Informe sobre el software

Medidor de CO2: Aire Nuevo WIFI

Lenguaje de programación

Se está utilizando el lenguaje de programación Arduino para programar las placas, mediante el entorno de desarrollo oficial de Arduino que se puede descargar en https://www.arduino.cc/en/software

Preparando el IDE para usar NodeMCU/esp8266

- Se debe abrir el IDE e ir a preferencias. Agregar el siguiente enlace https://arduino.esp8266.com/stable/package_esp8266com_index.json a la sección de Additional Board Manager URLs.
- Abrir el Boards Manager desde el menú de Placas en las herramientas, buscar ESP8266 e instalar el paquete de ESP8266 Community
- Ir al Library Manager desde el menú de Sketch e instalar el paquete ESP8266 Microgear de Chavee Issariyapat.
- Seleccionar la placa ESP8266 Boards (3.0.2) - Generic ESP8266 Module.

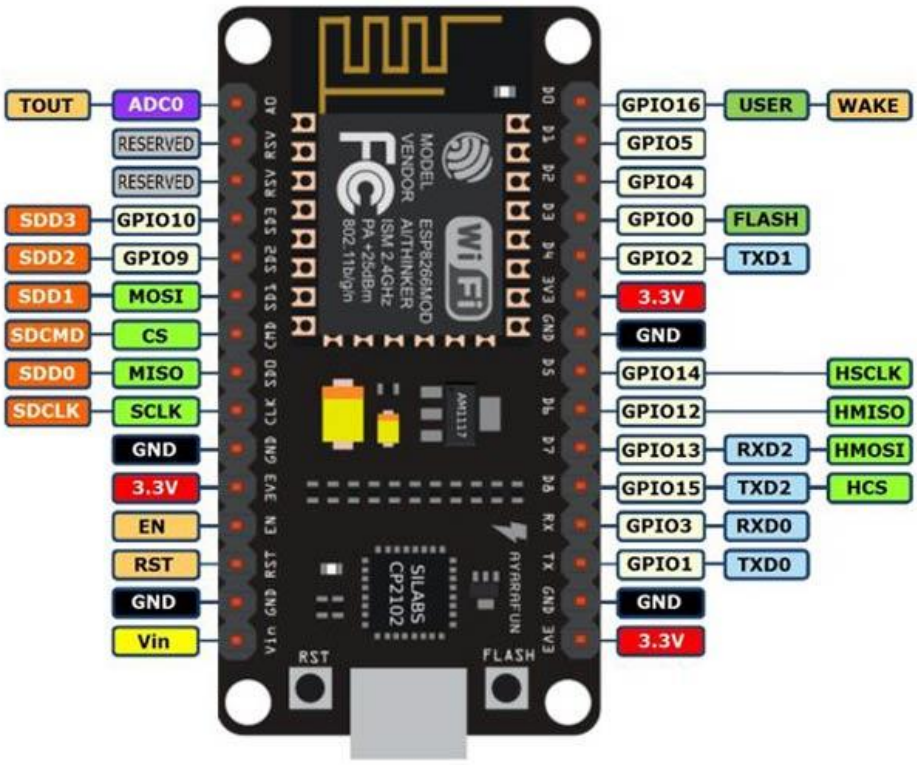
Librerías utilizadas

- Para el uso del sensor **MH-Z19C** se está utilizando la librería **mhz19_uart** que se puede descargar desde el siguiente enlace https://github.com/piot-jp-Team/mhz19_uart
- Para el uso del display **LCD LiquidCrystal I2C** se está utilizando la librería **Arduino-LiquidCrystal-I2C-library** que se puede descargar desde el siguiente enlace https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-library
- Para el uso del WIFI van a ser necesarias varias librerías la principal es **ESP8266WIFI** que viene incluida con **el link del primer paso de preparando el IDE.**
- Para la conexión como **Punto de Acceso** se utiliza la librería **WiFiManager** que se puede descargar desde el IDE de Arduino o desde el siguiente enlace https://github.com/tzapu/WiFiManager.
- Para la conexión con la API ThingSpeak se utiliza la librería **ThingSpeak,** en el IDE de Arduino elija Sketch / Incluir biblioteca / Administrar bibliotecas. Haga clic en la biblioteca ThingSpeak de la lista y haga clic en el botón Instalar.

