Istruzioni Standard Arduino

Qui di seguito sono riportate le istruzioni standard supportate dal linguaggio di programmazione di Arduino.

STRUTTURA

Il codice di qualsiasi programma per Arduino è composto essenzialmente di due parti:

void setup() - Questo è il posto dove mettiamo il codice di inizializzazione. Inizializza tutte le impostazioni e le istruzioni della

scheda (gli INPUT e OUTPUT) prima che il ciclo principale del programma si avvii.

void loop() - E' il contenitore del codice principale del programma. Contiene una serie di istruzioni che possono essere ripetute

una dopo l'altra fino a quando non spegniamo la scheda Arduino.

COSTANTI

Nella scheda Arduino è inserita una serie predefinita di parole chiave con valori speciali. High e Low sono usati per esempio quando si vuole accendere o spegnere un Pin di Arduino. INPUT e OUTPUT sono usate per definire se uno specifico Pin deve essere un dato di entrata o un dato di uscita. True e False indicano il rispettivo significato italiano: se abbiamo un'istruzione, la condizione può essere vera o falsa.

VARIABILI

Sono aree della memoria di Arduino dove si possono registrare dati e intervenire all'interno del programma. Come dice il nome stesso, le variabili possono essere cambiate tutte le volte che vogliamo. Quando si dichiara una variabile bisogna dichiararne anche il tipo. Questo significa dire al processore le dimensioni del valore che si vuole memorizzare. Ne esistono di diversi tipi:

boolean - Può assumere solamente due valori: vero o falso.

char - Contiene un singolo carattere. L'Arduino lo registra come un numero (ma noi vediamo il testo). Quando i caratteri sono

usati per registrare un numero, possono contenere un valore compreso tra -128 e 127.

byte - Può contenere un numero tra 0 e 255. Come un carattere usa solamente un byte di memoria.

int - Contiene un numero compreso tra -32'768 e 32'767. E' il tipo di variabile più usata e usa 2 byte di memoria.

unsigned int - Ha la stessa funzione di int, solo che non può contenere numeri negativi, ma numeri tra 0 e 65.535.

long - E' il doppio delle dimensioni di un int e contiene i numeri da -2'147'483'648 a 2'147'483'647.

unsigned long - Versione senza segno di long va da 0 a 4'294'967"295.

float - Può memorizzare numeri con la virgola. Occupa 4 bytes della RAM.

double - A doppia precisione in virgola mobile con valore massimo di 1'7976931348623157x10^308.

string - Un set di caratteri ASCII utilizzati per memorizzare informazioni di testo. Per la memoria, usa un byte per ogni carattere

della stringa, più un carattere NULL che indica ad Arduino la fine della stringa. Esempio:

*char string1[] = "Hello"; // 5 caratteri+carattere NULL
char string2[6]="Hello" // La stessa cosa di sopra
*

N.B. ogni istruzione deve sempre terminare con ";" in tale linguaggio. Inoltre "//" è usato per inserire commenti che aiutano a comprenderlo.

array - un elenco di variabili accessibili tramite un indice. Vengono utilizzate per creare tabelle di valori facilmente accessibili. Come

esempio se si vuole memorizzare diversi livelli di luminosità di un LED possiamo creare 4 variabili, una per ogni livello di luminosità.

Si può utilizzare una semplice array come:

int Luce[5]={0,25,50,100};

Nel tipo della variabile la parola "array" non si dichiara, ma si usano i simboli [] e {}.

STRUTTURE DI CONTROLLO

Il linguaggio di Arduino include parole chiave per controllare il progetto logico del nostro codice.

If…else - Permette di prendere delle decisioni all'interno del programma, ma deve essere seguito da una domanda sotto forma di

espressione tra parentesi. Se la domanda è vera tutto ciò che segue verrà eseguito. Se falso verrà eseguito tutto il codice che

segue else. If è possibile usarlo senza usare necessariamente else. Esempio:

*if (val=1){digitalWrite(LED, HIGH);} // (val=1) è la domanda se è vera esegue ciò che è fra parentesi
*

For - Ripete il codice per un numero predefinito di volte.

for(int i=0;i<10;i++){Serial.print("Ciao");} //stampa 10 volte "Ciao"

Switch - E' come un interruttore nel corso del programma. Fa prendere al programma diverse direzioni in base al valore della

variabile (il suo nome deve essere messo tra parentesi dopo switch). E' utile perché può sostituire lunghe serie di if.

*switch(valore sensore){
case 38:
digitalwrite(12, High);break;
case 55:
digitalwrite(3, High);break;
default: // si usa per*

*indicare tutti i casi in cui non è ne 38 ne 55
digitalwrite(12, Low);
digitalwrite(13, Low);}
*

While - Esegue un blocco di codice fino a quando una certa condizione posta tra le parentesi è vera.

