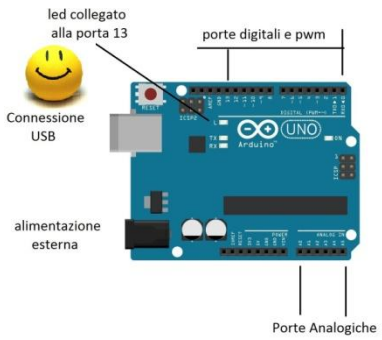
Arduino: manuale di programmazione wiring



Scrivere un programma non e' difficile, basta avere le idee chiare sull'obiettivo e sulla strada per raggiungerlo.

Per iniziare ad utilizzare Arduino basta leggere l'**introduzione** di questo manuale e poi passare subito a replicare i primi esercizi di questa raccolta.

Leggendo e copiando i programmi, si arrivera' a comprenderli, quindi a modificarli ed infine a scriverne di nuovi.

E' piu' facile di quel che sembra. Basta non scoraggiarsi e magari leggere un paio di volte qualche passaggio che sembra difficile.

Basta solo un po' di volonta', e solo all'inizio. E poi sara' solo una piacevole strada in discesa

In questo breve manuale sono trattate solo alcune delle istruzioni previste dal linguaggio wiring.

Sono state considerate solo le piu' diffuse o meglio, quelle considerate piu' utili. Con questo sottoinsieme di istruzioni e' possibile affrontare tutte le problematiche di gestione di sensori ed attuatori e scrivere programmi anche di notevole complessita'.

Per chiarimenti o suggerimenti scrivere a giocarduino@libero.it

I capitoli di questo manuale

Introduzione

Wiring e' il linguaggio di programmazione, derivato dal C++, utilizzato per scrivere programmi installabili sul microcontrollore ATmega328 che equipaggia Arduino uno.

Il microcontrollore legge, attiva e disattiva le porte di Arduino seguendo le istruzioni contenute in uno ***sketch***, e cioe' un programma scritto in ***wiring*** e ***compilato*** (e cioe' tradotto in linguaggio macchina) attraverso l'**IDE**, un ambiente di sviluppo gratuito operante su pc.

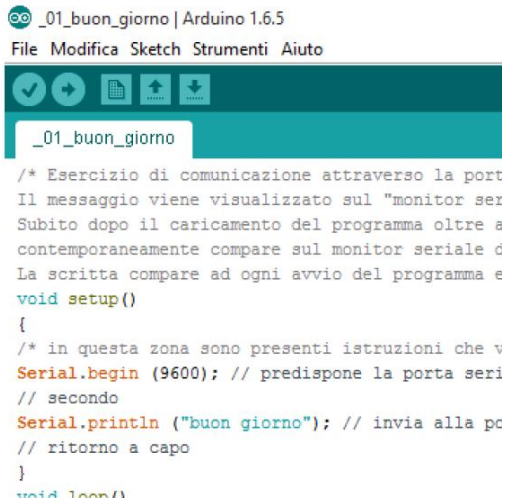
L'IDE e' quindi indispensabile per operare su Arduino ed e' liberamente scaricabile da qui:

https://www.arduino.cc/en/main/software

L'IDE consente di utilizzare il PC per scrivere il programma, compilarlo e trasferirlo, tramite una connessione USB, su Arduino.

L'IDE presenta una finestra riservata alla scrittura del programma, alcune icone per le funzioni di verifica, compilazione, carico e salvataggio dei programmi ed una serie di menu' a tendina abbastanza intuitivi. Nella sezione "aiuto" e' anche presente un'esaustiva spiegazione (in inglese) di ogni singolo elemento di programmazione (variabili, funzioni, istruzioni e relativa sintassi).

Sotto l'aspetto pratico, una volta scaricata, installata ed aperta l'IDE, bisogna specializzarla (una sola volta, al primo utilizzo), selezionando "strumenti" e poi "scheda arduino…." e quindi, dal menu che si apre, il tipo di scheda Arduino di cui si dispone (Arduino uno, Arduino mega, Arduino due, ecc.).



La testata dell'IDE

Ora, per compilare, caricare e lanciare un programma si deve:

- Collegare Arduino al pc mediante un cavo usb. Appena inserito il cavo alcune spie di Arduino si illumineranno, a conferma dell'avvenuto collegamento.
- Copiare (o digitare) il programma nella zona di editing dell'IDE, parzialmente gia' occupata da alcune istruzioni predefinite (che dovranno essere cancellate e sostituite dal programma copiato).
- Premere il pulsante di compilazione (la freccia rivolta a destra, piazzata nella riga dei comandi, in alto a sinistra)
- Attendere il completamento della compilazione, rilevabile dalle scritte bianche che compaiono nella parte bassa dell'IDE e dal pulsare, per pochi attimi, dei led di Arduino.

Il programma, una volta compilato viene automaticamente caricato sulla scheda ed a questo punto Arduino inizia immediatamente a lavorare e a compiere le azioni per le quali e' stato programmato.

Come primo programma da far girare e' consigliabile ricorrere al "buon giorno" reperibile qui'. Il programma, molto semplice e facilmente interpretabile, non richiede alcun particolare componente (solo Arduino, il cavo usb ed il pc) e non solo consente di rendersi conto di cos'e' e di come funziona la compilazione, ma anche di impratichirsi nell'uso del monitor seriale, residente sull'IDE.

Se durante la compilazione appaiono delle segnalazioni di colore rosso/arancione nella zona bassa dell'IDE, significa che qualcosa non e' andata bene per cui bisogna interpretare le segnalazioni e provvedere alle necessarie azioni correttive.

Qualche volta, soprattutto al primo utilizzo della scheda, viene proposta una segnalazione di errore poiche' la porta usb alla quale e' collegato Arduino non e' stata correttamente indirizzata dal pc. Per rimediare e' normalmente sufficiente sconnettere e riconnettere il cavo USB e poi rilanciare la compilazione.

Poiche' la lettura di un manuale di programmazione e' assolutamente poco appassionate, e' opportuno, per i neofiti, limitare a questo capitolo la lettura di questo manuale (in pratica limitarsi ad installare ed utilizzare l'IDE) e quindi passare alla sperimentazione pratica delle potenzialita' di Arduino, eseguendo i primi esercizi di questa raccolta copiando ed incollando, nella finestra dell'IDE, i programmi proposti in ogni esercizio.

Dopo aver sperimentato (e magari interpretato) i primi due o tre esercizi e' opportuno leggere i prossimi quattro capitoli e limitarsi ad accedere alle sezioni successive solo per approfondire la conoscenza delle istruzioni e delle funzioni via via utilizzate.

Dopo aver sperimentato la prima decina di esercizi ed averne ovviamente compreso ogni riga di codifica, non sara' probabilmente piu' necessario ricorrere a questo tomo che, per quanto sintetico e scritto in forma piana, e' di rara aridita'.

