UNITÀ 1: PROGRAMMI BASE

OBIETTIVI

L'obiettivo di questa unità è creare alcuni programmi iniziali che consentono di lavorare in modo rapido e semplice con input e output digitali (I/O). Prenderemo in considerazione i programmi scritti nel linguaggio di programmazione Arduino in modo da poter vedere ciò che appare e poi scriverne uno da soli. È un linguaggio sofisticato con le proprie regole, affermazioni e sintassi. Prima di tutto esamineremo le istruzioni di base necessarie per utilizzare l'input e l'output digitali; si trovano sulla scheda BASIC I / O Arduino.

SEZIONE TEORICA

- COM'È FATTO UN PROGRAMMA
  - COMMENTI
  - VARIABILI E FUNZIONI
  - ATTIVITA' DI CONFIGURAZIONE
  - CORPO PRINCIPALE
  - MESSAGGI
- ISTRUZIONI DI PROGRAMMA
  - funzione set up()
  - funzione loop()
  - funzione pinModel()
  - funzione digitalRead()
- funzione digitalWrite()
- OPERATORI LOGICI
  - Operatore NOT
  - Operatore AND
  - Operatore OR
  - Combinare gli operatori

SEZIONE PRATICA

- ESEMPIO 1 Illuminazione di un LED da 1 volt

- ESEMPIO 2: Illuminazione di un LED da 2 volt
- ESEMPIO 3: Illuminazione di un LED da 3 volt
- ESEMPIO 4: Illuminazione inversa di un LED
- ESEMPIO 5: Comando a due pulsanti
- ESEMPIO 6: Lavoro individuale