Se recomienda descargar las librerías utilizadas desde los enlaces y no desde el gestor de bibliotecas del **IDE** porque existen varias librerías distintas con nombres iguales y distintos funcionamientos.

Placa utilizada

Actualmente se está utilizando la placa **NodeMCU V1.0 ESP8266 (ESP-12E)** a continuación, se muestra el diagrama de las salidas que posee.



Driver del microcontrolador

Para usar el NodeMCU es necesario instalar el driver del chip adaptador de serial a USB. Esta placa dispone de un chip CP2102 como conversor serial a USB y por lo tanto deberemos usar el driver de este chip y que podemos encontrar su última versión en https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers.

Funcionamiento del código

El funcionamiento del código consiste en tres partes. La creación de un objeto medidor (**medidor.h**), la definición de funciones utilizadas (**medidor.cpp**) y el programa general que hace funcionar todo (**aire_nuevo**).

• Medidor.h

Si el medidor no está definido se define e incluye la librería de arduino.
Se define el largo del string, y la nota que va a utilizar el buzzer.

```
#ifndef MEDIDOR_H
#define MEDIDOR_H
#include <Arduino.h>

#define STR_LEN 17
#define NOTE_C7  2093
```

Se crea la clase Medidor y se setean las variables que se van a utilizar.

```
class Medidor {
private:
  byte rx_pin;
  byte tx_pin;
  byte ledR_pin;
  byte ledG_pin;
  byte ledB_pin;
  byte buzzer_pin;
  byte pulsador_pin;
  String numero_de_serie;

  char str_to_print[STR_LEN]={'A','i','r','e',' ','N','u','e','v','o',' ','W','i','f','i'};
  void displayPrint(int posicion, int linea, String texto);
  void scrollingText(uint8_t scrolled_by);
```

A continuación, define las funciones que va a utilizar:

```
public:
  Medidor(void);
  void imprimirCO2(int co2ppm);
  void logoUNAHUR();
  void scrollAireNuevo();
  void calibrar();
  void rgb(char color);
  void sonarAlarma(int duracionNota);
  void alarma(int veces, int duracionNota, char color);
  void alarmaCO2(int veces, int duracionNota);
  void conectar();
  void reconectar();
  void iniciar();
  void calentar();
  void verificarEstadoPulsador();
  void presentarMedidor();
  void sensarCO2();
};
```

• Medidor.cpp

Primero incluimos las librerías que vamos a utilizar:

- **MHZ19_uart.h** permite controlar el sensor
- **Wire.h** es necesaria para la librería que maneja el display
- **LiquidCrystal_I2C.h** como se mencionó antes, es la librería que nos permite controlar el display
- **WiFiManager.h** controla la conexión WIFI ya sea por Punto de acceso o como reconexión automática
- **ESP8266WiFi.h** permite controlar el modulo WIFI de la placa NodeMCU
- **ThingSpeak.h** permite la conexión y envío de datos a la API ThingSpeak

```
#include <MHZ19_uart.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <WiFiManager.h>
#include <ESP8266WiFi.h>
#include <ThingSpeak.h>
```

Definimos un nombre para la librería del sensor con el cual luego la llamaremos, después definimos el display a utilizar y luego establecemos el wifi como cliente. Por último, definimos un nombre para utilizar más adelante WifiManager.

```
MHZ19_uart sensor;
LiquidCrystal_I2C display(0x27,16,2);
WiFiClient client;
WiFiManager wifiManager;
```

Para la conexión con la API es necesario obtener los datos del canal que se quiere escribir

```
unsigned long myChannelNumber = Número de canal de thinkspeak;
const char * myWriteAPIKey = " API key de thinkspeak ";
```

Luego definimos las constantes para los distintos pines.

- **rx_pin** y **tx_pin** son los pines en los que está conectado el sensor

- **pinLed** es el pin donde está conectado un led simple de color rojo
- **pinBuzzer** es el pin en el cual está conectado el buzzer
- **pinCalib** es el pin en el que está conectado un pulsador que se utiliza para la calibración del dispositivo
- **numeroSerie** contiene el número de serie del medidor, este número se imprime tanto por serial como por el display

```
const int rx_pin = 14;
const int tx_pin = 12;
ledR_pin = 02;
ledG_pin = 10;
ledB_pin = 16;
buzzer_pin = 13;
pulsador_pin = 15;
numero_de_serie = "0000";
```

Para referirse a los pines hay que usar la nomenclatura GPIO

En “Medidor::conectar()” tenemos la configuración de los pines del led, el buzzer y se inicia el display. Luego se enciende el led Rojo y suena la alarma de encendido.