Che cosa e’, come si progetta e come si scrive un programma

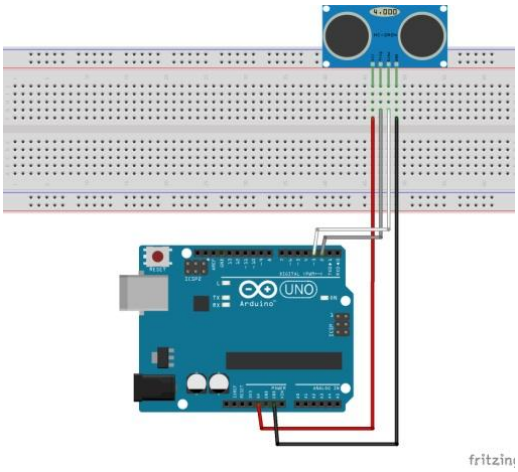
Un programma altro non e’ che un elenco di istruzioni elementari, che il computer esegue, uno dopo l’altra, in maniera acritica. Nello scrivere un programma si devono rispettare alcune regole ortografiche, grammaticali e sintattiche e si devono mettere in fila le istruzioni in maniera che il computer arrivi, passo dopo passo, esattamente al risultato voluto.

Sembra difficile, ma e’ in realta’ e’ tutto abbastanza facile. Basta pensare ad un programma come ad un romanzo composto da tre capitoli. Nel primo capitolo si descrivono i personaggi con le loro caratteristiche e magari i loro limiti, nel secondo capitolo si descrivono gli antefatti e nel terzo ed ultimo capitolo si descrivono i fatti, le azioni che coinvolgono i vari personaggi. Analogamente in un programma si possono identificare tre sezioni, la prima nella quale vengono definite le librerie, le costanti e le variabili, e cioe’ i personaggi e gli alleati (le librerie) che poi prenderanno parte al gioco, nella seconda parte si prepara l’ambiente in cui si svolgera’ l’azione e quindi si definiscono e si inizializzano i canali di comunicazione e si specializzano le funzioni fornite dalle librerie mentre nel terzo si descrivono le azioni che i vari personaggi (le costanti) aiutati dalle librerie e dalle routine (pezzi di programma che svolgono una funzione e che per comodita’ sono stati spostati in una zona diversa), devono eseguire per arrivare al risultato desiderato.

Appare quindi chiaro che prima di scrivere un programma e’ indispensabile non solo avere un’idea chiara di quale sia l’obiettivo, ma anche di quale sia il percorso per raggiungerlo. Si deve quindi “progettare” un programma, magari dividendolo in parti come ad esempio:

- Acquisizione dati dai sensori: quali sensori analizzare, in quale sequenza e con quali modalita’
- Decisioni da prendere, sulla base dei dati provenienti dai sensori
- Attivazione o disattivazione degli attuatori: quali attuatori attivare/disattivare ed in quale sequenza

E’ anche utile disegnare preventivamente lo schema delle connessioni tra Arduino, i sensori e gli attuatori, definendo le porte utilizzate dai vari componenti e le relative modalita’ di utilizzo (input oppure output). Un ottimo programma gratuito per il disegno degli schemi e’ reperibile qui: http://fritzing.org/download/



A lato viene proposto lo schema (disegnato con fritzing) e nelle prossime righe e’ riportata la codifica di un programma utilizzabile per misurare una distanza tramite un generatore di ultrasuoni.

Nel programma di esempio sono stati utilizzati i colori per facilitare l’individuazione e la comprensione delle varie parti ed e’ stata commentata ogni riga, in modo da rendere chiaro significato e scopo di ogni istruzione.

Coloro che non conoscono la programmazione possono limitarsi ad osservare le varie parti, leggere i commenti e magari tentare di decifrare qualche istruzione.
Informazioni di dettaglio sulla struttura di un programma, sul significato e sul funzionamento di ogni singola istruzione saranno comunque fornite piu’ avanti.

Legenda dei colori utilizzati nel programma di esempio:

Grigio: **Note e commenti**, che non influenzano il funzionamento del programma ma che lo rendono piu’ facilmente comprensibile e gestibile

Viola: **Variabili**, e cioe' zone di lavoro in cui vengono memorizzati i dati utilizzati dal programma. Ogni variabile e' caratterizzata da un nome e da un codice che ne definisce la tipologia.

Rosso: **Routine**, chiamate anche "metodi", ovvero istruzioni che non fanno parte del programma principale e che vengono eseguite solo nel momento in cui sono richiamate ("lanciate") da un'altra routine o dal programma principale. L'utilizzo delle routine, consente di isolare la gestione di singole funzioni e rende il programma principale piu' facilmente interpretabile

Verde: **Istruzioni iniziali** o di ***setup***, eseguite solo una volta, all'avvio del programma

Blu: **Parte principale** (o, meglio, ***loop***) del programma, eseguita e ripetuta in continuazione, fino a quando non viene tolta l'alimentazione alla scheda o viene premuto il pulsante reset

```
/*---------------------------------------------- programma di esempio ----------------------------------------------

Questo programma verifica tramite il modulo HC-SR04 la distanza di un ostacolo. Il modulo, lancia
un fascio di ultrasuoni e si mette in attesa di ricevere un segnale di ritorno.   Il tempo intercorso tra
il momento del lancio ed il momento del ritorno determina la distanza dell'ostacolo che, convertita
in centimetri, viene poi evidenziata sul monitor seriale (residente su PC). Lo schema vede il
"trigger" del modulo collegato alla porta 2 di Arduino, l'"echo" alla porta 3, il negativo alla terra ed il
positivo all'alimentazione da 5 volt. */

float cm = 0;                  // variabile di tipo float e chiamata "cm", in cui viene memorizzata la
                               //  distanza dall'ostacolo, in centimetri. Con questa idichiarazione viene
                               // definita  la variabile e le viene inizialmente assegnato il valore zero

void misurazionedistanza (void)     //********** inizio della routine "misurazionedistanza" *******
{                              // graffa di apertura, che delimita l'inizio delle istruzioni della routine
  digitalWrite(2, LOW);        //disattiva, sulla porta 2, il lancio del fascio di ultrasuoni (qualora fosse
//                               attivo)
  delayMicroseconds(2);        // attende 2 microsecondi (per lasciar spegnere eventuali echi presenti
//                               nell'ambiente
  digitalWrite(2, HIGH);       // attiva, sulla porta 2,  il lancio del fascio di ultrasuoni
  delayMicroseconds(10);       // attende 10 microsecondi (tempo richiesto dal modulo HC-SR04 per
                                 ricevere ed elaborare il segnale (l'eco) di ritorno)
  digitalWrite(2, LOW);        // disattiva, sulla porta 2, il lancio del fascio di ultrasuoni
  cm = pulseIn(3, HIGH) / 58.0; // rileva, sulla porta 3, il segnale di ritorno, lo converte in centimetri
//                               e lo memorizza nella variabile "cm" (il divisore 58.0 e' una costante
//                               trovata in rete e sperimentalmente verificata)
}                              // graffa di chiusura delle istruzioni della routine

void comunicazionedistanza (void)  // ****** inizio della routine "comunicazionedistanza" *******
{                              // graffa di apertura, che delimita l'inizio delle istruzioni della routine
  Serial.print(cm);            // visualizza sul monitor seriale il valore contenuto nella variabile cm
  Serial.println(" cm");       // visualizza la dicitura " cm" e posiziona il cursore sulla riga successiva
}                              // graffa di chiusura delle istruzioni della routine

void setup()                   //  fase di setup, e cioe' istruzioni eseguite solo a inizio programma
{                              // graffa di apertura, che delimita l'inizio delle istruzioni di setup
  Serial.begin(9600);          // inizializza il monitor seriale
  pinMode(2, OUTPUT);          // definisce la porta digitale 2 (collegata al trigger) come porta di output
  pinMode(3, INPUT);           // definisce la porta digitale 3 (collegata all'echo) come porta di input
}                              // graffa di chiusura delle istruzioni di setup

void loop()                          // fase di loop e cioe' parte principale del programma, ripetuta in
```

```
//                                  continuazione fino a quando non viene tolta l'alimentazione o
//                                  premuto il tasto reset
{                                  // graffa di apertura, che delimita l'inizio delle istruzioni di loop
  misurazionedistanza ();          // lancia la routine di misurazione della distanza
  comunicazionedistanza ();        // lancia la routine di comunicazione della distanza
  delay (2000);                    // attende due secondi (2000 millisecondi) prima di ripetere il ciclo
}                                  // graffa di chiusura delle istruzioni di loop
```

