Variables en C para Arduino

Básico

Avanzado (uso de la memoria)

Operadores * y **&** (punteros)

Son una serie de esquemas de apoyo para alumnado de ESO y Bachillerato
que ya ha hecho algunos programas en Arduino y quiere entender un poco
más en profundidad como funciona el lenguaje C

J.A.Manceras 2020

Variables en C para Arduino

1 BÁSICO

Una **variable** es una caja en la que puedo guardar valor

120

La **variable** tiene un nombre para diferenciarla del resto de variables que puedo tener en memoria

120
pasos_robot

En lenguaje C (pero no en todos) las variables (cajas) no valen para cualquier cosa y hay que decir el **tipo** de dato que contendrán (si es un número entero, decimal, un texto...)

120
pasos_robot
tipo: número entero

2 Operaciones básicas con variables

Declararación:
Decir cual es su nombre nombre y tipo antes de usarla la primera vez.

```
// Declaración de variable de tipo número entero
int pasos_robot;
```

Asignación:
Hacer que contenga un valor

```
// Variable a la izq. del signo igual
pasos_robot=120;
// Se pueden hacer a la vez declaración y asignación
int pasos_robot=120;
```

Lectura:
Usar su contenido

```
// A la derecha de un signo igual o en una expresión
posicion_final=pasos_robot+10;
if (pasos_robot>100)
{...........}
```

3 Tipos de variables en C

Crear una variable es declararla
los tipos básicos de C en Arduino son

byte 0

```
byte sensor_temperatura=0;
```

int 8

```
int contador=8;
```

char 'A'

```
char caracter='A';
```

long 355.433

```
long segundos_desde_inicio=355433;
```

float 3,45

```
float diametro=3,45;
```

array

3	mi_array[0]
5	mi_array[1]
7	**mi_array[2]**
\0	*código fin de array]*

```
int mi_array[3] = { 3 , 5 , 7};
```

String

'H'	mi_cadena[0]
'o'	mi_cadena[1]
'l'	mi_cadena[2]
'a'	mi_cadena[3]
'.'	mi_cadena[4]
\0	*código fin de cadena*

```
String mi_cadena = "Hola.";
```

4 Memoria de un ordenador

La informacion en la memoria de un ordenador se almacena en bytes
cada byte está compuesto de 8 bits. Cada bits vale 1 o 0

1 0
bit

byte 1 1 1 1 1 1 1 1

En un byte hay 256 combinaciones de valores diferentes.
Si las ordenamos según el sistema numérico binario, van del 0 al 255

Memorias RAM

Las memorias RAM son en las que se ejecutan los programas.
En los PCs vienen en Megabytes (MB) o Gygabytes (GB) , hoy en día lo normal son 4 GB

Una memoria RAM de 4 Gigabytes (4 GB) tiene 4.294.967.296 bytes.
Esto son más de 4.000 millones de byes.

RAM Arduino

Los Arduinos suelen tener memorias RAM más pequeñas, de:
- 32 Kilobytes (32KB), eso son 32.768 bytes
- 256 Kilobytes (256 KB), eso son 262.144 bytes.

Tamaño de los tipos de variables

Cada tipo de variable necesita una cantidad de bytes para almacenar su valor.
Depende de lo que queramos almacenar necesitaremos más o menos bytes.
La cantidad también puede depender del tipo de ordenador, estos valores son para Arduino

byte (8 bits) Puede tener valores entre 0 (00000000) y 255 (11111111)

int (2 bytes) Puede tener valores entre -32.768 y 32.767
con 2 bytes hay 65.536 combinaciones

char (1 byte) En código ASCII, 256 códigos para letras, la 'A' es el código 65 (01000000)

long (4 bytes) Puede tener valores ntre -2.147.483.648 y 2.147.483.647

float (4 bytes) Puede tener valores entre 3.4028235E+38 y -3.4028235E+38
23 bits mantisa, 8 bit exponente y 1 signo

array (1+x bytes) Como son matrices de otros tipos de variables su tamaño en variable

String (1+x bytes) Cada letra (codificada en ASCII) ocupa un byte (256 caracteres distintos)

Variables en C para Arduino (Avanzado I)