El display imprime los mensajes **"Buscando Señal WIFI Punto De Acceso"** y **WifiManager.**autoConnect comienza las rutinas de conexión, intentará auto conectarse a la última **red guardada** y si no encuentra una conexión disponible éste pasa a **Punto de Acceso.** En caso de encontrar una conexión disponible el display indicará un mensaje de **Conectando…** luego de 10 segundos indicará **Estás conectado**. Por último iniciamos **ThingSpeak.begin(client) “ThingSpeak en modo cliente”**.

```
void Medidor::conectar() {
  pinMode(ledR_pin, OUTPUT);
  pinMode(buzzer_pin, OUTPUT);
  rgb('a');
  display.begin();
  display.backlight();
  alarma(1, 250, 'r');

 //wifiManager.resetSettings(); //< Descomentar para resetear configuración

  Serial.println("Buscando Señal WIFI / Punto De Acceso");
  displayPrint(0, 0, "Buscando WIFI");
  displayPrint(0, 1, "Punto De Acceso");
  wifiManager.autoConnect("ESP8266Temp");
  display.clear();
  Serial.println("Conectando...");
  displayPrint(0, 0, "Conectando...");
  delay(10000);
```

```
  display.clear();
  Serial.println("Ya estás conectado");
  displayPrint(0, 0, "Estas Conectado");

  ThingSpeak.begin(client);
}
```

En “Medidor::reconectar()” comenzamos limpiando el display, en caso de perder la conexión Wifi comenzaría la rutina de reconexión, sonaría una alarma de 4 pitidos seguido del led encendiendo en color Rojo, el display mostraría el texto **"No hay conexión Buscando WIFI"** durante 60 segundos, en caso de auto conectarse mostraría el mensaje **"Conectado"** junto con 2 pitidos y una luz **led Verde**, en caso contrario mostraría el mensaje **"No hay conexión Reiniciar"** buscando que se reinicie el medidor o que se ingrese al punto de acceso y se reconecte mediante la contraseña del Wifi requerido.

```
void Medidor::reconectar() {
 display.clear();
 while(WiFi.status() != WL_CONNECTED){
  alarma(4, 250, 'r');
  Serial.println("No hay conexión Buscando WIFI");
  displayPrint(0, 0, "No hay Conexion");
  displayPrint(0, 1, "Buscando WIFI");
  delay(60000);
  display.clear();
  Serial.println("No hay conexión Reiniciar");
  displayPrint(0, 0, "No hay Conexion");
  displayPrint(0, 1, "Reiniciar");
  wifiManager.autoConnect("ESP8266Temp");
  display.clear();
  alarma(2, 250, 'g');
  Serial.println("Conectado");
  displayPrint(0, 0, "Conectado");
  delay(5000);
  }
}
```

En “Medidor::iniciar()” tenemos la **configuración de los pines** del **led**, el **buzzer** y el **pulsador para la calibración**. El **led** y el **buzzer** están como **salida** y el **pulsador** como **INPUT_PULLUP**, esto básicamente indica que en ese pin tenemos un pulsador para que se pueda sensar cuando está siendo presionado.

Luego **limpiamos la pantalla** y **prendemos la luz trasera** de esta. Suena una **alarma** indicando que el dispositivo se encendió y se muestra el **número de serie** y un cartel indicando que el equipo está

iniciando. Esto se muestra tanto por serial como por pantalla. Imprimimos en pantalla el **logo de la UNAHUR** y luego de 10 segundos llamamos a la función “calentar()”.

```
void Medidor::iniciar() {
  pinMode(ledR_pin, OUTPUT);
  pinMode(ledG_pin, OUTPUT);
  pinMode(ledB_pin, OUTPUT);
  rgb('a');
  pinMode(buzzer_pin, OUTPUT);
  pinMode(pulsador_pin, INPUT);
  display.begin();
  display.clear();
  display.backlight();
  alarma(1, 250, 'b');
  Serial.print("N° de serie " + numero_de_serie + "\n");
  Serial.print("INICIANDO \n");
  displayPrint(0, 0, "N/S: " + numero_de_serie);
  displayPrint(0, 1, "INICIANDO");
  logoUNAHUR();
  delay(10000);
  calentar();
}
```

