Obligatorio 4 - Funciones y programación modular. Bluetooth

IIE - Facultad de Ingeniería - Universidad de la República

Tallerine Biónico 2024

El objetivo de este obligatorio es comprender los conceptos de función y programación modular. Además se busca familiarizarse con el módulo bluetooth de la placa Microbit y la conexión con una aplicación de celular para realizar proyectos inalámbricos.

Se trabajará con 2 patas para ir adquiriendo sentido común en los movimientos.

Para finalizar, se procederá al armado de 4 patas.

1. Funciones

Una función es un código implementado para cumplir una tarea concreta y devolver un resultado en caso de ser necesario, a partir de parámetros de entrada.

Ejemplos de funciones existentes:

delay(n); A partir del parámetro “n” su tarea es esperar “n” milisegundos. No devuelve ningún valor.

digitalRead(p); A partir del parámetro “p”cumple la tarea de leer el pin “p” y devolver su valor booleano (HIGH o LOW).

Además de utilizar funciones ya existentes, es posible declarar funciones propias. Esto permite estructurar un programa en códigos más concretos, lo que facilita la organización del programa, la comprensión del mismo y por lo tanto hacer más simple la corrección de errores. Son muy útiles cuando se requiere realizar una misma tarea muchas veces. En el código para declarar una función es como si fuera un subprograma dentro del programa. El formato general se muestra en la figura 1.

Declaración de funciones

```
1 Tipo_de_dato nombreFuncion ( Lista_de_par metros )
2     {
3     //Variables locales;
4     //C digo de la funci n;
5     //Retorno de valores;
6     }
```

Figura 1. *Declaración de funciones.*

En el cuadro 1 se describen las partes nombradas en la declaración de la figura 1.

Tipo de dato	Tipo de dato del valor que devuelve la función (int, bool,....)
nombreFuncion	Nombre elegido para la función. Es un texto a elección.
Lista de parámetros	Conjunto de datos de entrada.
Variables locales	Definición de variables que sólo tienen sentido dentro de la función.
Código de la función	Código propio de la función.
Retorno de valores	Declara el valor a retornar por la función.

Cuadro 1: Descripción de las partes de la declaración de una función.

En el siguiente trozo de código se presenta un ejemplo que declara la función "sum_func", la cual recibe 2 números enteros y devuelve su suma como otro número entero.

```
int sum_func (int x, int y)
{
     int z=0;    // variable local a la función.
     z=x+y;
     return z; // valor a retornar
}
```

La declaración de una función se puede hacer en cualquier parte del código, tanto por encima de la función setup() como por debajo de la función loop() o entre ellas.

Declaración antes de loop()

```
1  int result = 0;
2  int sum_func (int x, int y)
3  {
4      int z = 0;
5      z = x+y ;
6      return z; // valor a retornar
7  }
8  void setup ()
9  {}
10 void loop ()
11 {
12     result = sum_func (5,6);
13 }
```

Declaración después de loop()

```
1  int result = 0;
2  void setup ()
3  {}
4  void loop ()
5  {
6      result = sum_func (5,6) ;
7  }
8  int sum_func (int x, int y)
9  {
10     int z = 0;
11     z = x+y ;
12     return z; // valor a retornar
13 }
```

Figura 2. *Declaración de la función* Sum_func(x,y) *antes y después de* loop().

Tipos de variables

Hasta ahora las variables se declaran en el programa principal o dentro de las funciones setup() y loop() sin mayor explicación. Cuando se definen funciones, es importante comprender el alcance de las variables. A continuación se explican los tipos de variables y su alcance:

- **Variables globales:** Son variables que se declaran fuera de una función, por lo que pueden ser "vistas" y modificadas por cualquier función. Son las variables que se venían declarando fuera de setup() y loop().
  En el proyecto se propondrá declarar ciertas variables globales las que serán modificadas por muchas funciones.

