Programación para Sistemas Embebidos

3. Tipos de datos, variables y constantes

¿Qué es una variable?

Una variable es una ubicación de almacenamiento en memoria que se encuentra vinculada a un nombre simbólico. Esta área de memoria reservada puede ser llenada con datos o dejarse vacía. Básicamente, se utiliza para almacenar diferentes tipos de valores. Algo muy útil con las variables es que podemos cambiar su contenido (el valor) durante la ejecución del programa; esto es también la razón por la que se llaman variables, en comparación con las constantes que también almacenan valores, pero que no pueden ser modificadas mientras el programa se está ejecutando.

¿Qué es un tipo de dato?

Las variables (y constantes) están asociadas con un tipo de dato. Un tipo de dato, también llamado simplemente tipo, define la naturaleza posible de los datos que puede almacenar una variable. También ofrece una forma práctica de reservar directamente un espacio con un tamaño definido en la memoria. C cuenta con alrededor de 10 tipos principales de datos que pueden ser ampliados, como veremos aquí.

Para la programación en Arduino, estudiaremos deliberadamente solo los tipos que utilizaremos con frecuencia. Estos tipos cubren aproximadamente el 80 por ciento de los tipos de datos comunes en C, y serán más que suficientes para nuestro propósito.

Utilizamos un tipo de dato cuando declaramos una variable como se muestra aquí:

```
int ledPin; // declara una variable del tipo int con el nombre "ledPin"
```

Al reservar una variable con un tipo de dato específico, se asigna un espacio de memoria de un tamaño particular (relacionado con el tipo int en este caso). Como se puede observar, si solo escribimos esa línea de código, todavía no se ha almacenado ningún dato en la variable. Sin embargo, tengamos en cuenta que se ha reservado un espacio en la memoria, listo para ser utilizado para almacenar valores.

Algunos de los tipos de datos más comunes en C:

1. Enteros (int): Representan números enteros, tanto positivos como negativos, sin decimales. Por ejemplo:

```
int edad = 25;
int temperatura = -5;
```

2. Flotantes (float y double): Representan números con decimales. Los valores de tipo `float` tienen una precisión menor que los de tipo `double`. Por ejemplo:

```
float peso = 68.5;
double pi = 3.14159265358979323846;
```

3. Caracteres (char): Representa un solo carácter, como una letra, un número o un símbolo, entre comillas simples. Por ejemplo:

```
char letra = 'A';
char simbolo = '$';
```

4. Cadenas de caracteres (char[]): Representa una secuencia de caracteres. Cada carácter se almacena en un elemento del array. Por ejemplo:

```
char nombre[] = "Juan";
```

5. Booleanos (bool): Representa valores de verdadero (`true`) o falso (`false`). Por ejemplo:

```
bool esMayorDeEdad = true;
bool tieneDescuento = false;
```

Este tipo de datos es útil para representar estados lógicos utilizados en los pines de entrada o salida de Arduino. Un valor lógico false se traduce a 0V en el pin y un valor lógico true a 5 V.

Tabla resumen de tipos de datos

Tipo	Longitud en Bytes	Rango de Valores
boolean	1	true, false
char	1	-128 a +127
unsigned char	1	0 a 255
byte	1	0 a 255
int	2	-32,768 a 32,767
unsigned int	2	0 a 65,535
word	2	0 a 65,535
long	4	-2,147,483,648 a 2,147,483,647
unsigned long	4	0 a 4,294,967,295
float	4	-3.4028235E+38 a 3.4028235E+38
double	4	-3.4028235E+38 a 3.4028235E+38
string	?	Una secuencia de caracteres terminada con un nulo ('\0')
array	?	Una secuencia de valores del mismo tipo, contenidos con nombre de variable

Es importante tener en cuenta que el tamaño en bytes y el rango de valores pueden variar dependiendo de la plataforma y el compilador utilizado. Además, la longitud del tipo `string` no está especificada en bytes, ya que es un tipo de dato que está compuesto por una secuencia de caracteres terminada con un carácter nulo ('\0'). La longitud de un `string` puede variar según la cantidad de caracteres que contenga y no está limitada a un tamaño fijo en bytes.
Utilizamos variables para almacenar valores y constantes para valores que no cambian durante la ejecución del programa. Las variables se declaran indicando su tipo y su nombre, y pueden ser modificadas durante la ejecución del programa. Por otro lado, las constantes se declaran utilizando la palabra clave `const` y su valor no puede ser cambiado después de su declaración.

