Obligatorio 4 - Funciones, programación modular y 4 patas

IIE - Facultad de Ingeniería - Universidad de la República

Tallerine Biónico 2025

El objetivo de este obligatorio es comprender los conceptos de función y programación modular. Además, se irá adquiriendo sentido común en los movimientos con 2 patas para finalmente proceder al armado de 4 patas y trasladar los movimientos aprendidos.

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” su tarea es 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, su comprensión y 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.

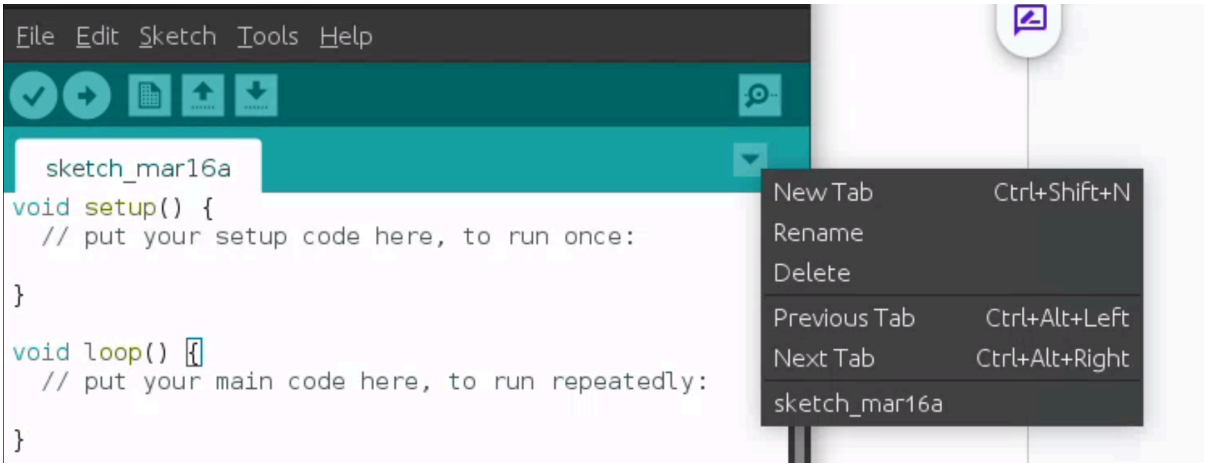

***Figura 3**. Crear un nuevo archivo.*

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. Armado del "Bicho"

Por simplicidad, al cuadrupedo se le va a llamar "Bicho". Cada grupo además debe apodarlo con el nombre que consideren. Una vez que se arman 2 caderas con sus 2 patas cada una, se las debe unir con la columna tal como se muestra en la figura 4.

Al no disponer de tornillos para sujetar la columna a las caderas del Bicho, se deben atar con los hilos de cable UTP suministrado. **No deben pegar las piezas con silicona u otro pegamento** para que el "Bicho" pueda ser desensamblado y utilizado en el futuro por otros estudiantes.

Se recuerda que se deben devolver los materiales tal como se les fue entregado, con las piezas separadas, todo desconectado y ordenado dentro de la caja.

