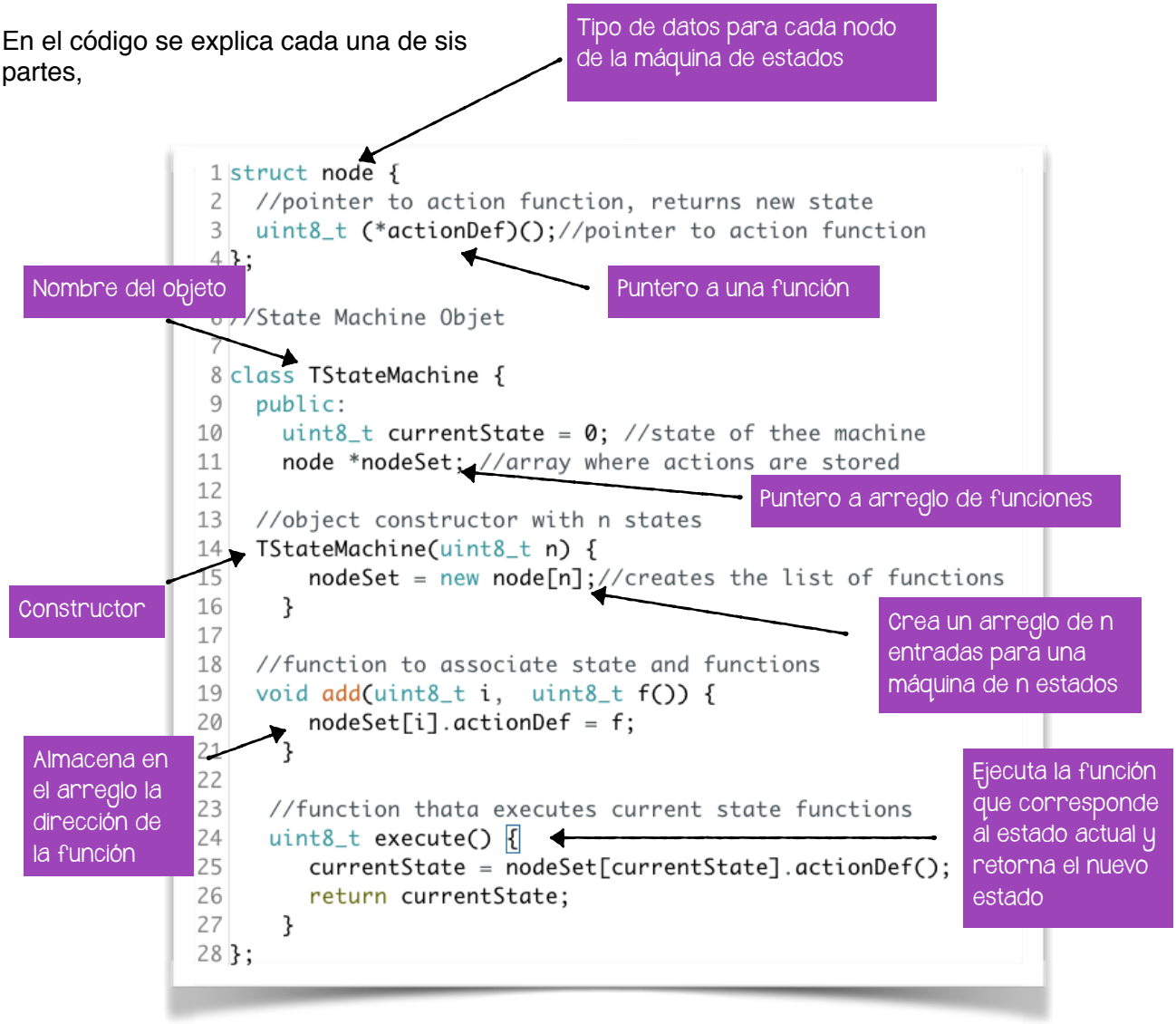
La orientación a objetos lleva consigo bastante teoría que no voy a explicar. Lo que les mostré es suficiente para comenzar a crear objeto, pero les recomiendo leer libros un poco más avanzados sobre el tema.

Código de Máquina de Estados

Tomás de Camino Beck, Ph.D.

Código en C++

En el código se explica cada una de sis partes,




```
1  struct node {
2    //pointer to action function, returns new state
3    uint8_t (*actionDef)();//pointer to action function
4  };
   //State Machine Objet
7
8  class TStateMachine {
9    public:
10     uint8_t currentState = 0; //state of thee machine
11     node *nodeSet; //array where actions are stored
12
13   //object constructor with n states
14   TStateMachine(uint8_t n) {
15       nodeSet = new node[n];//creates the list of functions
16     }
17
18   //function to associate state and functions
19   void add(uint8_t i,  uint8_t f()) {
20       nodeSet[i].actionDef = f;
21     }
22
23    //function thata executes current state functions
24    uint8_t execute() {
25       currentState = nodeSet[currentState].actionDef();
26       return currentState;
27     }
28 };
```

Comentario final

Cuando se utiliza el objeto, la única condición es que las funciones que se construyan retornen el nuevo estado luego de aplicar la función.