Taller Introducción a la Ingeniería Eléctrica
Robot y comunicaciones basadas en Microcontrolador Arduino

Instituto de Ingeniería Eléctrica

Taller 3: Funciones

3 de Setiembre de 2024

Agenda

¿Qué vimos la semana pasada?

¿Qué vimos la semana pasada?

- Estuvimos trabajando con el Arduino, un microcontrolador que maneja entradas analógicas y digitales y permite hacer muchas cosas a partir de eso.
- Nos focalizamos en el manejo de entradas digitales y analógicas, usando las salidas para prender y apagar leds, para sintonizar una frecuencia de parpadeo y generar el efecto dimmer.
- Aprendimos distintas estructuras de control.
- Aprendimos y repasamos conceptos varios trabajando con un display 7 segmentos y la cuenta regresiva.
- Idea de *sensar* el entorno y *actuar* en consecuencia.

Herramientas y conceptos importantes de programación: las funciones y el pasaje de parámetros.

Divide y vencerás!

Para qué?

Herramientas y conceptos importantes de programación: las funciones y el pasaje de parámetros.

Divide y vencerás!

Para qué?

Una *función* es un conjunto de declaraciones, definiciones, expresiones y sentencias que realizan una **tarea específica**, es decir, un código implementado para cumplir una tarea o tareas determinadas.

Formato general para definir una función

```
EspecificadorDeTipo nombreFuncion( listaParametros )
{
    variables locales;
    cOdigo de la funciOn;
    retorno de valores;
}
```

¿Por qué se deben utilizar? ¿Por qué conviene segmentar el código en funciones?

- El caso típico para crear un función es cuando se necesita realizar la **misma acción múltiples veces** dentro de un mismo programa.
- Las funciones ayudan al programador a ser **organizado**. Además, ayudan a conceptualizar el programa.
- Las funciones codifican una acción en un lugar, así que **sólo deben ser depuradas de errores una sola vez!!!**.
- Reducen las posibilidades de error.

Hay dos funciones que deben estar en todo sketch de Arduino: setup() y loop(). Las invoca el sistema.

Todas las demás funciones deben ser definidas fuera de las llaves de estas dos funciones. Hay funciones definidas en **bibliotecas** (*libraries*) de Arduino (*math, Serial, etc*).

La función delay(x) la cual ya ha sido invocada en los talleres, es una función implementada por Arduino.

FUNCIONES

Recordando: Estructura avanzada de un código

```
inclusiOn de bibliotecas particulares;

declaraciOn e inicializaciOn de variables;

encabezado de funciones propias implementadas;

void setup()
{
        seteo de pines;
        inicializaciOn de la comunicaciOn serial;
        lo que se ejecuta una sola vez;
}

void loop()
{
        lo que se va a hacer todo el tiempo;
}

encabezado e implementaciOn de funciones propias;
```

Ejemplo 1:

```
float encontrar_promedio(int num1, int num2)
{
    float promedio;

    promedio = (num1 + num2) / 2.0;
    return(promedio);
}
// Desde otro programa o funcion se llama a la funcion
k = encontrar_promedio (4, 5);
```

Observación:

```
float encontrar_promedio(int num1, int num2)
{
    return( (num1 + num2) / 2.0 );
}
```

FUNCIONES

Ejemplo 2

Como usar la función anterior en el Arduino (además, se recuerda la biblioteca Serial):

```
float encontrar_promedio(int num1, int num2);
void setup(){
  Serial.begin(9600);
}
void loop(){
  int i = 2;
  int j = 3;
  float k;
  k = encontrar_promedio(i, j); // k ahora contiene 2.5
  Serial.println(k); // envia el valor de k al PC.
  delay(5000);
}
float encontrar_promedio(int num1, int num2){
    float promedio;
    promedio = (num1 + num2) / 2.0;
    return(promedio); //intruccion return. Necesaria cunado
        la funcion retorna un valor.
}
```

Ejercicio 1 - Clase

A partir de los ejercicios realizados en el pasado taller.

a) Escribir una función que tenga el siguiente encabezado:

```
float convertir_entrada(int valor);
```

que reciba el valor entero leído de una entrada analógica por el parámetro int valor y devuelva el voltaje correspondiente (en volts).
Para probar el funcionamiento, implementar una función void loop() donde sea llamada la función *convertir_entrada*() e se imprima el valor que retorna.

b) Escribir una función:

```
int convertir_salida(float volts);
```

que reciba en el parametro float volts el voltaje deseado en una salida PWM y devuelva el correspondiente entero a imponer en dicha salida (0-255).
Para probar el funcionamiento, implementar una función void loop() donde sea llamada la función *convertir_salida*() e imprimir el valor que retorna.

c) Tomando como referencia el Ejercicio 5 del taller 2: cumplir el mismo objetivo pero utilizando las funciones realizadas en la partes a) y b).

