ESEMPI DI PROGRAMMAZIONE SULL'IDE DI ARDUINO

Questo documento è un approfondimento delle due dispense su Arduino scaricabili ai seguenti link:

Arduino partendo da zero https://danielepostacchini.it/wp-content/uploads/2019/10/Arduino-da-zero-1.pdf

Arduino partendo da uno https://danielepostacchini.it/wp-content/uploads/2019/11/Arduino-partendo-da-uno.pdf

Chi non ha dimestichezza con la programmazione o con il linguaggio C, può trovare qualche difficoltà con l'approccio iniziale, per questo mi aiuterò con i diagrammi di flusso per descrivere i semplici algoritmi che realizzeremo.

INSTALLAZIONE ED OPERAZIONI INIZIALI

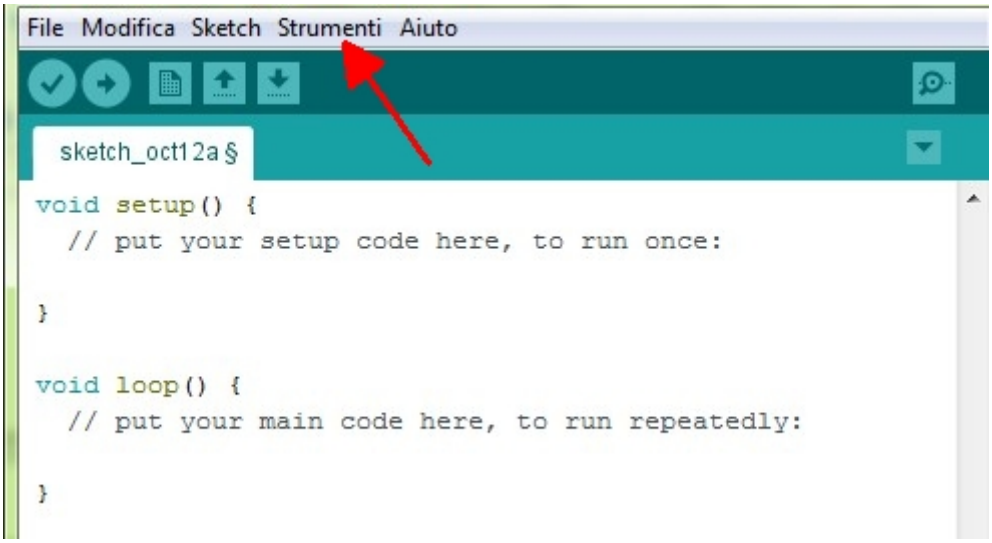
La I.D.E. di Arduino è la piattaforma software che ci consente di editare, compilare, programmare e "debuggare" (fare il debugging, cioè individuare e correggere gli errori) i programmi che vengono scritti per le schede appartenenti a questa famiglia.

L'acronimo I.D.E. sta per **I**ntegrated **D**evelopment **E**nvironment e cioè ambiente di sviluppo integrato.

Di fatto è un programma che si può scaricare ed installare dal seguente sito: https://www.arduino.cc/en/Main/Software

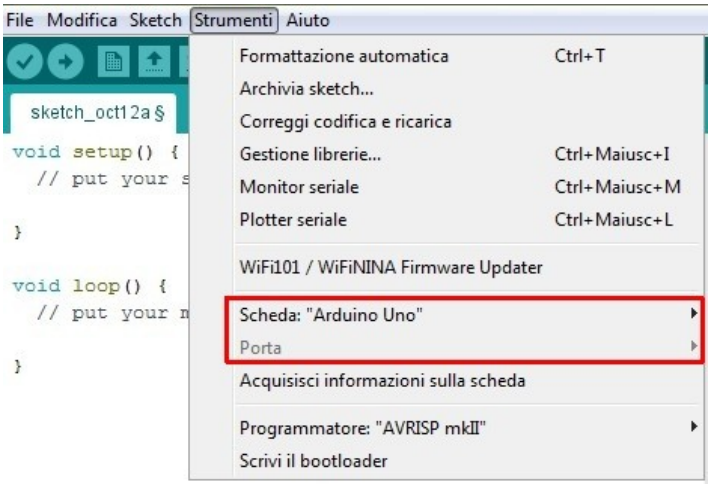
Dopo aver installato il programma, lo stesso si può avviare dall'elenco dei programmi, dove risulterà presente questa icona.

Una volta avviato il programma dovremo prima di tutto selezionare quale scheda stiamo utilizzando e la porta seriale (COM) dove la scheda è collegata. Di fatto noi collegheremo la scheda con un cavo USB, ma nella realà sulla scheda è presente un convertitore USB-RS232, quest'ultima sigla, RS232, indica uno standard seriale asincrono, presente nei vecchi PC non più disponibile però nei nuovi computer.
La scelta della scheda va fatta sulla voce "strumenti" presente nel menù in alto.

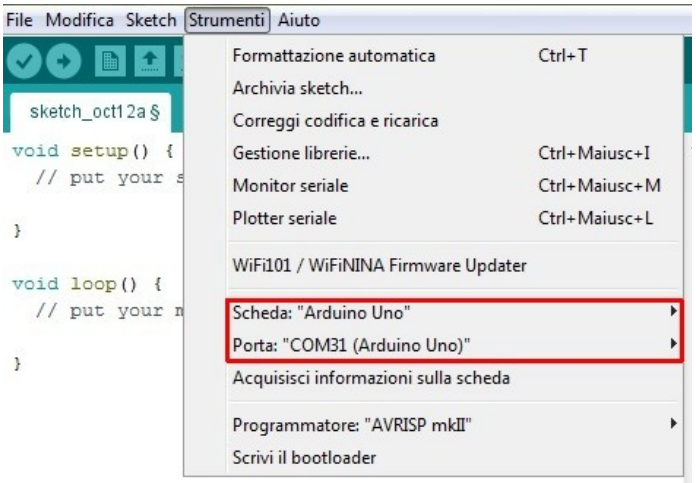


Se colleghiamo la scheda potremo impostare entrambe le cose, e cioè tipo di scheda e porta seriale.
Se invece la scheda non è collegata la porta seriale non verrà visualizzata.

