Corso di programmazione per le scuole con Arduino – PARTE 2

Nella scorsa puntata di questo corso abbiamo accennato ai concetti fondamentali della programmazione, dalle variabili alle funzioni per leggere il valore dei sensori. In questa seconda lezione approfondiamo le funzioni e gli oggetti più utili nel mondo di Arduino. Vedremo come utilizzare i sensori digitali e come accedere a pagine web per estrarre informazioni. È il concetto di API web, molto più semplice nella pratica di quanto la teoria possa far supporre, una abilità fondamentale per realizzare oggetti "intelligenti", in grado di reagire a informazioni che ricevono dall'esterno. Vedremo anche che il controllo di un servomotore non è troppo diverso da quello di un led, quindi gli studenti potranno pensare alla realizzazione di sistemi in movimento.

Naturalmente, qui presentiamo le principali caratteristiche di Arduino con degli esempi pratici, adattabili a vari ambiti, dalla scuola all'arte e il design. Gli insegnanti devono ricordare che si tratta più che altro di spunti utili soprattutto per capire la logica di Arduino e la programmazione. Naturalmente ciascuno dei nostri progetti può essere modificato, per esempio usando sensori diversi. Per imparare davvero come funzioni Arduino, infatti, la cosa migliore consiste proprio nel fare molti tentativi, in modo da prenderci la mano e soprattutto sviluppare la fantasia necessaria a risolvere creativamente i problemi che possono presentarsi.

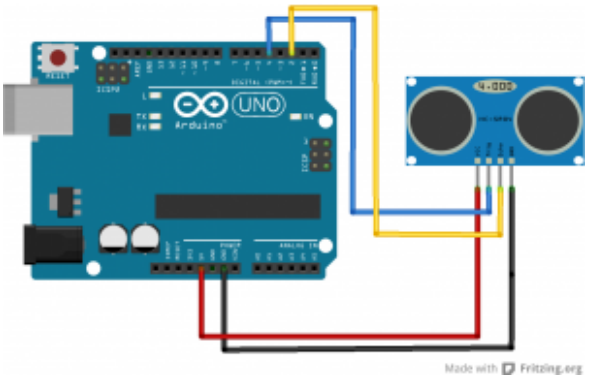
Table of Contents

3 Accendere un led in dissolvenza man mano che ci si avvicina ad Arduino

Una cosa interessante dei led è che possono essere accesi “a dissolvenza”. Cioè, invece di passare in un attimo dal completamente spento al completamente acceso, possono aumentare gradualmente luminosità fino ad accendersi completamente (e anche viceversa, possono diminuire luminosità gradualmente fino a spegnersi). Un tale comportamento è ovviamente **analogico**, non digitale, perché il digitale contempla solo lo stato acceso e quello spento, non lo stato acceso al 30% oppure acceso all’80%. Arduino non ha pin analogici di output, li ha solo di input, ma ha alcuni pin digitali che possono simulare un output analogico. Questi si chiamano **PWM** (Pulse With Modulation), e sono contraddistinti sulla scheda Arduino tra i vari pin digitali dal simbolo tilde (cioè ~). Se vogliamo poter accendere in dissolvenza un led, dobbiamo collegarlo ad uno dei pin digitali indicati dal simbolo ~, e poi assegnare a tale pin un valore compreso tra 0 e 255 utilizzando non la funzione digitalWrite, che abbiamo visto nell’esempio della puntata precedente, ma la funzione **analogWrite**. Naturalmente, per aggiungere una certa

interattività, ci serve un sensore: possiamo utilizzare il sensore **HC RS04**. Si tratta di un piccolo sensore ad ultrasuoni capace di leggere la distanza tra il sensore stesso e l'oggetto che ha di fronte (rileva oggetti tra i 2cm ed i 400cm di distanza da esso, con una precisione massima di 3mm). Il suo pin **Vcc** (primo da sinistra) va collegato al 5V di Arduino, mentre il pin **GND** (primo da destra) va collegato al GND di Arduino. Il pin **Trig** (secondo da sinistra) va collegato al pin digitale 9 di Arduino, mentre il suo pin **Echo** (secondo da destra) va collegato al pin digitale 10 di Arduino. Il codice del programma che fa ciò che vogliamo è il seguente:

Come si può ormai immaginare, il programma inizia con la solita dichiarazione delle variabili. La variabile **triggerPort** indica il pin di Arduino cui collegato il pin **Trig** del sensore, così come la **echoPort** indica il pin di Arduino cui è collegato il pin **Echo** del sensore. La variabile **led** indica il pin cui è collegato il led: ricordiamoci che deve essere uno di quelli contrassegnati dal simbolo ~ sulla scheda Arduino.



Il sensore di distanza HCSR04 ha 4 pin da collegare ad Arduino

La funzione **setup** stavolta si occupa di impostare la modalità dei tre pin digitali, chiamando per tre volte la funzione **pinMode** che abbiamo già visto. Stavolta, però, due pin (quello

del trigger del sensore e quello del led) hanno la modalità **OUTPUT**, mentre il pin dedicato all'echo del sensore ha modalità **INPUT**. Infatti, il sensore funziona emettendo degli ultrasuoni grazie al proprio pin trigger, e poi ascolta il loro eco di ritorno inviando il segnale ad Arduino tramite il pin echo. Il principio fisico alla base è che più distante è un oggetto, maggiore è il tempo necessario affinché l'eco ritorni indietro e venga rilevato dal sensore: è un sonar, come quello delle navi o dei delfini.