Limpiamos la pantalla e **iniciamos** el **sensor** indicándole los pines en los que está conectado. **Imprimimos** tanto por serial como por pantalla un mensaje indicando que el dispositivo se **está calentando**, esto es porque el sensor necesita **un minuto** de calentamiento para poder empezar a funcionar **correctamente**. Una vez pasado este minuto, se vuelve a limpiar la pantalla y suena una **alarma** de **tres** pitidos indicando que ya **terminó el calentamiento**.

```
void Medidor::calentar() {
  display.clear();
  sensor.begin(rx_pin, tx_pin);
  sensor.setAutoCalibration(false);
  Serial.print("Calentando, espere 1 minuto \n");
  displayPrint(0, 0, "Calentando");
  displayPrint(0, 1, "Espere 1 minuto");
  delay(60000);
  display.clear();
  alarma(3, 250,'g');
}
```

La función “**verificarEstadoPulsador()**” chequea si el pulsador está siendo presionado. En caso afirmativo hace sonar una alarma y llama a la función “**calibrar()**”.

```
void Medidor::verificarEstadoPulsador() {
```

```
  if (digitalRead(pulsador_pin) == HIGH) {
    alarma(1, 250, 'b');
    calibrar();
  }
}
```

La función “**presentarMedidor()**” imprime tanto por serial, como por display los mensajes “AireNuevo UNAHUR” y “MEDIDOR de CO2”.

```
void Medidor::presentarMedidor() {
  Serial.print("AireNuevo UNAHUR \n");
  Serial.print("MEDIDOR de CO2 \n");
  display.clear();
  displayPrint(0, 0, "AireNuevo UNAHUR");
  displayPrint(0, 1, "MEDIDOR de CO2");
  delay(5000);
}
```

La función “sensarCO2()” se encarga de medir el co2 en el ambiente y dependiendo el resultado decide si tiene que hacer sonar la alarma, la frecuencia, y el color del led.
Si la concentración de co2 es menor a 600 el led permanecerá verde indicando que los niveles están bien, mientras que de 600 a 800 el led se volverá amarillo avisando que se está alcanzando el límite de concentración de co2 y se recomienda ventilar el ambiente.
Si la concentración que se encuentra es de 800 o mayor se prende un led rojo y la alarma va a ir sonando cada vez con más frecuencia. Llegado el caso que al sensar la concentración es mayor o igual 1200ppm la alarma va a hacer un sonido intermitente para alertar que se debe abandonar y ventilar el lugar con urgencia.

```
void Medidor::sensarCO2() {
  display.clear();
  displayPrint(0, 0, "Aire Nuevo");
  while(sensor.getPPM() >= 1200) {
    alarmaCO2(1, 250);
    imprimirCO2(sensor.getPPM());
  }
  int co2ppm = sensor.getPPM();
  imprimirCO2(co2ppm);
  if(co2ppm >= 1000){
    alarmaCO2(4, 500);
  }
  else if(co2ppm >= 800) {
    alarmaCO2(2, 1000);
```

```
  }
  else if(co2ppm >= 600) {
    rgb('y');
  }
  else if(co2ppm < 600) {
    rgb('g');
  }
  displayPrint (13,1," ");
  displayPrint (14,1," ");
  displayPrint (15,1," ");
  scrollAireNuevo();
  delay(5000);
}
```

"**displayPrint (int posicion, int linea, String texto)**" usa 3 parámetros para indicar que se va a imprimir en el display, en que línea, y en qué posición del display.

```
void Medidor::displayPrint(int posicion, int linea, String texto) {
  display.setCursor(posicion, linea);
  display.print(texto);
}
```

"**imprimirCO2(int co2ppm)**" utiliza la función "displayPrint (…)" y la variable generada en "sensarCO2()" para mostrar tanto en serial, como por display el valor de los ppm actuales.