La semplice osservazione del programma consente alcune considerazioni:

Innanzi tutto la sequenza delle varie parti (le note iniziali, le variabili, le routine, il setup ed il loop), la cui collocazione all'interno del programma deve essere rispettata non solo per ottenere un corretto funzionamento, ma anche per facilitare la comprensione e la manutenzione del programma stesso.

E poi le note, apparentemente esagerate, diffuse e qualche volta ridondanti. In questo caso le note sono state volutamente ampliate nel tentativo di rendere piu' facilmente leggibile il programma ma anche nei normali programmi e' opportuno specificare il significato di ogni singolo passaggio. Le note sembrano un esercizio inutile, ma un programma, soprattutto se complesso, e' soggetto a modifiche e le note sono indispensabili per non fare danni in fase di modifica.

L'ultima considerazione riguarda le parentesi graffe. Sono essenziali e segnano l'inizio e la fine del setup, del loop, delle routine e delle parti di codice subordinate ad una condizione. Non solo sono indispensabili, ma contribuiscono in maniera determinate a rendere un programma piu' facilmente comprensibile.

In ogni caso, al di la dell'interpretazione di ogni singola istruzione che con le cognizioni attuali puo' risultare difficile, e' possibile, semplicemente leggendo le note, i titoli delle varie parti ed osservando le parentesi graffe, rendersi conto di come il programma sia stato diviso in parti e di quali funzioni siano svolte da ogni parte.

Struttura di un programma

Un programma wiring e' normalmente composto da tre parti separate da due dichiarazioni (*void setup()* e *void loop()*). La struttura standard di un programma e' questa:

```
/* prima parte, per la dichiarazione delle librerie, delle variabili, delle costanti e per la codifica delle
routine e cioe' parti di codice che vengono eseguite solo quando sono richiamate da una specifica
istruzione*/
/* -----( note esplicative sulle finalita' e sul funzionamento del programma ) -----*/
/*-----( dichiarazione delle eventuali librerie utilizzate dal programma )-----*/
/*-----( dichiarazione degli oggetti utilizzati, ad esempio display a cristalli liquidi o  servomotori )---*/
/*-----( dichiarazione delle costanti )-----*/
/*-----( dichiarazione delle variabili )-----*/
/*-----( codifica di eventuali routine )-----*/

void setup()   /* seconda parte o parte di setup, eseguita solo all'avvio del programma*/
{  /* ----( inizio del setup )------*/
   /*  ----( dichiarazione delle porte di input e di output )----*/
   /*  ----( eventuali istruzioni da eseguire all'avvio del programma )----*/
}/*   ----( fine del setup )---*/

void loop()   /*terza parte o parte di loop, parte principale del programma, che viene eseguita e
ripetuta fino al termine dell'alimentazione o fino alla pressione del pulsante reset*/
{ /* ----( inizio del loop )----*/
   /* ----( istruzioni di programma )-----*/
} / *----( fine del loop )----*/
```

Alcuni programmatori inseriscono le routine anche al termine della sezione di loop o tra la sezione di setup e quella di loop. Questa pratica e' consentita ma e' biasimevole, poiche' tende a rendere piu' difficile e dispersiva l'interpretazione del programma.

Ortografia, grammatica e sintassi

Il linguaggio di programmazione e' dotato di regole che devono essere scrupolosamente seguite:

- Ogni istruzione termina con un ";"

- Le parentesi tonde e quadre delimitano gli operatori di un'istruzione mentre le parentesi graffe delimitano una serie di istruzioni riferibili ad una condizione, a una routine o a una parte di programma. Se da un'istruzione dipende l'esecuzione di altre istruzioni, le istruzioni "subordinate" sono di norma racchiuse tra parentesi graffe;

- Ad ogni parentesi aperta deve corrispondere una parentesi chiusa. L'assenza di una parentesi di chiusura o di apertura qualche volta impedisce la compilazione (e quindi l'esecuzione) del programma ed in ogni caso ne rende imprevedibile i funzionamento;

- La combinazione di caratteri /* indica l'inizio di una zona di note, che puo' estendersi su piu' righe e che deve necessariamente essere chiusa dalla combinazione */

- La combinazione // indica l'inizio di una zona di note che si protrae fino alla fine della riga;

- Le indentazioni, non obbligatorie, sono comunque utili per rendere piu' facilmente comprensibile un programma. Nella sezione "strumenti" dell'IDE esiste la funzione "formattazione automatica", utile appunto per ottenere l'indentazione automatica del codice;

- Le variabili e le costanti devono essere dichiarate prima (in termini di posizione fisica nel programma) del loro utilizzo. Per questo motivo e' buona norma concentrare la loro definizione in testa al programma, prima del setup e delle eventuali routine.

Le parole chiave

Il compilatore riconosce alcune parole chiave alle quali assegna particolari significati:

- *HIGH* e *LOW* sono parole chiave utilizzate nella gestione delle porte di Arduino e nella gestione “digitale” di una varibile. *LOW* implica che alla variabile (o alla porta) sia associato il valore 0 mentre *HIGH* implica che sia associato il valore 1.