Alcance de una variable:

Variables **globales** vs. variables **locales**.

- Según donde se definan la variables, resulta el **alcance** de las mismas.
- Variables que se definen **dentro de una función**, solo pueden ser accedidas dentro de éstas y se le llaman variables locales a tal función. ("Nacen y mueren" en la ejecución de la función).
- Las variables definidas en el **programa principal** (fuera de las funciones que se implementen) son **variables globales**. Pueden ser modificadas también dentro de la implementación de las funciones (en general, no recomendable).
- OJO: evitar darle el mismo nombre a variables locales que a variables globales hasta no tener claro el concepto, ya que, dentro de la función, la variables a la cual se va a referir serán las locales. Luego de afianzados, con variables típicas como contadores (i, j, etc.) obviamente repetirán nombres.

- Se realiza una **copia** de las variables globales **pasadas** a la función en variables locales de la función. A esto se le llama **pasaje por valor**.

- Para conocer el valor de una variable local de la función es necesario retornar el valor con ***return***.

Permite conocer la tensión y además deja pronto el valor para una salida PWM en la misma variable donde se había guardado lo leído por la entrada analógica.

```
float voltaje(int val);  // Pasaje por valor.
```

FUNCIONES

Parámetros: Por Valor - Ejemplo

```
float resultado = 0; // variable global
int valor = 0; // variable global

float voltaje(int val);

void setup(){}

void loop(){
  valor = analogRead(A0);  // almaceno valor leido desde
      entrada analogica.
  resultado = voltaje(valor);   //se llama a la funcion
      pasando el valor almacenado en la variable global
      valor.
}

float voltaje(int val){ // Pasaje por valor
  float res;
  res = val*5.0 / 1023;
  return res;
}
```

- Existe la posibilidad de modificar variables globales dentro de la función sin acceder directamente a éstas, dando como resultado una forma de **retornar más de un valor** desde una función. Se le llama **pasaje por referencia**.

- En lugar de almacenarse el contenido de la variable global en la variable de la función (definida en el parámetro), ambas son la misma. Con lo cual, la propia variable global es la que se modifica en la ejecución de la función.

- Para avisarle al programa que tome dicha consideración en la declaración de la función, antes del parámetro que se quiere pasar por referencia, se agrega el símbolo &. No así para arreglos y matrices que siempre se pasan como referencia.

Permite conocer la tensión y además deja listo el valor para una salida PWM en la misma variable donde se había guardado lo leído por la entrada analógica.

```
float voltaje(int &val); // Pasaje por referencia.
```

FUNCIONES

Parámetros: Por Referencia - Ejemplo 1

```
float resultado = 0; // variable global
int valor = 0 ; // variable global

float voltaje(int &val);

void setup(){}

void loop(){
  resultado = voltaje(valor); //se llama a la funcion
      pasando la variable global valor.
}

float voltaje(int &val){ // Pasaje por referencia.
  float res;
  val = analogRead(A0); // la pude pasar para dentro de la
      funcion
  res = val*5.0 / 1023;
  return res;
}
```