Prima di concludere la funzione setup, provvediamo ad aprire una comunicazione sulla **porta seriale**, così potremo leggere dal computer il dato esatto di distanza dell'oggetto.

Inizia ora la funzione loop, qui scriveremo il codice che utilizza il sensore a ultrasuoni.

Il sensore a ultrasuoni
Quando si vuole realizzare qualcosa di interattivo ma non troppo invasivo, il sensore a ultrasuoni è una delle opzioni migliori. Noi ci siamo basati sull'HCSR04, uno dei più comuni ed economici, si può trovare su Ebay ed AliExpress per più o meno 2 euro. Misurando variazioni nella distanza il sensore può anche permetterci di capire se la persona posizionata davanti ad esso si stia muovendo, quindi possiamo sfruttarlo anche come sensore di movimento.

L'impulso ad ultrasuoni

Abbiamo detto che il sonar funziona inviando un impulso a ultrasuoni e ascoltandone l'eco. Dobbiamo quindi innanzitutto

inviare un impulso: e lo faremo utilizzando la funzione **digitalWrite** per accendere brevemente il pin cui è collegato il trigger del sensore.

Prima di tutto il sensore deve essere spento, quindi impostiamo il pin al valore **LOW**, ovvero 0 Volt. Poi accendiamo il sensore in modo che emetta un suono (nelle frequenze degli ultrasuoni) impostando il pin su **HIGH**, ovvero dandogli 5 Volt. Ci basta un impulso molto breve, quindi dopo avere atteso 10 microsecondi (cioè 0,01 millesimi di secondo) grazie alla funzione **delayMicroseconds**, possiamo spegnere di nuovo il pin che si occupa di far emettere il suono al sensore portandolo di nuovo agli 0 Volt del valore **LOW**. Insomma, si spegne l'emettitore, lo si accende per una frazione di secondo, e lo si spegne: questo è un impulso.

Ora dobbiamo misurare quanto tempo passa prima che arrivi l'eco dell'impulso. Possiamo farlo utilizzando la funzione **pulseIn**, specificando che deve rimanere in attesa sul pin echo del sensore finché non arriverà un impulso di tipo **HIGH**, ovvero un impulso da 5V (impulsi con voltaggio inferiore potrebbero essere dovuti a normali disturbi nel segnale). La funzione ci fornisce il tempo in millisecondi, quindi per esempio 3 secondi saranno rappresentati dal numero 3000. Possiamo registrare questo numero nella variabile **durata**, che dichiariamo in questa stessa riga di codice. La variabile è dichiarata come tipo **long**: si tratta di un numero intero molto lungo. Infatti, su un Arduino una variabile di tipo **int** arriva al massimo a contenere il numero **32768**, mentre un numero intero di tipo long può arrivare a **2147483647**. Visto che la durata dell'eco può essere un numero molto grande, se utilizzassimo una variabile di tipo int otterremmo un errore. Ci si potrebbe chiedere: perché allora non si utilizza direttamente il tipo long per tutte le variabili? Il fatto è

che la memoria di Arduino è molto limitata, quindi è meglio non intasarla senza motivo, e usare **long** al posto di **int** soltanto quando è davvero necessario.

Come abbiamo visto per il sensore di temperatura, anche in questo caso serve una semplice formula matematica per ottenere la distanza (che poi è il dato che ci interessa davvero) sulla base della durata dell'impulso. Si moltiplica per 0,034 e si divide per 2, come previsto dai produttori del sensore che stiamo utilizzando. E anche in questo caso la variabile **distanza** va dichiarata come tipo **long**, perché può essere un numero abbastanza grande.

Ora possiamo scrivere un messaggio sulla porta seriale, per comunicare al computer la distanza rilevata dal sensore.

Naturalmente dobbiamo tenere conto di un particolare: il sensore ha dei limiti, non può misurare la distanza di oggetti troppo lontani. Secondo i produttori, se il sensore impiega più di 38 secondi (ovvero **38000** millisecondi) per ottenere l'eco, significa che l'oggetto che si trova di fronte al sensore è troppo distante. Grazie ad un semplice **if** possiamo decidere di scrivere un messaggio che faccia sapere a chi sta controllando il monitor del computer che siamo fuori portata massima del sensore.

Continuando l'if con un **else**, possiamo dire ad Arduino che se, invece, il tempo trascorso per avere un impulso è inferiore ai 38000 millisecondi, la distanza rilevata è valida. Quindi possiamo scrivere la distanza, in centimetri, sulla porta seriale, affinché possa essere letta dal computer collegato ad Arduino tramite USB.

Mappare i numeri

È finalmente arrivato il momento di eseguire la dissolvenza sul led collegato ad Arduino. Noi vogliamo che la luminosità del led cambi in base alla durata dell'impulso (la quale è, come abbiamo detto, proporzionale alla distanza dell'oggetto più vicino). Però la variabile **durata** può avere un qualsiasi valore tra 0 e **38000**, mentre il led accetta soltanto valori di luminosità che variano tra 0 e **255**.

Poco male: abbiamo già visto la volta scorsa che per risolvere questo problema possiamo chiamare la funzione **map**, indicando la variabile che contiene il numero da tradurre e i due intervalli. Il risultato viene memorizzato in una nuova variabile di tipo **int** (del resto è comunque un numero piccolo) che chiamiamo **fadeValue**. Questa variabile conterrà, quindi, un numero da 0 a 255 a seconda della distanza del primo oggetto che si trova di fronte al sensore.