- *INPUT* e *OUTPUT* sono parole chiave utilizzate per definire se una specifica porta deve essere considerata di entrata (una porta alla quale e’ collegato un sensore) o di uscita (una porta alla quale e’ collegato un attuatore).

- *true* e *false* sono parole chiave utilizzate per verificare l’esito di una condizione.

Le istruzioni

Nella scrittura di un programma sono normalmente utilizzate:

- **variabili** e cioe' zone di memoria in cui sono depositati i dati
- **istruzioni** e cioe' i comandi, a loro volta classificabili in
    - istruzioni di **struttura e controllo**
    - operatori **matematici, booleani e computazionali**
- **funzioni,** a loro volta classificabili in:
    - funzioni di **input/output**
    - funzioni di **comunicazione**
    - funzioni di **tempo**
    - funzioni **matematiche**
    - funzioni di **generazione di numeri a caso**

Le variabili

Sono aree di memoria ove vengono memorizzati i dati necessari al funzionamento del programma.

Il programmatore assegna ad ogni variabile un nome ed un codice, che ne definisce la tipologia.

Come gia' detto, in un programma le variabili devono essere definite **prima** delle istruzioni che le utilizzano ed e' pertanto opportuno definirle tutte nella zona iniziale, prima del ***void setup ()*** e prima delle routine.

Nota *(un po' ostica, da rileggere magari quando si conosce un po' meglio il linguaggio di programmazione): le variabili definite all'interno di una sezione (all'interno del setup, oppure all'interno del ciclo di loop oppure ancora all'interno di un ciclo di for) mantengono il loro significato solo all'interno di detti cicli. A di fuori di questi cicli possono essere nuovamente definite ed assumere nuovi e differenti significati.*

Questa particolarita' e' utile quando si usa "assemblare" un programma copiando parti di codice e routine estratte da altri programmi. In questo caso ogni routine e' corredata dalle variabili da lei utilizzate e puo' quindi essere facilmente inserita in un qualunque programma.

Non e' pero' particolarmente utile se si scrive un programma partendo dall'inizio e, soprattutto nei programmi complessi, e' foriera di errori e confusione.

In linea di massima e come e' stato detto, e' opportuno dichiarare le variabili sempre all'inizio del programma ed utilizzarle ovunque, ma solo per lo scopo per le quali sono state dichiarate.

Se si dichiara che una variabile e', ad esempio utilizzata come indice di scorrimento di una tabella, e' opportuno, ai fini della comprensibilita' del programma, utilizzarla esclusivamente in quella veste e non, ad esempio, come deposito temporaneo di informazioni in routine che nulla hanno a che vedere con la suddetta tabella.

Le piu' comuni tipologie di variabili sono:

byte occupa un byte di memoria e puo' contenere un numero tra 0 e 255 non segnato.

int usa 2 byte di memoria e puo' contenere un numero compreso tra -32768 e 32767.

unsigned int Ha la stessa funzione di int, ma puo' contenere solo numeri positivi tra 0 e 65535.

long usa 4 byte e puo' contenere un numero compreso tra -2.147.483.648 a 2.147.483.647.

unsigned long Versione senza segno di long e puo' contenere un numero che va da 0 a 4.294.967.295.

float Può memorizzare numeri con la virgola. Utilizza 4 bytes e puo' contenere numeri compresi tra 3.4028235E+38 e -3.4028235E+38.

char occupa un byte di memoria. Se la si usa come un numero puo' contenere un valore che va da -128 a +127 se invece la si usa come testo puo' contenere un qualunque carattere ASCII. Se si aggiungono due parentesi quadre ***char[]*** diventa una variabile di tipo ***string*** nella quale e' possibile memorizzare un testo. Utilizza un byte per ogni carattere di testo più un carattere NULL che indica la fine del testo.

Esempio:

```
char saluto[] = "ciao"; // la variabile di tipo string denominata "saluto" contiene la parola "ciao"
                        // ed occupa 4 caratteri di testo + il carattere NULL e quindi 5 caratteri
```

Nota: *nell'esempio e' stato utilizzato il segno "=" per assegnare alla variabile saluto il valore "ciao". Il segno "=", se non e' accompagnato da un altro operatore matematico o condizionale, viene sempre interpretato come un'istruzione di assegnazione e quindi, nel nostro caso, assegna alla variabile saluto il valore "ciao". Qualora lo si voglia utilizzare in una condizione (per verificare, ad esempio: se a e' uguale a 10) il segno "=" deve essere*

raddoppiato e quindi deve essere utilizzata la combinazione "==" e quindi: if (a == 10)

All'interno delle parentesi quadre di una variabile di tipo *char* puo' essere inserito un numero per definire in maniera preventiva la lunghezza della variabile. Questa opzione risulta utile quando si vuole definire una variabile inizialmente vuota.

Esempio:

```
char area [10];      // variabile di tipo string denominata "area". E' vuota, ma rende disponibile al
                     // programma una zona nella quale potra' memorizzare parole
                     //  lunghe fino ad un massimo di 10 caratteri
```

array Non si tratta di un tipologia di variabile, ma del nome utilizzato per indicare una tabella, e cioe' un elenco di variabili accessibili tramite un indice.

Un *array* e' una variabile di tipo *int* o *char* seguita da una parentesi quadra aperta, un valore numerico (il numero di elementi) ed una parentesi quadra chiusa. Si utilizza la parola chiave *int* quando si vuole definire una tabella che conterra' dei numeri interi di valore compreso tra -32768 e 32767, mentre si utilizza la parola chiave *char* se si vuole definire una tabella che conterra' dei caratteri.

E' anche possibile definire preventivamente i valori di ogni elemento della tabella, facendo seguire alle parole chiave *int* o *char* le parentesi quadre con il numero di elementi, il segno di uguale ed i valori, separati da una virgola e racchiusi tra parentesi graffe.

Se, ad esempio vogliamo memorizzare quattro valori possiamo creare una array del tipo:

```
int tab1[4]={10,25,50,100};      // l'array (la tabella) denominato "tab1" e' lungo 5 byte (4 numeri
                                 // + il null finale) e contiene valori accessibili tramite un indice.
                                 // in questo esempio l'indice puo' assumere un valore che va
                                 // da 0 a 3; Con zero si accede al primo valore (10) mentre con 3
                                 // si accede al quarto valore (100) .
```

Per accedere ai valori presenti in tabella e' sufficiente utilizzare il nome della variabile associato all'indice. Esempio:

```
int val;          // definisce una variabile denominata val che conterra' un numero intero
val = tab1 [2];   // inserisce in val il valore contenuto nel terzo elemento della tabella del
                  // precedente esempio, e quindi 50
                  // (ricordarsi che con l'indice pari a zero si accede al primo elemento, con l'indice
                  // pari ad 1 si accede al secondo elemento e con l'indice pari a 2 si accede al
                  //  terzo  elemento….)
```

#define nome valore non e' la definizione di una variabile, ma solo la definizione di un *valore,* che sara' poi utilizzato dal compilatore in sostituzione della parola *nome*

Esempio:

```
#define pinled 5  //  in fase di compilazione il compilatore inserira' il valore 5 ad ogni occorrenza
//                   della parola pinled
```

Le istruzioni di struttura

Sono due istruzioni o meglio due dichiarazioni, che delimitano le parti che compongono un programma.

setup () che associata alla definizione *void* indica l'inizio della zona di inizializzazione del programma

loop () che associata alla definizione *void* indica l'inizio della zona dedicata al corpo del programma

Tra le istruzioni di struttura dovrebbe anche essere annoverata la definizione ***void*** che, pur essendo classificata come una tipologia di dati, contrassegna di fatto l'inizio di una routine.