VARIABLES

Una **variable** es una caja en la que puedo guardar valor

La **variable** tiene un nombre para diferenciarla del resto de variables que puedo tener en memoria

pasos_robot

En lenguaje C (pero no en todos) las variables (cajas) no valen para cualquier cosa y hay que decir el **tipo** de dato que contendrán (si es un número entero, decimal, un texto...)

pasos_robot
tipo: número entero

Tipos de variables en C

Crear una variable es declararla y los tipos básicos de C en Arduino son

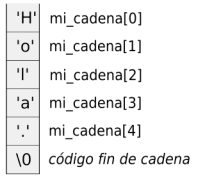
byte sensor_temperatura=0;

int contador=8;

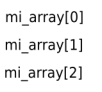
char caracter='A';

float diametro=3,45;

long segundos_desde_inicio=355433;



String mi_cadena = "Hola.";



int mi_array[3] = { 3 , 5 , 7};

5 AVANZADO

La caja (variable) es un trozo de memoria del ordenador

Los bytes de la memoria RAM están numerados, a eso se llama su ***dirección de memoria***

El compilador de C es el que anota cual es la posición de memoria que corresponde al identificador(nombre) de la variable para poder encontrarla cuando la uso en el programa

tipo	bytes	identificador	posición memoria
int	2	pasos_robot	8101
char	1	caracter	8103
byte	1	dia_mes	10002
int	2	sensor_temperatura	10003
long	4	segundos_desde_inicio	10005
array	7	mi_array	10009
float	6	diametro	10017
String	6	mi_cadena	10016

Memoria RAM 256 KB

	Decimal	Hexadecimal
	0000	000000
...		
...	8099	
	8100	
120	8101	001FA5
	8102	001FA6
'A'	8103	001FA7
...	8104	001FA8
...	10000	
	10001	
13	10002	002712
8	10003	002713
	10004	
	10005	002715
355.433	10006	
	10007	
	10008	
3	10009	002719
	10010	
5	10011	
	10012	
7		
\0	10015	00271F
'H'	10016	002720
'o'	10017	002721
...		
...		
	262143*	03FFFF

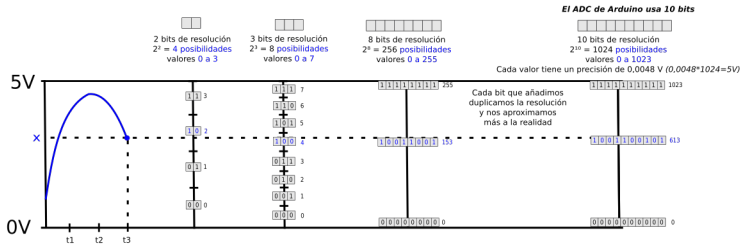
* 256 KB = 262.144 bytes

6 De analógico a digital en Arduino

Usando la electrónica podemos conectar Arduino con el exterior para leer señales de dos tipos:
- Digitales (1 y 0, creados mediante voltaje en el circuito, 5V (1), 0V (0)
- ***Analógicas, variaciones de voltaje entre 0 y 5V***

Para converir un determinado valor de voltaje entre 0 y 5V en un número digital, Arduino usa un circuito llamado ***ADC (Analog to Digital Converter)*** que muestrea la señal y devuelve un valor digital. ¿Cuántos valores de voltaje diferentes hay entre 0 y 5V?. Pues infinitos.

No podemos reservar memoria para posibilidades infinitas en un ordenador, habra que hacer una aproximación con una determinada ***resolución***.
¿Qué voltaje (x) hay en t3?. Depende de los bits de resolución que empleemos.