- **Variables locales:** Son variables que se declaran dentro de una función y solo tienen sentido dentro de esta. No pueden ser utilizadas por otras funciones.
  Existen 2 tipos de variables locales:

  - **Dinámicas:** Su valor se pierde entre llamados consecutivos a la función. Se declaran igual que hasta ahora, pero dentro de la función.
    Ej: int dato = 0; // Cada vez que se llama la función, dato=0

- **Estáticas:** Su valor se mantiene entre llamados consecutivos a la función. Se declaran igual que hasta ahora, pero dentro de la función y con la palabra clave “static” delante. Ej: `static int dato = 0;` El “0” se carga solo la primera vez que se llama a la función.

En el ejemplo anterior (figura 2), result es una variable global y z es una variable local dinámica.

No confundir entre tipo de variable y tipo de datos de una variable. El tipo de variable refiere al alcance de la variable, mientras que el tipo de datos refiere al dato que representa.

2. Programación modular

Cualquier programa útil excede la cantidad de líneas de código que es cómodo leer en un solo archivo. Por eso es práctico separar el código en varios módulos (o archivos). Cada uno de estos módulos se encarga de resolver una problemática concreta. La idea es que los módulos sean, dentro de los posible, independientes entre sí.

Para crear un nuevo archivo *clickear* en la flecha ubicada en la parte superior y seleccionar *New Tab*. Luego elegir el nombre y se creará el archivo.

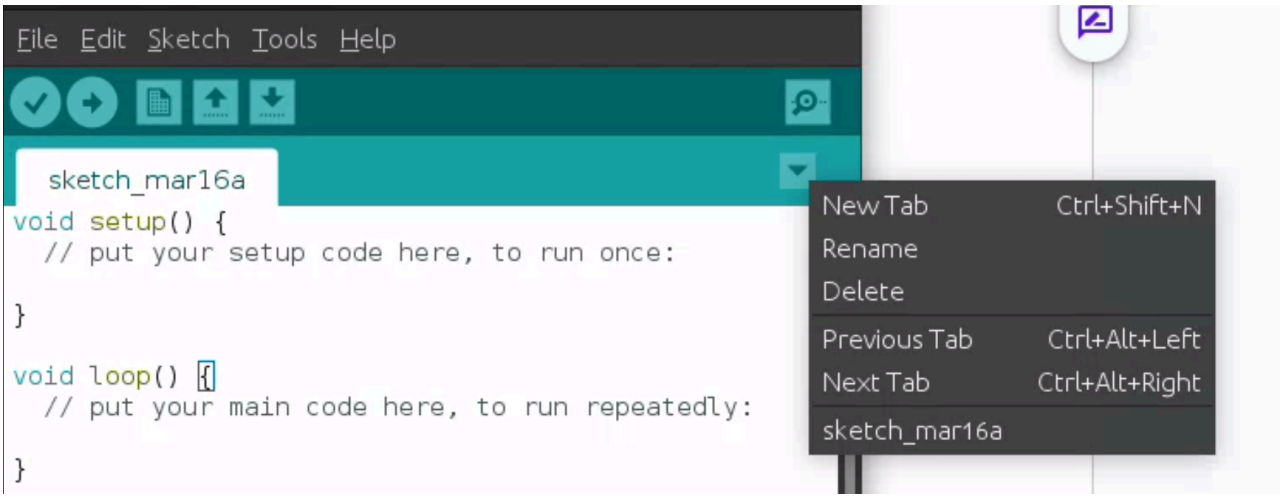

***Figura 3**. Crear un nuevo archivo.*

Para crear un módulo, se deben crear 2 archivos del mismo nombre con extensiones “*.cpp*” y “*.h*” respectivamente. Por ejemplo, para crear el módulo llamado *movimientos*, el cual se encarga de realizar los movimientos del “bicho” mediante órdenes a los servomotores, requiere crean los archivos "*movimientos.h*" y "*movimiento.cpp*". El archivo "*movimientos.h*" solo debe incluir las cabeceras de las *funciones que el módulo ofrece; y el archivo "movimientos.cpp" debe implementar la lógica de dichas funciones*.