In quest'ultimo caso la sua struttura e':

void nome_della_routine ***(void)***

{….. istruzioni della routine ….. ; */* la routine viene eseguita solo nel momento in cui il percorso seguito dal programma incontra l'istruzione di lancio della routine (e cioe' nome_della_routine ();). A questo punto il programma esegue le istruzioni della routine e, al termine, ritorna al normale iter eseguendo le istruzioni immediatamente successive all'istruzione di lancio della routine */*
}

Per chiamare (o meglio lanciare) una routine e' sufficiente scriverne il nome seguito da una parentesi aperta ed una chiusa. Esempio:

nome_della_routine ***();***

Nota: al momento del lancio, all'interno delle parentesi puo' essere inserito il nome di una variabile che la routine utilizza o modifica. Quest'opzione non sembra particolarmente utile in un programma che non nasce dall'assemblaggio di routine e funzioni estratte da altri programmi e pertanto viene di fatto ignorata in queste note.

Le istruzioni di controllo

Sono istruzioni che, al verificarsi di una condizione, lanciano l'esecuzione di specifiche parti di programma.

if.. else…

Permette di prendere delle decisioni. L'istruzione ***if*** deve essere seguita da una condizione racchiusa parentesi e da una o piu' istruzioni racchiuse tra parentesi graffe. Se la condizione è vera verranno eseguite le istruzioni contenute tra parentesi graffe mentre se la condizione e' falsa verranno eseguite le istruzioni (sempre tra parentesi graffe) contenute immediatamente dopo la parola chiave ***else***. In entrambi i casi verranno poi eseguite le istruzioni successive alla fine delle istruzioni dipendenti dalla ***else***. E' possibile usare l'istruzione ***if*** anche senza la parola chiave ***else***. In quest'ultimo caso se la condizione e' vera verranno eseguite le istruzioni racchiuse tra le parentesi graffe che seguono la ***if***, se invece e' falsa e si passera' direttamente alle istruzioni successive alla parentesi graffa di chiusura.

Struttura:

```
If (..condizione..)
  { ..codice da eseguire solo se la condizione e' vera..; }
else
  { ..codice da eseguire solo se la condizione e' falsa..; }
…. codice che verra' eseguito in ogni caso….
```

Esempio:

```
if (val==1)                    // se la variabile "val" contiene il valore "1"
{
   digitalWrite(3, HIGH);      // pone in stato "HIGH" (e cioe' attiva) la componente
                               // di OUTPUT che in fase di inizializzazione del programma e' stata
 }                             // associata alla porta 3 (ad esempio un led)

else
{
  digitalWrite (3, LOW);       // se invece "val" contiene un valore diverso da "1" pone in stato di
                               //  "LOW" (disattiva) la componente di OUTPUT associata alla porta 3
}
```

Nota: Come gia' precedentemente accennato, la combinazione di segni *"=="* e' necessaria per differenziare una condizione da un'assegnazione. Piu' in dettaglio, l'espressione *val = 1* viene interpretata dal compilatore come un assegnazione e quindi "inserisci 1 nella variabile val" mentre l'espressione *val == 1*, preceduta da un'istruzione di condizione (come ad esempio una *if*) viene interpretata come una condizione e quindi "se val e' uguale ad 1"

L'istruzione ***if*** puo' essere utilizzata anche senza le parentesi graffe; in questo caso, se la condizione e' vera, viene eseguita l'istruzione immediatamente successiva alla condizione mentre se e' falsa viene saltata la prima l'istruzione successiva ed eseguita la seconda successiva.

for…

l'istruzione ***for*** ripete una serie di istruzioni fino a quando una condizione risulta vera.

Struttura:

for (**..inizializzazione di una variabile..**;** ..condizione..**;** ..modifica della variabile.. **)
{** .. codice da eseguire e ripetere fino a quando la condizione e' vera.. **}

Esempio:

```
for (i=0; i<10; i++)
  { Serial.print ("Ciao"); }
/*  i=0 e' una assegnazione: assegna il valore iniziale zero alla variabile "i".  "i" e' una variabile
intera che deve essere stata predefinita nella zona di dichiarazione delle variabili, in caso contrario
e' possibile definirla al momento e quindi scrivere int i = 0 al posto di i=0;

i<10 e' la condizione: se "i" contiene un valore minore di 10 vengono eseguite le istruzioni
racchiuse tra parentesi graffe

i++  e' l'istruzione di modifica della variabile "i".  "++" e' un operatore computazionale che viene
interpretato dal programma come "aumenta di 1 il contenuto della variabile"

Serial.print ("Ciao"); e' il codice da eseguire e ripetere fino a quando "i" e' minore di 10 */
```

Nota: L'istruzione ***for*** puo' essere utilizzata anche senza le parentesi graffe; in questo caso se la condizione e' vera viene eseguita l'istruzione immediatamente successiva alla parentesi di chiusura della condizione e poi si ritorna all'istruzione di gestione della variabile del ciclo (nel nostro esempio alla i++) e subito dopo di nuovo al test.

switch… case… break… default

E' un'istruzione composta da piu' istruzioni elementari. E' utilizzata per eseguire parti di programma a seconda del valore contenuto in una variabile il cui nome e' indicato, tra parentesi, subito dopo la parola *switch*.

Struttura:

switch (*..nome di una variabile..* ***)***
{ case *XX:* *…codice..* ***; break; case*** *YY:* ***..codice.. ; break; default:*** *..codice..* ***}***

Esempio:

```
switch(sensore)          // inizio dell'istruzione switch; le istruzioni che seguono (fino alla parentesi
                         // graffa chiusa) sono subordinate al valore contenuto nella variabile
                         // denominata "sensore"
{
  case 38: digitalwrite(12, HIGH);break; // nel caso in cui la variabile sensore contenga il
                         // valore 38, attiva il componente collegato alla porta 12
                         // (digitalwrite(12, HIGH);) e poi interrompe l'esecuzione dell'istruzione
                         // (break;) saltando alle istruzioni successive alla parentesi graffa di
                         // chiusura
  case 55: digitalwrite(13, HIGH);break; // nel caso in cui la variabile sensore contenga il
                         // valore 55, attiva il componente collegato alla porta 13
                         // (digitalwrite(13, HIGH);) e poi interrompe l'esecuzione (break;) saltando
                         // alle istruzioni successive alla parentesi graffa di chiusura
  default: digitalwrite(12, LOW); digitalwrite(13, LOW); // in ogni altro caso (e cioe' se la
                         // variabile sensore non contiene ne' 38 ne' 55) disattiva (e cioe' pone
                         // in stato LOW) i componenti collegati alle porte 12 e 13
}
```

while…

Esegue una serie di istruzioni racchiuse tra parentesi graffe e le ripete fino a quando una condizione è vera.