*while(valore sensore<500){
digitalWrite(13, HIGH);
delay(100);
digitalWrite (13, HIGH);*

*
delay(100);
Valoresensore=analogRead(1);
*

Do…While - E' uguale a while solo che il codice è avviato prima che la condizione sia verificata. Si usa quando si vuole eseguire il

codice almeno una volta prima che la condizione sia valutata. Esempio:

*do {
digitalWrite(13,HIGH);
delay(100);
digitalWrite(13,HIGH);
DELAY (100);*

*
valore sensore=analogread(1);
}*

Break - Questo termine consente di bloccare il ciclo e continuare ad eseguire il codice fuori dal ciclo. Viene utilizzato anche per

separare le varie condizioni nella funzione Switch.

Continue - Questo comando fa saltare il resto del codice all'interno del ciclo, e riavvia il ciclo. Esempio:

*for(luminosità=0; luminosità<200; luminosità++)
f((x>120) && (x<180)) continue;
analogWrite(PWMPin, luminosità);
delay(20);
}*

Return - Ferma una funzione che si sta eseguendo e restituisce un risultato. E' possibile infatti usarlo per restituire un valore da

una funzione. Esempio chiama una funzione “calcolaumidità” e ritorna il valore dell’umidità.

*Int calcolaumidità() {
Int umidità=0;*
*
umidità0(analogread(0)+45/100)/100;
return umidità;
}*

OPERAZIONI ARITMETICHE

Si può usare Arduino per compiere operazioni matematiche complesse con una semplice sintassi: + e – indicano addizione e

sottrazione, * indica la moltiplicazione, e / la divisione.
C’è un operatore in più in questo linguaggio chiamato “Modulo” che è un comando che restituisce il resto di una divisione. Esempio:

*a=3+3; luminosità=((12*valore sensore)/4);*

OPERATORI DI COMPARAZIONE

Quando si specificano delle condizioni ci sono vari operatori che tu puoi usare:

== Uguale a

> maggiore di

< minore di

!= diverso da

<= minore o uguale

>= maggiore o uguale

OPERATORI BOOLEANI

Sono usati quando si vogliono combinare più condizioni, ad esempio se vogliamo verificare se il valore di un sensore è tra 1 e 5

basta scrivere:

if(sensore=>1) && (sensore=<=5);

Esistono tre tipi di operazioni booleane: &&(And), ||(Or), !(Not).

OPERATORI COMPUTAZIONALI

Servono a ridurre la mole di un codice e a renderlo più semplice e chiaro per operazioni semplici come incrementare o

decrementare una variabile.

Esempio: val=val+1; è come dire val++

incremento (++) e decremento (--)

++ e -- incrementano/decrementano una variabile di 1; lo stesso è applicabile a +=, -=, *=, /= .

Esempio: le seguenti espressioni sono equivalenti:

val=val+5;

Val+=5;

FUNZIONI INPUT E OUTPUT

Arduino include funzioni per la gestione degli Input e degli Output.

pinMode(pin,mode) - Riconfigura un pin digitale a comportarsi come uscita o come entrata.

pinMode(13,INPUT) - imposta il pin 13 come Input.

digitalWrite(pin,value) - imposta un pin digitale ad ON o a OFF.

digitalWrite(7,HIGH) - imposta come digitale il pin 7.

int digitalRead(pin) - Legge lo stato di un input Pin, ritorna HIGH se il Pin riceve della tensione oppure LOW se non c'è tensione

applicata.

```
Val=digitalRead(7); // legge il pin 7 dentro a val
```

Int analogRead(pin) - Legge la tensione applicata a un ingresso analogico e ritorna un numero tra 0 e 1023 che rappresenta le

tensioni tra 0 e 5 V.

```
val=AnalogRead(0); // legge l'ingresso analogico 0 dentro a val
```

analogWrite(pin,value) - Cambia la frequenza PWM su uno dei pin segnati PWM, nella voce pin si può mettere 11 10 9 6 5 3,

value invece può essere un valore da 0 a 255 che rappresenta la scala da 0 a 5 V.

```
analogWrite(9,128);
```

shiftOut(dataPin, clock, Pin, bit, Order, value) - Invia i dati ad un registro. Questo protocollo usa un pin per i dati e uno per il

clock. bitOrder indica l'ordine dei bytes (least significant byte=LSB, most significant byte=LMB) e value è il byte da inviare. Esempio:

```
shiftOut(dataPin, Clock Pin, LSBFIRST, 255);
```

insigned long pulseIn(pin, value) - misura la durata degli impulsi in arrivo su uno degli ingressi digitali. E' utile ad esempio per

leggere alcuni sensori a infrarossi o alcuni accelerometri che emettono impulsi di diversa durata.

```
Tempo=pulsin(8,HIGH);
```

FUNZIONI DI TEMPO

Arduino include alcune funzioni per misurare il tempo trascorso e anche per mettere in pausa il nostro programma.

