Progetto 2

1 Introduzione 3

1.1 Informazioni sul progetto 3

1.2 Abstract 3

1.3 Scopo 3

Analisi 4

1.4 Analisi del dominio 4

1.5 Analisi e specifica dei requisiti 4

1.6 Use case 6

1.7 Pianificazione 6

1.8 Analisi dei mezzi 6

1.8.1 Software 6

1.8.2 Hardware 6

2 Progettazione 7

2.1 Design dell’architettura del sistema 7

2.2 Design dei dati e database 7

2.3 Design delle interfacce 7

2.4 Design procedurale 7

3 Implementazione 8

4 Test 8

4.1 Protocollo di test 8

4.2 Risultati test 9

4.3 Mancanze/limitazioni conosciute 9

5 Consuntivo 9

6 Conclusioni 9

6.1 Sviluppi futuri 9

6.2 Considerazioni personali 9

7 Bibliografia 9

7.1 Bibliografia per articoli di riviste: 9

7.2 Bibliografia per libri 9

7.3 Sitografia 9

8 Allegati 10

# Introduzione

## Informazioni sul progetto

Autore: Matan Davidi e Filippo Finke

Scuola: Arti e Mestieri Trevano

Classe: I3AA

Anno scolastico: 2018/19

Sezione: Informatica

Materia: Modulo 306

Docenti responsabili: Adriano Barchi, Luca Muggiasca, Francesco Mussi, Elisa Nannini, Massimo Sartori

Data di inizio: 14.11.2018

Data di consegna: 08.02.2018

## Abstract

How many times has a programmer decided to learn a new language only to be discouraged for whatever reason at the beginning of the road? How many people may actually be interested in learning to program but are afraid because it looks too complicated?

This document contains the technical documentation of a user-friendly library for new programmers to help them get comfortable with programming in Arduino, a programming language based on C++, at their own pace.

Please note that this document contains the initial analysis of the project and of the current situation, the hardware and software that were used to implement it, the design of the library and associated circuits, an explanation of the code contained in the library and of the realization of the associated electronic circuits and the test cases that were run to ensure the correct functioning of the library.

## Scopo

Creare una libreria di codice utilizzabile tramite un modello base di [Digispark USB Development Board](http://digistump.com/products/1) per avvicinare nuovi utenti, per esempio studenti delle scuole medie senza previe conoscenze di programmazione, al mondo dell’informatica e dell’elettronica.

## Analisi

## Analisi del dominio

Adesso come adesso, prima della realizzazione del nostro progetto, la programmazione in Arduino implica come prerequisiti delle conoscenze base di programmazione in linguaggi C-like e di montaggio di circuiti elettronici. Questo rischia di allontanare i nuovi utenti a questo mondo che combina programmazione con utilità pratica. La nostra libreria è progettata per aiutare le persone che si interfacciano agli Arduino oppure alla programmazione per la prima volta senza dover scrivere troppo codice, in modo da potersi concentrare sulla comprensione di quello che si scrive.

Idealmente, questa libreria è stata pensata per essere utilizzata da studenti delle scuole medie, che non posseggono alcuna conoscenza né di programmazione né di elettronica, che vengono a fare una giornata informativa alla Scuola Arti e Mestieri di Trevano in modo che possano portare a casa un lavoro in cui sia il montaggio del circuito elettronico sia la programmazione della logica di funzionamento siano state fatte da loro.

## Analisi e specifica dei requisiti

|  |  |
| --- | --- |
| **ID: REQ-001** | |
| Nome | Libreria Arduino |
| Priorità | 1 |
| Versione | 1.0 |
| Note |  |
| *Sotto-requisiti* | |
| 001 | Bisogna realizzare una libreria compatibile con il linguaggio Arduino |
| 002 | La libreria deve essere realizzata in linguaggio C++ |

|  |  |
| --- | --- |
| **ID: REQ-002** | |
| Nome | LedLib |
| Priorità | 1 |
| Versione | 1.0 |
| Note |  |
| *Sotto-requisiti* | |
| 001 | È necessario implementare una libreria per controllare lo stato di un LED |
| 002 | La libreria deve implementare un metodo che ottiene lo stato del LED associato |
| 003 | La libreria deve implementare un metodo che ottiene lo stato analogico del LED associato |
| 004 | La libreria deve implementare un metodo che imposta lo stato del LED associato a un valore definito |
| 005 | La libreria deve implementare un metodo che imposta lo stato del LED associato a ALTO |
| 006 | La libreria deve implementare un metodo che imposta lo stato del LED associato a BASSO |
| 007 | La libreria deve implementare un metodo che inverte lo stato del LED associato |
| 008 | La libreria deve implementare un metodo che imposta lo stato del LED associato a un valore analogico definito |

|  |  |
| --- | --- |
| **ID: REQ-003** | |
| Nome | ButtonLib |
| Priorità | 1 |
| Versione | 1.0 |
| Note |  |
| *Sotto-requisiti* | |
| 001 | È necessario implementare una libreria per controllare lo stato di un bottone |
| 002 | La libreria deve implementare un metodo che ottiene lo stato del bottone associato |

|  |  |
| --- | --- |
| **ID: REQ-004** | |
| Nome | PhotocellLib |
| Priorità | 1 |
| Versione | 1.0 |
| Note |  |
| *Sotto-requisiti* | |
| 001 | È necessario implementare una libreria per controllare lo stato di una fotocellula |
| 002 | La libreria deve implementare un metodo che ottiene lo stato della fotocellula associata |