Possiamo concludere impostando il valore appena calcolato, memorizzato nella variabile **fadeValue**, come valore del pin cui è collegato il LED (identificato dalla variabile led che avevamo dichiarato all'inizio del programma). Per farlo utilizziamo la funzione **analogWrite**, che per l'appunto non si limita a scrivere "acceso" o "spento" come la **digitalWrite**, ma piuttosto permette di specificare un numero (da 0 a 255, proprio come quello appena calcolato) per indicare la luminosità desiderata del led. Ora il ciclo else è completato, lo chiudiamo con una parentesi graffa.

Prima di concludere definitivamente la funzione loop, con una ultima parentesi graffa chiusa, utilizziamo la funzione **delay**

per dire ad Arduino di aspettare **1000** millisecondi, ovvero 1 secondo, prima di ripetere la funzione (e dunque eseguire una nuova lettura della distanza con il sensore ad ultrasuoni).

Controllare anche i led da illuminazione
Arduino fornisce, tramite i suoi pin digitali, al massimo 5Volt e 0,2Watt, che sembrano pochi ma sono comunque sufficienti per accendere i classici led di segnalazione ed alcuni piccoli led da illuminazione che si trovano nei negozi di elettronica. Ma è comunque possibile utilizzare Arduino per accendere led molto più potenti, anche fino a 90Watt (ed una normale lampadina led domestica ha circa 10W), utilizzando un apposito Shield, ovvero un circuito stampato da montare sopra ad Arduino: https://www.sparkfun.com/products/10618.

4 Leggere l'ora da internet e segnalarla con una lancetta su un servomotore

In questo progetto affrontiamo due temi importanti: il movimento e internet. Arduino, infatti, può manipolare molto facilmente dei servomotori, e dunque si può utilizzare per applicazioni meccaniche di vario tipo, anche per costruire dei robot. Inoltre, Arduino può essere collegato a internet: esistono diversi metodi per farlo, a seconda della scheda che si sta utilizzando. Un Arduino Uno può essere collegato ad internet tramite lo shield Ethernet, acquistabile a parte sia in versione semplice che con PowerOverEthernet (Arduino verrebbe alimentato direttamente dal cavo ethernet). Questo è lo scenario più tipico, ed è quello su cui ci basiamo per il nostro esempio. In alternativa, si può ricorrere a un WemosD1, che è fondamentalmente un Arduino Uno con un chip wifi

integrato, abbastanza facile da programmare e economico (costa poco più di un Arduino Uno). Chiaramente, il WiFi va configurato usando le apposite funzioni indicando la password di accesso alla rete, cosa che non serve per le connessioni ethernet. Esiste anche l'ottimo Arduino Yun, che integra sia una porta ethernet che una antenna WiFi, ed ha una comoda interfaccia web per configurare la connessione, senza quindi la necessità di configurare il WiFi nel codice del proprio programma. Arduino Yun è il più semplice da utilizzare, ma è un po' costoso (circa 50 euro), mentre la coppia Arduino Uno + Ethernet shield è molto più economica (la versione ufficiale arriva al massimo a 40 euro, e si può comprare una riproduzione made in China per meno di 10 euro su AliExpress). Lo shield deve semplicemente essere montato sopra ad Arduino. Il servomotore, invece, va collegato ad Arduino in modo che il polo positivo (**rosso**) sia connesso al pin **5V**, mentre il negativo (**nero**) sia connesso al pin **GND** di Arduino. Infine, il pin del segnale del servomotore (tipicamente **bianco** o in un altro colore) va collegato ad uno dei pin digitali **PWM** di Arduino, quelli contrassegnati dal simbolo ~ che abbiamo già visto per la dissolvenza del led. Nel nostro progetto di prova, realizzeremo un orologio: per semplificare le cose, utilizzeremo il servomotore per muovere soltanto la lancetta delle ore, tralasciando quella dei minuti. Il codice è il seguente:

Sono necessarie tre librerie esterne: **SPI** ed **Ethernet** ci servono per utilizzare la connessione ethernet. Invece, **Servo** è utile per controllare il servomotore.

Proprio per utilizzare il servomotore si deve creare una variabile speciale, che chiamiamo **myservo**, di tipo **Servo**. Questa non è proprio una variabile, è un "oggetto". Gli **oggetti** sono degli elementi del programma che possono avere una serie di proprietà (variabili) e di funzioni tutte loro. Per esempio, l'oggetto **myservo** appena creato avrà tra le sue

proprietà la posizione del motore, mentre l'oggetto **client** avrà tra le sue funzioni "personali" una funzione che esegue la connessione a internet. Gli oggetti servono a risparmiare tempo e rendere la programmazione più intuitiva: lo vedremo tra poco.

Una scheda Arduino Uno con lo shield ethernet montato sopra di essa

La pagina web da leggere

Dichiariamo due variabili che ci permettono di impostare il pin cui è collegato il segnale del servomotore, ed il MAC dell'Ethernet shield. Ogni dispositivo ethernet ha infatti un codice MAC che lo identifica: possiamo scegliere qualsiasi cosa, anche se in teoria dovremmo indicare quello che è stampato sull'ethernet shield.

È importante non avere nella propria rete locale due dispositivi con lo stesso codice MAC, altrimenti il router non riesce a distinguerli. Quindi basta cambiare un numero (esadecimale) qualsiasi nel codice dei programmi che si scrivono.