Struttura:

While (** ..condizione.. **)
{** .. codice da ripetere sino a quando la condizione risulta vera.. **}

Esempio:

```
while(sensore<500)      // se il valore contenuto nella variabile "sensore" e' minore di 500 il
                        // programma esegue le istruzioni contenute tra parentesi graffe, in caso
                        // contrario prosegue partendo dall'istruzione successiva alla parentesi
                        // graffa di chiusura
{
  digitalWrite(13, HIGH); //attiva il componente collegato alla porta 13 ed
  delay(100);             // attende 100 millisecondi
  digitalWrite(13, LOW);  // disattiva il componente collegato alla porta 13 ed
  delay(100);             // attende 100 millisecondi
  sensore=analogRead(1);  // inserisce nella variabile sensore il valore del componete collegato
                          // alla porta analogica 1  e ritorna all'istruzione while
}
```

La codifica in esempio puo' essere utilizzata per lanciare un allarme (facendo lampeggiare un led o emettendo un suono intermittente tramite un buzzer collegati alla porta 13) se il componente collegato alla porta analogica 1 (ad esempio un rilevatore di temperatura) restituisce un valore inferiore a 500. L'allarme cessa (ed il programma continua il suo percorso) solo se il sensore fornisce un valore pari o superiore a 500

do…while

E' un'istruzione identica alla istruzione *while*, solo che il codice viene eseguito anche prima che la condizione sia verificata. Si usa quando si vuole eseguire il codice almeno una volta prima che la condizione sia valutata.

Struttura:

do
{ *..codice da ripetere fino a quando la condizione di while risulta vera..* ***}***
while (*..condizione..* ***);***

Esempio:

```
do
 {
        digitalWrite(13,HIGH); delay(100);
        digitalWrite(13,LOW); delay (100);
        valoresensore=analogRead(1);
 }
while (valoresensore < 500);
```

Nell'esempio viene attivato e disattivato un dispositivo (ad esempio viene fatto lampeggiare un led o prodotto un suono intermittente) sino a quando il valore fornito da un sensore analogico e' inferiore a 500. La differenza, rispetto al precedente esempio dell'istruzione *while* e' che il lampeggio o il suono inizia prima della verifica del valore fornito dal sensore e quindi, nel nostro esempio, l'allarme lampeggia o suona almeno una volta, indipendentemente dal valore fornito dal sensore

Break

Questa istruzione consente di bloccare un ciclo di *for.., do..* oppure *while…* e continuare ad eseguire il codice fuori dal ciclo. Viene anche utilizzato nell'istruzione *switch* per interrompere l'analisi delle condizioni.

Continue

Usato in un ciclo *do.., for..* o *while…* consente di interrompere l'esecuzione del codice interno al ciclo e tornare alla verifica della condizione.

Esempio:

```
for(lum=0; lum<200; lum++)
{
        if((lum>120) && (lum<180)) continue;
        analogWrite(porta1, lum);
        delay(20);
}
```

Nell'esempio si suppone che alla porta di output denominata *porta1* sia collegato un led la cui luminosita' varia progressivamente da 0 a 200. La progressione pero' si interrompe quando il valore della luminosita' e' compreso tra 120 e 180 *(lum> 120* ***&&*** *lum< 180*).

&& e' la combinazione utilizzata per rappresentare l'operatore booleano "and"

Operatori matematici

Nella scrittura di un programma possono essere utilizzati i normali operatori matematici:

+ per le addizioni

- per le sottrazioni

/ per le divisioni

***** per le moltiplicazioni

= per l'assegnazione del risultato

C'è anche uno strano operatore, il "***%***"che anziche' calcolare una percentuale (come ci si potrebbe aspettare), restituisce il resto di una divisione.

Esempio:
x = 19%7 // x contiene il valore 5 e cioe' il **resto** di 19 diviso 7

Nota: se il risultato di una divisione viene assegnato ad una variabile di tipo float, il numero di decimali del risultato e' pari al numero di decimali del divisore.

Esempio:
float risultato; *// definisce una variabile di tipo float in cui sara' inserito il risultato*
// di una divisione
risultato = (11/3.00); *// per effetto del numero di decimali del divisore, il risultato della divisione*
// viene limitato a due decimali e quindi e' 3.67

Operatori di comparazione o di condizione

Sono operatori utilizzabili nelle condizioni.

== uguale a (il doppio segno di uguale e' indispensabile per differenziare una condizione da un'assegnazione)

> maggiore di

< minore di

!= diverso da

<= minore o uguale di

>= maggiore o uguale di

Operatori booleani

Sono operatori usati per combinare più condizioni, ad esempio se vogliamo verificare se il valore di un sensore è compreso tra 1 e 5 basta scrivere:

```
if(sensore>=1) && (sensore<=5)
   { . codice da eseguire se il valore di sensore e' maggiore o uguale a 1 e minore o uguale a 5 .. }
else
   { … codice da eseguire se il valore di sensore e' minore di 1 oppure maggiore di 5… }
```

Esistono tre tipi di operatori booleani:

&& corrisponde alla “and”,

|| corrisponde alla “or”,

! corrisponde al “not”.

Nota: gli operatori booleani e soprattutto la “or” ed il “not” combinati insieme, sono di difficile utilizzo. Se impropriamente utilizzati, possono portare a risultati imprevedibili. In linea di massima e' sempre opportuno evitare di utilizzare l'operatore “or” in combinazione con l'operatore “not” poiche' il risultato e' quasi sempre diverso da quello che un utilizzatore frettoloso o impreparato potrebbe aspettarsi.

Operatori computazionali

sono operatori utilizzati per semplificare alcune operazioni elementari, come ad esempio incrementare o decrementare una variabile.

++ somma 1 alla variabile che precede l'operatore
-- sottrae 1 alla variabile che precede l'operatore

Esempio:
val++ corrisponde all'operazione *val* = *val* + *1* (incrementa di 1 il valore di val)
val-- corrisponde all'operazione *val* = *val* – *1* (decrementa di 1 il valore di val)

Qualora si voglia utilizzare un valore diverso da 1 si dovranno utilizzare i seguenti operatori computazionali

+=, -=, *=, /=

Esempio: le seguenti espressioni sono equivalenti:
val=val+5; // aumenta di 5 il valore di val
val+=5; // aumenta di 5 il valore di val

Librerie e funzioni

le **funzioni** sono istruzioni o meglio macroistruzioni mediante le quali e' possibile interagire con i sensori e gli attuatori oppure eseguire alcune attivita' (tipicamente calcoli) che restituiscono un valore.

