Arduino: Programmazione

Programmazione formalmente ispirata al **linguaggio C** da cui deriva.

I programmi in **ARDUINO** sono chiamati **“Sketch”**.

Un programma è una **serie di istruzioni** che vengono lette dall’alto
verso il basso e convertite in eseguibile e poi trasferite sulla scheda Arduino.

Arduino: Programmazione

Linguaggio C: Struttura di un programma

```
main()
{
      // istruzioni di programma da eseguire
}
```

Arduino: Programmazione

Linguaggio Arduino: Struttura di un programma

```
void setup()
{
    // istruzioni da eseguire una sola volta
}

void loop()
{
    // istruzioni da eseguire ciclicamente
}
```

Arduino: Programmazione

Linguaggio C: Compilazione ed esecuzione

Per eseguire un programma in linguaggio C è necessario compilarlo.

Il processo di compilazione trasforma le istruzioni scritte in linguaggio naturale in una serie di istruzioni macchina eseguibili dal calcolatore.

Questa operazione converte il “codice sorgente” in “codice oggetto” eseguibile dalla macchina (codice macchina)

Per ARDUINO avviene la stessa cosa ma la fase di compilazione si traduce e completa essenzialmente nella fase di caricamento del codice sulla scheda stessa.

Arduino: Programmazione

Linguaggio Arduino: Struttura di un programma

setup()

Fase di preparazione

loop()

Fase di esecuzione

Arduino: Programmazione

Istruzioni:

Le istruzioni di programma vanno terminate con il punto e virgola.

Un gruppo di istruzioni deve essere racchiuso tra parentesi graffe.

Le istruzioni del preprocessore NON vanno terminate con il punto e virgola.

Arduino: Programmazione

Un esempio di programma per ARDUINO:

```
#define LED 13

int INGRESSO = 7;

void setup() {
   pinMode(LED, OUTPUT);
   pinMode(INGRESSO, INPUT);
}

void loop() {
   if (digitalRead(INGRESSO)) digitalWrite(LED, HIGH);
   if (!digitalRead(INGRESSO)) digitalWrite(LED, LOW);
}
```

Arduino: Programmazione

Per comprendere il programma di esempio visto è necessario introdurre i seguenti concetti:

- **Variabili**, **Costanti** e **Tipo di dato**
- **Funzioni** (parametri di ingresso e parametri di ritorno)
- **Operazioni Aritmetiche** e **Operatori Logici** e di **Confronto**
- **Istruzioni condizionali**
- Istruzioni di **preprocessore**
- *Costanti di Arduino*
- *Funzioni di Arduino*

Arduino: Programmazione

Le variabili:

La variabile è una locazione di memoria alla quale attribuiamo un nome e nella quale memorizziamo un valore.

Le variabili prima di essere usate devono essere dichiarate.

Le variabili hanno una portata di visibilità più o meno ampia a seconda di dove si trova la loro dichiarazione.

Arduino: Programmazione

Un esempio di programma per ARDUINO:

```
#define LED 13

int INGRESSO = 7;

void setup() {
   pinMode(LED, OUTPUT);
   pinMode(INGRESSO, INPUT);
}

void loop() {
   if (digitalRead(INGRESSO)) digitalWrite(LED, HIGH);
   if (!digitalRead(INGRESSO)) digitalWrite(LED, LOW);
}
```

Arduino: Programmazione

Tipi di dato:

Tipi di dichiarazione	Rappresentazione	N. di byte	Intervallo
Bolean			True – False / On –Off / High - Low
Char	Carattere	1 (8 bit)	- 127 +127
Byte	Carattere	1 (8 bit)	0 +255
Int	Numero intero	2 (16 bit)	-32.768 + 32.767
Unsigned int	Numero intero	2 (16 bit)	0 + 65.535
Short	Numero intero "corto"	2 (16 bit)	
Long	Numero intero "lungo"	4 (32 bit)	-2.147.483.648 +2.147.483.647
Float	Numero reale	4 (32 bit)	-3.4028235E+38 a + 3.4028235E+38
Double	Numero reale "lungo"	8 (64 bit)	$1.7976931348623157 \times 10^{308}$

Arduino: Programmazione

Le costanti:

La costante è una locazione di memoria alla quale attribuiamo un nome e nella quale memorizziamo un valore.

Le costanti prima di essere usate devono essere dichiarate.

Esistono delle costanti in ARDUINO pre-dichiarate e sempre disponibili in qualsiasi punto del programma.

Arduino: Programmazione

Un esempio di programma per ARDUINO:

```
#define LED 13

int INGRESSO = 7;

void setup() {
   pinMode(LED, OUTPUT);
   pinMode(INGRESSO, INPUT);
}

void loop() {
   if (digitalRead(INGRESSO)) digitalWrite(LED, HIGH);
   if (!digitalRead(INGRESSO)) digitalWrite(LED, LOW);
}
```

Arduino: Programmazione

Le COSTANTI in Arduino:

Il linguaggio Arduino ha una serie di parole chiave predefinite con valori speciali.