Ora specifichiamo due informazioni: il **nome del server** che

vogliamo contattare, e la **pagina** che vogliamo leggere da esso. Per far leggere ad Arduino la pagina web http://codice-sorgente.it/UTCTime.php?zone=Europe/Rome, dobbiamo dividere il nome del server dal percorso della pagina, perché questo è richiesto dal protocollo HTTP. Questa pagina è un classico esempio di **API** web, ovvero una pagina web messa a disposizione di programmatori per ottenere informazioni varie. In particolare, questa pagina ci fornisce in tempo reale la data e l'ora per il fuso orario che abbiamo selezionato: nell'esempio scegliamo Roma, ma potremmo indicare Atene scrivendo Europe/Athens.

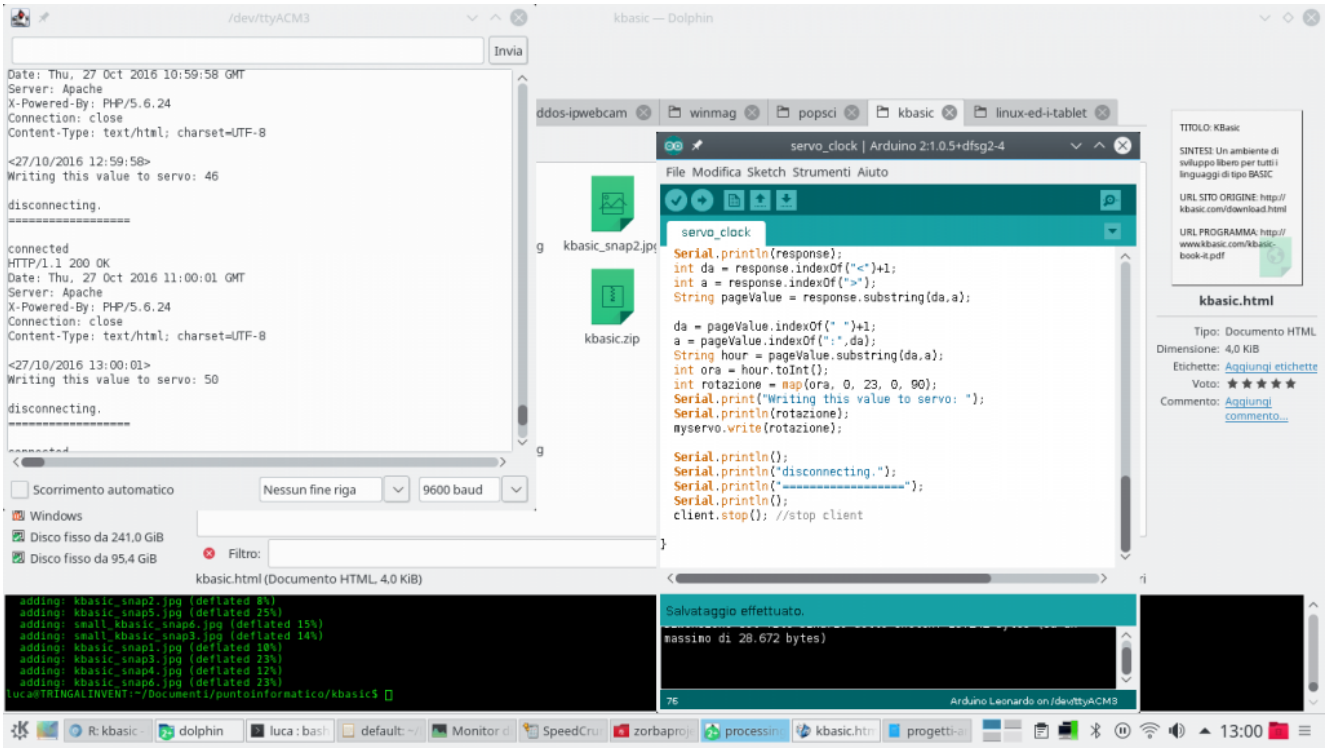
Altre API web

Abbiamo introdotto il concetto di API web, ovvero di una pagina web che contiene un semplice testo a disposizione dei programmatori. Nel nostro esempio ne utilizziamo una che fornisce l'ora attuale, ma esistono pagine web simili per qualsiasi cosa: uno tra i fornitori più importanti è Google, che offre la possibilità di eseguire ricerche con il suo motore, sfruttare i servizi di Maps, addirittura inviare email. Ma anche Twitter offre API con cui leggere od inviare tweet. Ne esistono migliaia, ed il sito migliore per trovarle è https://www.publicapis.com/. Molte di queste API richiedono un nome utente, che in genere è gratuito ma necessita di una iscrizione al sito web.

Se volete provare a realizzare una vostra pagina web che contenga l'ora attuale, come quella che sfruttiamo nel nostro esempio, dovete soltanto copiare questo file PHP:
https://pastebin.com/JeMf8p84
sul vostro server web (per esempio, Altervista ed Aruba supportano PHP).

La solita funzione **setup** comincia aprendo una comunicazione sulla porta seriale, così potremo leggere i messaggi di Arduino dal nostro computer, se vogliamo. Poi inizia con una

condizione **if**: questa ci permette di tentare la connessione al router, utilizzando la funzione **Ethernet.begin**, che richiede come argomento il codice **MAC** che avevamo scritto poco fa.