Oltre a quelle standard, previste dal manuale di programmazione di Arduino, esistono una miriade di altre funzioni gestite da **librerie**, spesso scritte da utenti, che possono essere utilizzate per svolgere particolari attivita', soprattutto in presenza di sensori o attuatori di utilizzo complesso (come ad esempio un motore passo passo o un display lcd).

Per utilizzare una libreria e le funzioni da essa gestite bisogna dichiararla (includerla) nella parte iniziale del programma (prima del setup e prima delle routine).

Per includere una libreria in un programma bisogna innanzi tutto inserirla (se non gia' presente) nell'archivio delle librerie di arduino e poi richiamarla nel programma attraverso la definizione

#include <nome_della_libreria.h>

da posizionare, come gia' detto, in testa al programma.

Per inserire una nuova libreria nell'archivio delle librerie bisogna andare nell'IDE, seguire il percorso *Sketch->importa libreria->add library*, selezionare la cartella od il file compresso (zippato) che contiene la libreria e quindi premere il pulsante "apri".

Una volta caricata una nuova libreria in archivio e' opportuno chiudere e riaprire l'IDE per essere certi che la libreria e le sue nuove funzioni siano "visibili" al programma.

Come gia' detto esistono molte librerie, specifiche per quasi ogni tipo di device (per ogni tipo di sensore o di attuatore). Dette librerie sono normalmente reperibili on line, e sono rintracciabili semplicemente ricercando informazioni (datasheet o anche esempi e note di utilizzo) sul device che si intende utilizzare.

Funzioni di INPUT e di OUTPUT

Il linguaggio di programmazione comprende le funzioni per la gestione delle porte di Arduino. Attraverso queste funzioni e' possibile indicare il senso di utilizzo di una porta (INPUT oppure OUTPUT), attivare o disattivare una porta digitale, rilevare il segnale fornito da una porta analogica o attivare una porta digitale in modalita' PWM, una modalita' che simula l'emissione di un segnale analogico.

pinMode(porta,utilizzo);

questa istruzione e' utilizzata per configurare una delle 14 porte digitali; nella variabile *porta* deve essere inserito il numero della porta digitale (da 0 a 13) che si intende definire mentre in *utilizzo* deve essere indicato il tipo di utilizzo (*INPUT* oppure *OUTPUT*).

Esempio:

```
pinMode(13,INPUT);      // configura la porta digitale 13 come una porta di INPUT
pinMode(12,OUTPUT);  // configura la porta digitale 12 come porta di OUTPUT
/*  Nota: Al posto del valore 12 oppure13 puo' ovviamente essere utilizzata una variabile
contenente il numero della porta */
```

digitalWrite(porta,valore);

attiva (*HIGH*) o disattiva (*LOW*) una porta digitale di OUTPUT. L'attivazione implica che la porta in oggetto venga alimentata con una tensione di 5 volt mentre la disattivazione implica che attraverso la porta non circoli alcuna tensione (in pratica attiva o disattiva l'attuatore collegato alla porta)

Esempio:

```
digitalWrite(7,HIGH);       // attiva (pone cioe' in stato "HIGH", accende) il componente collegato
                            // alla porta digitale 7.
digitalWrite(led, LOW);     // disattiva (pone cioe' in stato di "LOW", spegne) il componente
                            // collegato alla porta il cui numero e' memorizzato in una variabile
                            // denominata led
```

*variabile = **digitalRead(**numero della porta**);***

rileva lo stato di una porta digitale di INPUT. Inserisce il valore 1 (*HIGH*) in *variabile* se sulla porta rileva una tensione superiore a 3 volt, inserisce 0 (*LOW*) se rileva una tensione inferiore a 1,5 volt e lascia inalterato il valore di *variabile* se rileva una tensione compresa tra 1,5 e 3 volt.

Esempio:

```
int val = 0;                //definisce una variabile intera di nome val
…….
…….
val=digitalRead(7);         // rileva la tensione fornita dal sensore collegato alla porta 7. Se rileva una
                            // tensione maggiore di 3 volt inserisce 1 (HIGH) nella variabile val; se
                            // rileva una tensione inferiore a 1,5 inserisce 0 (LOW) mente se rileva una
                            // tensione compresa tra 1,5 e 3 volt lascia inalterato il valore di "val"
```

variabile = ***analogRead(***porta***);***

rileva la tensione presente su di una porta analogica e inserisce in *variabile* un numero, compreso tra 0 e 1023, proporzionale alla tensione rilevata. 0 corrisponde ad una tensione pari a 0 mentre 1023 corrisponde ad una tensione pari a 5 volt.

Esempio:

```
int val = 0;                //definisce una variabile intera di nome val
…….
…….
val=analogRead(0);      // rileva la tensione presente sulla porta analogica 0  ed inserisce
                        // nella variabile val  un valore compreso tra 0 e 1023, proporzionale
                        // alla tensione rilevata
```

analogWrite(*porta****,****valore****);***

Con questa istruzione e' possibile utilizzare una porta digitale in maniera PWM e cioe' come una porta analogica di OUTPUT. L'istruzione e' in grado di fornire in uscita sulla porta in questione una tensione (tra 0 e 5 volt) proporzionale al numero (compreso tra 0 e 255) inserito nella variabile *valore*.

Nota: su Arduino Uno le porte digitali utilizzabili come porte analogiche di OUTPUT sono **solo** le porte: 11, 10, 9, 6, 5, e 3.

Esempio di utilizzo delle istruzioni di input e di output (programma completo, per la gestione della della luminosita' di un led):

```
/* Materiale necessario: una resistenza variabile, un led ed una resistenza da 220 ohm. Schema:
collegare il positivo del led alla porta 9 ed il pin centrale di una resistenza variabile alla porta
analogica 3; collegare il negativo del led ad una resistenza da 220 ohm a sua volta collegata a
terra; collegare i pin estremi della resistenza variabile a terra e all'alimentazione da 5 volt.

Ruotando la manopola della resistenza variabile, variera' l'intensita' luminosa del led  /*

int ledPin      = 9;   // definisce una variabile denominata "ledPin",  il cui valore e' 9
int analogPin = 3;   // definisce una variabile denominata "analogPin",  il cui valore e' 3
int val         = 0;   // definisce una variabile denominata "val",  il cui valore iniziale e' 0

void setup()
{
pinMode(ledPin, OUTPUT);   // definisce la porta digitale 9 (il led) come porta di OUTPUT
}

void loop()
{
val = analogRead(analogPin);   // Arduino rileva la tensione fornita dalla resistenza variabile
                               // collegata alla porta analogica 3 (analogPin contiene il valore 3)
                               // ed inserisce in "val"  un valore proporzionale alla tensione
                               // rilevata e  compreso tra 0 e 1023
analogWrite(ledPin, val / 4);      // invia alla porta ledPin (la porta 9) una tensione che varia da 0
                               // a 5  volt in funzione del valore contenuto in val/4 (piu'
                              // esattamente una tensione pari a (5/255)*(val/4) volt;
}
```

Funzioni di comunicazione seriale

Arduino, attraverso le funzioni di tipo ***Serial,*** puo' comunicare con il monitor seriale (video e tastiera) presente nell'IDE (l'ambiente di sviluppo installato sul PC) o con altri device collegati tramite le porte seriali.