MATERIALE RICHIESTO

-Lap top o desktop computer

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

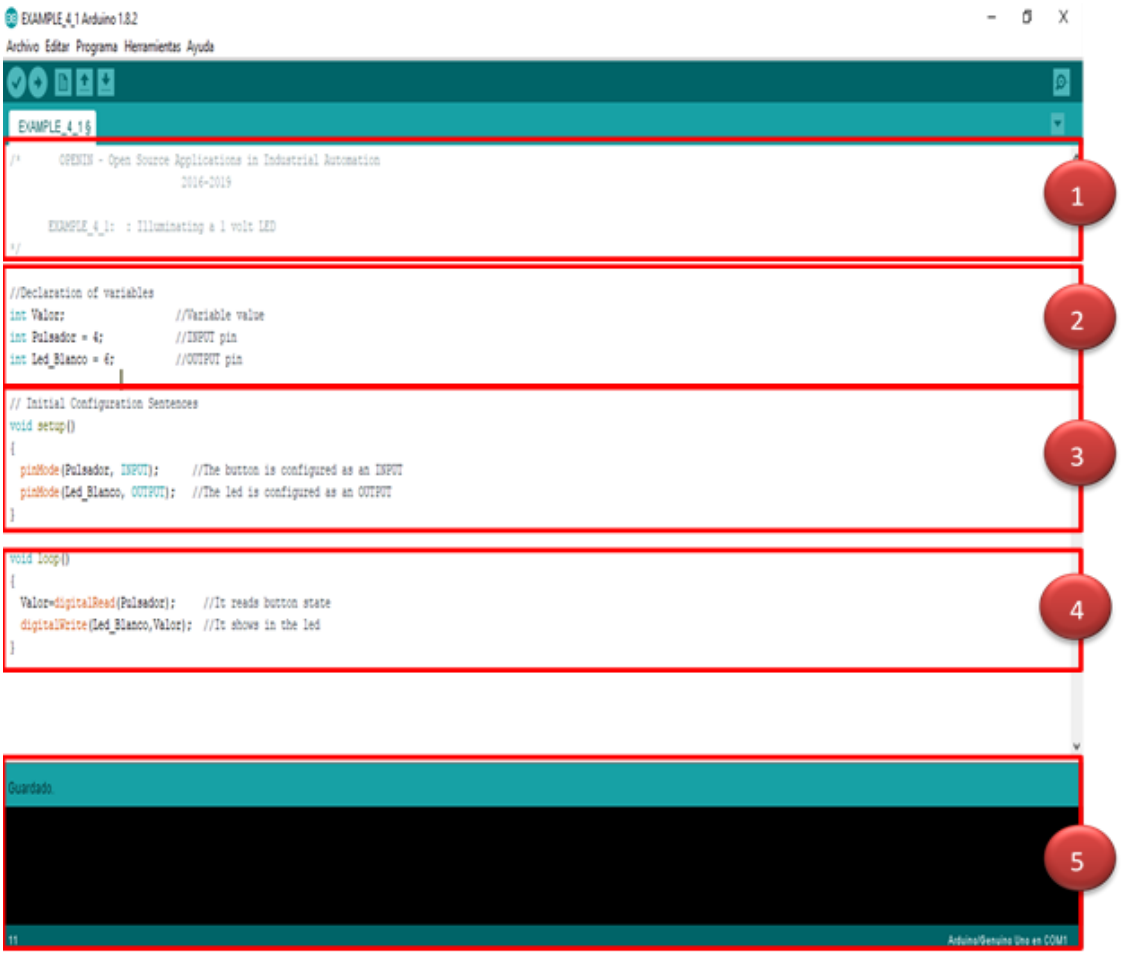
-Arduino UNO controller

-Un cavo USB

SOMMARIO

SEZIONE TEORICA

1. COM'E' FATTO UN PROGRAMMA



A. COMMENTI

Tutti i programmi dovrebbero iniziare fornendo alcune informazioni. In questo caso, ad esempio, questo sarebbe il nome del percorso, la data, l'autore, la società, ecc. Sarebbe anche interessante inserire una spiegazione di ciò in cui consiste il programma o che cosa faccia.

Queste informazioni sono chiamate "Commenti dell'intestazione". Possono essere inclusi tutti i commenti desiderati ovunque si voglia e appaiono tra i simboli "/ *" e "* /". Diamo uno sguardo da vicino all'esempio precedente.

Possono essere inclusi anche semplici commenti di una linea.

EXAMPLE_4_1

```
/*       OPENIN - Open Source Applications in Industrial Automation
                              2016-2019

      EXAMPLE_4_1:   : Illuminating a 1 volt LED
*/
```

1

Figure 1

Abituiamoci a scrivere commenti sui programmi. Potremmo averne bisogno in futuro per ricordarci di quello che abbiamo fatto.

B. DICHIARAZIONE DI VARIABILI E FUNZIONI

Consigliamo di utilizzare questa seconda sezione per dichiarare variabili e funzioni. Pensiamo a una variabile come una sorta di casella o di un recipiente cui assegnare un nome e possibilmente un valore. Lo si salva nella memoria RAM del controller e potremo utilizzarla più volte.

Una funzione comprende una serie di altre istruzioni o funzioni. Diamo anche a questo gruppo un nome e potremo usarlo tutte le volte che ne avremo bisogno.

```
//Declaration of variables
int Valor;                    //Variable value
int Pulsador = 4;             //INPUT pin
int Led_Blanco = 6;           //OUTPUT pin
```

2

Figure 2

Nell'esempio nella figura precedente sono state create tre variabili: "Valore", "Pulsante" e "LED_Bianco". L'input digitale verrà salvato nella prima. La seconda variabile, "Pulsante", ha un valore di 4 che corrisponde al numero del pin di input cui il pulsante è collegato; dobbiamo leggere lo stato del pulsante. Abbiamo dato un valore di 6 alla variabile "LED_Bianco". Questo corrisponde all'output del pin cui è stato collegato il LED. Il LED bianco è quello sulla scheda di sperimentazione BASIC I/O.

Generalmente, entrambe le variabili e le funzioni devono essere dichiarate PRIMA di essere utilizzate nel programma.

C. ATTIVITA' DI CONFIGURAZIONE

Normalmente i programmi scritti in lingua Arduino iniziano eseguendo alcune istruzioni o funzioni di configurazione. Questi determinano quali pin devono essere usati come input e quali come output.

```
// Initial Configuration Sentences
void setup()
{
  pinMode(Pulsador, INPUT);       //The button is configured as an INPUT
  pinMode(Led_Blanco, OUTPUT);    //The led is configured as an OUTPUT
}
```

3

Figure 3

La variabile "Pulsante" nella figura 2, che precedentemente aveva un valore di 4, è configurato come INPUT. Il pin variabile "LED_Bianco", o pin 6, è configurato come OUTPUT.

In generale, le istruzioni di configurazione o funzioni vengono eseguite una sola volta quando si reimposta il sistema o lo si collega ad una fonte di alimentazione.