Con il monitor seriale di Arduino IDE possiamo leggere i messaggi che Arduino invia al nostro computer

La funzione **Ethernet.begin** fornisce una risposta numerica: il numero indica lo stato attuale della connessione, e se lo stato è **0**, significa che non c'è connessione. Quindi l'istruzione **if** può riconoscere questo numero e, nel caso sia **0**, prendere provvedimenti. In particolare, oltre a scrivere un messaggio di errore sulla **porta seriale**, così che si possa leggere dal computer collegato ad Arduino, se non c'è una connessione ethernet viene iniziato un **ciclo while infinito**: questo ciclo dura in eterno, impedendo qualsiasi altra operazione ad Arduino. Lo facciamo per evitare che il programma possa andare avanti se non c'è una connessione. Praticamente, Arduino rimane in attesa di essere spento. Un ciclo è una porzione di codice che viene eseguita più volte: nel caso di un ciclo while (che significa "mentre") il codice contenuto nel ciclo viene ripetuto finché la condizione posta

è vera (**true**). In questo caso il ciclo è sempre vero, perché come condizione abbiamo indicato proprio il valore **true**, e non contiene nessun codice, quindi il suo solo effetto è bloccare Arduino. Approfondiremo più avanti i cicli.

Prima di concludere la funzione **setup**, chiamiamo una funzione dell'oggetto **myservo**, che rappresenta il servomotore connesso ad Arduino. La funzione si chiama **attach** e serve a specificare che il nostro servomotore, d'ora in poi rappresentato in tutto e per tutto dal nome **myservo**, è attaccato al pin digitale di Arduino contraddistinto dal numero **3** (numero che avevamo indicato nella variabile **servopin**).

Connettersi al server web

La funzione **loop**, stavolta è molto semplice, perché deleghiamo buona parte del codice ad un'altra funzione: la funzione **sendGET**, che viene chiamata ogni 3 secondi.

In altre parole, la funzione **loop**, che viene eseguita continuamente, non fa altro che aspettare 3000 millisecondi, ovvero 3 secondi, e poi chiamare la funzione **sendGET**: è quest'ultima a fare tutto il lavoro, ed adesso dovremo scriverla.

Le parentesi graffe
Su sistemi Windows, con tastiera italiana, si può digitare una parentesi graffa premendo i tasti **AltGr + Shift + è** oppure **AltGr + Shift + +**. Infatti, i tasti **è** e **+** sono quelli che contengono anche le parentesi quadre. Quindi, premendo solo **AltGr + è** si ottiene la parentesi quadra, mentre premendo **AltGr + Shift + è** si ottiene la parentesi graffa. Su un sistema GNU/Linux, invece, si può scrivere la parentesi graffa

premendo i tasti **AltGr + 7** oppure **AltGr + 0**.

Questa è la nostra funzione **sendGET**, nella quale scriveremo il codice necessario per scaricare la pagina web con Arduino e analizzarla in modo da scoprire l'ora corrente.

Per cominciare si deve aprire una connessione HTTP con il server in cui si trova la pagina web che vogliamo scaricare. Le connessioni HTTP sono (in teoria) sempre eseguite sulla porta **80**, e il nome del server è memorizzato nella variabile **serverName** che avevamo scritto all'inizio del programma: queste due informazioni vengono consegnate alla funzione **connect** dell'oggetto **client**. Se la connessione avviene correttamente, la funzione **connect** fornisce il valore **true** (cioè vero), che viene riconosciuto da **if** e quindi si può procedere con il resto del programma. Come avevamo detto, l'oggetto **client** possiede alcune funzioni (scritte dagli autori di Arduino) che facilitano la connessione a dei server di vario tipo, e per utilizzare le varie funzioni basta utilizzare il punto (il simbolo **.**), posizionandolo dopo il nome dell'oggetto (ovvero la parola **client**).

Per esempio, un'altra funzione molto utile dell'oggetto **client** è **print** (e la sua simile **println**). Esattamente come le funzioni **print** e **println** dell'oggetto **Serial** ci permettono di inviare messaggi sul cavo USB, queste funzioni permettono l'invio di messaggi al server che abbiamo appena contattato. Per richiedere al server una pagina gli dobbiamo inviare un messaggio del tipo **GET pagina HTTP/1.0**. Visto che avevamo indicato la pagina all'inizio del programma, stiamo soltanto inserendo questa informazione nel messaggio con la funzione **print** prima di terminare il messaggio con **println**, esattamente come succede per i messaggi su cavo USB diretti al nostro computer (anch'essi vanno terminati con **println**, altrimenti

non vengono inviati).

È poi richiesto un messaggio ulteriore, che specifichi il nome del server da cui vogliamo prelevare la pagina, nella forma **Host: server**. Anche questo messaggio viene inviato come il precedente.

Ora la connessione è completata ed abbiamo richiesto la pagina al server, quindi il blocco **if** è terminato.
Naturalmente, possiamo aggiungere una condizione **else** per decidere cosa fare se la connessione al server non venisse portata a termine correttamente (ovvero se il risultato della funzione **client.connect** fosse **false**).

In questo caso basta scrivere sul monitor seriale un messaggio per l'eventuale computer collegato ad Arduino, specificando che la connessione è fallita. Il comando **return** termina immediatamente la funzione, impedendo che il programma possa proseguire se non c'è una connessione.

La risposta del server

Il blocco **if** e anche il suo **else** sono terminati. Quindi se il programma sta continuando significa che abbiamo eseguito una connessione al server e abbiamo richiesto una pagina web. Ora dobbiamo ottenere una risposta dal server, che memorizzeremo nella variabile **response**, di tipo **String**. Come abbiamo già detto, infatti, le **stringhe** sono un tipo di variabile che contiene un testo, e la risposta del server è proprio un testo.

La stringa **response** è inizialmente vuota, la riempiremo man mano.