Ejemplos de declaración de variables y constantes:

```
int edad = 30; // Variable entera
float altura = 1.75; // Variable flotante
const int NUMERO_MAXIMO = 100; // Constante entera
char inicial = 'J'; // Variable caracter
char nombre[] = "Maria"; // Variable cadena de caracteres
```

Operadores y expresiones en C

En Arduino existen diversos operadores y expresiones que permiten realizar cálculos y manipular datos.

Operadores Aritméticos

- Suma (+): Realiza la adición de dos operandos.

- Resta (-): Realiza la resta de dos operandos.

- Multiplicación (*): Realiza la multiplicación de dos operandos.

- División (/): Realiza la división de dos operandos.

- Módulo (%): Devuelve el resto de la división entre dos operandos.

Operadores de Asignación

- Asignación (=): Asigna el valor de la expresión del lado derecho a la variable del lado izquierdo.

- Suma y asignación (+=): Suma el valor del lado derecho al valor actual de la variable y guarda el resultado en la variable.

- Resta y asignación (-=): Resta el valor del lado derecho al valor actual de la variable y guarda el resultado en la variable.

- Multiplicación y asignación (*=): Multiplica el valor del lado derecho al valor actual de la variable y guarda el resultado en la variable.

- División y asignación (/=): Divide el valor actual de la variable por el valor del lado derecho y guarda el resultado en la variable.

- Módulo y asignación (%=): Realiza la operación de módulo con el valor del lado derecho y guarda el resultado en la variable.

Operadores de Comparación

- Igual (==): Comprueba si dos operandos son iguales.

- Diferente (!=): Comprueba si dos operandos son diferentes.

- Mayor que (>): Comprueba si el operando de la izquierda es mayor que el de la derecha.

- Menor que (<): Comprueba si el operando de la izquierda es menor que el de la derecha.

- Mayor o igual que (>=): Comprueba si el operando de la izquierda es mayor o igual que el de la derecha.

- Menor o igual que (<=): Comprueba si el operando de la izquierda es menor o igual que el de la derecha.

Operadores Lógicos

- AND lógico (&&): Devuelve verdadero si ambas expresiones son verdaderas.

- OR lógico (||): Devuelve verdadero si al menos una de las expresiones es verdadera.

- NOT lógico (!): Invierte el valor de verdad de la expresión (si es verdadera, se convierte en falsa, y viceversa).

Operadores de Incremento y Decremento

- Incremento (++): Aumenta en uno el valor de la variable.

- Decremento (--): Disminuye en uno el valor de la variable.

Expresiones

Una expresión es una combinación de variables, constantes y operadores que se evalúan para obtener un valor. Por ejemplo:

```
int a = 5;
int b = 10;
int resultado = a + b; // La expresión a + b se evalúa y se almacena en la variable resultado
```

Podemos utilizar los operadores enumerados para construir cualquier expresión.

Más sobre variables, el concepto de ámbito (Scope)

El ámbito puede ser definido como una propiedad particular de una variable (y funciones, como veremos más adelante).

Considerando el código fuente, **el ámbito de una variable es aquella parte del código donde esta variable es visible y utilizable.**

Una variable puede ser global y, en ese caso, es visible y utilizable en cualquier lugar del código fuente. Pero también una variable puede ser local, declarada dentro de una función, por ejemplo, y solamente es visible dentro de esta función en particular.

La propiedad de ámbito para una variable es establecida implícitamente por el lugar de declaración de la variable en el código.

Probablemente acabas de comprender que cada variable puede ser declarada globalmente pero es una buena práctica hacer que cada parte del código acceda solamente las variables que debe conocer y nada más.

Tratar de minimizar el ámbito de las variables es definitivamente el enfoque correcto. Observa el siguiente ejemplo:

```
// Esta variable se declara en el nivel más alto, haciéndola visible en todas partes
int enteroGlobal;
void setup() {
  // ... algún código
}
void loop() {
  int a; // a es visible solo dentro de la función loop
  algunaFuncion(); // llamando a la función global algunaFuncion
  // ... algún otro código
}
void algunaFuncion() {
  // ... otro código más
  int variableLocal; // variableLocal es visible solo en la función algunaFuncion
}
```