UNITÀ 4: PROCESSI DECISIONALI E FUNZIONI DI CONTROLLO

OBIETTIVI

Studiare le funzioni che controllano il flusso e l'esecuzione di un programma. Sono chiamate anche "funzioni di controllo" e sono disponibili in tutti i linguaggi di programmazione; Arduino non fa eccezione.

Gli esempi che avete svolto finora sono programmi che comprendono una serie di funzioni eseguite sequenzialmente dalla prima all'ultima. Da ora in poi i programmi avranno una certa quantità di "intelligenza" e di capacità decisionale; ci saranno funzioni che vengono eseguite solo in determinate circostanze.

SEZIONE TEORICA

- Operatori di confronto
- Operatori booleani
- Operatori composti
- La funzione if(...)
- La funzione if(...) else
- La funzione for(...)
- La funzione while(...)
    - Alter forme di while
- The switch(…) / case function
- Other control functions
- La funzione switch(...)
- Altre funzioni di controllo
    - la funzione do … while(…)
    - la funzione break
    - la funzione goto

SEZIONE PRATICA

- ESEMPIO 1: La campana elettrica V2
- ESEMPIO 2: Le luci Fairy V2
- ESEMPIO 3: Il segnale di traffico V3
- ESEMPIO 4: Il segnale elettrico

- ESEMPIO 5: Il segnale di traffico V4
- ESEMPIO 6: Bursts
- ESEMPIO 7: La campana elettrica V3
- ESEMPIO 8: Il misuratore
- ESEMPIO 9: Tempistica

MATERIALI PRATICI

-Lap top o desktop computer

-Arduino IDE; questo dovrebbe includere il materiale supplementare già installato e configurato.

-Arduino UNO controller board

-Un cavo USB

SOMMARIO

SEZIONE TEORICA

Noi esseri umani siamo responsabili delle decisioni che prendiamo. Siamo in grado di analizzare contesti e circostanze e prendere decisioni basate sui nostri interessi, sentimenti, abilità, intuizioni, impegni ecc. Queste decisioni ci portano ad agire in modi diversi a seconda dei casi.

I programmi su cui hai lavorato finora sono stati molto sequenziali. Tutte le istruzioni vengono eseguite una dopo l'altra dal primo all'ultimo senza alcuna altra considerazione.

Tuttavia, Arduino, proprio come qualsiasi altro controller, ha la capacità di prendere decisioni e di eseguire programmi o svolgere compiti adeguati per ogni singolo caso. Sicuramente sapete che Arduino non ha alcun sentimento o intuito; non è nemmeno intelligente. È una macchina e tutto ciò che sa fare è lavorare con i numeri. Prende decisioni basate su di essi: calcoli logici e aritmetici, confronti, stati di segnali di ingresso, valori analogici, lettura di un sensore ecc ...

1. OPERATORI DI COMPARAZIONE

Come sapete, Arduino può eseguire operazioni aritmetiche come addizione e sottrazione. Sapete anche che i risultati possono essere espressi come costanti o variabili o una combinazione di entrambi.

Ma Arduino sa anche fare confronti tra i numeri o tra risultati di alcune funzioni. Ecco i vari operatori di confronto e i segni che li rappresentano:

OPERATORE	SEGNO	OPERATORE	SEGNO
Uguale a	= =	Diverso da	!=
Meno di	<	Più grande di	>
Meno di o uguale a	< =	Più grande di o uguale a	>=

Qualunque sia il confronto, ci sono solo due possibili risultati: "vero" o "falso". Esaminiamo attentamente i seguenti esempi:

Supponiamo che….

char Letter = 'J';	**int B = 12345;**
byte A = 13;	**float PI = 3.1416**

Quindi...

Letter == 'J'	**//Vero**
Letter != 'Q'	**//Vero**
18 < A	**//Falso**
A == 8 + 5	**//Vero**
B >= A	**//Vero**
PI * 2 > 8.16	**//Falso**
B – 1000 <= A * 12	**//Falso**
digitalRead(4)==1	**//Vero se pin4 è su livello "1"**

2. OPERATORI BOLEANI

È anche possibile relazionare tra loro i precedenti confronti. Arduino ha tre operatori logici o "booleani" per questo scopo; qui ci sono con i loro simboli:

OPERATORE	SEGNO
NOT	!
AND	&&
OR	||

Allo stesso modo, ci sono solo due possibili risultati quando due o più espressioni sono correlate utilizzando questi operatori logici: "true" o "false". Suggerisco di utilizzare parentesi graffe per raggruppare ogni relazione: farà leggere più facilmente e evitare errori. Date un'occhiata ai seguenti esempi:

(Letter == 'X') && (A > 10) **//Falso**

(A == 10+3) && (B >= 12345) && (Letter != 'Q') **//Vero**

(B > 12300) || (PI = 3.1412) **//Vero**

(A == B) || (A > 10 + 4) **//Falso**

!(A == B) **//Vero**

(digitalRead(4) ==1) && (digitalRead(8)==1) **//Vero se pin 4 e 8 sono su livello "1"**

3. OPERATORI COMPOSTI

Molte volte farai operazioni molto semplici con una variabile e il risultato sarà la stessa variabile. Ricorda che è possibile utilizzare i cosiddetti "operatori composti" per semplificare queste espressioni. Ecco un riepilogo di essi nella tabella seguente:

OPERATIONE	OPERATORE	ESEMPIO	UGUALE A
++	Aumenta una unità	X++	X= X + 1
- -	Diminuisce un numero	Y- -	Y = Y - 1
+ =	Addizione	X+=Y	X = X + Y
-=	Sottrazione	X-= 3	X = X - 3
***=**	Montiplicazione	X *= Y	X = X * Y
/=	Divisione	X /= 5	X = X / 5

4. FUNZIONE IF(...)

Questa è la funzione di controllo più importante e fondamentale. Permette di valutare le espressioni e di prendere decisioni. Nel calcolo, il segno nella figura a sinistra è usato per rappresentare la presa di una decisione nei diagrammi noti come diagrammi di flusso. Il controller valuta un'espressione come complicata o più complicata di quelle già viste. Se il risultato è "true" esegue tutte le funzioni all'interno delle parentesi graffe "{...}". Se il risultato è "false", il controller non li esegue e il programma continua.

Ricorda: qualsiasi espressione può essere formata da operazioni aritmetiche tra costanti e / o variabili correlate tra loro da operatori di confronto che possono a loro volta essere correlati utilizzando operatori logici o booleani. Prenditi il tuo tempo per pensare a questo e dare un'altra occhiata agli esempi precedenti. Questo è importante ... non c'è fretta!

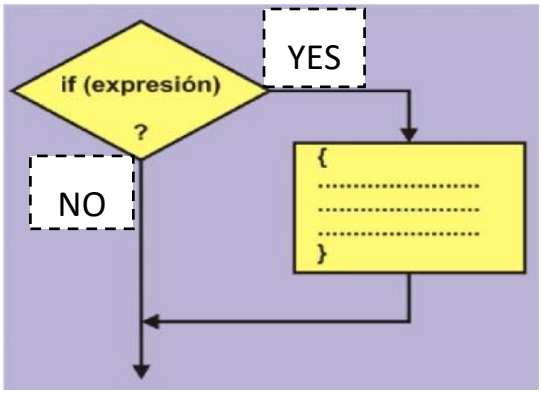

Figura 1

Sintassi:

if(expression)

{

….

….

}

espressione:stabilisce la condizione che il controller Arduino deve valutare. Potrebbe essere il rapporto tra costanti e / o variabili. Ci possono essere uno o più confronti legati tra loro da operatori logici. Si possono anche confrontare i risultati delle operazioni aritmetiche o altre funzioni del linguaggio Arduino.

parentesi graffe: le parentesi graffe racchiudono le funzioni che il controller deve eseguire se la condizione "vera" è soddisfatta. Non è necessario usarle se esiste un'unica funzione da eseguire.

Esempio:

```
 void loop()
{
if((A>B) || (C < 25))                    //Se la condizione è vera…
{
```

```
        digitalWrite(6,HIGH);          //Sposta su pin 6
C=25;                                  //Un valore di 25 è memorizzato nella variabile C
}
....                                   //Continua l'esecuzione....
}
```

Suggerimento: poiché l'uso della parentesi graffa è vario, è una buona pratica di programmazione digitare la parentesi di chiusura immediatamente dopo aver digitato quella di apertura, quando si inserisce un costrutto che richiede graffe. Quindi inserire alcuni spazi tra le parentesi graffe e iniziare a inserire le istruzioni. Nell'esempio le parentesi graffe che aprono e chiudono la funzione **loop()** sono facilmente distinguibili dalle parentesi graffe che racchiudono le funzioni da eseguire se la condizione **if()** è vera.