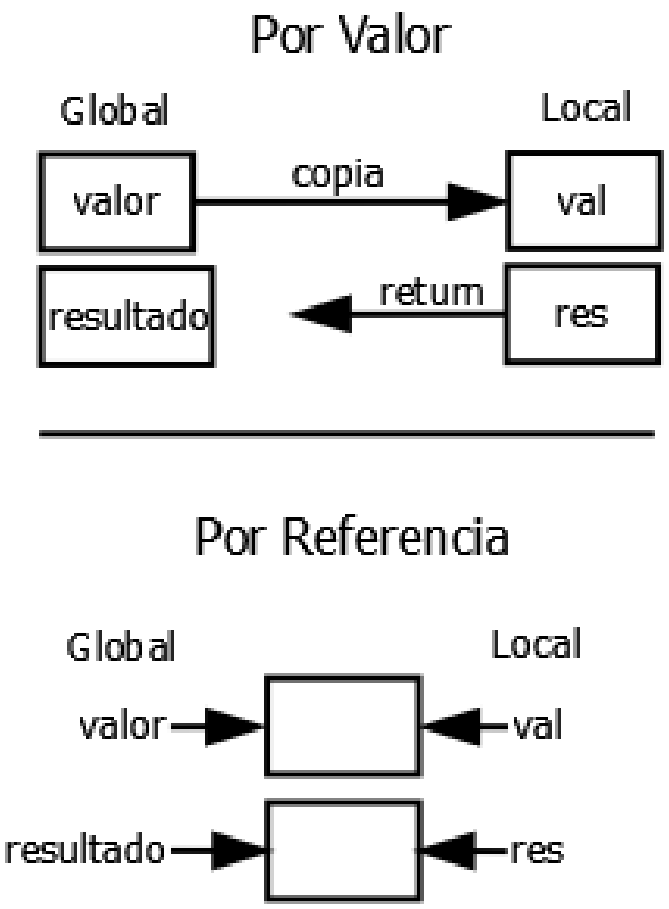
```
float resultado = 0; // variable global
int valor = 0; // variable global

void voltaje(float &res, int &val);

void setup(){}

void loop(){
  voltaje(resultado, valor);
}

void voltaje(float &res, int &val){
  //float res; ya no es necesario
  val = analogRead(A0);
  res = val*5.0 / 1023;
  //return res; tampoco es necesario
}
```


Por Valor
Global
Local
valor
copia
val
resultado
return
res
Por Referencia
Global
Local
valor
val
resultado
res

Ejercicio 2

Clase

1. Partiendo del Ejercicio 5 del taller 2 (Dimmer), eliminar el LED y la resistencia y conectar un motor de corriente continua (*Motor CC* en el simulador) entre el pin al que estaba conectado el LED y tierra -vamos a profundizar sobre motores de corriente continua en los siguientes talleres-. Probar el funcionamiento del nuevo sistema.
2. Conectar la mayor cantidad posible de LEDs a los pines digitales con sus respectivas resistencias (no utilizar los pines 0 ni 1, ni tampoco el pin con PWM que tiene al motor). Se pide que los leds se coloquen sobre el protoboard alineados y que los primeros sean verdes y los últimos rojos.
3. Imponer el modo dentro de la función *void setup()* de todos los pines.
4. Implementar una función con el siguiente encabezado:

```
void ledsToON(byte nivel, byte pinV1, byte pinV2, ... ,
    byte pinR1, byte pinR2, ...);
```

Dicha función debe cumplir el objetivo de ir prendiendo los leds de manera consistente a la velocidad del motor. El valor de velocidad será pasado a la función como un valor en la variable *byte nivel*.

5. Completar la función *void loop()* utilizando las funciones implementadas en el ejercicio 1:

```
float convertir_entrada(int valor);
int convertir_salida(float volts);
```

y la función implementada en la parte anterior:

```
void ledsToON(byte nivel, byte pinV1, byte pinV2, ... ,
    byte pinR1, byte pinR2, ...);
```

para probar el funcionamiento del TACÓMETRO.

Ejercicio

Para próxima clase

Partiendo del ejercicio 3 planteado en el taller 2, sobre mostrar una cuenta regresiva haciendo uso de un display de 7 segmentos. Se pide:

1. Implementar la función con el siguiente encabezado.

```
byte mapeoDigitoTo7segEnBinario(byte digito);
```

   La cual, dado un dígito del 0 al 9, debe retornar un byte cuya representación binaria indique con 1 los segmentos a encender para poder mostrar ese dígito en el display.
   Nota: investigar sobre la sentencia de control de flujo: "*Switch-case*"

2. Implementar la función con el siguiente encabezado.

```
void encenderDisplay(byte digitoEnbinario);
```

   La cual, dada la representación binaria de un dígito del 0 al 9 (salida de la función anterior), debe encender los segmentos respectivos del display.

3. Actualizar el código del ejercicio para hacer uso de estas funciones. Observar el cambio en la cantidad de líneas y la claridad del nuevo código.

Resumen para la próxima clase:

1. Si no se terminaron los ejercicios para hacer en este taller, terminarlos.
2. Tener funcionando el Ejercicio de deberes para compartir en la siguiente clase.
3. Por dudas utilizar el *Foro de consultas*. Considerar que, a demanda, están los horarios de consulta en la portada del curso.
4. Queda disponible un cuestionario sobre esta clase, que deberá ser completado en el sitio EVA. Lo deberá hacer cada estudiante individualmente!!
5. Se recomienda continuar con la lectura de la documentación sugerida en la sección de *Introducción* en el sitio de EVA. En particular, leer sobre comunicación serial.