|  |  |
| --- | --- |
| **ID: REQ-005** | |
| Nome | LED – Bottone |
| Priorità | 2 |
| Versione | 1.0 |
| Note |  |
| *Sotto-requisiti* | |
| 001 | È necessario realizzare tre circuiti di esempio che contengano una combinazione del LED e del bottone |
| 002 | Uno dei circuiti da realizzare deve accendere il LED quando viene premuto il bottone |
| 003 | Uno dei circuiti da realizzare deve invertire lo stato del LED quando viene premuto il bottone |
| 004 | Uno dei circuiti da realizzare deve invertire continuamente lo stato del LED velocemente da quando viene premuto il bottone fino a quando non viene rilasciato |

|  |  |
| --- | --- |
| **ID: REQ-006** | |
| Nome | LED – Fotocellula |
| Priorità | 2 |
| Versione | 1.0 |
| Note |  |
| *Sotto-requisiti* | |
| 001 | È necessario realizzare tre circuiti di esempio che contengano una combinazione della fotocellula e del LED |
| 002 | Uno dei circuiti deve accendere o spegnere il LED in base al valore rilevato dalla fotocellula: se il valore è al di sopra di una soglia il LED viene acceso; se il valore è al di sotto di una soglia il LED viene spento |
| 003 | Uno dei circuiti deve regolare l’intensità del LED in base al valore rilevato dalla fotocellula: più è alta la luminosità rilevata, maggiore è l’intensità del LED. |
| 004 | Uno dei circuiti deve regolare la frequenza di lampeggiamento del LED in modo inversamente proporzionale al valore rilevato dalla fotocellula: più è alta la luminosità rilevata, minore è la frequenza di lampeggiamento del LED. |

|  |  |
| --- | --- |
| **ID: REQ-007** | |
| Nome | LCD – Display a segmenti liquidi |
| Priorità | 2 |
| Versione | 1.0 |
| Note |  |
| *Sotto-requisiti* | |
| 001 | È necessario realizzare tre circuiti di esempio che contengano il display LCD |
| 002 | Uno dei circuiti deve stampare sullo schermo LCD la scritta “Hello World”. |
| 003 | Uno dei circuiti deve stampare del testo all’interno del display LCD e mostrare il lampeggiamento del cursore, attivandolo e disattivandolo ad intervalli regolari. |
| 004 | Uno dei circuiti da realizzare deve stampare del testo sullo schermo LCD e dargli un effetto di scorrimento attraverso il display a segmenti liquidi. |

## 

## Pianificazione

|  |
| --- |
| Figura 1: Diagramma di Gantt utilizzato per la pianificazione.  La pianificazione si divide in 5 fasi distinte: Analisi, Progettazione, Implementazione, Testing e Consegna; ognuna delle quali si suddivide nuovamente in attività. Analisi   Figura 2: Ingrandimento della pianificazione della fase di analisi  In questa fase ricadono tutte le attività preliminari che servono per capire la situazione attuale e i requisiti del cliente sotto ogni aspetto. Dopodiché vengono stilati i requisiti e viene effettuata l’analisi di costi e benefici per definire se vale la pena lavorare al progetto. In questo progetto sono state svolte 5 attività:   * L’analisi del dominio, dove è stata analizzata la situazione corrente prima della realizzazione del progetto * L’intervista con il cliente, grazie alla quale è stato possibile capire i requisiti da seguire meticolosamente durante la realizzazione del progetto * La risoluzione dei dubbi, in cui abbiamo potuto smussare tutti gli angoli dei requisiti grazie alle domande poste al cliente in modo da implementare il progetto esattamente come vuole * L’analisi dei requisiti, che porta a stilare i requisiti che è possibile vedere nel capitolo Analisi e specifica dei requisiti * L’analisi dei costi e dei benefici, che permette di valutare qualora valga la pena o meno svolgere il progetto  Progettazione   Figura 3: Ingrandimento della pianificazione della fase di progettazione  All’interno di questa fase si trovano le attività che definiscono ogni aspetto del progetto prima dell’implementazione:   * La realizzazione del diagramma di Gantt, che permette una suddivisione visiva chiara del tempo e dei costi per ogni aspetto del progetto * La progettazione delle tre librerie per gli attuatori definiti nei requisiti: ButtonLib, LedLib e PhotocellLib, con funzionalità e metodi specifici, e dei tre circuiti di esempio per ognuna e realizzazione dei diagrammi UML per le classi da implementare. Non viene contata la libreria per il display LCD perché una libreria uguale esiste già.  Implementazione   Figura 4: Ingrandimento della pianificazione della fase di implementazione  Dentro questa fase vi sono tutte le attività relative all’implementazione del progetto esattamente come descritto nella progettazione:   * La creazione delle tre librerie per gli attuatori: LED, Interruttore e Fotocellula. Non viene contata la libreria per il display LCD perché una libreria uguale esiste già.  Testing   Figura 5: Ingrandimento della pianificazione della fase di testing  Nella fase di testing ricadono tutte le verifiche di funzionamento effettuate come descritto nel capitolo 4, Test:   * I test per ogni circuito realizzato: il primo per la coppia di attuatori LED – Bottone, il secondo per la coppia LED – Fotocellula e il terzo per il display a cristalli liquidi (LCD)  Consegna   Figura 6: Ingrandimento della pianificazione della fase di consegna  All’interno dell’ultima fase, quella di consegna, si trovano le attività di preparazione per la consegna del progetto implementato al cliente:   * La realizzazione di una guida di utilizzo per ogni circuito realizzato in modo che ne sia spiegato il funzionamento * La documentazione delle singole librerie per far capire cosa fa ogni membro di ogni classe * La documentazione del progetto nel suo intero per sapere come è stato realizzato e cosa contiene * La realizzazione della presentazione del progetto |