Scheda non collegata, la porta non è visibile

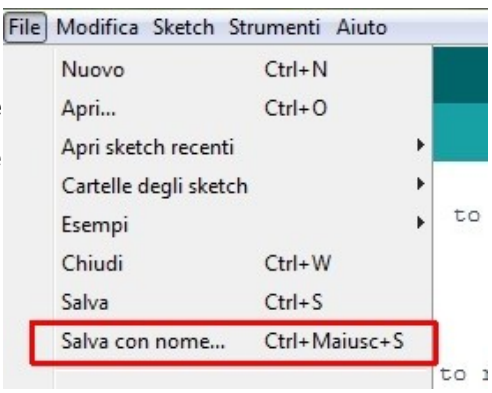


Scheda collegata si deve selezionare la porta dove c'è scritto Arduino Uno o il nome della scheda se differente



Questa operazione può essere fatta anche successivamente, ma sempre prima di caricare il programma, altrimenti verrà segnalato un errore di caricamento.

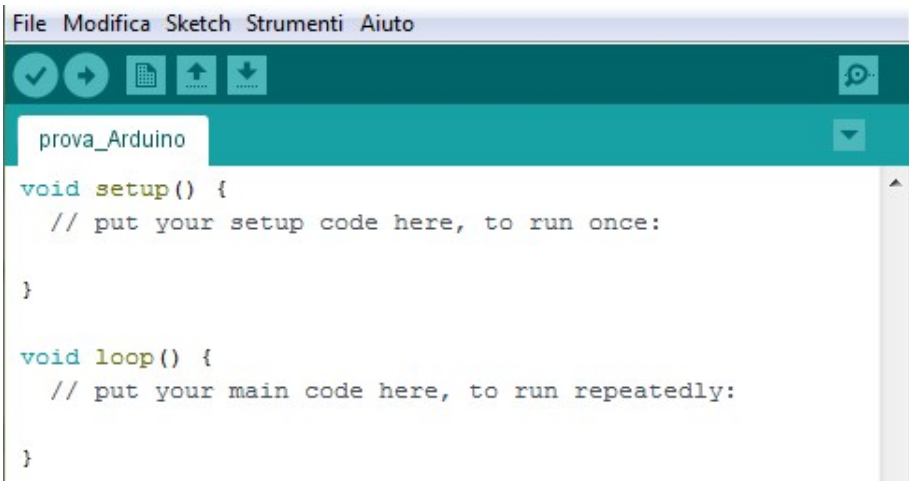
Un'altra cosa che conviene fare subito, è creare una cartella dove scrivere il programma, e prima di cominciare a scrivere il codice salvare il file dandogli un nome (salva con nome) nella cartella creata.



A questo punto siamo pronti a scrivere i nostri programmi.

PROGRAMMAZIONE

Entriamo ora nel vivo della programmazione. All'avvio o quando si fa **File – Nuovo**, per creare un nuovo programma, si apre questa schermata.



Per chi non conosce il linguaggio C, queste scritte potrebbero non avere senso, cerchiamo perciò di spiegare cosa rappresentano.

Questa è una parte di programma o meglio, è uno dei posti dove scrivere le righe del nostro programma.

```
void setup() {
  // put your setup code here, to run once:

}
```

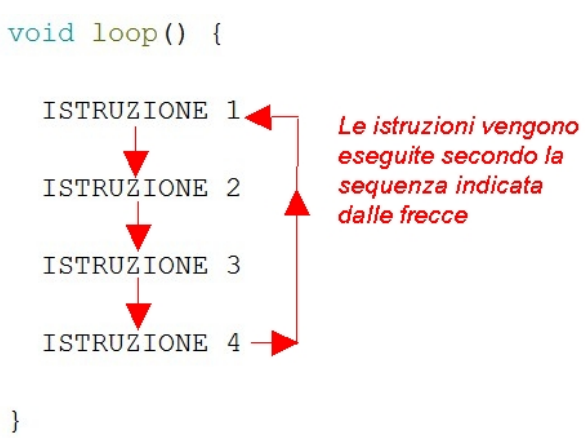
Le istruzioni che scriveremo, andranno scritte nello spazio compreso tra le due parentesi graffe, e come recita la scritta in inglese, tutto ciò che scriveremo in questo spazio, sarà eseguito solo una volta all'avvio.

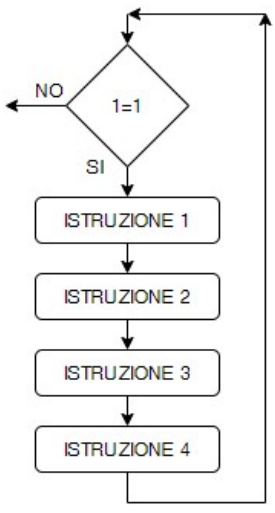
La prima cosa che il microcontrollore presente sulla scheda eseguirà, saranno le istruzioni presenti all'interno di questo spazio, dove ovviamente metteremo le impostazioni principali, e cioè quelle operazioni che vanno eseguite solo una volta.