Prima di cominciare a leggere la risposta del server dobbiamo assicurarci che ci stia dicendo qualcosa: potrebbe essere necessario qualche secondo prima che il server cominci a trasmettere. Un ciclo **while** risolve il problema: i cicli while vengono eseguiti continuamente finché una certa condizione è valida. Nel nostro caso, vogliamo due condizioni: una è che il client deve essere ancora connesso al server, cosa che possiamo verificare chiamando la funzione **client.connected**. L'altra condizione è che il client non sia disponibile, perché se non lo è significa che è occupato dalla risposta del server e quindi la potremo leggere. Possiamo sapere se il client è disponibile con la funzione **client.available**, la quale ci fornisce il valore **true** quando il client è disponibile. Però noi non vogliamo che questa condizione sia vera: la vogliamo falsa, così sapremo che il client non è disponibile. Possiamo negare la condizione utilizzando il **punto esclamativo**. In altre parole, la condizione **client.available** è vera quando il client è disponibile, mentre **!client.available** è vera quando il client non è disponibile. Abbiamo quindi le due condizioni, **client.connected** e **!client.available**. La domanda è: come possiamo unirle per ottenere una unica condizione? Semplice, basta utilizzare il simbolo **&&**, la doppia e commerciale, che rappresenta la congiunzione "e" ("and" in inglese). Quindi la condizione inserita tra le parentesi tonde del ciclo **while** è vera solo se il client è contemporaneamente connesso ma non disponibile. Significa che appena il client diventerà nuovamente disponibile oppure si disconnetterà, il ciclo while verrà fermato ed il programma potrà proseguire.

Ora, se ci troviamo in questo punto del programma, significa che il ciclo precedente si è interrotto, e il client ha quindi registrato tutta la risposta del server alla nostra richiesta di leggere la pagina web. Possiamo leggere la risposta una lettera alla volta utilizzando la funzione **client.read()**. La lettera viene memorizzata in una variabile di tipo **char**, ovvero un carattere, che viene poi aggiunta alla variabile

response. Il simbolo **+=**, infatti, dice ad Arduino che il carattere **c** va aggiunto alla fine dell'attuale stringa **response**. Se il carattere è "o" e la stringa è "cia", il risultato dell'operazione sarà la stringa "ciao".
Ovviamente, per leggere tutti i caratteri della risposta del server abbiamo bisogno di un ciclo che ripeta la lettura finché il client non ha più lettere da fornirci, un ciclo **while**. Per quante volte si deve ripetere il ciclo, cioè con quale condizione lo dobbiamo ripetere? Semplice: dobbiamo ripeterlo finché il client è connesso al server (perché è ovvio che se la connessione è caduta non ha più senso continuare a leggere), oppure finché il client è disponibile (perché se non è più disponibile alla lettura significa che la risposta del server è terminata e non c'è più niente da leggere). Stavolta abbiamo due condizioni, come nel ciclo precedente: una sarà **client.connected** e l'altra **client.available**, ma non le vogliamo necessariamente entrambe vere. Infatti, ci basta che una delle due sia vera per continuare a leggere. Insomma, l'una oppure l'altra. E il simbolo per esprimere la congiunzione "oppure" è **||**, cioè la doppia pipe (il simbolo che sulle tastiere italiane si trova sopra a \).

Se anche questo ciclo while è terminato, significa che tutta la risposta del server è ormai contenuta nella variabile **response**, quindi possiamo inviarla all'eventuale computer connesso ad Arduino tramite cavo USB.

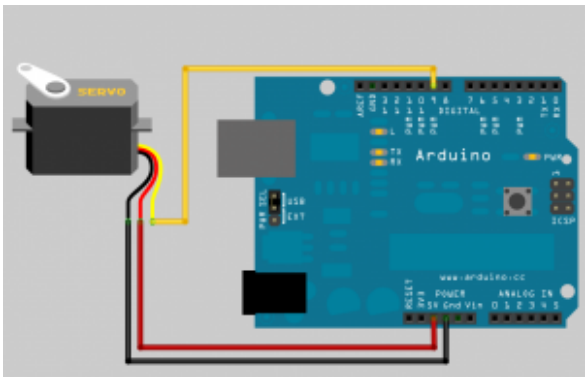
Estrarre l'informazione dalla pagina web

Ora possiamo cominciare a manipolare la risposta del server per estrarre le informazioni che ci interessano.

Innanzitutto, dobbiamo estrarre il contenuto della pagina web: se leggete la risposta completa del server, vi accorgerete che sono presenti anche molte informazioni accessorie che non ci interessano. Dobbiamo identificare il contenuto della pagina web: se provate a visitare la pagina http://codice-sorgente.it/UTCTime.php?zone=Europe/Rome noterete che il suo contenuto è qualcosa del tipo **<07/03/2019 21:33:56>**. Quindi, il suo contenuto è facilmente riconoscibile in quanto compreso tra il simbolo **<** e il simbolo **>**. Possiamo cercare il simbolo iniziale e quello finale all'interno della stringa **response** utilizzando la funzione **indexOf** tipica di ogni stringa, (funzione cui si accede con il solito simbolo **.**). Per esempio, scrivendo **response.indexOf("<")** ci viene fornita la posizione del simbolo **<** all'interno della stringa **response**. La posizione è un numero, per esempio potrebbe essere **42** se il simbolo **<** fosse il quarantaduesimo carattere dall'inizio del testo. Similmente, si può trovare la posizione del simbolo **>**, ed entrambe le posizioni si memorizzano in due variabili di tipo **int** (visto che sono numeri), che chiamiamo rispettivamente **da** e **a**. Da notare che alla prima posizione abbiamo sommato **1**, altrimenti sarebbe stato considerato anche il simbolo **<**, invece noi vogliamo tenere conto dei caratteri che si trovano dopo di esso. Utilizzando queste due posizioni, possiamo poi sfruttare la funzione **substring** della stessa **stringa** per ottenere tutto il testo compreso tra i due simboli. La funzione **substring** ci fornisce direttamente una stringa, che possiamo inserire in una nuova variabile chiamata **pageValue**.