## 

## Analisi dei mezzi

### Software

Il progetto è stato sviluppato su un sistema operativo Windows 10 Home a 64 bit versione 10.0.17134 build 17134 e macOS Mojave 10.14.1 utilizzando il seguente software:

* Arduino 1.8.7
* Atom 1.32.2
* Fritzing 0.9.3
* GanttProject 2.8.9
* GitHub Desktop 1.5.0
* Google Chrome 70.0.3538.110
* Microsoft Visio 2010 14.0.4756.1000
* Microsoft Visual Studio Code 1.29.1
* Microsoft Word 16.0.10730.20102
* Mozilla Firefox 63.0.3
* SourceTree 3.0.8

Le librerie utilizzate comprendono:

* Arduino (Arduino.h), versione incorporata nell’IDE Arduino 1.8.5 per il linguaggio C++, che permette di scrivere codice in C++ utilizzando i metodi e le funzioni di Arduino
* LiquidCrystal\_I2C (LiquidCrystal\_I2C.h) 1.5.8A utilizzata per la gestione del display LCD.
* TinyWireM (TinyWireM.h) 1.0.1 utilizzata per interfacciarsi con il bus I2C.

### Hardware

* HP Pavilion - 15-au147nz:
  + Monitor da 15.6 pollici 1920x1080
  + Intel® Core™ i7-7500U 2.7 GHz Dual-core
  + 16 GB RAM
  + NVIDIA® GeForce® 940MX
  + Intel® HD Graphics 620
* MacBook Pro 2018 13”:
  + Monitor da 13 pollici 2560×1600
  + Intel® Core™ i5-8259U 2.3 Ghz Quad-core
  + 8 GB RAM 2133 MHz LPDDR3
  + Intel Iris Plus Graphics 655 1536 MB
* Digispark USB Development Board ([specifiche](http://digistump.com/products/1" \l "product_description))
  + Alimentazione tramite USB o fonte esterna - 5v o 7-35v (12v o meno consigliato, selezione automatica)
  + Regolatore da 500mA e 5V incorporato
  + USB incorporato
  + 6 Pin I/O (2 vengono utilizzati per USB solo se il programma comunica attivamente tramite USB, altrimenti è possibile utilizzare tutti e 6 anche se si sta programmando via USB)
  + Memoria Flash da 8k (circa 6k dopo il bootloader)
  + Pin di I2C e SPI
  + PWM su 3 pin (altri possibili con il software PWM)
  + ADC su 4 pin
  + LED di alimentazione e LED di stato/test

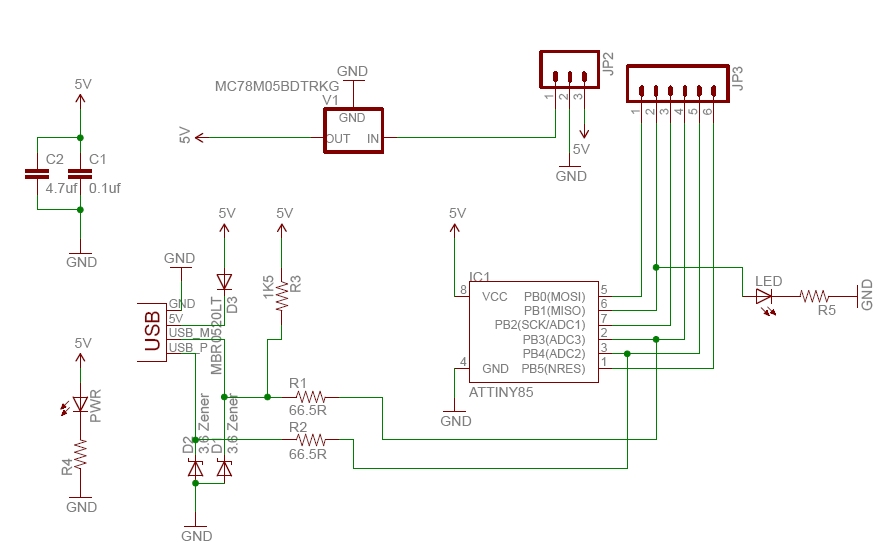


Figura 7: schema elettrico del Digispark USB Development Board

# Progettazione

## Design dell’architettura del sistema

|  |  |
| --- | --- |
| Figura 8: Bottone - LED, schema di circuito  All’interno del circuito sono presenti quattro componenti: 1 bottone, 1 LED e 2 resistenze.  Il bottone è collegato in pull-down tramite una resistenza da 10kΩ, il suo stato viene letto attraverso il pin “P0” del micro controllore. Il LED è attaccato al pin “P1” del Digispark attraverso una resistenza da 330Ω. | Figura 9: Fotocellula - LED, schema di circuito  All’interno del circuito sono presenti quattro componenti: 1 fotocellula, 1 LED e 2 resistenze.  La fotocellula è collegata in pull-down attraverso una resistenza da 10kΩ al pin “P3”. Il LED è attaccato al pin “P1” del Digispark attraverso una resistenza da 330Ω. |
| Figura 10: Bottone - LED, schema elettrico  Schema elettrico del circuito “Bottone – LED”. | Schema elettrico del circuito “Fotocellula – LED”.  Figura 11: Fotocellula - LED, schema elettrico |
| Figura 12: LCD, schema di circuito  All’interno del circuito sono presenti due componenti:  1 display a cristalli liquidi (LCD) e 1 shift register.  Lo shift register viene alimentato dal digispark ed è  collegato ai relativi pin SCL(“P2”) e SDA(“P0”).  Mentre il display LCD è collegato interamente allo shift  register. | |
| Figura 13: LCD, schema elettrico  Schema elettrico del circuito “LCD”. | |