L'altra parte fondamentale dove scrivere il programma, è il **loop.**

```
void loop() {
  // put your main code here, to run repeatedly:

}
```

Anche qui il meccanismo è lo stesso e cioè il programma va scritto tra le due parentesi graffe, preferibilmente un'istruzione per ogni riga, solo che in questo caso il programma sarà eseguito ciclicamente. Qui metteremo ovviamente il corpo principale del programma.



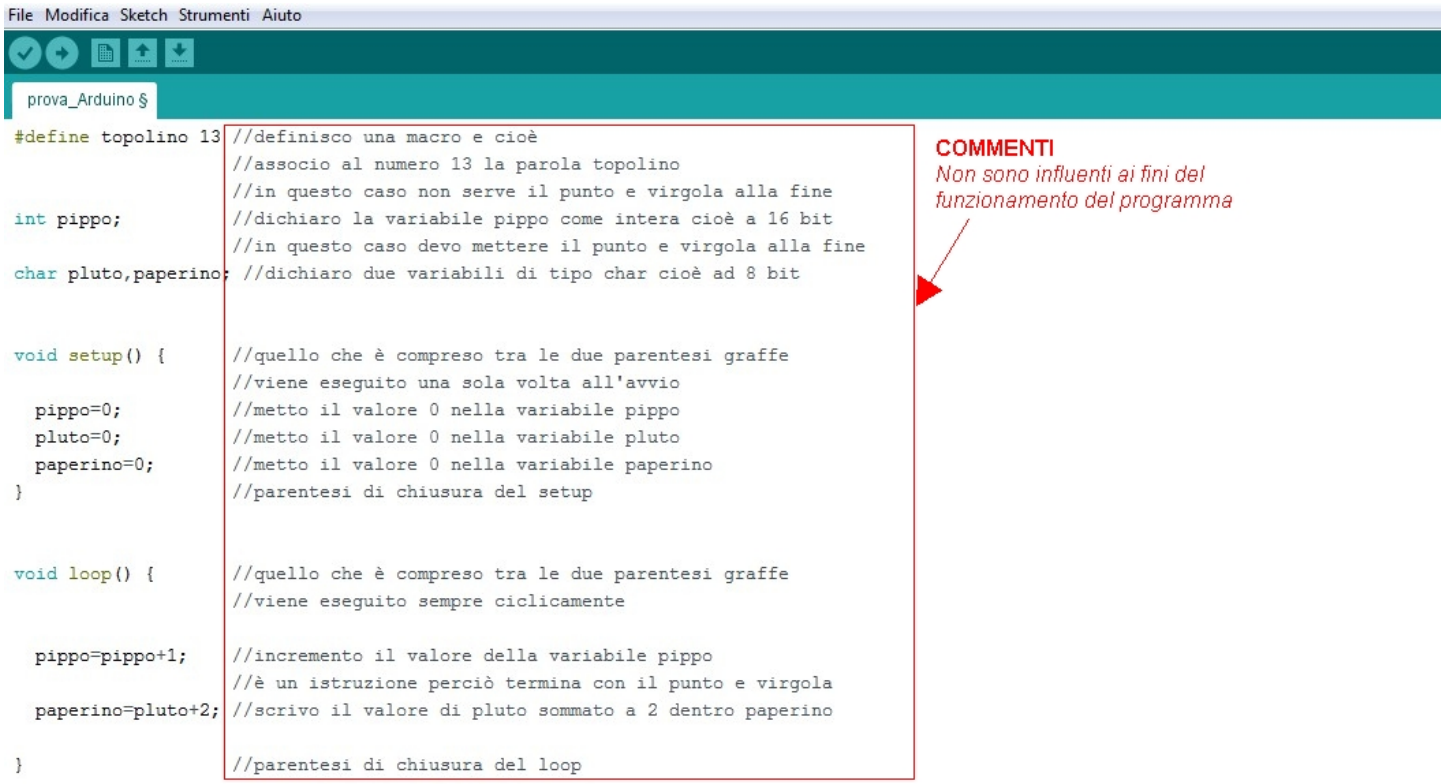


E' come se ci fosse un ciclo iterativo che dipende da una condizione sempre vera, come ad esempio 1=1.

Più avanti vedremo come strutturare altri cicli iterativi (come ad esempio ***do-while, while-do, for***) all'interno del **loop**. Limitiamoci per ora ad utilizzare l'unica ripetizione fornita dal ciclo loop.
Prima di scrivere un programma inoltre occorre conoscere delle regole fondamentali per il linguaggio C, che sono le seguenti:

- ogni istruzione deve terminare con il punto e virgola **;**
- i commenti vanno preceduti dal doppio slash **//**
- ogni parentesi graffa va successivamente sempre chiusa
- le variabili globali e cioè utilizzabili in ogni parte del programma, vanno dichiarate all'inizio prima del setup
- le costanti vanno anch'esse dichiarate all'inizio.

Un esempio di programma con dichiarazione di variabili e di una costante è il seguente:



Il programma ovviamente non ha alcuna funzione, è solo un esempio.
Possiamo vedere ad esempio inizialmente la definizione di una macro (#define) e la dichiarazione di 3 variabili di tipo diverso.
Questa prima parte non contiene istruzioni ma delle direttive per il compilatore. La dichiarazione di variabili o la definizione di macro NON SONO ISTRUZIONI che il microcontrollore eseguie, ma SONO DIRETTIVE per il compilatore che dovrà tradurre il programma in linguaggio macchina.
Le istruzioni sono invece quelle contenute in **setup** ed in **loop**. Nel linguaggio C è possibile inserire altre parti racchiuse tra parentesi graffe come setup e loop, chiamate **funzioni.**
L'ambiente di sviluppo di Arduino, ci mette a disposizione delle istruzioni, per gestire i piedini digitali presenti sulla scheda come ad esempio pinMode, e digitalWrite, o altre funzioni per gestire dei ritardi come ad esempio delay.