L'attuale contenuto della stringa **pageValue** è qualcosa del tipo 07/03/2019 21:33:56. Noi siamo interessati soltanto al numero che rappresenta le ore, quindi possiamo ripetere la stessa procedura cambiando i valori delle variabili **da** ed **a**. In particolare, li dobbiamo cambiare affinché ci permettano di identificare il numero delle ore: questo è delimitato a sinistra da uno **spazio**, ed a destra dai **due punti**. Quindi,

utilizzando questi due simboli, la funzione **substring** riesce ad estrarre soltanto il numero delle ore, e inserirlo in una nuova variabile. Da notare che, anche se l'ora attuale è ai nostri occhi un numero, per il momento è ancora considerata un testo da parte di Arduino visto che finora lavoravamo con le stringhe. Possiamo trasformare questa informazione in un numero a tutti gli effetti grazie alla funzione **toInt** della stringa stessa, che traduce il testo "21" nel numero 21. Adesso, la variabile di tipo **int** (un numero intero) chiamata **ora** contiene l'orario attuale (solo le ore, non i minuti ed i secondi).



Collegare un servomotore ad Arduino è facile, basta ricordare che il pin digitale deve essere contrassegnato dal simbolo ~

Muovere il servomotore

Adesso abbiamo l'orario attuale, quindi possiamo spostare il servomotore in modo da direzionare la nostra lancetta. Per muovere un servomotore basta dargli una posizione utilizzando la sua funzione **write**. Quindi, nel nostro caso basta chiamare la funzione **myservo.write**.

Però c'è un particolare, le ore che ci fornisce il sito web

vanno da **0** a **23**, mentre le posizioni di un servomotore vanno da **0** a **180**. Infatti, un normale servomotore può ruotare di 180°: se chiamiamo la funzione **myservo.write(0)**, il motore rimarrà nella posizione iniziale, **se chiamiamo myservo.write(45)** verrà posizionato a 45° rispetto alla posizione iniziale. Il problema è: come traduciamo le ore del giorno in una serie di numeri compresa tra 0 e 180? Semplice, basta utilizzare la funzione **map**, che abbiamo già visto più volte. In questo modo le ore **23** diventerebbero **180**°, le ore **12** diventerebbero **45**°, e così via.

La funzione di mappatura

Abbiamo sempre usato la funzione map predefinita in Arduino. Ma, come esercizio di programmazione, possiamo anche pensare di scrivere una nostra versione della funzione. Che cos'è, quindi questa funzione? È una semplice proporzione, come quelle che si studiano a scuola:

Infatti, basta fare un rapporto tra i due range (quello del valore di partenza e quello del valore che si vuole ottenere). La funzione che abbiamo appena definito fornisce un numero con virgola (double), ma se preferiamo un numero intero ci basta modificare la definizione della funzione come tipo int.

Naturalmente, se vogliamo rappresentare le ore in un ciclo di 12 invece che di 24, basta dividere il numero per 12 tenendo non il quoziente ma il resto. Il resto della divisione si ottiene con l'operatore matematico %. Insomma, basta sostituire la variabile **ora** con la formula **(ora%12)**. In questo modo, le 7 rimangono le 7, mentre le 15 diventano le 3 (perché 15 diviso 12 ha il resto di 3).

Prima di completare la funzione, possiamo scrivere qualche

messaggio sul cavo USB per far sapere all'eventuale computer connesso ad Arduino che stiamo terminando la connessione al server che ha ospitato la pagina web. Poi la semplice funzione **stop** dell'oggetto **client** ci permette di concludere la connessione, e con la solita parentesi graffa chiudiamo anche la funzione, ed il programma è terminato.
Abbiamo detto che il nostro servomotore ruota di 180°: esistono anche altri servomotori modificati per poter eseguire una rotazione di 360°, chiamati **continuous rotation servo**, ma questi solitamente non si possono posizionare a un angolo preciso, piuttosto ruotano con velocità differenti. Ma sono casi particolari: un normale servomotore ruota di 180°, quindi la nostra lancetta dell'orologio si comporterebbe non tanto come un orologio normale, ma piuttosto come la lancetta di un contachilometri nelle automobili. È anche possibile cercare di modificare un servomotore da 180° per fare in modo che riesca a girare di 360°: http://www.instructables.com/id/How-to-modify-a-servo-motor-for-continuous-rotatio/?ALLSTEPS.
Ovviamente, per migliorare il nostro esempio, con un altro servomotore si può fare la stessa cosa per i minuti, ricordando però che sono 60 e non 24, quindi la funzione **map** va adattata di conseguenza.

Il codice sorgente

Potete trovare il codice sorgente dei vari programmi presentati in questo corso di introduzione alla programmazione con Arduino nel repository:
https://www.codice-sorgente.it/cgit/arduino-a-scuola.git/tree/
I file relativi a questa lezione sono, ovviamente, **dissolvenza-led-distanza-ultrasuoni.ino** e **ethernet-servomotore.ino**.