## Design procedurale

Classe Button (ButtonLib):

* button.cpp
* button.h
* keywords.txt

Campi:

* int \_pin, che contiene il numero del pin al quale è collegato il LED

Metodi:

* + Button(int pin), che istanzia nuovi oggetti di tipo Button, accettando come parametro il numero del pin da cui leggere lo stato del bottone
  + bool getState(), che ottiene lo stato del bottone: 1 se premuto, 0 se non premuto.

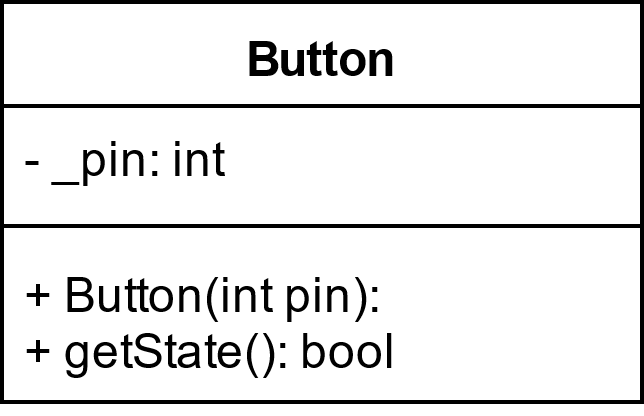


Figura 14: Schema UML della classe Button

Classe Led (LedLib):

* led.cpp
* led.h
* keywords.txt

Campi:

* int \_pin, che contiene il numero del pin al quale è collegato il LED

Metodi:

* Led(int pin), che istanzia nuovi oggetti di tipo Led, accettando come parametro il numero del pin al quale è collegato il LED
* void on(), che imposta lo stato del LED a 1: acceso
* void off(), che imposta lo stato del LED a 0: spento
* void toggle(), che inverte lo stato del LED: da acceso a spento e da spento ad acceso
* void setState(bool state), che imposta lo stato del LED al valore passato come parametro, ‘true’ lo accende e ‘false’ lo spegne
* void setAnalogState(int value), che imposta lo stato del LED con un valore analogico, da 0 a 1023
* bool getState(), che restituisce lo stato del LED, ‘true’ è acceso e ‘false’ è spento

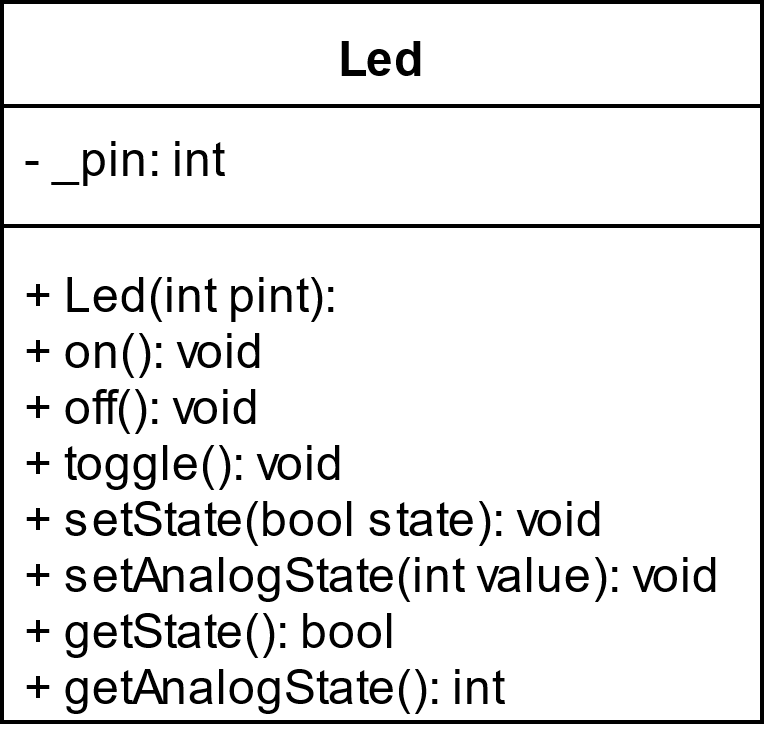


Figura 15: Schema UML della classe Led

Classe Photocell (PhotocellLib):

* photocell.cpp
* photocell.h
* keywords.txt

Campi:

* int \_pin, che contiene il numero del pin al quale è collegato il LED

Metodi:

* Photocell(int pin), che istanzia nuovi oggetti di tipo Photocell, accettando come parametro il numero del pin da cui leggere il valore restituito dalla fotocellula
* int getLux(), che restituisce il valore della luminosità rilevato dalla fotocellula, da 0 a 1023