HIGH e **LOW** si usano, quando si vuole accendere o spegnere un **pin** di Arduino.

INPUT e **OUTPUT** si usano per impostare un determinato **pin** come ingresso o uscita.

TRUE / FALSE indica il fatto che una condizione o un'espressione è vera o falsa.

Arduino: Programmazione

I commenti:

I commenti inseriti all'interno del sorgente tra le istruzioni di programma permettono di documentare le parti del programma.

I commenti sono ignorati dal processore.

Ce ne sono di due tipi commenti a blocchi e commenti a singola linea, naturalmente con sintassi diversa.

Arduino: Programmazione

Un esempio di programma per ARDUINO:

```
#define LED 13 // preprocessore: definisco una costante

int INGRESSO = 7; /*  variabile intera
                      con valore assegnato
                      uguale a 7
                   */

void setup() {
   pinMode(LED, OUTPUT);
   pinMode(INGRESSO, INPUT);
}

void loop() {
   if (digitalRead(INGRESSO)) digitalWrite(LED, HIGH);
   if (!digitalRead(INGRESSO)) digitalWrite(LED, LOW);
}
```

Arduino: Programmazione

Funzioni (parametri di ingresso e parametri di ritorno**):**

Una funzione è un **blocco di codice** che ha un nome ben definito, quindi è un blocco di istruzioni che vengono eseguiti quando la funzione viene chiamata.
Le funzioni sono utilizzate per eseguire **operazioni ripetitive** in modo da ridurre il codice programma ed evitare quindi confusione nel programma stesso.
Le funzioni sono dichiarate all'inizio del programma e specificate dal tipo di funzione.
La struttura della funzione è la seguente:

<**Tipo del valore restituito**> <**nome funzione**> (<**elenco dei parametri**>)

Dopo il tipo, occorre dichiarare il nome dato alla funzione e tra parentesi i parametri che vengono passati alla funzione.

Arduino: Programmazione

Un esempio di dichiarazione di Funzione per ARDUINO:

```
int delayVal()  // funzione senza parametri
{
   int v;         // dichiaro la variabile
   v = analogRead(pot); // funzione arduino
   v = v / 4;     // espressione aritmetica
   return v;      // valore di ritorno della funzione
}
```

Arduino: Programmazione

Un esempio di Blocco di Istruzioni per ARDUINO:

```
{
   digitalWrite (pin, HIGH);  // il 'pin' è su
   delay (1000);              // un secondo di pausa
   digitalWrite (pin, LOW);   // il 'pin' è giù
   delay (1000);              // un secondo di pausa
}
```

Arduino: Programmazione

Operazioni aritmetiche:

Gli operatori aritmetici sono addizione, sottrazione, moltiplicazione e divisione.

Esempi:

```
y = y + 3;
x = x - 7;
i = j * 6;
r = r / 5;
```

Arduino: Programmazione

Compound assignments - Assegnazioni compound:

Si tratta di **operatori speciali** che si usano per rendere più conciso il codice di programma.
Esso combina un'operazione aritmetica con un'assegnazione di variabile.

Esempi:
a = a + 1 si può scrivere **a++**
a = a + 2 si può scrivere **a += 2**

Attenzione!
Se scrivo: **value++**, prima valuta la variabile value e poi la incrementa di 1.
Se invece scrivo: **++value**, prima incrementa di 1 e poi lo valuta.
Lo stesso vale per –- (meno meno).

Arduino: Programmazione

Compound assignments - Assegnazioni compound:

Questi operatori speciali si trovano comunemente nei cicli **for.** Le assegnazioni più comuni includono:

x++; // è uguale a x = x + 1, incrementa la variabile x di +1
x –; // è uguale a x = x - 1, decrementa x di -1
x += y; // è uguale a x = x + y, incrementa x di + y
x -= y; // è uguale a x = x - y, decrementa x di -y
x *= y; // è uguale a x = x * y, moltiplica x per y
x /= y; // è uguale a x = x / y, divide x per y

Nota: per esempio, x *= 3 da come risultato il triplo del valore di x e poi viene riassegnato alla variabile x.

Arduino: Programmazione

Operatori di confronto:

Alcune volte si ha bisogno di confrontare una variabile o una costante con un'altra. Il confronto si usa nelle istruzioni condizionate **if**, **while** e **for** per verificare se una determinata condizione è vera.

Esempi:

```
x == y      // x è uguale a y
x != y      // x è diverso da y
x < y       // x è minore di y
x > y       // x è maggiore di y
x <= y      // x è minore o uguale a y
x >= y      // x è maggiore o uguale a y
```

Arduino: Programmazione

Gli operatori logici o operatori booleani:

Gli operatori logici sono un modo per confrontare due espressioni. Si usano anche quando si vogliono combinare diverse condizioni. Restituiscono una funzione TRUE o FALSE.
Ci sono tre operatori logici AND, OR e NOT, che vengono utilizzati in istruzioni if:

AND logico:
if (x > 0 **&&** x < 5) // vera solo se entrambe le espressioni sono vere