Además de esto último también utiliza "ThingSpeak.setField(…)" para seleccionar un campo del canal de la API y luego utiliza "ThingSpeak.writeFields(… , …)" para escribir los ppm actuales en la API.

```
void Medidor::imprimirCO2(int co2ppm) {
  Serial.print("CO2: " + String(co2ppm) + "ppm \n");
  displayPrint(0, 1, "          ");
  displayPrint(0, 1, "CO2: " + String(co2ppm) + "ppm");
  logoUNAHUR();
  delay (5000);
  ThingSpeak.setField(1, co2ppm);
  if (co2ppm >0){
  int x = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
  if (x == 200) {
    Serial.println("Channel update successful.");
  }
  else {
    Serial.println("Problem updating channel. HTTP error code " + String(x));
  }
}
```

```
}
```

Esta función sirve para dibujar el **logo** de la **UNAHUR** en el **display**. Para ello primero creamos unos **caracteres personalizados** en **bytes**, luego posicionamos el **cursor** del **display** en las posiciones correspondientes y los escribimos, en este caso **no** es posible reutilizar la función **displayPrint()** porque para imprimir caracteres personalizados se debe utilizar la función **write()** de la librería del display, en cambio, en los mensajes regulares se utiliza **print()**

```
void logoUNAHUR() {
  byte UNAHUR1[] = {
    B11100,
    B11110,
    B11111,
    B11111,
    B11111,
    B11111,
    B01111,
    B00111
  };
  byte UNAHUR2[] = {
    B11111,
    B11111,
    B11111,
    B11111,
    B11111,
    B11111,
    B11111,
    B11111
  };
  byte UNAHUR3[] = {
    B00111,
    B01111,
    B11111,
    B11111,
    B11111,
    B11111,
    B11110,
    B11100
  };
  display.createChar(0, UNAHUR1);
  display.createChar(1, UNAHUR2);
  display.createChar(2, UNAHUR3);
  display.setCursor(13, 0);
  display.write(0);
  display.setCursor(14, 0);
  display.write(1);
  display.setCursor(15, 0);
```

```
  display.write(2);
  display.setCursor(13, 1);
  display.write(2);
  display.setCursor(14, 1);
  display.write(1);
  display.setCursor(15, 1);
  display.write(0);
}
```

Para el mensaje que va moviéndose tenemos que crear un **array** con los caracteres y dos funciones, **scrollingText()** se encarga de imprimir todos los caracteres una vez, y **aireNuevo()** se encarga de hacer esa impresión durante 11 veces para imprimir los 10 caracteres y que queden de nuevo en la posición inicial con un espacio en blanco al final.

```
void Medidor::scrollingText(uint8_t scrolled_by) {
  for (uint8_t i=0;i<16;i++) {
    display.setCursor(i,0);
    if (scrolled_by>=16) scrolled_by=0;
    if (scrolled_by<15) display.print(str_to_print[scrolled_by]);
    else display.print(' ');
    scrolled_by++;
  }
}
```

```
Void Medidor::scrollAireNuevo() {
  for (uint8_t i=0;i<STR_LEN;i++) {
    scrollingText(i);
    delay(500);
  }
}
```

La siguiente función la usamos para **calibrar** el equipo, primero definimos la cantidad de **segundos a esperar**, este número es el tiempo que el equipo va a esperar para ejecutar la **calibración**, lo **recomendado** por el fabricante es que sean **al menos 20 minutos**, nosotros esperamos **30 minutos** para estar más **seguros**. Este número se maneja en segundos asi que convertimos 30 minutos a **1800 segundos**.

Luego se imprime un mensaje notificando que **comienza la calibración** que se puede ver durante 10 segundos, luego empieza el proceso de calibración. **Mientras** los **segundos pasados** son **menores** a los **segundos a esperar** mencionados anteriormente, se va mostrando por **pantalla** y por **serial** cuantos minutos van transcurriendo, **cada un minuto** se muestra el **CO2** que está sensando en el momento.