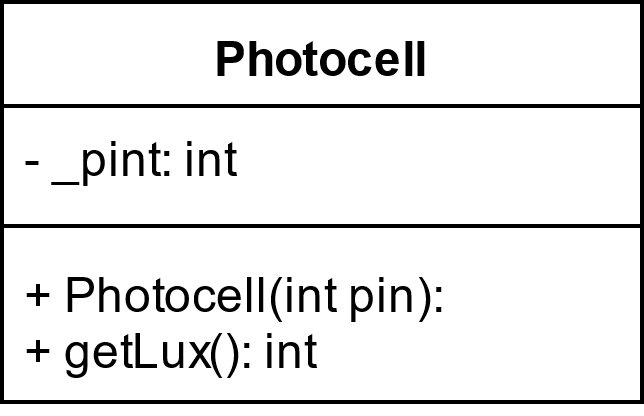


Figura 16: schema UML della classe Photocell

# Implementazione

Tutte le librerie descritte in seguito importano la libreria di Arduino, i cui metodi sono dichiarati nel file header ‘Arduino.h’.

## ButtonLib

Questa libreria viene usata per controllare lo stato di un bottone attraverso i metodi presenti nella classe Button, quindi ogni bottone presente all’interno di un circuito dovrà corrispondere ad un’istanza di questa classe. Questa classe è composta dal file ‘button.cpp’, ‘button.h’ e ‘keywords.txt’. La libreria è composta dai seguenti metodi:

### Costruttore

Button::Button(int pin)

Il metodo costruttore Button istanzia un nuovo oggetto di tipo Button, permettendo di specificare il pin al quale è attaccato il filo che permette di leggere lo stato del bottone.

### getState

bool Button::getState()

Il metodo getState permette di ottenere lo stato del bottone. Esso ritorna, infatti, un valore booleano che può assumere il valore true quando il bottone è premuto, oppure false quando il bottone non è premuto.

Il valore viene ricavato attraverso un metodo della libreria Arduino utilizzando il seguente pezzo di codice:

**return** digitalRead(\_pin);

## LedLib

Questa libreria viene usata per controllare lo stato di un LED attraverso i metodi presenti nella classe Led, quindi ogni LED presente all’interno di un circuito dovrà corrispondere ad un’istanza di questa classe. Questa classe è composta dal file ‘led.cpp’, ‘led.h’ e ‘keywords.txt’. La libreria è composta dai seguenti metodi:

### Costruttore

Led::Led(int pin)

Il metodo costruttore Led istanzia un nuovo oggetto di tipo Led, permettendo di specificare il pin al quale è attaccato il filo che permette di leggere e/o scrivere lo stato del LED.

### on

void Led::on()

Il metodo on permette di impostare lo stato del LED a 1, che significa accendere il LED fino al momento in cui non viene spento.

Questo metodo utilizza il seguente pezzo di codice per richiamare un altro metodo della classe stessa in modo da poter accendere il LED (vedi setState):

setState(HIGH);

### off

void Led::off()

Il metodo off permette di impostare lo stato del LED a 0, che significa spegnere il LED fino al momento in cui non viene acceso.

Questo metodo utilizza il seguente pezzo di codice per richiamare un altro metodo della classe stessa in modo da poter spegnere il LED (vedi setState):

setState(LOW);

### toggle

void Led::toggle()

Il metodo toggle permette di impostare lo stato del LED al valore inverso rispetto a quello corrente, che significa accendere il LED se è correntemente spento, oppure spegnerlo se dovesse essere acceso. Questo stato viene mantenuto fino al prossimo cambiamento di stato provocato da una chiamata a uno dei metodi on, off o toggle.

Questo metodo fa utilizzo di due metodi della classe stessa per ricavare il valore corrente, invertirlo ed infine impostarlo. Il tutto viene effettuato utilizzando il seguente blocco di codice (vedi setState e getState):

setState(!getState());

### setState

void Led::setState(bool state)

Il metodo setState permette di impostare lo stato del LED al valore booleano passato come parametro state. I valori accettabili per il parametro state di questo metodo sono true per accendere il LED e false per spegnerlo.

Questo metodo utilizza un metodo predisposto dalla libreria di Arduino per poter impostare lo stato del LED:

digitalWrite(pin, state);

### setAnalogState

void Led::setAnalogState(int value)

Il metodo setAnalogState permette di impostare lo stato del LED al valore del numero intero passato come parametro value. Questo permette di regolare l’intensità della luce emanata dal LED con un valore da 0 a 255, dove 0 significa completamente spento e 255 significa acceso alla massima luminosità.

Questo metodo utilizza un metodo predisposto dalla librearia di Arduino per poter impostare un valore analogico ad un determinato pin:

analogWrite(pin, value);

### getState

bool Led::getState()

Il metodo getState permette di ottenere un valore che rappresenta lo stato del LED. Esso ritorna, infatti, un valore booleano che simboleggia lo stato del LED: se il valore ritornato è true il LED è acceso; se il valore ritornato è false il LED è spento.

Questo metodo utilizza un metodo predisposto dalla libreria di Arduino per poter ricavare lo stato di un determinato pin:

**return** digitalRead(pin);

### getAnalogState

int Led::getAnalogState()

Il metodo getAnalogState permette di ottenere un valore che rappresenta lo stato analogico del LED, che significa l’intensità della luce che emana. Esso ritorna, infatti, un valore compreso tra 0, che significa che il LED è completamente spento, e 255, che significa che il LED è acceso alla massima intensità.