5. FUNZIONE IF(…) ELSE

Questa è un'altra funzione di controllo; è una derivata della precedente funzione **if(...)**. Indica esattamente cosa fare se la condizione (**else**) è falsa. Date un'occhiata alla figura 2. La funzione valuta l'espressione o la condizione. Se è "true", tutte le funzioni delle parentesi graffe "{...}" vengono eseguite allo stesso modo della funzione **if(...)**

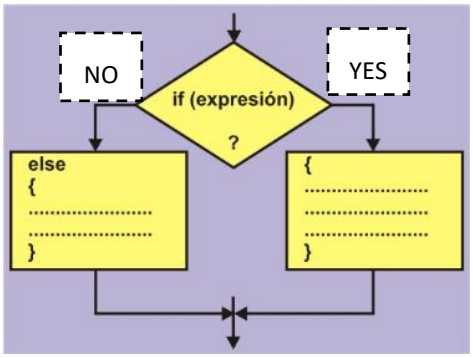

Figura 2

Se la condizione è "falsa" vengono eseguite tutte le funzioni racchiuse nelle parentesi **else{...}**. Una volta che le funzioni sono state eseguite, indipendentemente dal fatto che la condizione sia vera o falsa, il programma continua.

Sintassi:

if(expression)

{

….

….

}

else

{

….

….

}

Espressione: stabilisce la condizione che il controller Arduino deve valutare. Potrebbe essere un confronto tra costanti e/o variabili. Ci possono essere uno o più confronti legati tra loro da operatori logici. Può anche confrontare risultati di operazioni aritmetiche o altre funzioni linguistiche di Arduino.

Parentesi graffe: contengono tutte le funzioni che il controller deve eseguire se la condizione **(if)** è vera o falsa **(else)**. Non è necessario includerle se c'è una sola funzione da eseguire in entrambi i casi.

Esempio

```
if(digitalRead(4) == 1)          //Se il pin 4 è su "1" …
  {
digitalWrite(6,HIGH);            //Attiva pin 6
tone(2,2000,400);                //Genera un tono 2 KHz sul pin 2
}
else                             //…se non…
digitalWrite(6,LOW);             //Disabilita il pin di output 6
….                               //Continua a eseguire il programma
…
```

6. FUNZIONE FOR():

Questa funzione ci permette di creare cicli controllati. Un ciclo ripete un blocco di funzioni racchiuse in parentesi graffe.

Una figura ha più valore di mille parole. Date un'occhiata alla figura 3. Dichiariamo un valore iniziale di 1, ad esempio, per la variabile "N". Il valore viene testato e se è inferiore a 5 tutte le funzioni racchiuse nelle parentesi graffe sono eseguite. Il valore della variabile "N" viene modificato automaticamente. Viene aumentato di un'unità nell'esempio (N++). Si testa nuovamente. Se è vero, il blocco di istruzione e l'incremento vengono eseguiti e quindi la condizione viene testata di nuovo.

Quando la condizione diventa falsa, il ciclo termina. Le funzioni racchiuse nelle parentesi nella figura vengono eseguite quattro volte: è quanto il massimo numero "N" inferiore a 5.

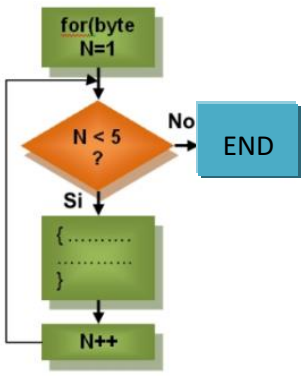

Figure 3

Sintassi:

for(initialization,condition,increment)

{

….

….

}

inizializzazione: questa è un'espressione che consente di stabilire il valore di una variabile. Accade solo una volta all'inizio del ciclo.

condizione: è la condizione che viene testata. Se è "true", viene eseguito l'incremento. Quando la condizione diventa "falsa", il ciclo termina e il programma continua. La condizione viene testata ogni volta che viene ripetuto il ciclo.

incremento: questa espressione consente di modificare il valore della variabile **i**. Questa espressione viene eseguita ogni volta che viene ripetuto il ciclo.

parentesi graffe: racchiudono tutte le funzioni che compongono il ciclo. Le funzioni vengono eseguite un numero di volte definito.