D. CORPO PRINCIPALE DEL PROGRAMMA

Dobbiamo scrivere tutte le istruzioni e le funzioni che costituiscono il nostro programma. Ci sono solo due funzioni nell'esempio che stiamo considerando al momento, ma teniamo presente che un programma ne può contenere centinaia o addirittura migliaia.

```
void loop()
{
  Valor=digitalRead(Pulsador);     //It reads button state
  digitalWrite(Led_Blanco,Valor);  //It shows in the led
}
```

4

Figura 4

La prima funzione, "Valore=digitalRead(Pulsante)", legge lo stato digitale del pulsante collegato al pin 4, quello precedentemente configurato come input. Lo stato del pulsante viene salvato nella variabile "Valore" nella memoria RAM.

La seconda funzione "digitalWrite(Led_Bianco,Valore);" scrive il contenuto della variabile "Valore" sul pin 6. Ricordiamo che questo pin, il numero 6, era stato precedentemente configurato come output.

Il controller esegue tutte le funzioni che compongono il corpo principale del programma il più velocemente possibile. Le esegue costantemente e indefinitamente dalla prima all' ultima.

Importante: Quando si dichiarano le variabili, le funzioni di configurazione e del programma principale, è bene accertarsi che terminino SEMPRE con questo simbolo: ";".

E. MESSAGGI

In quest'ultima sezione, l'ambiente di lavoro IDE Arduino visualizzerà una serie di messaggi: ci dirà se stiamo salvando, compilando o registrando un programma nella memoria del controller. Ci informerà inoltre se ci siano stati errori di compilazione, il tipo di errore e dove si siano verificati.

Questo esercizio ha funzionato perfettamente. Ci viene detto che il programma occupa un totale di 1.066 dei 32.256 byte disponibili di memoria FLASH. 15 dei 2.048 byte disponibili di memoria RAM sono stati utilizzati per i dati (a causa delle variabili che abbiamo creato).

Quindi ci sono ancora 2033 byte disponibili.

Ricordiamo: "Compilare" un programma significa tradurre quello che abbiamo scritto utilizzando un linguaggio complesso Arduino in codice binario (noto anche come lingua macchina); questo è ciò che è effettivamente registrato sulla memoria FLASH del controller.

2. ISTRUZIONI DI PROGRAMMA

Ora abbiamo visto in teoria com'è fatto un programma. Abbiamo visto che è costituito da una serie di istruzioni chiamate anche "FUNZIONI"; un programma può contenerne migliaia. Ogni linguaggio di programmazione ha le proprie funzioni, la propria sintassi e le proprie regole di programmazione. Daremo un'occhiata al linguaggio di programmazione Arduino; abbiamo già usato cinque diverse istruzioni o funzioni. Ora guardiamo più da vicino.

A. LA FUNZIONE SETUP()

Questa funzione e tutte le altre funzioni che si trovano al suo interno vengono eseguite quando il sistema viene reimpostato. Ciò avviene ogni volta che si preme il pulsante RESET o si accende il sistema.

In genere ci sono altre funzioni all'interno di questa funzione: configurazione del pin di input e output, alcune impostazioni di variabili, librerie, ecc. In termini generali, è possibile inserire qualunque istruzione e funzione che desideri nella funzione **setup ()** purché siano racchiuse da parentesi graffe: "{...}". Dobbiamo sempre creare una funzione **setup ()** anche se non c'è niente all'interno: basta lasciare vuote le parentesi graffe.

La cosa importante da ricordare è che tutte le funzioni contenute nella funzione setup () vengono eseguite solo una volta: quando si ripristina (RESET).

Sintassi:

Void setup()

{

.....}

Example

Void setup()

{

pinMode(INPUT, Pulsante); *// Il PIN del pulsante è configurato come input*

pinMode(LED_Bianco,OUTPUT); *// Il PIN del pulsante è configurato come output*

}

La funzione principale del **loop()** può includere decine, centinaia o migliaia di altre funzioni. L'esempio precedente ne comprende solo due. La funzione loop () può anche essere vuota senza alcuna funzione racchiusa tra le parentesi graffe: {}. Ma non fa nessuna differenza: devi sempre averne una.

Tutte le altre funzioni racchiuse nella funzione loop () sono eseguite costantemente dalla prima all'ultima, una dopo l'altra, finché non si disconnette il sistema.

B. LA FUNZIONE PINMODE()

Serve a configurare uno dei pin del controller Arduino come input o output. Di solito si trova all'inizio di un programma ed è incluso nella funzione setup ().