Questo metodo utilizza un metodo predisposto dalla libreria di Arduino per poter ricavare il valore analogico di un determinato pin:

**return** analogRead(pin);

## PhotocellLib

Questa libreria viene usata per controllare lo stato di una resistenza fotovoltaica attraverso i metodi presenti nella classe Photocell, quindi ogni fotocellula presente all’interno di un circuito dovrà corrispondere ad un’istanza di questa classe. Questa classe è composta dal file ‘photocell.cpp’, ‘photocell.h’ e ‘keywords.txt’. La libreria è composta dai seguenti metodi:

### Costruttore

Photocell::Photocell(int pin)

Il metodo costruttore Photocell istanzia un nuovo oggetto di tipo Photocell, permettendo di specificare il pin al quale è attaccato il filo che permette di leggere lo stato della fotoresistenza.

### getLux

int Photocell::getLux()

Il metodo getLux restituisce il valore misurato dalla fotoresistenza. Infatti esso ritorna un valore intero tra 0 e 1023, dove 0 significa che non è stata rilevata alcuna luce e 1023 significa che è stata rilevata una luminosità maggiore o uguale al valore massimo rilevabile dalla fotoresistenza.  
Questo metodo utilizza un metodo predisposto dalla libreria di Arduino per poter ricavare il valore analogico di un determinato pin:

**return** analogRead(pin);

# Test

## Protocollo di test

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-001  REQ-005 | **Nome:** | Uno dei circuiti deve accendere il LED quando viene premuto il bottone |
| **Descrizione:** | Uno dei circuiti di esempio per la coppia di attuatori LED – Bottone deve permettere di premere un interruttore per accendere il LED, che si deve spegnere una volta rilasciato il pulsante. | | |
| **Prerequisiti:** | * + - Un montaggio elettrico contenente un Digispark USB collegato ad un interruttore e un LED (vedi Figura 2: Bottone - LED, schema di circuito)     - Una versione della libreria “ButtonLib” implementata nel progetto     - Una versione della libreria “LedLib” implementata nel progetto | | |
| **Procedura:** | 1. Avviare il codice di esempio “LedOnOff.ino” 2. Tenere premuto l’interruttore collegato al Digispark 3. Rilasciare l’interruttore collegato al Digispark | | |
| **Risultati attesi:** | Quando l’interruttore viene premuto, il LED si accende fino al momento in cui il pulsante non viene rilasciato, dopodiché si spegne. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-002  REQ-005 | **Nome:** | Uno dei circuiti deve invertire lo stato del LED quando viene premuto il bottone |
| **Descrizione:** | Uno dei circuiti di esempio per la coppia di attuatori LED – Bottone deve permettere di premere un interruttore per invertire lo stato LED, che si deve accendere se è spento e spegnere se è acceso. | | |
| **Prerequisiti:** | * + - Un montaggio elettrico contenente un Digispark USB collegato ad un interruttore e un LED (vedi Figura 2: Bottone - LED, schema di circuito)     - Una versione della libreria “ButtonLib” implementata nel progetto     - Una versione della libreria “LedLib” implementata nel progetto | | |
| **Procedura:** | 1. Avviare il codice di esempio “LedToggle.ino” 2. Premere l’interruttore collegato al Digispark 3. Premere nuovamente l’interruttore collegato al Digispark | | |
| **Risultati attesi:** | Quando l’interruttore viene premuto per la prima volta, il LED si accende fino al momento in cui il pulsante non viene premuto la seconda volta, dopodiché si spegne. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-003  REQ-005 | **Nome:** | Uno dei circuiti deve invertire lo stato del LED da quando viene premuto il bottone fino a quando non viene rilasciato |
| **Descrizione:** | Uno dei circuiti di esempio per la coppia di attuatori LED – Bottone deve permettere di tenere premuto un interruttore per cominciare a invertire lo stato LED velocemente fino a quando il pulsante non viene rilasciato, dopodiché il LED assumerà l’ultimo stato in cui si trova. | | |
| **Prerequisiti:** | * + - Un montaggio elettrico contenente un Digispark USB collegato ad un interruttore e un LED (vedi Figura 2: Bottone - LED, schema di circuito)     - Una versione della libreria “ButtonLib” implementata nel progetto     - Una versione della libreria “LedLib” implementata nel progetto | | |
| **Procedura:** | 1. Avviare il codice di esempio “LedFlickering.ino” 2. Premere l’interruttore collegato al Digispark 3. Rilasciare l’interruttore collegato al Digispark | | |
| **Risultati attesi:** | Quando l’interruttore viene premuto il LED comincia a cambiare rapidamente il proprio stato, alternando tra acceso e spento fino al momento in cui il pulsante non viene rilasciato, dopodiché rimane sull’ultimo valore che ha assunto prima che fosse rilasciato il pulsante. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-004  REQ-006 | **Nome:** | Uno dei circuiti deve accendere o spegnere il LED in base al valore rilevato dalla fotocellula |
| **Descrizione:** | Uno dei circuiti di esempio per la coppia di attuatori LED – Fotocellula deve impostare lo stato di un LED in base al valore rilevato da una fotocellula: quando il valore misurato scende sotto una certa soglia il LED viene spento; quando il valore misurato sale al di sopra di una certa soglia il LED viene acceso. | | |
| **Prerequisiti:** | * + - Un montaggio elettrico contenente un Digispark USB collegato ad una fotocellula e un LED (vedi Figura 3: Fotocellula - LED, schema di circuito)     - Una versione della libreria “PhotocellLib” implementata nel progetto     - Una versione della libreria “LedLib” implementata nel progetto | | |
| **Procedura:** | 1. Avviare il codice di esempio “PhotocellThreshold.ino” 2. Oscurare la fotocellula posizionandovi sopra un materiale opaco 3. Rimuovere il materiale opaco dalla fotocellula | | |
| **Risultati attesi:** | Quando la fotocellula viene oscurata il LED si spegne; quando viene rimosso il materiale opaco il LED si accende nuovamente. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-005  REQ-006 | **Nome:** | Uno dei circuiti deve regolare l’intensità del LED in base al valore rilevato dalla fotocellula. |
| **Descrizione:** | Uno dei circuiti di esempio per la coppia di attuatori LED – Fotocellula deve impostare la luminosità di un LED in base al valore rilevato da una fotocellula: maggiore è la luminosità rilevata, maggiore è la luminosità del LED. | | |
| **Prerequisiti:** | * + - Un montaggio elettrico contenente un Digispark USB collegato ad una fotocellula e un LED (vedi Figura 3: Fotocellula - LED, schema di circuito)     - Una versione della libreria “PhotocellLib” implementata nel progetto     - Una versione della libreria “LedLib” implementata nel progetto | | |
| **Procedura:** | 1. Avviare il codice di esempio “PhotocellToLEDIntensity.ino” 2. Oscurare la fotocellula posizionandovi sopra gradualmente un materiale opaco 3. Rimuovere gradualmente il materiale opaco dalla fotocellula | | |
| **Risultati attesi:** | Quando il materiale opaco oscura completamente il sensore il LED è spento, man mano che il materiale viene rimosso il LED si accende con un’intensità sempre maggiore fino a quando il corpo opaco non oscura più la fotocellula e il LED brilla alla luminosità massima. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-006  REQ-006 | **Nome:** | Uno dei circuiti deve regolare la frequenza di lampeggiamento del LED in modo inversamente proporzionale al valore rilevato dalla fotocellula |
| **Descrizione:** | Uno dei circuiti di esempio per la coppia di attuatori LED – Fotocellula deve impostare la frequenza di lampeggiamento di un LED in modo inversamente proporzionale al valore rilevato dalla fotocellula: maggiore è la luminosità rilevata, minore è la velocità di sfarfallio del LED. | | |
| **Prerequisiti:** | * + - Un montaggio elettrico contenente un Digispark USB collegato ad una fotocellula e un LED (vedi Figura 3: Fotocellula - LED, schema di circuito)     - Una versione della libreria “PhotocellLib” implementata nel progetto     - Una versione della libreria “LedLib” implementata nel progetto | | |
| **Procedura:** | 1. Avviare il codice di esempio “PhotocellDelay.ino” 2. Oscurare la fotocellula posizionandovi sopra gradualmente un materiale opaco 3. Rimuovere gradualmente il materiale opaco dalla fotocellula | | |
| **Risultati attesi:** | Quando il materiale opaco oscura completamente il sensore il LED rimane sull’ultimo stato che ha assunto, man mano che il materiale viene rimosso il LED si comincia a lampeggiare con una frequenza sempre maggiore fino a quando il corpo opaco non oscura più la fotocellula e il LED lampeggia alla velocità massima. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-007  REQ-007 | **Nome:** | Uno dei circuiti deve stampare sullo schermo LCD la scritta “Hello World”. |
| **Descrizione:** | Uno dei circuiti di esempio per il display a cristalli liquidi (LCD) deve stampare la scritta “Hello, World” sul display LCD. | | |
| **Prerequisiti:** | * + - Un montaggio elettrico contenente un Digispark USB collegato ad uno Shift register e un display a cristalli liquidi (vedi Figura 6: LCD, schema di circuito)     - Una versione della libreria “LcdLib” implementata nel progetto | | |
| **Procedura:** | 1. Avviare il codice di esempio “LCDHelloWorld.ino” | | |
| **Risultati attesi:** | Il display a cristalli liquidi (LCD) mostra la scritta “Hello World”. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-008  REQ-007 | **Nome:** | Uno dei circuiti deve stampare del testo all’interno del display LCD e mostrare il lampeggiamento del cursore |
| **Descrizione:** | Uno dei circuiti di esempio per il display a cristalli liquidi (LCD) deve mostrare una stringa di testo all’interno del display LCD ed evidenziare il lampeggiamento del cursore | | |
| **Prerequisiti:** | * + - Un montaggio elettrico contenente un Digispark USB collegato ad uno Shift register e un display a cristalli liquidi (vedi Figura 6: LCD, schema di circuito)     - Una versione della libreria “LiquidCrystal\_I2C” implementata nel progetto | | |
| **Procedura:** | 1. Avviare il codice di esempio “LCDCursorBlink.ino” | | |
| **Risultati attesi:** | Il display a cristalli liquidi (LCD) mostrera la scritta “Non blinka!” con un cursore che rimane acceso per 5 secondi, dopodiché mostra il testo “Blinka!” e il cursore passa da acceso a spento a intervalli regolari. Questi due stati del display si alternano a intervalli di 5 secondi. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-009  REQ-007 | **Nome:** | Uno dei circuiti da realizzare deve stampare del testo sullo schermo LCD e dargli un effetto di scorrimento attraverso il display. |
| **Descrizione:** | Uno dei circuiti di esempio per il display a cristalli liquidi (LCD) deve stampare del testo sul display dandogli un effetto di scorrimento lungo la sua superficie da un lato verso l’altro. | | |
| **Prerequisiti:** | * + - Un montaggio elettrico contenente un Digispark USB collegato ad uno Shift register e un display a cristalli liquidi (vedi Figura 6: LCD, schema di circuito)     - Una versione della libreria “LiquidCrystal\_I2C” implementata nel progetto | | |
| **Procedura:** | 1. Avviare il codice di esempio “LCDScroll.ino” | | |
| **Risultati attesi:** | Il display a cristalli liquidi (LCD) mostra la scritta “Hello, World” che scorre da un lato del display verso l’altro tornando al lato inziale quando esce dal LCD. | | |