OR logico:
if (x > 0 **||** y> 0) // vero se uno delle due espressioni è vera

NOT logico:
if (**!**x > 0) // vera solo se l'espressione è falsa

Arduino: Programmazione

Controllo del flusso di esecuzione del codice di programma:

1. **Controllo di flusso (IF) -** istruzione condizionale
2. **Controllo di flusso (IF … ELSE) -** istruzione condizionale
3. **Controllo di flusso (FOR) –** ciclo “finito” e determinato
4. **Controllo di flusso (WHILE) –** ciclo a condizione iniziale
5. **Controllo di flusso (DO … WHILE) –** ciclo a condizione finale

Arduino: Programmazione

Controllo di flusso (IF):

L'istruzione **if** verifica se una certa condizione.
Se l'espressione è vera lo sketch esegue le istruzioni che seguono.
Se falsa il programma ignora la dichiarazione.

Un esempio:

```
if (variabile == valore)
{
   // esegue blocco di istruzioni
}
```

Arduino: Programmazione

Controllo di flusso (IF … ELSE):

Se l'espressione contenuta all'interno delle parentesi tonde è vera,
viene eseguito il codice di programma che segue. Se l'espressione è falsa vengono eseguite le righe di codice subito dopo l'istruzione else.

```
#define Acceso 1 // definisce Acceso = 1
#define Spento 0 // definisce Spento = 0

if (pulsante == ON)
{
   digitalWrite(rele, acceso);
}
else
{
   digitalWrite(rele, spento);
}
```

Arduino: Programmazione

Controllo di flusso (FOR):

Il for viene utilizzato per ripetere un blocco di istruzioni racchiuso tra parentesi graffe un determinato numero di volte. Viene utilizzato un contatore per incrementare e terminare il ciclo. Esso è composto da tre parti, separate da punto e virgola (;):

```
for (inizializzazione; condizione; espressione)
{
      esegui istruzioni;
}
```

Esempio:

```
for (int A = 0; A < 10; A++)
{
   // esegui le istruzioni;
}
```

Arduino: Programmazione

Controllo di flusso (WHILE):

È un comando simile a If.
Il ciclo while esegue all'infinito le istruzioni racchiuse tra le parentesi graffe fino a quando la condizione racchiusa dentro le parentesi tonde diventa falsa.
Qualcosa deve far cambiare la variabile in esame, o il ciclo while non potrà mai uscire.

Esempio:

```
while (someVariable < 200) // verifica se meno di 200
{
    doSomething;
    // esegue le istruzioni racchiuse tra le parentesi graffe
    SomeVariable++;
    // la variabile viene incrementata di una unità
}
```

Arduino: Programmazione

Controllo di flusso (DO … WHILE):

Il ciclo do-while si comporta come l'istruzione while ma con una importante differenza: esegue almeno una volta l'istruzione all'interno del ciclo do-while. Quindi viene usato quando si vuole che il codice dentro le parentesi graffe venga eseguito almeno una volta prima di verificare la condizione.

Esempio:

```
do
{
   x = readSensors();  // assegna il valore di readSensors() a x
   delay (50);         // pausa di 50 millisecondi
} while (x < 100);     // ciclo finché x è minore di 100
```

Arduino: Programmazione

Ingressi / Uscite digitali:

- **pinMode** (pin, mode)

Utilizzato in void setup (), serve per configurare un determinato pin e stabilire se deve essere un ingresso o un'uscita.

- **digitalRead** (pin)

L'istruzione permette di leggere lo stato di un pin di input e restituisce un valore HIGH se al pin è applicato un tensione o un valore LOW se non è applicato nessun segnale. Il pin può essere specificato come una variabile o costante (0-13).

- **digitalWrite** (pin, valore)

Attiva o disattiva un pin digitale, quindi l'istruzione pone il pin di uscita a livello logico HIGH o LOW. Il pin può essere specificato come una variabile o una costante (0-13).

Arduino: Programmazione

Ingressi / Uscite analogici:

- **analogRead** (pin)

Legge il valore di tensione applicato ad un pin di input analogico con una risoluzione pari a 10 bit. Questa funzione restituisce un numero intero compreso tra 0 e 1023.

- **analogWrite** (pin, value)

Cambia la percentuale della modulazione di larghezza di impulso (Pulse Width Modulation - PWM) su uno dei pin contrassegnati dalla sigla PWM.

Arduino: Programmazione

Orologio Interno:

- delay (ms)

Mette in pausa un programma per la quantità di tempo specificato in millisecondi. Il valore 1000 è pari a 1 secondo.

Esempio: delay (1000); // attende un secondo

- DelayMicroseconds(us)

Mette in pausa il programma per la quantità specificata di microsecondi.

Esempio: delayMicroseconds (1000); // attende un millesimo di secondo

- Millis ()

Restituisce il numero di millisecondi da quando la scheda Arduino ha iniziato l'esecuzione del programma corrente. Il tipo di dato è un unsigned long.

value = Millis (); // imposta la variabile 'value' al numero di millisecondi Millis()

Riferimenti

valentino.stampone@gmail.com

https://www.facebook.com/valentino.stampone

http://about.me/valentino.stampone