Una vez **pasados los 30 minutos**, se ejecuta la función **calibrateZero()** de la librería del **sensor** y se notifica por **serial** y por **pantalla**, **luego** de un minuto se ejecuta **de nuevo** la misma función y notifica que se hizo una **segunda calibración**. Esto es para estar seguros en caso de que hubiera algún problema con la primera, **no es necesario** que pasen **30 minutos entre ambas calibraciones**, los 30 minutos **solo son necesarios desde que se inicia hasta la primera calibración.**

Cuando ambas calibraciones son ejecutadas, se notifica por **serial**, por **pantalla**, y suena una **alarma** de **cinco** pitidos indicando que el proceso fue **finalizado**.

```
void Medidor::calibrar()
{
  const long segundosEspera = 1800;
  long segundosPasados = 0;
  // Print por serial
  Serial.print("COMIENZA CALIBRACION \n");
  // Print por display
  display.clear();
  displayPrint(0, 0, "COMIENZA");
  displayPrint(0, 1, "CALIBRACION");
  delay(10000);

  while(segundosPasados <= segundosEspera) {
    if (++segundosPasados % 60 == 0) {
      Serial.print(String(segundosPasados / 60) + " minutos \n");
    }
    display.clear();
    displayPrint(0, 0, "CALIBRANDO");
    displayPrint(0, 1, String(segundosPasados / 60) + " minutos");
    delay(1000);
    }
  sensor.calibrateZero();
  Serial.print("PRIMERA CALIBRACION \n");
  display.clear();
  displayPrint(0, 0, "PRIMERA");
  displayPrint(0, 1, "CALIBRACION");
  alarma(1, 250, 'b');
  delay(60000);
  sensor.calibrateZero();
  Serial.print("SEGUNDA CALIBRACION \n");
  display.clear();
  displayPrint(0, 0, "SEGUNDA");
  displayPrint(0, 1, "CALIBRACION");
  alarma(1, 250, 'b');
  delay(10000);
  Serial.print("CALIBRACION TERMINADA \n");
  display.clear();
  displayPrint(0, 0, "CALIBRACION");
  displayPrint(0, 1, "TERMINADA");
  alarma(5, 250, 'g');
```

```
  delay(10000);
}
```

La función rgb se encarga de crear los colores que van a ser asignados al led. Los valores en LOW son los valores activos, mientras que los HIGH están desactivados. Esto se puede ver claramente en el caso a “apagado” donde los 3 colores están en modo HIGH.

```
void Medidor::rgb(char color) {
  switch (color) {
    case 'r': //red
      digitalWrite(ledR_pin, LOW);
      digitalWrite(ledG_pin, HIGH);
      digitalWrite(ledB_pin, HIGH);
      break;
    case 'g': //green
      digitalWrite(ledR_pin, HIGH);
      digitalWrite(ledG_pin, LOW);
      digitalWrite(ledB_pin, HIGH);
      break;
    case 'y': //yellow
      digitalWrite(ledR_pin, LOW);
      digitalWrite(ledG_pin, LOW);
      digitalWrite(ledB_pin, HIGH);
      break;
    case 'b': //blue
      digitalWrite(ledR_pin, HIGH);
      digitalWrite(ledG_pin, HIGH);
      digitalWrite(ledB_pin, LOW);
      break;
    case 'a': //apagado
      digitalWrite(ledR_pin, HIGH);
      digitalWrite(ledG_pin, HIGH);
      digitalWrite(ledB_pin, HIGH);
      break;
  }
}
```

La función “alarma” se utiliza para notificar cuando el medidor se enciende, cuando termino de calentar, y para notificar el comienzo de calibración.

```
void Medidor::alarma(int veces, int duracionNota, char color) {
  rgb(color);
  for(int i=0; i<veces; i++) {
    sonarAlarma(duracionNota);
    delay(duracionNota);
  }
}
```

Las funciones “alarmaCO2” y “sonarAlarma” se utilizan cuando al censar el ambiente se detecta un nivel elevado de CO2.

```
void Medidor::alarmaCO2(int veces, int duracionNota) {
  for(int i=0; i<veces; i++) {
    rgb('r');
    sonarAlarma(duracionNota);
    if(i<veces-1 or veces==1) {
      rgb('a');
    }
    delay(duracionNota);
  }
}
```

```
void Medidor::sonarAlarma(int duracionNota) {
  tone(buzzer_pin, NOTE_C7, duracionNota);
  delay(duracionNota);
  noTone(buzzer_pin);
}
```

• aire_nuevo

aire_nuevo es el centro de todo el programa. Se encarga de utilizar las funciones y objetos antes creados para hacer funcionar el medidor. Comienza por incluir el medidor, lo nombra como “medidor” y setea los loops en 0.

```
#include "Medidor.h"
Medidor medidor;
long loops = 0;
```