Para poder utilizar el módulo en nuestro programa ambos archivos deben ir en la misma carpeta que el sketch y debemos incluir el archivo .h de la siguiente forma en el programa, para el ejemplo del módulo movimiento: `#include "movimientos.h".`

El ejemplo "modular.ino" explica más en detalle lo anterior.

3. Bluetooth en el celular

Hasta ahora hemos recibido comandos desde el Monitor Serial. Se busca recibir estos comandos desde un celular por medio de Bluetooth. Para esto se debe instalar en el celular una aplicación que

permite enviar texto a través de Bluetooth.

La microbit cuenta con una radio bluetooth de baja energía (BLE, Bluetooth Low Energy) incorporada que se puede comunicar de forma sencilla con la aplicación **Adafruit Bluefruit Connect** la cual se encuentra disponible para Android e iOS (*Play Store* y *App Store* respectivamente).

Se puede acceder por más información de la aplicación al siguiente enlace: https://learn.adafruit.com/bluefruit-le-connect

Previamente, se necesitará tener instalada la biblioteca **Adafruit Microbit** y un código ejecutándose en la Microbit que la utilice para poder detectar y conectarse a la placa desde el celular. Esto se explica en el siguiente punto.

Una vez instalada la aplicación, se abre y se elige el dispositivo a conectarse (figura 4) . Luego se debe seleccionar el módulo "UART" (figura 5). Por último, en los 3 puntitos, seleccionar "Display Mode" como "Text" (figura 6).