Esistono numerose funzioni seriali (l'elenco completo e' presente nella guida reperibile nel menu' aiuto dell'IDE) e tra queste le piu' diffuse sono:

Serial.begin(*velocita****)****;* *// istruzione da utilizzare in fase di setup, prepara Arduino a mandare*
// e a ricevere dati tramite la porta seriale. In "velocita" si inserisce
// normalmente il valore 9600 (9600 bit per secondo) ma e' possibile
// utilizzare anche altri valori, fino ad un massimo di 115200

Serial.print(*valore****);*** *// Invia al monitor seriale il contenuto della variabile valore e*
// mantiene il cursore del monitor sulla linea corrente, in attesa
// di un nuovo valore.

Serial.println(*valore****);*** *// invia al monitor seriale il contenuto della variabile valore seguito*
// dal segnale di ritorno a capo, in modo da esporre i successivi
// messaggi su di una nuova riga

val = ***Serial.available();*** *// inserisce in val (una variabile intera) la lunghezza della parola digitata*
// sulla tastiera collegata serialmente ad Arduino (normalmente la tastiera
// del monitor seriale e cioe' la tastiera del pc collegato con il cavo usb)

val = ***Serial.read();*** *// legge (inserisce in val, una variabile normalmente di tipo chr) cio' che*
// e' stato battuto sulla tastiera collegata serialmente ad arduino

Come abbiamo visto dalle istruzioni sopra riportate, la porta seriale opera sia in entrata che in uscita e quindi attraverso questa porta e' possibile non solo esportare dati e messaggi ma anche ricevere dati utilizzabili dal programma operante su Arduino.

Funzioni di tempo

Wiring include alcune funzioni di gestione del tempo. Le piu' interessanti sembrano essere

millis();

Fornisce i millisecondi trascorsi dall'inizio del programma.

Esempio:

```
tempo=millis();    // inserisce nella variabile tempo i millisecondi trascorsi dal momento
                   // dell'accensione della scheda
```

delay(pausa);

Mette in pausa il programma per un numero di millisecondi specificato in *pausa.*

Esempio:

```
delay(1000);     // interrompe per un secondo (1000 millisecondi) l'esecuzione del programma
```

Funzioni matematiche

il linguaggio include funzioni matematiche alcune delle quali sono qui' rappresentate. L'elenco completo delle istruzioni e delle funzioni e' reperibile nella sezione "aiuto" dell'ambiente di sviluppo.

*gamma = **min (**alfa**,**beta**);***

inserisce in *gamma* il valore minore tra i due valori contenuti nelle variabili *alfa* e *beta*

Esempio:
val=min(50,30); // in val sara' presente il valore 30

Analogo ragionamento vale per le seguenti funzioni

*val = **max(**x**,**y**);*** // inserisce in val il valore maggiore tra i valori contenuti in x e y.

*val = **abs(**x**);*** // inserisce in val il valore assoluto di x (toglie il segno a x)

*val = **constrain(**x**,**a**,**b**);*** // x e' un valore variabile; a e' il minimo valore accettabile e b e' il
// massimo valore accettabile.
// inserisce in val il valore x se x e' compreso tra a e b;
// inserisce in val il valore a se x è minore di a;
// inserisce in val il valore b se x è maggiore di b.

*val = **pow(**base**,**esponente**);*** // calcola una potenza e cioe' inserisce in val la base elevata
// all'esponente. Attenzione: val deve essere una variabile di tipo double

*val = **sqrt(**x**);*** // calcola la radice quadrata di x. Attenzione, val deve essere una
// variabile di tipo double.

val =map(x, daMinimo, daMassimo, aMinimo, aMassimo); // inserisce in val un valore
// compreso tra aMinimo e aMassimo, calcolato mantenendo la
// proporzione esistente tra il valore x e daMinimo e daMassimo.
// In pratica riparametra il valore x (compreso tra daMinimo e
// daMassimo) su di una scala diversa e compresa tra aMinimo
// ed aMassimo

Funzioni di generazione di numeri a caso (random)

Wiring e' dotato di un generatore di numeri a caso:

val = ***random*** *(max)* *// inserisce in val un numero a caso compreso tra 0 ed il*
// valore contenuto in max

Poiche' i numeri generati sono in realta' pseudo casuali (fanno cioe' parte di una enorme sequenza predefinita), per evitare di ripetere la medesima sequenza di numeri ad ogni avvio del programma e' opportuno inizializzare il generatore di numeri a caso utilizzando l'istruzione:

randomSeed(*seme***);** *// inizializza il generatore di numeri a caso*

Inserendo in *seme* un numero sempre diverso e quindi derivato, ad esempio, da una funzione di tempo applicata ad un'azione umana (come il tempo intercorso tra l'avvio del programma e la pressione di un pulsante) oppure il valore fornito da una porta analogica non utilizzata. In questo modo ad ogni avvio del sistema l'istruzione *random ()* restituira' numeri a caso in sequenze sempre diverse.

Conclusioni

Qui' si conclude questa breve panoramica sul linguaggio di programmazione di Arduino.

E' forse possibile trovare presso librerie specializzate (Feltrinelli o meglio Hoepli) alcuni manuali in italiano sull'argomento.

Poiche' il linguaggio utilizzato per la scrittura degli sketch si chiama **Wiring** la ricerca dovrebbe essere indirizzata verso un eventuale manuale di Wiring.

Wiring e' un C++ "contaminato" da JAVA per cui e' anche forse possibile ricorrere ad un manuale C++ e/o Java (ipotesi pero' sconsigliabile, viste le dimensioni di questi due tomi e visto che wiring e' solo un sottoinsieme di questi linguaggi).

Probabilmente il modo migliore per approfondire le caratteristiche di un'istruzione o di una funzione resta il semplice ricorso alla guida presente nell'IDE (in inglese, ma assolutamente ben fatta ed esaustiva) o una ricerca su internet.

Ultima, ma non ultima nota, occorre precisare che anche *Wiring* e' dotato di un suo ambiente di sviluppo (una sua IDE) che e' possibile trovare e scaricare qui:

http://wiring.org.co/download/index.html

L'IDE di wiring puo' essere utilizzata altrettanto bene quanto l'IDE di Arduino per scrivere, compilare e caricare i programmi.