Esempio:

```
for (int N = 1; N < 5; N=N+1)          //Stabilisce le condizioni del loop
{
digitalWrite(6,HIGH);                  //Attiva il pin 6
delay (150);                           //Sospende il programma per 0.15”
digitalWrite(6,LOW);                   //Disabilita il pin 6
delay (1000);                          //Sospende il programma per 1”
}
….                                     //Continua l’esecuzione del programma
```

La sequenza di "attivazione, pausa, disattivazione, pausa" sul pin 6 viene ripetuta quattro volte.

È possibile utilizzare tutti i tipi di espressioni che abbiamo studiato all'inizio di questa unità per impostare l'inizializzazione, la condizione e l'incremento di qualsiasi ciclo **for()**. Potrebbero essere aritmetica, logica o booleana o confronti tra variabili e / o costanti. Supponiamo quanto segue:

```
int A = 5;
for(byte N=A+3; N <= A * 2 + 8; N = N + 3)
{
….
….
}
```

Ora è il tuo turno:

Qual è il valore iniziale? _____

E quello finale? _____

Quanto aumenta ogni volta? _____

Quante volte è ripetuto il ciclo? _____

7. FUNZIONE WHILE():

I cicli **while** sono una variazione delle funzioni **for()**, quindi vengono utilizzati anche per cicli in cui un gruppo di funzioni viene eseguito un certo numero di volte.

Sintassi:

while(condizione)

{

….

….

}

condizione: questa è l'espressione condizionale. Il ciclo continuerà all'infinito, finché l'espressione all'interno delle parentesi graffe, {}, diventa falsa. Quando avviene, il ciclo finisce e il programma continua.

Esempio:

int N = 6 while (N > 0) *//While N è maggiore di 0*

…

{

digitalWrite(6,HIGH); *//Attiva il pin 6*

```
delay (150);                    //Sospende per 0.15”
digitalWrite(6,LOW);            //Disattiva il pin 6
delay (1000);                   //Sospende per 1”
n--;                            //Il prossimo valore N ( N = N – 1)
}
….
….                              //Continua a eseguire il programma
```

A. ALTRE FORMA DI WHILE()

È molto comune usare la funzione **while()** come un ciclo all'interno di se stesso. Non ci sono parentesi graffe e il ciclo esegue la funzione fino a quando la condizione è falsa. Esaminiamo attentamente i seguenti esempi e notiamo dove è posizionato il ';'.

while(digitalRead(4)==1);

La funzione **while()** viene eseguita costantemente finché il pin 4 è sul livello "1". Ovvero, la funzione è sospesa finché il pin 4 va a livello "0" per continuare l'esecuzione del programma.

while(! digitaRead(4));

Questo è proprio come l'esempio precedente, ma al contrario. La funzione **while()** viene eseguita costantemente finché il pin 4 NON è al livello "1", o in altre parole, finché è sul livello "0". Attende quindi che il pin 4 vada al livello "1" per continuare ad eseguire il programma.

while(digitalRead(4)==0);

Questo è lo stesso dell'esempio precedente: attende che il perno 4 vada al livello "1" per continuare l'esecuzione del programma.

while(1);

Un ciclo infinito. La condizione è sempre vera (1), quindi la funzione **while()** viene eseguita costantemente e indefinitamente. Non esegue alcuna funzione successiva.

8. FUNZIONE SWITCH() / CASE

Questa funzione consente di scegliere tra diversi "modi" di esecuzione di vari gruppi di funzioni. Un'istruzione di commutazione confronta il valore di una variabile con i valori specificati nelle istruzioni di caso. Quando il valore dato corrisponde a quello della variabile, viene eseguito il codice.

La parola chiave di *interruzione* esce dall'istruzione **switch()** e viene utilizzata alla fine di ogni casistica. Qui di seguito:

"Se il valore della variabile è X, eseguire queste funzioni. Se il valore della variabile è Y, eseguire altre funzioni. Se è Z, altre ancora, ecc ... "

Sintassi:

switch(variable)

{

case X:

….;

break;

case n:

….;

….;

break;

default:

….;

}

variable: questo è il valore della variabile che verrà confrontata con quelle di ciascun "caso" menzionato.

case: questo definisce tutti i valori che verranno confrontati con il contenuto della variabile. Quando il contenuto della variabile coincide con uno dei valori, vengono eseguite tutte le funzioni tra questo caso e l'espressione di *interruzione*.

default: optional. Se nessuno dei valori corrisponde, tutte le funzioni vengono eseguite.