Vediamo due esempi che realizzano lo stesso programma.

I due programmi sono praticamente identici, solo che nel secondo caso, invece di indicare il numero del pin di Arduino, do un nome al pin **13** e cioè **LED**, mediante la direttiva **#define** all'inizio del programma, in pratica si crea una costante di valore 13.

File Modifica Sketch Strumenti Aiuto

Verifica

prova_Arduino §

```
void setup() {
  pinMode(13,OUTPUT);     //imposto il pin LED cioè 13 come uscita
}//chiusura del setup

void loop() {
 digitalWrite(13,HIGH);   //metto ad un livello alto il pin 13
 delay(500);               //attendo 500msec
 digitalWrite(13,LOW);    //metto ad un livello basso il pin 13
 delay(500);              //attendo 500msec
}//chiusura del loop
```

In ogni punto del programma se scriverò LED è come se scrivessi 13. La funzione svolta è la stessa nei due casi, e corrisponde al lampeggio di un LED collegato sul piedino 13 della scheda Arduino.

Nel setup, c'è solo un istruzione, che è la definizione del funzionamento del pin13, che potrebbe essere un ingresso o un'uscita, nel nostro caso dovendo accendere o spegnere un led, il pin 13 dovrà essere un'uscita. Questa scelta viene fatta con la funzione **pinMode.**

File Modifica Sketch Strumenti Aiuto

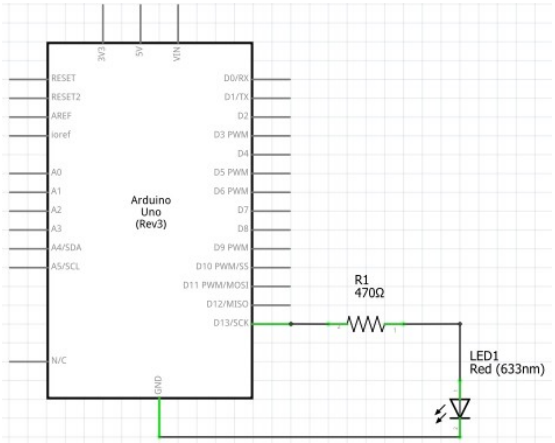
Verifica

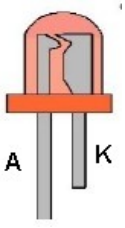
prova_Arduino §

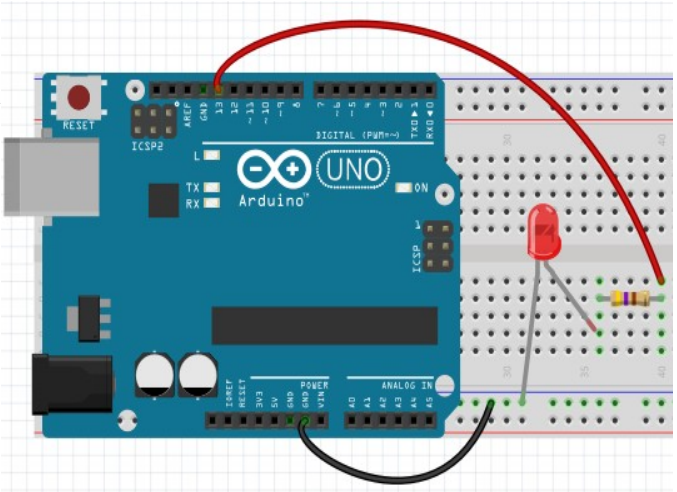
```
#define LED 13            //associo alla parola LED il numero 13
                          //se nel programma scriverò LED
                          //è come se scrivessi 13

void setup() {
  pinMode(LED,OUTPUT);    //imposto il pin LED cioè 13 come uscita
}//chiusura del setup

void loop() {
 digitalWrite(LED,HIGH);  //metto ad un livello alto il pin 13
 delay(500);              //attendo 500msec
 digitalWrite(LED,LOW);   //metto ad un livello basso il pin 13
 delay(500);             //attendo 500msec
}//chiusura del loop
```

Successivamente nel ciclo loop, utilizzeremo l'istruzione **digitalWrite** per mettere a livello alto (accendere) o a livello basso (spegnere) il pin 13 e conseguentemente il LED. Nel mezzo c'è un'istruzione **delay** che ferma il programma per il tempo indicato tra le parentesi espresso in milli secondi.

Lo schema di collegamento è il seguente, bisogna ricordars che il led ha un verso, l'anodo **(A)** dove entra la corrente ed il catodo **(K)** dove esce.





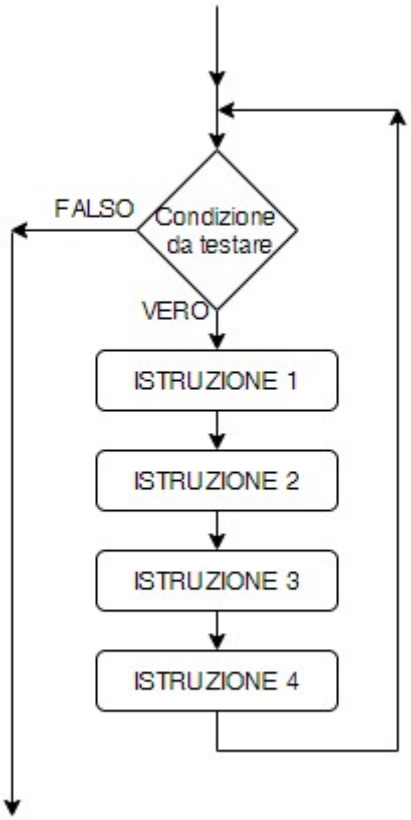


CICLI ITERATIVI

Nel programma di esempio precedente l'unico ciclo è il **loop**. Nel linguaggio C invece esistono 3 principli cicli iterativi; il ciclo **do-while**, il ciclo **while-do** ed il ciclo **for**.