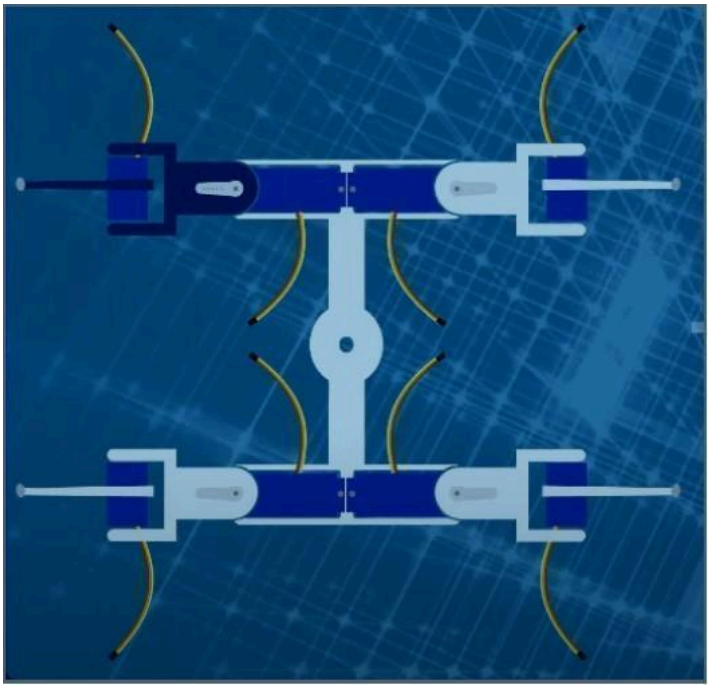

***Figura 4**: Esquema del Bicho armado*

El cuerpo del Bicho donde apoya la placa Microbit y demás elementos opcionales no se suministra. Se espera que cada grupo sea creativo a la hora de buscar o crear un soporte. Se sugiere usar materiales livianos y evitar tamaños excesivamente grandes que puedan entorpecer o enlentecer los movimientos del "Bicho". Buscar inspiración de los videos del EVA de años anteriores.

4. 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. Armar el "Bicho" con 4 patas. Recordar ajustar correctamente los ángulos de los servos y determinar el ángulo de error para cada uno. Tomando como referencia las funciones creadas para 2 patas, se deberán desarrollar las siguientes funciones para 4 patas:
   a. **Ini patas()**, que inicie todos los servos. Los valores iniciales son a elección, de forma que el Bicho queda parado y en buen equilibrio.
   b. **Saludar()**, que salude moviendo la pata derecha considerando las 4 patas. El bicho deberá

quedar parado en 3 patas.

6. Para lograr caminar hacia adelante, es necesario que el movimiento de varios servos se realice en forma simultánea. Cuando se indica que 2 servos se mueven en forma simultánea, se entiende que no hay un retardo (delay) entre ambos movimientos. Esto se puede generalizar a varios servos. Como primer punto, para que el Bicho camine las patas cruzadas deben moverse en forma similar y simultánea. Si se considera que inicialmente las patas se encuentran en una posición inicial en la cual el Bicho está parado, una posibilidad para que "camine" hacia adelante sería:

   - Levantar las piernas derecha delantera e izquierda trasera.
   - Mover hacia adelante los muslos derecho delantero e izquierdo trasero.
   - Mover hacia atrás los muslos izquierdo delantero y derecho trasero Bajar
   - a posición inicial las piernas derecha delantera e izquierda trasera.

   - Levantar las piernas izquierda delantera y derecha trasera.
   - Mover hacia adelante los muslos izquierdo delantero y derecho trasero.
   - Mover hacia atrás los muslos derecho delantero e izquierdo trasero.
   - Bajar a posición inicial las pierna izquierda delantera y derecha trasera.

   Observar que estos son 2 pasos, donde los movimientos individuales se pueden separar en 2 bloques iguales y simétricos.

   A. Desarrollar la función **Avanzar()** que camine hacia adelante 2 pasos, tal como se describe en la secuencia anterior.
   B. Ajustar la función **Avanzar()** mediante ensayo y error. Tratar de comprender cuáles movimientos deben ser simultáneos y cuáles no. Definir los valores de los retardos que considere necesarios.
   C. Desarrollar la función **Retroceder()** tomando como referencia la función avanzar.

2. Para completar la movilidad del Bicho se requiere implementar funciones que permitan rotar sobre sí mismo en sentido horario y antihorario:
   a. Partiendo de las funciones Avanzar() y Retroceder(), desarrollar la función **GiroHorario()** de forma que el Bicho realice 2 pasos girando en sentido horario.
   b. Desarrollar la función opuesta **GiroAntiHorario()**

3. Una vez ajustadas las funciones anteriores integrarlas en un programa que utilice una conexión bluetooth de forma que realice el movimiento consecutivo del último comando enviado.
   a. Si recibe la letra "a": camina hacia adelante.
   b. Si recibe la letra "b": camina hacia atrás.
   c. Si recibe la letra "c": gire en sentido horario.
   d. Si recibe la letra "d": gire en sentido antihorario.
   e. Si recibe la letra "e": vaya a posición inicial (parado).
   f. Si recibe la letra "f": salude en forma indefinida.
   g. Si recibe la letra "g": se quede quieto donde quedó .

Inicialmente se considera que fue oprimida la letra "e".