Esempio:

```
switch(A)                          //Variabile di paragone
{
case 1:                            //Se il valore di A è 1...
digitalWrite(6,HIGH);              //Attiva il pin 6
tone(2,200,200);                   //Suono sul pin 2
break;                             //Interruzione
case 3:                            // Se il valore di A è 3…
B=digitalRead(7);                  //Legge il pin 7
break;                             //Interruzione
case 124:                          //Se il valore di A è 124…
B=12*4;                            //Il valore di B è 48
digitalWrite(11,HIGH)              //Attiva il pin 11
break;                             //Interruzione
default:                           //Se non ci sono termini di paragone …
 digitalWrite(6.LOW);              //Disattiva il pin 6
 digitalWrite(11,LOW);             //Disattiva il pin 11
}
```

9. ALTRE FUNZIONI DI CONTROLLO

Le funzioni di controllo introdotte sono le più importanti che userai. Il linguaggio di programmazione Arduino ne prevede in ogni caso altre.

A. FUNZIONE DO…WHILE()

Il ciclo **do** funziona allo stesso modo del ciclo **while()**, ad eccezione che la condizione viene testata alla fine del ciclo, quindi il ciclo **do** verrà eseguito almeno una volta.

Sintassi:

```
do
{
```

….

} while(condition)

B. FUNZIONE BREAK

Break si usa per uscire da un ciclo **for()**,**while()** o **do()**, bypassando la condizione normale del ciclo. Viene utilizzato anche per uscire da un'istruzione **switch()/ case**.

Sintassi:

break;

C. FUNZIONE RETURN

Questo termina una funzione e restituisce un valore da qualsiasi funzione creata dall'utente. Imparerai a creare le tue funzioni un po' più avanti.

Sintassi:

return;

return value;

value: è il valore che la funzione restituisce quando torna al programma chiamante. È facoltativo e può essere una costante o formare parte di una variabile.

D. FUNZIONE GOTO

Trasferisce il flusso di programma a un punto contrassegnato nel programma.

Sintassi:

test:

….

goto test: *// Salta q qualsiasi punto l'etichetta indichi ("test") all'interno del programma.*

SEZIONE PRATICA

10. ESEMPIO 1: Campana elettrica V2

Questa è una versione migliorata dell'esempio che abbiamo visto nell'Unità precedente dove abbiamo dovuto riavviare l'intero sistema ogni volta che volevamo far suonare la campana.

Questa versione migliorata ci offre anche una buona opportunità di utilizzare la funzione **if()**. Il programma controlla se il pin 4 è abilitato.

Quando ciò accade, vengono emessi due suoni consecutivi di 400 e 300 Hz.

Se il pin 4 non è abilitato, il programma non fa nulla.

Diamo uno sguardo da vicino alle frecce rosse nella figura. E' consigliabile allineare le parentesi graffe in modo che la relazione tra le due sia chiara: una delle parentesi graffe si apre e l'altra si chiude.

```
void setup()
{
  pinMode( 2 , OUTPUT);  //D2 Salida del altavoz
}

void loop()
{
 if(digitalRead(4))    //Si se activa la patilla 4...
  {
   tone(2,400);        //Tono de 400 Hz
   delay(300);         //Durante 0,3"
   tone(2,300,300);    //Tono de 300 Hz durante 0.3"
  }
}
```

Figura 4

11. ESEMPIO 2: luci soffuse V2

Ecco un altro vecchio amico: le luminarie. Il movimento è da destra a sinistra o viceversa a seconda che il pin 4 sia a livello "0" o al livello "1". Questa è un'ottima opportunità per utilizzare la funzione **if(..) else**.

12. ESEMPIO 3: il semaforo V3

Continuiamo a migliorare alcuni degli esempi dell'ultima unità. Adesso è il momento del semaforo. In questa versione, la sequenza inizia quando il pedone premendo il pulsante si collega al pin 4. Non è necessario premere il pulsante RESET per riavviare il sistema come fatto nelle versioni precedenti.

Se nessuno preme il pulsante il semaforo rimane sul rosso. Questo è un altro buon esempio di come utilizzare la funzione **if (...) else**.

Suggerisco di confrontare questo esempio con le versioni precedenti. Le differenze sono piccole ma significative; miglioreremo a poco a poco il semaforo.

13. ESEMPIO 4: faro elettrico

This Questo è un buon esempio pratico che serve anche a introdurre la funzione **for(...)**. Tenta di simulare un faro che rilascia un certo numero di lampi di luce durante un determinato periodo di tempo.