Ciclo WHILE-DO

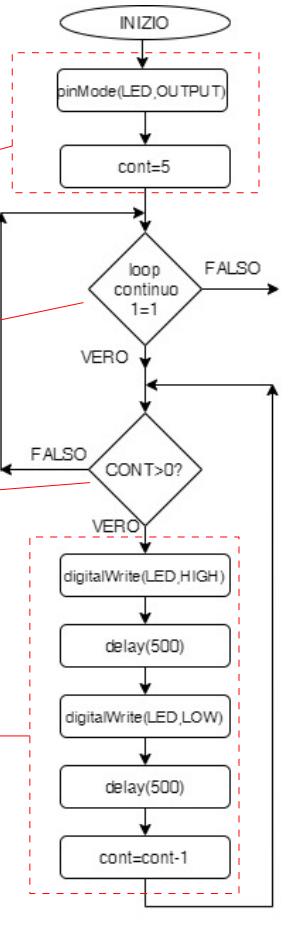
In questo caso vengono eseguite delle istruzioni all'interno di un ciclo, se una condizione è vera le istruzioni vengono eseguite, il controllo della condizione va fatto in testa, cioè all'inizio del ciclo.



Proviamo ad applicare questo ciclo nel programma del lampeggio del led visto precedentemente.
Scriviamo un programma per fare lampeggiare il led un numero di volte definito dal valore di una variabile.

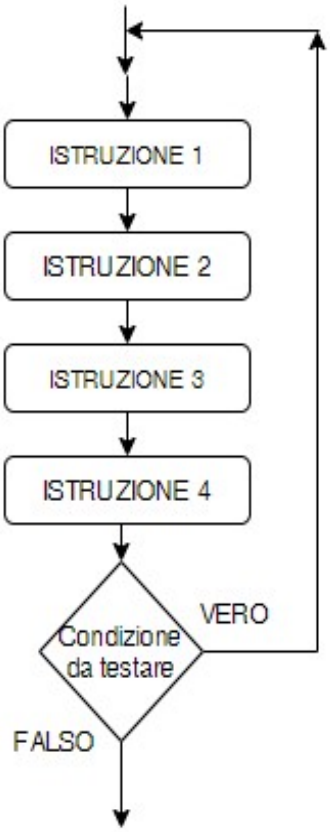
La variabile cont viene dichiarata all'inizio, e nel setup viene caricata con il valore 5.
Nel ciclo while-do, controlleremo il valore della variabile all'inizio, se è maggiore di zero, verrà eseguito un lampeggio, altrimenti non verrà eseguito nulla ed il flusso del programma si spostera direttamente alla fine del ciclo while.
In questo modo verranno eseguiti 5 lampeggi, oltre ai quali la variabile cont diverrà pari a 0, ed il flusso non entrerà più nel ciclo while.

```
#define LED 13

int cont;                  //dichiaro una variabile intera, 16 bit

//questa parte di programma viene eseguito all'avvio
void setup() {
  pinMode(LED,OUTPUT);     //imposto il pin LED cioè 13 come uscita
  cont=5;
}//chiusura del setup

//questa parte di programma viene eseguita ciclicamente
void loop() {
 while(cont>0){
    digitalWrite(LED,HIGH);  //metto ad un livello alto il pin 13
    delay(500);              //attendo 500msec
    digitalWrite(LED,LOW);   //metto ad un livello basso il pin 13
    delay(500);              //attendo 500msec
    cont=cont-1;             //decremento la variabile cont
 }//while
}//chiusura del loop
```



Ciclo DO-WHILE

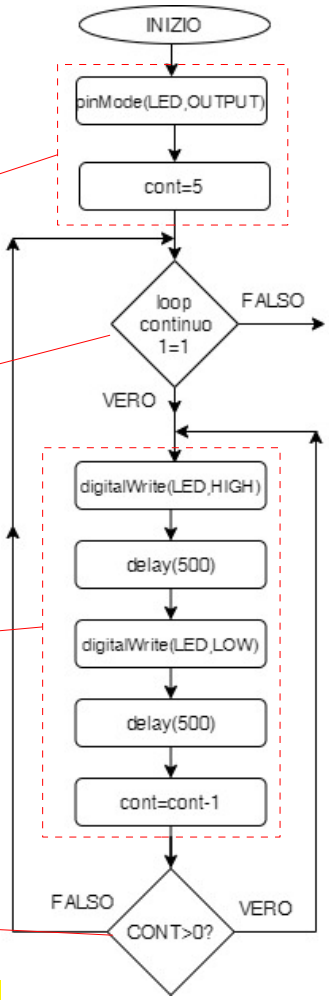
In questo caso il controllo della condizione avviene alla fine, per il resto il funzionamento è lo stesso, le istruzioni tra le parentesti graffe vengono eseguite ciclicamente se la condizione controllata alla fine è vera.



Se provassimo ad applicare lo stesso ragionamento con il lampeggio del LED, ci accorgeremmo che questo tipo di ciclo, non sarebbe adatto ad effettuare la funzione di prima.