Sintassi:

pinMode;

pin: questo è il numero del pin che stiamo per configurare; può essere tra 0 e 13 in Arduino UNO..

Mode: determina se funziona come INPUT o OUTPUT

Esempi:

int Button=4;

int White_LED=6;

pinMode (White_LED, OUTPUT); // Il pin 6 è un output

pinMode (9 OUTPUT); // Il pin 9 è un output

pinMode (Button, INPUT); // Il pin 4 è un input

pinMode (12, INPUT); // Il pin 12 è un input

Tutti i pin digitali vengono configurati automaticamente come ingressi quando si reimposta il sistema.

C. LA FUNZIONE DIGITAL READ()

Questa funzione legge e restituisce lo stato logico binario ("1" o "0", "HIGH" o "LOW") di ognuno dei pin del controller Arduino.

Sintassi:

digitalRead (pin);

pin: visualizza il numero del pin che intendiamo leggere; può essere tra 0 e 13 in Arduino UNO.

Esempi:

int Button = 4;

Value=digitalRead (Button); // legge lo stato del pin 4 e lo salva in "Valore"

X=digitalRead(8); // Legge lo stato del pin 8 e lo salva in "X"

D. LA FUNZIONE DIGITALWRITE()

La funzione digitalWrite scrive o imposta un valore binario ("1" o "0", "HIGH" o "LOW") tramite un pin di output.

Sintassi:

digitalWrite(pin, valore);

pin: ci dà il numero del pin di cui intendiamo impostare il valore; può essere tra 0 e 13 in Arduino UNO.

Value: indica il valore da impostare ("1" o "0", "HIGH" o "LOW").

Esempi:

int White_LED=6;

DigitalWrite(White_LED,valore); //Imposta il contenuto del "valore" "1" o "0" attraverso il pin 6

DigitalWrite(11, LOW); // Imposta il livello "0" attraverso l'output 11

DigitalWrite(7,1); //Imposta il livello "1"attraverso l'output 7

3. OPERATORI LOGICI

Non possiamo proseguire senza parlare di tre tipi di operatori che sono in grado di correlare diversi tipi di funzioni, operazioni, espressioni, ecc… tra loro. Sono conosciuti come "operatori logici" e vengono utilizzati molto nei sistemi digitali.

A. OPERATORE NOT

Questo operatore esprime una negazione (NOT) ed è rappresentato da un punto esclamativo (!). È più facile capirlo con un esempio. Diamo un'occhiata a questa funzione:

Value=digitalRead(Button);

La variabile "Valore" è equivalente al livello "1" quando il "pulsante" è anche su "1". In caso contrario, il "valore" corrisponde al livello "0". Diamo un'occhiata alla seguente funzione:

Value = ! digitalRead(Button);

La variabile "Valore" è equivalente al livello "1" quando il "Pulsante" viene letto e non è - **NOT** - sul livello "1", ovvero si trova in livello "0". Altrimenti, se il pulsante è sul livello "1", la variabile "Valore" sarà livello "0". Diamo un'occhiata all'esempio riportato di seguito e rispondiamo:

Value = ! digitalRead(12);

"Valore" equivale a livello "1" o "vero" se: _____

“Valore” equivale a livello “0” o “falso” se: _____

B. OPERATORE AND

Questa operazione genera un livello "1", noto anche come "vero", quando **TUTTI** gli elementi che si rapportano l'uno all'altro sono anche a livello "1" o sono veri. È rappresentato da questi simboli: "&&". Diamo un'occhiata all'esempio:

Value =digitalRead(4) && **digitalRead(12);**

La variabile "Value" è equivalente al livello "1", "true", quando gli inputs 4 e 12 sono letti e entrambi sono anche a livello "1". Se uno degli input è a livello "0" il risultato della variabile "Valore" è anche "0" o "falso". Diamo un'occhiata all'esempio seguente e rispondiamo:

Value =digitalRead(4) && **digitalRead(8)** && **digitalRead(12);**

“Valore” equivale a livello “1” o “vero” se: _____

“Valore” equivale a livello “0” o “falso” se:

C. L’OPERATORE OR

Questa operazione genera un livello "1", noto anche come "vero", quando QUALSIASI degli elementi che si collegano tra loro sono anche a livello "1" oppure sono veri. È rappresentato da questo simbolo: "||". Diamo un'occhiata all'esempio:

Value =digitalRead(4) || digitalRead(12);