La funzione **loop()** nella figura a destra contiene il programma principale. **for(...)** fissa il valore iniziale della variabile **(N=1)** per il numero di volte in cui il ciclo deve essere eseguito finché N non raggiunge il valore finale **(N<flash)**; esso fissa anche l'incremento di N ogni volta che viene eseguito il ciclo **(N ++)**.

Se diamo un'occhiata, vediamo che il ciclo deve essere ripetuto cinque volte. Ogni volta che viene ripetuto il ciclo, il LED bianco si accende per 0,15" e poi si spegne per altri 0,15". Una volta che il ciclo è terminato, c'è una pausa di 1,5 " e il processo viene ripetuto.

```
int Tiempo = 150;           //Tiempo de encendido y apagado
int Destellos = 5;          //Nº de destellos deseados

void setup()
{
pinMode(6,OUTPUT);           //D2 Salida
}

void loop()
{
for(byte N=1;N<=Destellos;N++)
  {
  digitalWrite(6,HIGH);       //El led blanco se enciende
  delay(Tiempo);              //Temporiza
  digitalWrite(6,LOW);        //El led blanco se apaga
  delay(Tiempo);              //Temporiza
  }
delay(1500);                  //Temporiza 1.5 s.
}
```

Figura 5

- **Ora è il tuo turno**

Questo programma è molto semplice. Proviamo a cambiare il numero dei lampeggiamenti, il tempo in cui la luce rimane accesa e spenta e il tempo tra un lampo e quello successivo. Sperimenta con diverse durate.

14. ESEMPIO 5: il semaforo V4

Dal momento che ora sapete come funziona **for...()**, questo è il momento ideale per progettare la versione finale (V4) del progetto del semaforo.

I due cicli **for()** generano la sequenza di toni ogni volta che si accendono le luci verdi e ambra. Il programma ha pertanto meno funzioni, utilizza meno memoria flash e quindi è più efficiente.

15. ESEMPIO 6: raffiche

I cicli **for(...)** possono essere inseriti uno dentro l'altro. Ciò significa che possiamo mettere un ciclo **for(...)** all'interno di altri cicli **for(...)**, e così via.

This program is a simple example of two nested **for(…)** loops. They make two lights flash on and off each time you press a pushbutton connected to pin 4.

Il LED rosso lampeggia una volta ogni cinque flash del LED bianco; la sequenza termina con il sesto flash.

Esistono due cicli **for(...)** all'interno della funzione **if(...).** I cicli vengono eseguiti ogni volta che qualcuno preme il pulsante collegato al pin 4. La variabile "B" controlla il ciclo interno. La variabile va da 1 a 5. Genera uno scoppio di cinque lampi del LED bianco.

La variabile "A" va da 1 a 6 e controlla il ciclo esterno **for(...)**. Poiché racchiude il ciclo interno, va in circolo sei volte. Ad ogni ciclo il LED rosso lampeggia.

Quando due o più cicli for(...) sono abbinati, il ciclo interno viene avviato prima e poi il successivo e così via fino al ciclo esterno.

```
void loop()
{
  if(digitalRead(4))  //Si la entrada D4 es verdad (nivel "1") ...
  {
    for(byte R=1; R<=6;R++)     //Mientras R <= 6
    {
      for(byte B=1;B<=5;B++)    //Mientras B <= 5 ...
        {
        digitalWrite(6,HIGH);   //Activa el led blanco
        delay(80);              //Temporiza
        digitalWrite(6,LOW);    //Desactiva el led oblanc
        delay(100);             //Temporiza
        }                       //Siguiente valor para B
     digitalWrite(11,HIGH);     //Activa el led rojo
     delay(100);                //Temporiza
     digitalWrite(11,LOW);      //Desactiva el led rojo
     delay(120);                //Temporiza
```

Figura 6

16. ESEMPIO 7: campana elettrica V3

Ecco la versione V3 della campana elettrica; è la versione finale e definitiva. La cosa importante di questo esempio è il modo in cui trattiamo il segnale di ingresso sul pin 4. Non è più sufficiente premere il pulsante e impostare un livello "1". Dobbiamo anche impostare il livello di riposo "0".

Questo compito è noto come "rilevamento degli impulsi". Viene utilizzato molto nei sistemi di controllo in cui il segnale di ingresso non è costituito da un livello logico statico come "1" o "0". Ci sono molte periferiche di input che emettono impulsi. Un impulso è una transizione completa dal livello "0" al livello "1" e viceversa "0" (0-1-0) o viceversa (1-0-1)

La funzione **while()** è ideale per rilevare queste situazioni e molte altre. Dai un'occhiata al programma nella figura a destra.