Infatti essendo la condizione in coda, le istruzioni verrebbero eseguite comunque, pertanto il flusso del programma entrerebbe sempre nel ciclo do-while, anche quanto cont vale 0.
E successivamente decrementando cont dal valore 0, esso assumerebbe valor negativi, ma le istruzioni di lampeggio contenute nel ciclo, verrebero comunque eseguite.

```
#define LED 13

int cont;                  //dichiaro una variabile intera, 16 bit

//questa parte di programma viene eseguito all'avvio
void setup() {
  pinMode(LED,OUTPUT);     //imposto il pin LED cioè 13 come uscita
  cont=5;
}//chiusura del setup

//questa parte di programma viene eseguita ciclicamente
void loop() {
 do{
   digitalWrite(LED,HIGH); //metto ad un livello alto il pin 13
   delay(500);             //attendo 500msec
   digitalWrite(LED,LOW);  //metto ad un livello basso il pin 13
   delay(500);             //attendo 500msec
   cont=cont-1;            //decremento la variabile cont
 }while(cont>0);
}//chiusura del loop
```



Cosi come strutturato questo programma NON RISPETTEREBBE quanto richiesto ed ottenuto con il ciclo while-do.

Ciò significa che la scelta tra ciclo **while-do** o **do-while,** dipende fortemente dall'algoritmo che si vuole realizzare.

Ciclo FOR

Nel ciclo FOR invece, vengono eseguite le istruzioni inserite tra le parentesi graffe del ciclo, per un numero di volte definito dalla variabile **cont**.

```
for(cont=0;cont<val;cont++){
        ISTRUZIONE 1
        ISTRUZIONE 2
}
```

Quanto scritto sopra sta a significare che ISTRUZIONE 1 e ISTRUZIONE 2 vengono eseguite fino a quando cont<val, e cont viene incrementato (cont++) ad ogni ciclo.
Non occorre mettere l'incremento della variabile cont, in quanto è implicito nella sintassi del ciclo FOR.

Applicato al nostro programma, anche questo ciclo non sortirebbe l'effetto voluto, in quanto il flusso del programma entrerà sempre all'interno del ciclo FOR, eseguendo le istruzioni per il numero di volte richiesto.

```
#define LED 13

int cont;                  //dichiaro una variabile intera, 16 bit
int val;                   //dichiaro una variabile intera, 16 bit

//questa parte di programma viene eseguito all'avvio
void setup() {
  pinMode(LED,OUTPUT);     //imposto il pin LED cioè 13 come uscita
  val=5;
}//chiusura del setup

//questa parte di programma viene eseguita ciclicamente
void loop() {
  for(cont=0;cont<val;cont++) {
    digitalWrite(LED,HIGH);
    delay(500);
    digitalWrite(LED,LOW);
    delay(500);
  }//ciclo for
}//chiusura del loop
```

Pertanto anche il ciclo FOR strutturato in questo modo non potrebbe rispettare quanto richiesto, e cioè 5 lampeggi del LED.

Come si è visto perciò la scelta del tipo di ciclo iterativo, è fortemente dipendente dall'algoritmo da realizzare.

STRUTTURE DI SELEZIONE

Oltre ai cicl iterativi, per il controllo del flusso del programma, ci sono le strutture di selezione, come ad esempio la struttura **if….else** , oppure **switch….case**.

Con queste due strutture si può decidere in che direzione dovrà andare il flusso del programma, se fare determinate operazioni o altre. Il tutto avviene sempre dopo aver controllato il valore di una variabile.

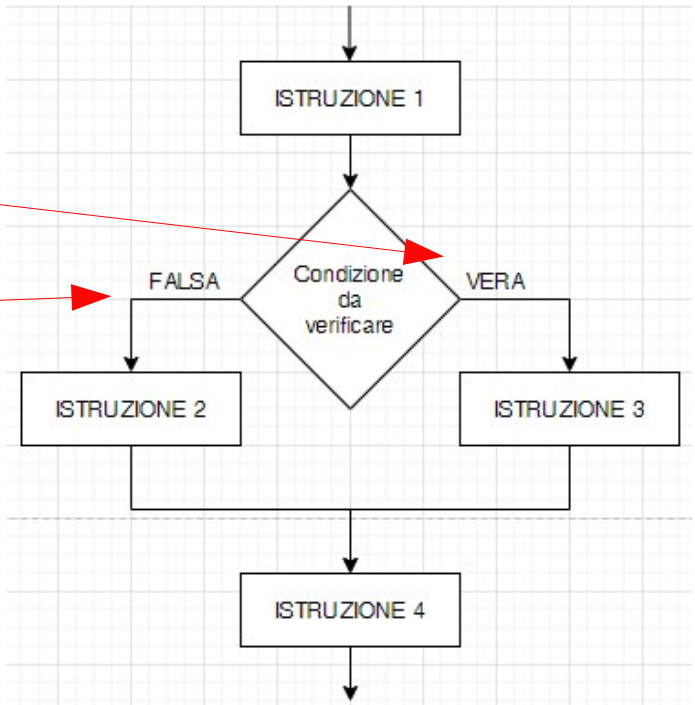
Nel cicli iterativi come il while-do, o nelle strutture di controllo, il confronto con un valore (ad esempio se una variabile vale 0) viene fatto con i seguenti operatori:

- **==** se uguale, ad esempio **if(a==b)** significa se **a=b**
- **>=** se maggiore o uguale si
- **<=** se minore 0 uguale di
- **!=** se diverso da

Struttura IF….ELSE

Nel linguaggio C questa struttura consente di selezionare due percorsi differenti del flusso del programma, in base ad una condizione se vera o falsa.

```
ISTRUZIONE 1;
if(condizione==vera){
   ISTRUZIONE 3;
 }//chiusura dell'IF
else{
   ISTRUZIONE 2;
 }//chiusura dell'ELSE
ISTRUZIONE 4;
```



Al'interno delle parentesi possono esserci ovviamente più istruzioni, e ogni istruzione sarà chiusa dal punto e virgola. Il punto e virgola non andrà messo invece sulla riga dell'IF e dell'ELSE.