La ***unidad de memoria es el byte***, así que los tipos de variables son múltiplos de bytes

1 byte 1 1 1 1 1 1 1 1 número de combinaciones 2^8=256 (0 a 255)

2 bytes 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 núm de combinaciones 2^{16}=65.536 (0 a 65.535)

Aunque hagamos una lectura con una variable de tipo entero (int) de 2 bytes (16 bits)

int lectura = **analogRead**(3); //los pines de lectura analógica son del 0 al 5

como la resolución de la lectura es de 10 bits, los valores siempre serán entre 0 y 1023

Variables en C para Arduino (Operadores * y &)

El compilador de C es el que anota cual es la posición de memoria que corresponde al identificador(nombre) de la variable

tipo	bytes	identificador	posición memoria
int	2	pasos_robot	8101
byte	1	dia_mes	10002
int	2	sensor_temperatura	10003
long	4	segundos_desde_inicio	10005

7 Punteros (* y &)

En C podemos crear un "***puntero o apuntador***" a una variable cualquiera poniendo un * delante de la variable que declaramos.
El puntero no es más que ***una variable cuyo contenido es una posición de memoria***

int *p_pasos_del_robot;
//sin inicializar, no apunta a ningún sitio

tipo	bytes	identificador	posición memoria
int *	4	p_pasos_del_robot	10009

?????
p_pasos_robot
tipo: puntero a entero

int pasos_del_robot=8;

120
pasos_robot
tipo: número entero

En C podemos saber la posición de memoria donde está una variable poniendo delante el sigo **&**. Si hago *printf(&pasos_del_robot)* saldría por pantalla 10003

// Asigno al puntero la dirección de la variable
p_pasos_del_robot **=** **&**pasos_robot;

8101
p_pasos_robot
tipo: puntero a entero

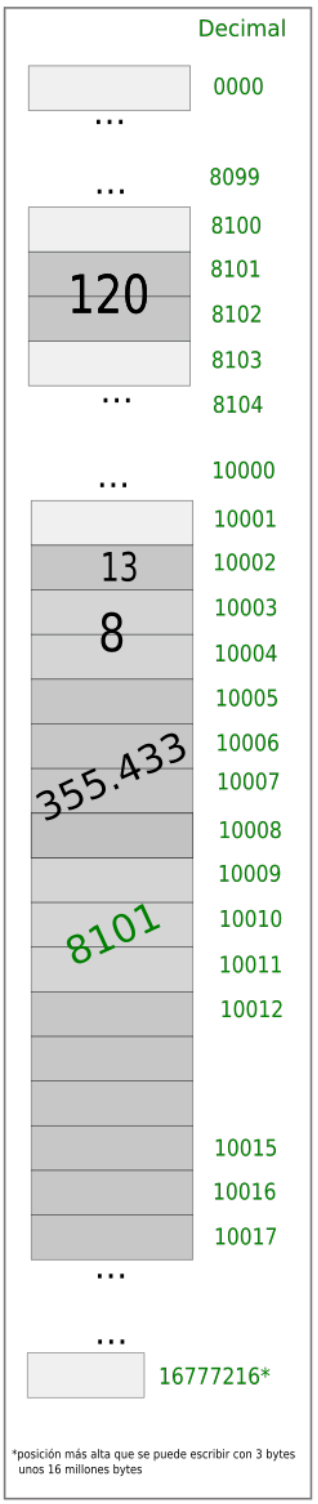
Ahora hay ***dos formas de acceder a la dirección de memoria*** que contiene a la variable.
1) A través de la propia variable. *printf(&pasos_robot);* *pinta 8101*
2) A través del puntero. *printf(p_pasos_del_robot);* *pinta 8101*

Y también hay ***dos formas de acceder al contenido de la variable*** pasos_robot.
1) A través de la propia variable. *printf(pasos_robot);* *pinta 120*
2) Usando el operador de dirección con el que declaramos el puntero, el asterisco (*).
**p_pasos_del_robot*, indica el contenido de la variable a la que apunta el puntero.
de modo que *printf(*pasos_del_robot);* *pinta 120*

Como podemos acceder al mismo sitio hay que tener cuidado, ya que
*p_pasos_del_robot=231;
cambia el contenido de la variable pasos_robot

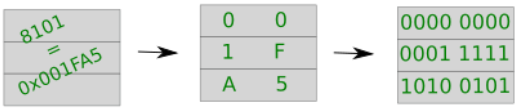
231
pasos_robot
tipo: número entero

¿Qué devuelve &p_pasos_del_robot? 10009, la dirección de esa variable



En informática lo más común es ecribir las ***posiciones de memoria en notación hexadecimal***,
en C para que un número sea hexadecimal basta poner 0x delante. El mismo número se puede escribir 8101 (decimal) o 0x1FA5 (hexadecimal)