Insigned long millis() - Ritorna il numero in millisecondi trascorsi dall'inizio del programma, esempio:

```
durata=millis()-tempo // calcola il tempo trascorso prima di "tempo"
```

delay(ms) - Mette in pausa il programma per un numero di millisecondi specificato.

```
delay(1000); //stoppa il programma per 1 secondo
```

delayMicroseconds(us) - Come delay mette in pausa il programma ma l'unità di misura è molto più piccola, parliamo di microsecondi.

```
delayMicroseconds(2000); // aspetta per 2 millisecondi (1000us=1ms)
```

FUNZIONI MATEMATICHE

Arduino include molte funzioni matematiche comuni. Servono, per esempio, per trovare il numero max o il numero min.

min (x,y) - Ritorna il più piccolo fra x e y. Esempio:

```
Val= min(5,20);  // val adesso è 5
```

max(x,y) - Ritorna il più grande fra x e y.

abs(x) - Ritorna il valore assoluto di x, ossia trasforma i numeri negativi in numeri positivi. Se x fosse 5 ritorna 5, ma anche se x

fosse -5 ritorna sempre 5. Esempio:

```
Val= abs(-5) // val vale 5
```

constrain(x,a,b) - Ritorna il valore "x" costretta tra "a" e "b". Ciò vuol dire che se "x" è minore di "a" ritornerà semplicemente

"a" e se x è maggiore di "b" restituirà semplicemente il valore di "b".

map(value, fromLow, fromHigh, toHigh) - Associa un valore che sta nel range fromLow e maxlow in un nuovo range che va

da toLow a toHigh. E' molto utile per processare valori provenienti da sensori analogici. Esempio:

```
val=map(analogRead(0),0,1023,100,200);  // associa il valore analogico 0 ad un valore tra 100 e 200
```

double pow(base,exponent) - Restituisce come risultato la potenza di un numero. Si deve indicare la base e l'esponente.

Double sqrt(x) - Restituisce la radice quadrata di un numero x.

Double sin(rad) - Restituisce il seno dell'angolo specificato in radianti. Esempio:

```
Double sine= sine(2); // circa 0.909297370
```

Double cos(rad) - Restituisce il coseno dell' angolo specificato in radianti.

Double tan(rad) - Restituisce il valore della tangente di un angolo specificato in radianti.

FUNZIONI NUMERI RANDOM

Se si ha bisogno di generare numeri random (a caso), Arduino ci viene incontro con alcuni comandi standard per generarli.

randomSeed(seed) - Anche se la distribuzione di numeri restituita dal comando random() è essenzialmente casuale, la sequenza

è prevedibile. randomSeed(seed) inizializza il generatore di numeri pseudo-casuali, facendola partire da un punto arbitrario nella sua

sequenza casuale.

Long random(min,max) - Restituisce un valore long intero di valore compreso fra min e max -1. Se min non è specificato il suo

valore minimo è 0. Esempio:

```
long random= random(13); // numero compreso fra 0 e 12
```

COMUNICAZIONE SERIALE

Queste sono le funzione seriali cioè quelle funzioni che Arduino usa per comunicare tramite la porta Usb del nostro Pc.

Serial.begin(speed) - Prepara Arduino a mandare e a ricevere dati tramite porta seriale. Possiamo usare generalmente 9600 bits

per secondo con la porta seriale dell'Arduino, ma sono disponibili anche altre velocità, di solito non si supera i 115.200 bps.

```
Serial.print(data)Serial.begin(9600);
```

Serial.print(data,codifica) - Invia alcuni dati alla porta seriale. La codifica è opzionale.

```
Serial.print(32);  // stampa 32

Serial.Print(32, DEC);   // stampa 32 come sopra

Serial.Print(32, OCT);  // 40  (stampa10 in ottale)

Serial.Print(32 , BIN);  // 100000 (stampa 10 in binario)

Serial.Print(32 , BYTE);  // "Space" valore associato nella tabella ASCII
```

Int Serial.available() - Ritorna quanti bytes non ancora letti sono disponibili sulla porta Serial per leggerli tramite la funzione

read(). Dopo aver read() tutti i bytes disponibili Serial.Available restituisce 0 fino a quando nuovi dati non giungono sulla Porta.

Int.Serial.read() - Recupera un byte di dati in entrata sulla porta Seriale.

```
int data= Serial.read();
```

Poichè i dati possono giungere nella porta seriale prima che il programma li possa leggere(per la velocità), Arduino salva tutti i dati

in un buffer. Se è necessario ripulire il buffer e aggiornarlo con i dati aggiornati, usiamo la funzione flush().

```
Serial.flush();
```