La scelta del flusso potrebbe derivare dalla verifica di più condizioni come ad esempio nei seguenti casi:

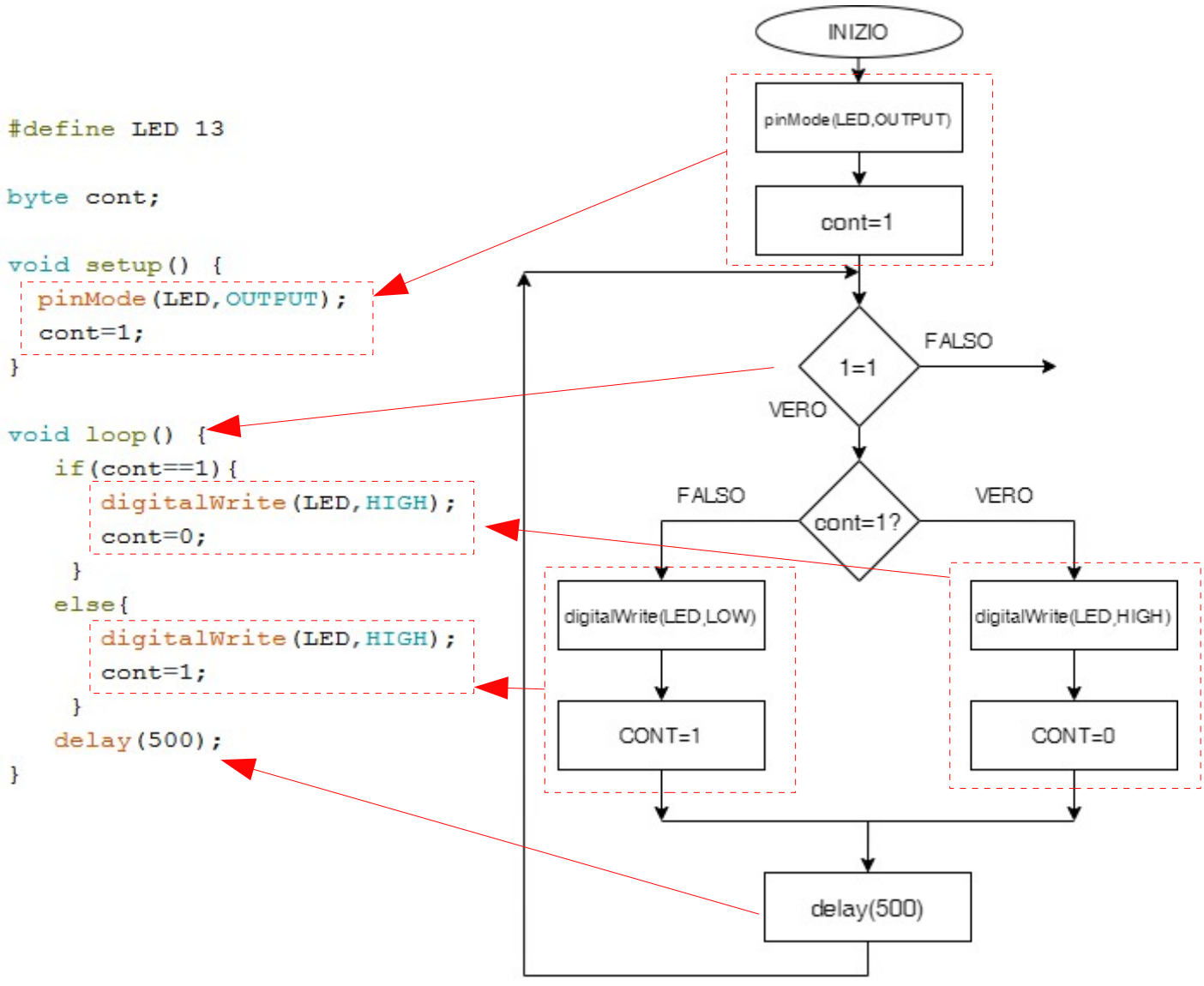
```
if((variabile1>10)&&(variabile1<100)){
 //viene eseguito se il valore della variabile1
 //è compreso tra 10 e 100
 //maggiore di 10 e minore di 100   ( && = AND )
}
```

```
if((variabile1<10)||(variabile1>100)){
 //viene eseguito se il valore della variabile1
 //non è compreso tra 10 e 100
 //minore di 10 o maggiore di 100   ( || = OR )
}
```

```
if((variabile1==10)&&(variabile1>=100)){
 //viene eseguito se il valore della variabile1
 //è uguale a 10 o maggiore uguale a 100
}
```