En el **setup()** se inicia el serial en 9600 baudios, luego se debe llama a la función “conectar” que comienza la rutina de conexión a una red wifi y luego de 2 segundos de finalizada la rutina de conexión llama a la función “iniciar” que configura si los pines son entrada o salida, se limpia la pantalla del display, etc. En el **loop()** está el **código principal** que se ejecuta en **bucle** mientras el dispositivo esté encendido.

```
void setup() {
 Serial.begin(9600);
 medidor.conectar();
 delay (2000);
 medidor.iniciar();
}
```

En el **loop** primero tenemos un **chequeo** para revisar el **estado de la conexión** con la función **“reconectar”** luego comienza el segundo chequeo para revisar el **estado del pulsador**, si está siendo **pulsado**, suena una **alarma** y se **ejecuta** la **calibración**.

Luego revisamos cuantas **veces** se ejecutó el **loop**, si fue **ejecutado 30 veces** se muestra por pantalla y serial, un **mensaje presentando al medidor.**

Después se limpia la pantalla. En la **linea superior** se imprime el mensaje **“Aire Nuevo”** y en la **línea inferior** se imprime el **CO2**, a la **derecha** de ambas líneas se dibuja el **logo** de la **UNAHUR**.

En caso de que el **CO2** exceda las **800ppm** (partes por millón) suena una **alarma** de **dos** pitidos **lentos**. Si el **CO2** excede los **1000ppm**, suena una alarma de **cuatro** pitidos un poco **más rápidos**. Si el **CO2** excedió los **1200ppm**, empieza a sonar un **pitido rápido** que **no cesa hasta que las partes por millón desciendan de los 1200**. Luego ejecuta el **mensaje animado** y se **espera 5 segundos** y para volver a ejecutar el loop. **Entre mediciones hay 10 segundos**, 5 segundos transcurren en el mensaje animado y 5 se esperan.

```
void loop() {
  medidor.reconectar();
  medidor.verificarEstadoPulsador();
  if(loops == 30) {
```

```
    medidor.presentarMedidor();
    loops = 0;
  }
  medidor.sensarCO2();
  loops++;
}
```

• ThingSpeak

Para la recopilación de **datos/PPM** en la nube utilizamos la **API ThingSpeak** que nos permite utilizar un espacio de un máximo de **4 canales gratuitos** los cuales **cada canal se dedica a recopilar los datos/PPM de un medidor a la vez**, o sea que **tenemos un espacio de hasta 4 medidores en la nube por cada usuario gratuito creado en ThingSpeak**, de ser necesario más espacio hay opciones pagas que brindan más espacio y más beneficios.

Para la creación del usuario hay que dirigirse al siguiente link https://thingspeak.com/ , una vez en la página ingresar a **Sign In**, crear una cuenta, seguir los pasos siguientes. Una vez creado el usuario elegir una cuenta para uso **personal, proyecto no comercial.** Ya está creada tu cuenta ThingSpeak gratis con posibilidad de crear y utilizar hasta 4 canales. Para la creación de un nuevo canal ingresar a **New Channel**, una vez dentro en el campo **Name** ingresar el ID del medidor, luego en **Field 1** colocar PPM, por último clickear en **Save Channel** para guardar el canal. Para una visualización numérica en el campo creado ir a **Add Widgets** y seleccionar **Numeric Display**.

Una vez configurado el usuario y el/los campos a utilizar sólo nos quedan buscar los datos identificadores del canal para poder conectar la API con el medidor en el código, éstos se encuentran en **Chanel ID: número de canal**, y en **API Keys** copiar la **Key** de **Write API Key** ya sólo queda reemplazarlas por sus variables correspondientes en el código.

• ACLARACIÓN IMPORTANTE

Es necesario aclarar que para el correcto funcionamiento del medidor **Aire Nuevo WIFI** hay que tener **obligatoriamente una conexión del tipo WIFI disponible**, ya que si no detecta ninguna conexión éste no va a funcionar. No importa si el router tiene conexión a internet o no ya que puede proporcionar conexión inalámbrica WIFI sin estar conectado a una red de internet, los beneficios de estar conectado a

internet son que el medidor va a recopilar los datos en una API y va a mostrar los datos/PPM en la nube en tiempo real.