La variabile "Valore" è equivalente al livello "1", "vero", quando gli input 4 o 12 o entrambi sono a livello "1". Se tutti gli input sono a livello "0" il risultato della variabile "Valore" è anche a livello "0" o "falso". Diamo un'occhiata all'esempio seguente e rispondiamo:

Value =digitalRead(4) || digitalRead(8) || digitalRead(12);

“Valore” equivale a livello “1” o “vero” se:

“Valore” equivale a livello “0” o “falso” se:

Value =digitalRead(4) || digitalRead(12);

The “Value” variable is equivalent to level “1”, “true”, when inputs 4 OR 12 OR both are at level “1”. If all of the inputs are at level “0” the result for the “Value” variable is also level “0” or “false. Have a look at the following example below and reply:

Value =digitalRead(4) || digitalRead(8) || digitalRead(12);

“*Value*” equals level “1” or “*true*” if:

“*Value*” equals level “0” or “*false*” if:

D. COMBINAZIONE OPERATORI

Naturalmente si possono combinare diversi operatori logici nella stessa funzione. Si devono usare le parentesi per stabilire l'ordine con cui devono essere valutati e calcolati. Diamo un'occhiata all'esempio riportato di seguito:

Value = ! digitalRead(4) && **digitalRead(7);**

La variabile "Valore" è equivalente al livello "1" o "vero" quando l'input 4 è **NOT** sul livello "1" **BUT** l'input 7 è… Diamo un'occhiata all'esempio seguente e rispondiamo:

Value = (digitalRead(8) && **! digitalRead(12)) || digitalRead(4));**

“Valore” equivale a livello “1” o “vero” se: _____

“Valore” equivale a livello “0” o “falso” se: _____

AREA PRATICA

4. ESEMPIO 1: Illuminare un led a 1 volt

Questo è lo stesso esempio che abbiamo studiato nell'area teorica. Ricordiamo che ciò che fa è leggere lo stato del pulsante collegato al pin 4 e registrarlo nel LED bianco della scheda di sperimentazione BASIC I / O collegata al pin 6 sul controller.

- **Compilazione e registrazione**

Come si può vedere dalla figura, tutto quello che dobbiamo fare è premere il pulsante contrassegnato 1. L'ambiente IDE Arduino compila il programma per noi traducendolo in codice macchina o binario; l'intero processo è abbastanza veloce e completamente automatico.

Se il programma è scritto correttamente e non ci sono errori, vedremo alcuni messaggi come quelli della finestra 2. L'abbiamo già visto prima: dicono quanto spazio occuperà nella memoria FLASH del controller così come la quantità di memoria RAM consumata dalle variabili che abbiamo utilizzato. Oltre a questo, se il programma è stato scritto correttamente, verrà registrato immediatamente nella memoria FLASH del controller.

Se ci sono errori, appariranno nella stessa finestra; vedremo i tipi di errori che commessi e dove sono nel programma. Basta correggerli e continuare a compilare il programma.

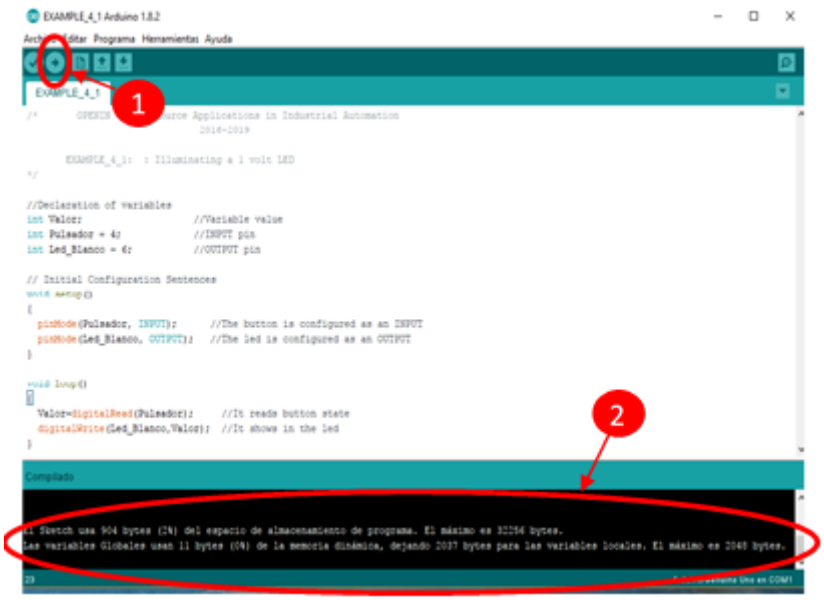


Figura 5

- **Controllo**

Anche se l'ambiente IDE Arduino non ti comunica errori quando hai completato la compilazione, non significa necessariamente che il programma funzioni correttamente. L'ambiente rileva solo errori sintattici: una funzione non corretta o non esistente, variabili che non sono state precedentemente dichiarate, dati non corretti, ecc.

