Buenas prácticas de programación

Nombres de variables

No usar abreviaciones. Ejemplo: activacionCalefaccion

No usar caracteres especiales ni espacios

descripcionObjeto. Ejemplo: salidaCalefaccion

Nombres funciones

funcionObjeto

La primera palabra será un verbo en infinitivo. Ejemplo: activarCalefaccion

Interfaz o prototipo de la función: nombre, tipo de dato que devuelve, tipo de datos son los parámetros.

En el prototipo no es necesario ponerle nombre. No se suele poner, sino solo el tipo de dato.

Ejemplo: void activarSalida(int) 🡨Interfaz

En este caso no devuelve nada (void) y que le tengo que ingresar un int

Punteros, devolución de matrices,

Nombres de clases

Primera letra mayúscula. Ejemplo Serial…

Definiciones

Se escribe todo en mayúsculas por ser el único lugar donde vamos a poner definiciones y vamos a arrancar con un guion bajo.

Valores fijos (Constantes)

const int pinSalidaCalefaccion

Tipos de datos

Usar como están en la referencia de arduino. No en ANSI sino en arduino.

<https://www.arduino.cc/reference/en/>

Todas las cadenas de caracteres que sean fijas van con la *macro F*

Ejemplo: Serial.print(f(“*cadena de carácter*”)) 🡨 cuando manejamos un string, si no usamos f la cadena de caracteres se carga en la RAM la cual no es abundante. De esta forma queda cargada solo en la FLASH. EMPROM SRAM FLASH.

En la EMPROM se graba en el micro, se pueden guardar algunos valores como usuarios.

El programa se guarda en la FLASH, código firmware. Memoria del programa.

Memoria de dato, EMPROM (máx. 100mil escrituras) no volátil, SRAM es volátil.

FLASH es donde se graba el programa, se guardan las constantes. Arduino no te permite escribir en la FLASH. La FLASH ya tiene un programa (uploader) cargado para la comunicación con la PC.

Entero corto 16bit 2 instancias de procesamiento

Entero largo 32bit 4 instancias de procesamiento

Dos BUS DE DATOS: uno para conectar la FLASH y otro para el resto de los periféricos

Estructura de código

Todo el código se tiene que ir indentando (presionamos TAB cuando usamos un IF, por ejemplo).

Siempre usar llaves.

Estructura de archivos

El .ino quede lo más limpio posible. Sólo la estructura principal. No debería tener más de 50 líneas.

Archivos *.h* Son los *header*: definiciones de bibliotecas, funciones y clases.

Archivos *.cpp* Archivos fuente. Que es lo que hace cada función.

Si definimos clases, en cada .h habrá una clase. El nombre del archivo será igual que el de la clase. Clase pepito archivo pepito.h y el archivo con la lógica de la clase será pepito.cpp

El archivo .h es una buena forma de ver que funciones tengo disponible. Si quisiera saber la lógica necesito acceder al .cpp.

Tratar de que cada módulo tenga sus variables locales y que si fuese necesario se pasen como argumentos/parámetros a otro lado.

No usar variables globales, para no usar RAM(mayor rendimiento).

Puede inducir un error en el programa entero si hay varias funciones que la utilicen.

CHAR o STRING

Cadena de caracteres: es un vector

String es un objeto de una clase (usa más memoria que una cad. de char)

Ventaja del string, se acomoda a lo ingresado, busca en la memoria el espacio requerido. Usa muchos recursos y puede fragmentar la memoria.

La cadena de caracteres es la forma nativa de uso de letras. Se perfectamente cuanto de la memoria estoy usando.

Convenio: Usamos STRINGS

Definición de clase

Cada vez que definamos una clase: usar la *plantilla de definición*

Usar en cada .h

#ifndef \_NOMBRECLASE (Pregunta si está definido en él .ino)

#define \_NOMBRECLASE (define si no lo está)

.

. *definición de clase*

.

#enif

El numeral es una instrucción de preprocesador, previo a la compilación. Nunca se hace el #include con un .cpp

Función del .ino

La idea es que desde el .ino se invoque a cada función como pueden ser los sensores y que el trabajo se haga dentro de cada módulo.

Entorno

VSCode (Las librerías se pueden instalar localmente en el proyecto, tiene la ventaja de que tenes la versión estática de la misma, ya que puede cambiar)

Intelicense

PlatformIO (previa instalación del VSCode)

Comentarios

Si el código está bien escrito no debería llevar comentarios. Pero no se cumple.

Comentario de línea: //

Comentario de bloque \*/ …………. \*/

Hacer un comentario de bloque en cada .h con lo que el mismo va a hacer.

Fecha, versión.

Los comentarios de línea van en el propio código.