Proviamo a fare un programma dove si testa il valore di una variabile, se essa è uguale a 0 viene spento un LED altrimenti viene acceso. Con questo programma si realizza comunque il lampeggio del LED come nell'esempio precedente.

```
#define LED 13

byte cont;                    //dichiaro una variabile ad 8 bit

//questa parte di programma viene eseguito all'avvio
void setup() {
  pinMode(LED,OUTPUT);     //imposto il pin LED cioè 13 come uscita
  cont=1;
}//chiusura del setup

//questa parte di programma viene eseguita ciclicamente
void loop() {
   if(cont==1){                  //test della variabile cont
      digitalWrite(LED,HIGH);    //netto ad 1 il pin del LED
      cont=0;                    //metto a 0 la variabile cont
    }//chiusura dell'IF
   else{
      digitalWrite(LED,HIGH);    //netto ad 0 il pin del LED
      cont=1;                    //metto a 1 la variabile cont
    }//chiusura dell'ELSE
   delay(500);                   //attesa di 500msec
}//chiusura del loop
```



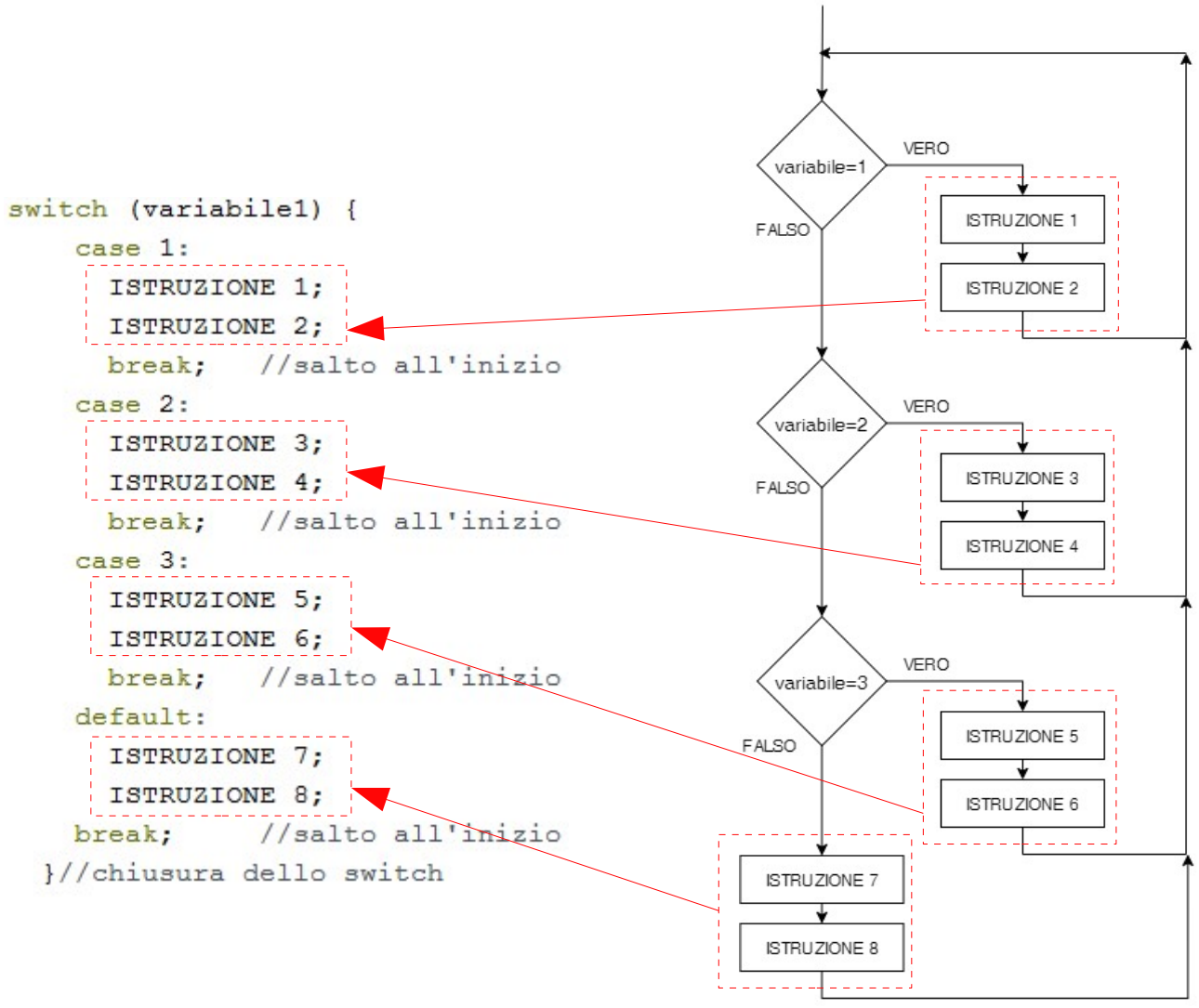
Struttura SWITCH...CASE

Con questa struttura si può controllare il valore di una variabile, ed in base al suo valore permettere più di due percorsi possibili, ad esempio volendo eseguire istruzioni differenti in base al valore di una variabile posso realizzare la seguente struttura:

```
switch (variabile1) {
    case 1:
      //inserire le istruzioni da fare
      //se variabile1=1
      break;
    case 2:
      //inserire le istruzioni da fare
      //se variabile1=2
      break;
    case 3:
      //inserire le istruzioni da fare
      //se variabile1=3
      break;
    default:
      //inserire le istruzioni da fare
      //negli altri casi
    break;
  }//chiusura dello switch
```

Vengono eseguite solo le istruzioni nel "**case"** associato al valore della **variabile1**.

L'istruzione **break**, consente di non testare gli altri **"case"** ma di tornare allo **switch** iniziale.



CONCLUSIONI

Ci sono ovviamente molti altri punti che verranno approfonditi con altre dispense.

In ogni caso, accedendo alla guida dall'IDE di Arduino, ci sono tutte le informazioni relative alle strutture, ai cicli ed alle funzioni disponibili.