Il test effettivo avviene quando si controlla che il controller esegua esattamente quello che abbiamo programmato. Questo esempio è molto semplice: supponendo che la scheda di esperimento BASIC I / O sia inserita correttamente sulla scheda di controllo Arduino UNO, basta premere il pulsante D4: se il LED bianco collegato al pin D6 si accende, tutto va bene; se non è così, c'è un errore.

Un ultimo controllo: quando registriamo un programma nella memoria FLASH, rimane lì indefinitamente anche se si spegne l'alimentazione. I programmi vengono eliminati solo quando se ne registra un altro. Ecco un altro dettaglio: una volta registrato un programma, la scheda Arduino non si basa più sul PC in alcun modo. E c'è un modo molto semplice per controllare tutto questo: togliere il cavo USB dalla scheda Arduino UNO e collegarla a un normale cavo di alimentazione oppure collegare una batteria come nella figura. Vedremo presto se tutto funziona proprio come ha fatto prima.

Possiamo pensare che tutto questo non sia molto utile e che non abbiamo bisogno di un controller per illuminare un LED con un pulsante. E' un errore. Abbiamo compiuto il primo passo: se abbiamo capito il concetto di un programma e delle sue funzioni, saremo in grado di fare cose molto più complicate e interessanti.

5. ESEMPIO 2: Illuminare un LED a 2 volt

Ecco un'altra versione del programma precedente. Controlla il LED bianco BASIC I / O collegato al pin 6 con il pulsante collegato al pin 4 sulla stessa scheda. Diamo un'occhiata alla figura e paragoniamola all'esempio precedente.

La differenza è che non abbiamo usato variabili per definire quali pin corrispondano al pulsante e al LED bianco.

I numeri di riferimento dei pin 4 e 6 appaiono nelle funzioni stesse: **pinMode(), digitalRead(), digitalWrite().**

In questo caso, i numeri del pin vengono espressi come "*costanti*" e non "*variabili*".

Come abbiamo fatto con l'esempio precedente, compiliamo e registriamo il programma; dopo di che assicuriamoci che funzioni correttamente.

Conclusioni

L'utilizzo di variabili o costanti per definire un pin è fondamentalmente una decisione soggettiva. Ad esempio, se il pulsante sarà permanentemente connesso al pin 4, utilizziamo questo numero come costante come abbiamo fatto in questo esempio; lo salveremo sulla memoria RAM. D'altra parte, se c'è una probabilità che questo pulsante o qualsiasi altra periferica possa essere soggetto a modifiche nelle connessioni del controller, è meglio definire questa connessione come una variabile: ad esempio "**intButton= 4;**". In questo modo, se in futuro il pulsante viene collegato al pin 12, l'unica cosa che dovremo cambiare sarà l'istruzione della variabile "**intButton=12;**".

6. ESEMPIO 3: Illuminare un LED a volt

Ecco un'ultima versione del programma per il controllo di un LED con un pulsante. La prima cosa da fare è compilare il programma e registrarlo sul controller; assicurarsi che funzioni correttamente.

È anche possibile verificare che il programma funzioni correttamente rimuovendo il cavo USB e alimentando la scheda BASIC I / O Arduino con una fonte diversa, ad esempio una batteria. Tutto dovrebbe funzionare correttamente. Allora, quali sono le differenze?