Esto se ha así porque en hexadecimal es mucho más fácil identificar los bytes que forman una posición de memoria, ya que cada byte se forma exactamente con dos cifras hexadecimales



Nuestro programa podrá manejar tanta memoria como bytes usemos para los punteros
Con 2 bytes por puntero sólo podremos manejar 64 KB (65536 posiciones)
Con 3 bytes por puntero 16 MB (16.777.216 posiciones)
Con 4 bytes por puntero 4 GB
Los PCs actuales usan 5 para poder ampliar la memoria a 8GB por ejemplo

Enun Arduino con 256K usamos 3 bytes (24 bits), aunque sólo se usan 18 para la máxima posición de memoria (03FFFF)

Variables en C para Arduino. Operadores * y **&** (y II)

El compilador de C es el que anota cual es la posición de memoria
que corresponde al identificador(nombre) de la variable

tipo	bytes	identificador	posición memoria
int	2	pasos_robot	8101
byte	1	dia_mes	10002
int	2	sensor_temperatura	10003
long	4	segundos_desde_inicio	10005
char	1	letra	8103
String	5	cadena	10012

8 Uso de punteros con cadenas de caracteres

Aunque en Arduino C podemos definir una cadena como:

```
String cadena="Hola";
```

Pero es muy común ver código con las notaciones clásicas para cadenas no modificables:

```
char cadena[]="Hola";
```
o
```
char *cadena="Hola";
```

10012
cadena
tipo: puntero a char

que define una cadena como un array de carácteres
y el nombre es la dirección del primer byte

También se suelen manipular cadenas usando los operadores * y &.

```
char *puntero_a_cadena = cadena;
```

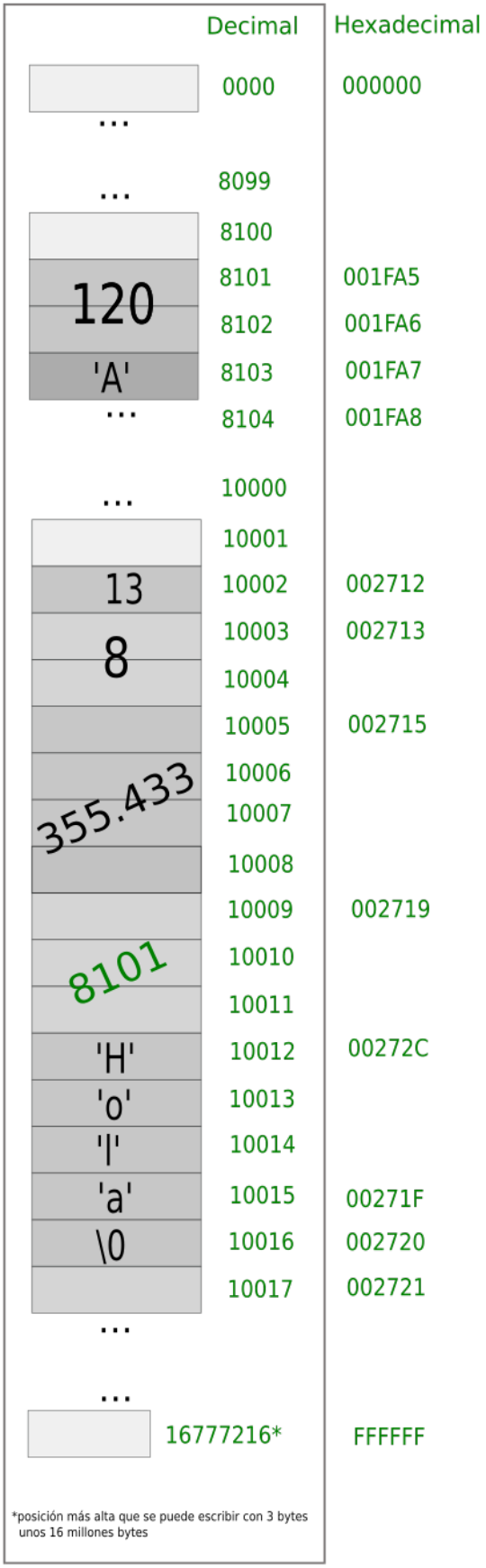
10012
puntero_a_cadena
tipo: puntero a char

me permite tener un puntero apuntando a la primera posición
Ahora a la segunda letra se puede acceder de dos formas:

```
cadena[1];
```
o
```
*(puntero+1);
```

que suma 1 a la posición (10012+1=10013) y consulta esa dirección de memoria
Ambos accesos devuelven el caracter 'o'.