La prima funzione **while()** attende finché si trova sul livello "0". La seconda funzione **while()** attende finché il pin di ingresso numero 4 si trova al livello "1". Conclusione: finché non si preme il pulsante (livello "1") e poi si solleva il dito (livello "0"), non emette alcun suono. Provalo e fai attenzione a questo dettaglio; questo è ciò che lo rende diverso dalla campana elettrica nella versione V2.

```
void setup()
{
}

void loop()
{
  while(! digitalRead(4));  //Espera mientras la patilla 4 está a "
  while(digitalRead(4));    //Espera mientras la patilla 4 está a "
  tone(2,400);              //Tono de 400 Hz
  delay(300);               //Durante 0,3"
  tone(2,300,300);          //Tono de 300 Hz durante 0.3"
}
```

Figure 7

17. ESEMPIO 8: metro

Questa è un'applicazione pratica. Immagina un sistema di controllo degli accessi per un auditorium con una capacità di dieci persone. Un sensore collegato al pin 4 rileva quando una persona passa e genera un "impulso" per ogni persona che entra. Quando l'auditorium è pieno, il dispositivo emette un segnale acustico per dire al pubblico che la performance sta per iniziare. Dai un'occhiata alla funzione **loop()** nella figura a sinistra.

Le funzioni contenute nella funzione **while()** vengono eseguite finché la variabile "contatore" è inferiore o equivalente a 10. Questa variabile aumenta ogni volta che viene rilevato un impulso 0-1-0 sul pin 4. Due funzioni separate **while()** sono anche utilizzate per il rilevamento.

Quando il sensore ha generato dieci impulsi, il che significa che ci sono dieci persone nell'auditorium, emette un segnale acustico.

```
void loop()
{
  byte Contador=1;          //Inicia la variable con 1
  while(Contador <= 10)
  {
    while(! digitalRead(4)); //Espera mientras la patilla 4 está a
    while(digitalRead(4));   //Espera mientras la patilla 4 está a
    Contador++;              //Pulso recibido, contador + 1
  }
  tone(2,400);              //Tono de 400 Hz
  delay(300);               //Durante 0,3"
  tone(2,300,300);          //Tono de 300 Hz durante 0.3"
}
```

Figura 8

- **Ora è il tuo turno**

Una volta registrato il programma, assicurati che funzioni correttamente. Tutto ciò che devi fare è premere il pulsante sul pin 4. Questo simula il sensore del contatore persone. Siamo certi che funzioni correttamente?

Se guardi bene, noterai che sembra contare meno persone di quante ce ne siano davvero. O in altre parole, quando cinque o sei persone sono passate, il cicalino si spegne. Questo difetto è dovuto a un problema noto come "effetto rebound" e si verifica spesso con pulsanti e interruttori.

Anche se si preme il pulsante una sola volta, il pin di input sul controller non riceve solo un singolo impulso 0-1-0, ma ne riceve diversi. Ciò è dovuto al fatto che i contatti metallici sul pulsante richiedono un po' di tempo per ristabilirsi ogni volta che si preme il pulsante. Dai un'occhiata alla figura.

Anche se questa volta è solo pochi millisecondi, il nostro Arduino è molto più veloce e rileva tutti quegli impulsi. Ciò significa che anche se si preme il pulsante una volta sola, Arduino potrebbe rilevare questo come uno, due o più impulsi.

Un modo per evitarlo è inserire una breve pausa. Quando viene rilevato un cambiamento di stato nel pulsante, inseriamo una pausa di 20 mS prima di attendere la modifica successiva. In questo modo evitiamo qualsiasi rimbalzo durante quel lasso di tempo. Apporta le seguenti modifiche al programma EXAMPLE_6_8, registralo e verifica se funziona correttamente.

while(! digitalRead(4)); *//Attende finchè il pin 4 è su “0”*

delay(20); *//Pausa anti-rebound*

while(digitalRead(4)); *//Attende finchè il pin 4 è su "1"*

delay(20); *//Pausa anti-rebound*

Counter ++; *//Impulso ricevuto, contatore + 1*

18. ESEMPIO 9: tempistica

Ti ricordi la funzione **millis()**? È stato quello che ti ha permesso di sapere quanto tempo passa tra il momento in cui connetti Arduino e / o il momento in cui lo riavvii. La useremo in combinazione con la funzione **switch(...) / case** per misurare diversi intervalli di tempo.