## Risultati test

|  |  |
| --- | --- |
| **Test case** | **Risultato** |
| TC-001 | Quando l’interruttore viene premuto, il LED si accende fino al momento in cui il pulsante non viene rilasciato, dopodiché si spegne. |
| TC-002 | Quando l’interruttore viene premuto per la prima volta, il LED si accende fino al momento in cui il pulsante non viene premuto la seconda volta, dopodiché si spegne. |
| TC-003 | Quando l’interruttore viene premuto il LED comincia a cambiare rapidamente il proprio stato, alternando tra acceso e spento fino al momento in cui il pulsante non viene rilasciato, dopodiché rimane sull’ultimo valore che ha assunto prima che fosse rilasciato il pulsante. |
| TC-004 | Quando la fotocellula viene oscurata il LED si spegne; quando viene rimosso il materiale opaco il LED si accende nuovamente. |
| TC-005 | Quando il materiale opaco oscura completamente il sensore il LED è spento, man mano che il materiale viene rimosso il LED si accende con un’intensità sempre maggiore fino a quando il corpo opaco non oscurerà più la fotocellula e il LED brilla alla luminosità massima. |
| TC-006 | Quando il materiale opaco oscura completamente il sensore il LED rimane sull’ultimo stato che ha assunto, man mano che il materiale viene rimosso il LED si comincia a lampeggiare con una frequenza sempre maggiore fino a quando il corpo opaco non oscurerà più la fotocellula e il LED lampeggerà alla velocità massima. |
| TC-007 | Il display a cristalli liquidi (LCD) mostra la scritta “Hello World”. |
| TC-008 | Il display a cristalli liquidi (LCD) mostrerà la scritta “Non blinka!” con un cursore che rimane acceso per 5 secondi, dopodiché mostrerà il testo “Blinka!” e il cursore passera da acceso a spento a intervalli regolari. Questi due stati del display si alternano a intervalli di 5 secondi. |
| TC-009 | Il display a cristalli liquidi (LCD) mostrerà la scritta “Hello, World” che scorre da un lato del display verso l’altro tornando al lato inziale quando esce dal LCD. |

# Consuntivo

Non abbiamo riscontrato particolari problemi nella gestione del tempo, infatti durante tutto il progetto siamo stati in orario con la pianificazione e siamo riusciti a finire in tempo per la consegna senza difficoltà. Infatti il Gantt di pianificazione e quello consuntivo corrispondono.

# Conclusioni

Essendo pensata per i programmatori alla prime armi o, addirittura, che non si sono mai affacciati al mondo della programmazione, questo progetto avrà un impatto piuttosto grande in chiunque si trovi in questa posizione e potrebbe portare loro a cominciare a programmare per passione e interessarsene sempre di più. Arduino potrebbe essere un primo passo verso linguaggi più usati, come Java o C++, da progetti semplici come Hello World a applicazioni web con integrazione a un database. Oppure l’esatto contrario: grazie a queste librerie qualcuno potrebbe capire che la programmazione non gli piace e cambiare strada. I risultati sono generali e facilmente implementabili in qualsiasi tipo di progetto Arduino che faccia utilizzo di uno o più degli attuatori descritti in queste librerie.

## Sviluppi futuri

Eventuali sviluppi futuri includono l’aggiunta di nuovi circuiti di esempio per gli attuatori già presenti, come un circuito che unisca un bottone con un display a cristalli liquidi; di nuovi attuatori con conseguenti librerie e, di conseguenza, circuiti di esempio, magari il potenziometro o il sensore di ultrasuoni; e di ulteriori funzionalità per le librerie implementate, per esempio si potrebbe aggiungere un metodo alla libreria del display LCD che mostri un’animazione di scrittura “lettera per lettera” di un testo passato come parametro.

## Considerazioni personali

In questo progetto abbiamo imparato a realizzare una libreria per Arduino e le basi della programmazione a oggetti in C++, che si è rivelata molto più simile a Java e C# di quanto pensassimo.

# Bibliografia

## Sitografia

* <https://digistump.com/wiki/digispark/tutorials/connecting>  
  *digispark:tutorials:connecting [Digistump Wiki]*  
  05 Dicembre 2018
* <https://learn.adafruit.com/photocells/arduino-code>  
  Arduino Code | Photocells | Adafruit Learning System

05 Dicembre 2018

* <https://digistump.com/wiki/digispark/tutorials/lcd>

*digispark:tutorials:lcd [Digistump Wiki]*

14 Dicembre 2018

# Allegati

* Diari di lavoro