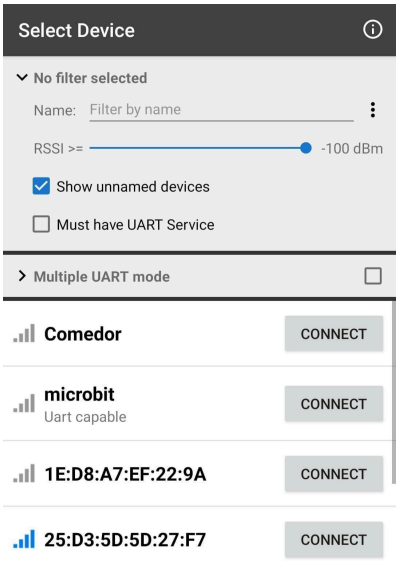

***Figura 4**. Detección de dispositivos*

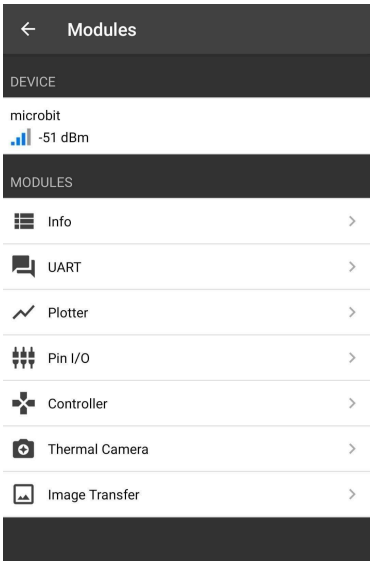

***Figura 5**. Módulos*

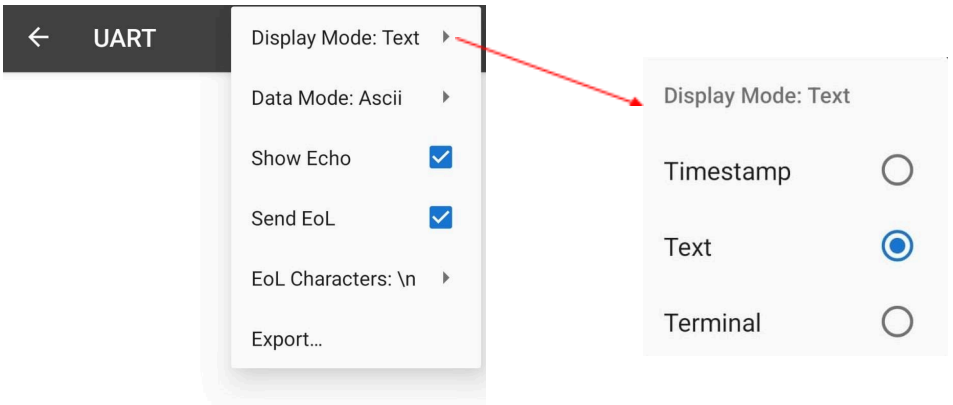

***Figura 6**. Display Mode*

4. Bluetooth en la Microbit

Para utilizar la comunicación Bluetooth de la placa no es necesario conocer sus detalles ya que se dispone de una biblioteca que lo resuelve: **Adafruit Microbit.**

La biblioteca **Adafruit Microbit** tiene varias funcionalidades, brinda una interfaz para el uso de la matriz LED, el sensor de temperatura, el acelerómetro y en este obligatorio se trabajará con ella pues brinda también una interfaz para usar con la aplicación **Adafruit Bluefruit Connect** a través de la conexión Bluetooth.

Para que todo funcione se deberán instalar las siguientes bibliotecas:

- Adafruit Microbit: https://www.arduino.cc/reference/en/libraries/adafruit-microbit-library/
- BLEPeripheral: https://www.arduino.cc/reference/en/libraries/bleperipheral/
- Adafruit GFX : https://www.arduino.cc/reference/en/libraries/adafruit-gfx-library/
- Adafruit_BusIO: https://www.arduino.cc/reference/en/libraries/adafruit-busio/

Para instalar cada librería en Arduino IDE seleccionar *Programa > Incluir Librería > Añadir biblioteca.ZIP...* Una vez realizado esto, reiniciar Arduino IDE y verificar si se encuentra en el listado de librerías, *Programa > Incluir Librería.*

5. Ejemplo de comunicación Bluetooth

El siguiente código **`ble_uart_microbit.ino`** (disponible en EVA) muestra cómo se comunica la placa Microbit a través de un puerto serie a la PC y un dispositivo inalámbrico (el celular en nuestro caso):

```
#include <Adafruit_Microbit.h>

Adafruit_Microbit microbit; //Define la variable microbit para utilizar las funciones.

void setup() {
Serial.begin(9600);
Serial.println("Microbit ready!");
microbit.BTLESerial.setLocalName("Microbit"); //Microbit es el nombre de la conexión
microbit.BTLESerial.begin();
}

void loop() {
  microbit.BTLESerial.poll(); //Permite escribir en el celular
  if (microbit.BTLESerial.available()) {
          Serial.write(microbit.BTLESerial.read());
      }
  if (Serial.available()){
        microbit.BTLESerial.write(Serial.read());
    }
}
```

Aclaración: El búfer interno UART RX de la microbit tiene 64 bytes, para asegurarse de leer los datos antes de que el búfer esté lleno y que los datos no se pierdan se recomienda enviar hasta un máximo de 10 caracteres desde la terminal serial de la PC.

6. Ejercicios

1. En este ejercicio se utiliza el sketch **`Variables.ino`** (disponible en EVA).

a. Identificar en el código los diferentes tipos de variables (global, local dinámica o local estática).

b. Cargar el código en la placa Microbit y verificar su funcionamiento.

c. Modificar el código quitando la palabra reservada static y volver a ejecutarlo. Observar que cambia. Justificar.

d. Modificar en el código cambiando `y = x - y;` por `y = x - result;`. ¿Qué sucede al intentar ejecutar el código? Justificar.

2. Disponiendo de 2 patas bien calibradas se pide implementar los movimientos **init_patas()**, **avanzar()**, **retroceder()** y **saludar(n)** utilizando funciones y el concepto de modularización. Recordar que se deben crear los archivos movimientos.cpp y movimientos.h, este módulo se encargará de resolver los movimientos del "bicho".