Dopo la sequenza di riavvio di Arduino misureremo quanto tempo trascorrerà. Dopo un secondo il LED bianco si accende. Dopo due, il LED verde si accende, dopo tre, il LED ambra si accende e dopo quattro, il LED rosso. Infine, quando sono trascorsi 8 secondi da quando il sistema è stato riavviato, tutti i LED si spengono e viene emesso un segnale acustico. Osserva da vicino la funzione **loop()** nella figura.

La funzione **A=milis()** memorizza i millisecondi che sono trascorsi dal riavvio del sistema nella variabile "A". La funzione **switch(A)** analizza il suo contenuto. Se il valore del contenuto è 1000 (**case 1000:**), il LED bianco si accende. È trascorso un secondo (1000 mS). Se il valore del contenuto è 2000 (**case 2000:**), il LED verde si accende; se il valore è 3000, la spia ambra si accende e se il valore è 4000, il LED rosso si accende. Infine, se il valore del contenuto è 8000 (**case 8000:**), saranno trascorsi 8 secondi. Tutti i LED si spengono e viene emesso un tono a 1 KHz.

Il programma continua a funzionare ma il tempo trascorso supera 8000 ms (8 secondi), quindi la variabile "A" non soddisfa più nessuna delle cinque condizioni.

Sai per quanto tempo dovrai aspettare prima che la variabile soddisfi nuovamente le cinque condizioni?

Calcoliamolo. La funzione **millis()** restituisce un numero intero lungo senza segno di 32 bit. In altre parole, restituisce un valore compreso tra 0 e 4.294.967.295 millesimi di secondo che equivalgono a 4.294.967 secondi. Tenendo presente che un giorno ha 86.400 secondi (24 * 60 * 60), occorrerebbero 50 giorni affinché la funzione **millis()** superi l'overflow e torni a 0.

Se si desidera tornare indietro e vedere la sequenza di accensione del LED, è possibile attendere tutto il tempo necessario o semplicemente riavviare il sistema.

```
void loop()
{
 A=millis();             //Lee el tiempo transcurrido tras el RESE
 switch(A)               //Analiza la variable
 {
   case 1000:              //Si vale 1000 (1") ...
     digitalWrite(6,HIGH); //Activa el led blanco
     break;
   case 2000:              //Si vale 2000 (2") ...
     digitalWrite(6,LOW); //Desactiva led blanco
     digitalWrite(9,HIGH); //Activa el led verde
     break;
   case 3000:              //Si vale 3000 (3") ...
     digitalWrite(9,LOW);  //Desactiva led verde
     digitalWrite(10,HIGH); //Activa el led ámbar
     break;
   case 4000:              //Si vale 4000 (4") ...
     digitalWrite(10,LOW); //Desactiva led ámbar
     digitalWrite(11,HIGH); //Activa el led rojo
     break;
   case 8000:              //Si vale 8000 (8") ...
     digitalWrite(11,LOW); //Desactiva led rojo
     tone(2,1000,300);     //Tono de 3 KHz durante 0.3"
     break;
 }
}
```

Figura 9

RIFERIMENTI

PUBBLICAZIONI

[1]. **EXPLORING ARDUINO**, Jeremy Blum, Ed.Willey
[2]. Practical ARDUINO, Jonathan Oxer & Hugh Blemings, Ed.Technology in action
[3]. Programing Arduino, Next Steps, Simon Monk
[4]. Sensores y actuadores, Aplicaciones con ARDUINO, Leonel G.Corona Ramirez, Griselda S. Abarca Jiménez, Jesús Mares Carreño, Ed.Patria
[5]. ARDUINO: Curso práctico de formación, Oscar Torrente Artero, Ed.RC libros
[6]. 30 ARDUINO PROJECTS FOR THE EVIL GENIUS, Simon Monk, Ed. TAB
[7]. Beginning C for ARDUINO, Jack Purdum, Ph.D., Ed.Technology in action
[8]. ARDUINO programing notebook, Brian W.Evans

SITI WEB

[1]. https://www.arduino.cc/
[2]. https://www.prometec.net/
[3]. http://blog.bricogeek.com/
[4]. https://aprendiendoarduino.wordpress.com/
[5]. https://www.sparkfun.com