Eso si, hay que tener en cuenta que el puntero "cadena" no se puede modificar,
es una constante creada en tiempo de compliación

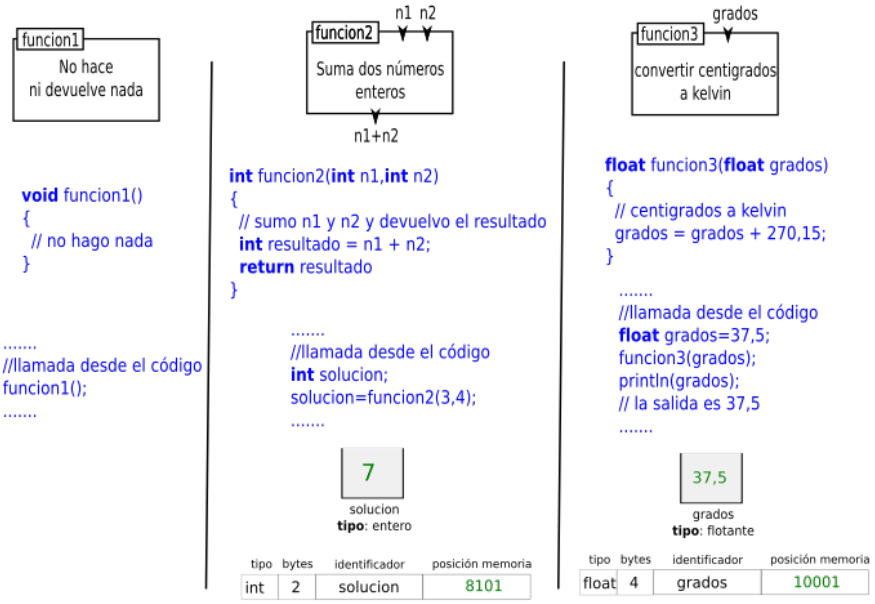


Variables en C para Arduino. Operadores * y **&** (y III)

9 Uso de punteros con funciones

Las funciones o procedimientos son un trozo de código al que ponemos un nombre y que podemos ejecutar las veces que queramos llamándolas desde cualquier punto del programa.

Ejemplos de 3 funciones en C:



Fíjate en la funcion3. Cuando la llamo modifica la variable parámetro grados, sin embargo, al volver de la función esta variable tiene el mismo valor que tenía. (37,5)

La razón es que para parámetros de tipos simples, C hace una copia y, aunque le ponga el mismo nombre, la función trabaja con una copia de la original (parámetro por valor). Mientras se ejecuta la función aparece una variable que se modifica y al terminar se borra:

tipo	bytes	identificador	posición memoria
float	4	grados	10005

TEMPORAL (FUNCIÓN)

Uso de punteros

Esa es la razon de que se usen tanto los punteros como parámetros de funciones. Al copiarse la dirección para trabajar en la función, tengo la dirección de la variable original para tocarla si es necesario (otra discusión es si es una buena práctica).

```
float funcion3(float *grados)
{
 // El parámetro es un puntero a flotante
 // centigrados a kelvin
 // con el * accedo al contenido
 *grados = *grados + 270,15;
}
```

```
.....
//llamada desde el código
float grados=37,5;
//No paso el valor sino la dirección de la variable
funcion3(&grados);
println(grados);
// la salida es 317,65
.....
```

A este modo de pasar parámetros se le llama parámetros por referencia. Hay que estar muy seguro de cuando se hace para mantener la integridad de un programa. Por otro lado es la única forma de pasar tipos complejos como los arrays, que no se copian sino que se pasa un puntero al array original (la copia podría ser enorme)

Memoria RAM 1 MB