- **Conclusioni**

Se studi bene il programma potrai vedere le differenze tra l'esempio 1 e l'esempio 2. Possiamo arrivare alle seguenti conclusioni:

- Non è stata utilizzata una variabile; ciò significa un risparmio sulla memoria RAM. In molti casi utilizzare variabili e costanti dipende più da ciò che ci interessa ottenere.
- I pin da utilizzare come ingressi digitali non hanno necessità di essere configurati usando **pinMode()**. Tutti i pin sono configurati automaticamente come input.
- Il risultato di una funzione può essere utilizzato come input per un'altra funzione senza passare attraverso la variabile intermedia "Valore" come negli altri esempi. Diamo un'occhiata alla seguente espressione:
  digitalWrite(6,digitalRead(4));
- C'è una funzione all'interno di un'altra. La funzione più interna è quella che viene eseguita prima: **digitalRead(4).** Il risultato viene passato alla seguente funzione **digitalWrite (,...);**

- Un singolo programma può essere scritto in diversi modi. Adesso basta preoccuparsi del fatto che il programma funzioni correttamente. Col tempo saremo in grado di ridurre la quantità di memoria FLASH e memoria dati RAM utilizzate e questo lo renderà più "**EFFICIENTE**".

È possibile fare un controllo molto semplice da soli. Compilare e registrare nuovamente ESEMPIO_4_1, ESEMPIO_4_2 e ESEMPIO_4_3. Guardiamo quanta memoria RAM e FLASH utilizziamo. Otteniamo queste informazioni dall'ambiente IDE Arduino ogni volta che finiamo di registrare un programma; esso appare in fondo. Completiamo la seguente tabella:

ESEMPIO	Memoria FLASH	Memoria RAM
ESEMPIO 1		
ESEMPIO 2		
ESEMPIO 3		

7. ESEMPIO 4: Illumination inversa di un led

L'idea è quella di utilizzare l'operatore logico **NOT** e creare un programma per illuminare il LED bianco collegato al pin 6 ogni volta che il pulsante collegato al pin 4 non è acceso.

Il corpo principale del programma è contenuto nella funzione **loop()** ed è stato evidenziato in giallo.

È molto simile all'esempio precedente. L'unica differenza è che la funzione **NOT**, rappresentata dal simbolo "**!**", inverte la lettura del pulsante collegato al pin 4.

Questo esempio potrebbe simulare una macchina con un motore che funziona sempre se non viene premuto un pulsante di emergenza.

Questo esempio (Figure 7) potrebbe simulare una macchina con un motore che funziona sempre se non viene premuto un pulsante di emergenza.

Figure 6

8. ESEMPIO 5: Comando a due pulsanti

Immaginiamo di voler controllare un allarme con due pulsanti che sono molto distanti. L'allarme dovrebbe spegnersi quando si preme uno o – **OR** - l'altro dei pulsanti (**OR** entrambi). L'allarme è simulato dal LED rosso sulla scheda di esperimento BASIC I/O collegato al pin 11. I due pulsanti sono collegati ai pin 4 e 12. Ecco il programma finale, osserviamolo attentamente alla figura 8.

Figure 7

9. ESEMPIO 6: lavoro individuale

Utilizziamo l'**ESEMPIO 6** come modello di programma o "Template". Effettuiamo le modifiche necessarie al modello per eseguire gli esercizi.

Mentre apprendiamo sempre di più la programmazione, ci renderemo conto che il controller è in grado di fare molto di più che illuminare un semplice LED ... questo è solo l'inizio!

Esercizio	Descrizione
1	Il LED ambra dovrebbe accendersi quando i pulsanti 7 e 12 vengono premuti.
2	Il LED verde si accende quando i pulsanti 12 e 8 vengono premuti ma non il pulsante 4.
3	Il LED bianco si accende quando viene premuto il pulsante 4 ma non il pulsante 7. Il LED rosso si accende quando vengono premuti i pulsanti 8 o 12.
4	Il LED verde si accende quando viene premuto il pulsante 4 ma non il pulsante 7 e quando viene premuto il pulsante 8 ma non il pulsante 12.
5	Il LED rosso si accende quando i pulsanti 4, 7 e 8 vengono premuti o, dall'altro lato, il pulsante 12.

Ricordiamo quanto segue: dobbiamo prendere nota di tutte le modifiche apportate a Arduino IDE, registrarle sul controller e assicurarci che funzioni secondo le specifiche per ogni esercizio. Completiamo la seguente tabella. Scriviamo tutte le istruzioni che abbiamo usato per la configurazione nella colonna **setup()**. Facciamo lo stesso nella colonna **loop(),** scrivendo tutte le funzioni che costituiscono il programma principale.

Completa la seguente tabella.

Esercizio	Funzione	Istruzioni del programma
1	setup()	
	loop()	
2	setup()	
	loop()	
3	setup()	
	loop()	
4	setup()	
	loop()	
5	setup()	
	loop()	

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