   Observación: como ahora el módulo movimientos se encarga de manejar los servos se deberá utilizar la librería de servomotores solamente en este archivo. Ver el ejemplo "modular.ino".

   Antes de realizar este ejercicio se debe disponer de 2 patas bien calibradas.

   a. Crear la función **Ini patas()** tal que inicialice simultáneamente los servos de las dos patas de forma que ambos muslos queden alineados con la cadera y ambas patas queden hacia abajo a 80° de la horizontal. Al final de esta función se debe agregar un retardo que corresponda a lo que demora un servo en moverse 90°. Se busca imitar la posición de “parado”. Para determinar el tiempo que demora un servo en moverse un determinado ángulo, referirse al ejercicio 1, parte c.

   b. Crear la función **Avanzar()** tal que dé un paso hacia adelante con la pierna derecha (referirse al obligatorio 3, ejercicio 4 parte c) y luego otro con la pierna izquierda. Los retardos a utilizar deben corresponder al tiempo que le toma al servo realizar el movimiento. Se busca imitar el caminar hacia adelante. En breve se verá que para caminar se deben mover ambas patas en forma simultánea y coordinada.

   c. Crear la función **Retroceder()** tal que realice el movimiento inverso a Avanzar().

   d. Crear una función **Saludar(n)** tal que levante su pierna derecha, adelante el muslo derecho y luego realice un movimiento de vaivén “n” veces con el muslo. Una vez finalizado, debe volver los servos a su posición inicial. El ángulo de vaivén a utilizar es a elección. Los retardos a utilizar deben estar acordes al movimiento.

3. Utilizando las funciones creadas en el ejercicio 3, realizar un programa que repita la secuencia:

   - Salude 2 veces
   - Camine 7 pasos hacia adelante
   - Camine 5 pasos hacia atrás
   - Salude 4 veces
   - Camine 2 pasos hacia atrás

   Se debe aplicar el concepto de modularidad para este ejercicio. Es decir, utilizar el módulo movimientos creado en el ejercicio 2 para los movimientos y resolver el problema en el sckech principal.

4. Realizar un programa que utilice el Monitor Serie para recibir “comandos” (letras) y cumpla lo siguiente:

Comando	Comportamiento de las 2 patas
‘a’	“Camine” hacia adelante
‘b’	“Camine” hacia atrás
‘c’	Permanezca en la posición inicial
“d”	“Saludar 5 veces”
en otro caso	Patas completamente estiradas

   Referirse al obligatorio 3, ejercicio 4, parte d.

   Al igual que en el ejercicio anterior, se debe aplicar el concepto de modularidad.

5. En un celular instalar la aplicación Adafruit Bluefruit Connect.
   a. Realizar la conexión Bluetooth entre el celular y la placa como se explica más arriba.
   b. Estudiar el código ble_uart_microbit.ino. Se debe comprender que se espera que suceda.
   c. Utilizar la aplicación del celular en Modo Terminal UART. Investigar el funcionamiento para enviar y recibir caracteres (“a”, “b”, etc)
   d. Probar el código en la placa.
   e. Modificar el código de forma tal que si el último dato enviado desde el celular es la letra “a” representar con un símbolo a elección en la matriz LED (y dejar encendido). En el caso de que el último comando recibido no sea la letra “a” se puede apagar la matriz LED o representar con otro símbolo que el dato enviado no es “a”.

Aclaración: Al abrir varias ventanas del IDE de Arduino puede darse el caso que no se seleccione el campo Softdevice: S110. Si sucede esto y se carga el programa, se borrará parte del código que tiene el micro:bit por defecto para manejar el bluetooth. Notarán que le cargan el programa y no responde. Para solucionar esto deben ir a https://makecode.microbit.org/ generar un proyecto nuevo, descargar un programa vacío, este termina en .hex y subirlo a la micro:bit. Esto volverá a cargar la parte borrada. Luego de esto volver a programar en arduino.