Corso di programmazione per le scuole con Arduino – PARTE 1

Quello che segue, e che proseguirà nelle prossime puntate, è un corso per assoluti principianti. È un corso per chi non ha mai scritto una linea di codice, ma vuole imparare a programmare. E come base per imparare i fondamenti della programmazione, abbiamo scelto C++ e Arduino. Perché è lo strumento più semplice da programmare e, soprattutto, concreto. Chi si avvicina alla programmazione ha bisogno di riscontri immediati, deve vedere immediatamente quale possa essere l'applicazione pratica di un concetto, altrimenti finirà per annoiarsi e rinunciare a studiare. Arduino è la soluzione perfetta per cominciare perché con un paio di righe di codice si possono ottenere risultati tangibili, e stimolare la curiosità per i capitoli successivi del corso di programmazione.

Per i docenti delle scuole (secondarie di primo e secondo grado): sentitevi pure liberi di utilizzare questa serie di articoli come "libro di testo".

Una introduzione per gli insegnanti

Nelle scuole italiane l'insegnamento dell'informatica è spesso trascurato, soprattutto per quanto riguarda la programmazione, che dovrebbe invece essere fondamentale per permettere agli

alunni lo sviluppo della logica e della capacità di risoluzione dei problemi.

Sicuramente, una parte di questo atteggiamento deriva dalla tendenza tutta italiana a considerare i computer soltanto come macchine da scrivere molto costose, invece che come strumenti davvero interattivi. è per questo motivo che dal ministero dell'istruzione, nonostante le riforme si rincorrano ogni 5 anni circa, non sono mai arrivate linee guida che suggeriscano come utilizzare i computer per l'insegnamento. E tutto viene lasciato nelle mani dei docenti, che per quanta buona volontà possano avere spesso non dispongono delle basi di informatica perché non hanno mai fatto parte della loro formazione.

Negli altri paesi non è così: la formazione digitale è considerata decisamente importante, e uno dei requisiti per diventare insegnante. Il rapporto del 2017 (http://ec.europa.eu/newsroom/document.cfm?doc_id=44390), a pagina 3, ci presenta tra le ultime posizioni sia in termini di competenze avanzate che come competenze basilari. A pagina 9 è persino possibile notare che, se consideriamo solo i lavoratori, le competenze digitali sono ancora minori, e ci fanno scendere ulteriormente in classifica, fino al terz'ultimo posto. E non è che nel nostro paese i computer siano davvero poco diffusi: tutte le scuole ne hanno a disposizione. Il problema è il modo in cui è pensata l'informatica a livello ministeriale, con i programmi che sono sempre troppo datati e troppo nozionistici, con poca attenzione all'aspetto pratico. Per cui è inevitabile che anche la formazione dei docenti stessi ne risenta, visto che la direzione verso cui si punta è sbagliata.

Nei rari istituti in cui si insegna la programmazione, solitamente scuole secondarie di secondo grado, lo si fa con strumenti e metodi obsoleti. Si utilizzano ancora Pascal e Fortran, linguaggi con cui uno studente oggi può fare ben poco di pratico e ai quali quindi non si affezionerà. Gli studenti vogliono qualcosa da poter esibire agli amici, e un programma

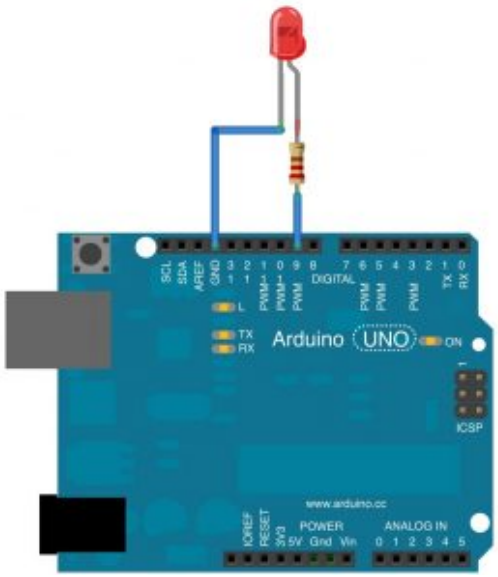
a riga di comando che somma un paio di numeri non è un gran trofeo. E non è nemmeno una cosa davvero utile, perché per la maggioranza delle operazioni che vengono descritte nei corsi di programmazione delle scuole esistono già altri programmi che risolvono il problema in modo molto più semplice e veloce. Siccome di solito le lezioni di programmazione sono tenute dai docenti di matematica, vengono descritte principalmente operazioni matematiche. Ma in realtà non ha senso: per imparare a fare la media di un elenco di numeri basta usare un foglio di calcolo, sarebbe una soluzione molto più adatta all'uso che nel mondo reale si fa dei computer. Se si vuole insegnare la programmazione bisogna prima di tutto spiegare il senso stesso della programmazione, cioè il motivo per cui valga la pena imparare come scrivere programmi. È un problema comune a molte discipline: gli studenti si annoiano sempre a realizzare riassunti nei compiti di italiano, e questo perché nessuno spiega loro qual è il senso stesso del riassunto (cioè imparare a estrarre informazioni da un testo e rielaborarle in modo proprio). Nessuno, studenti adolescenti in particolare, sarà mai ben disposto a fare qualcosa di cui non capisce l'utilità.

Procurarsi un arduino

Arduino è il minicomputer più diffuso tra gli artisti che vogliono rendere interattive le loro creazioni (è il motivo per cui venne creato in primo luogo), tra gli appassionati di modellismo che realizzano droni, ed anche tra i progettisti di dispositivi intelligenti (in particolare per i dispositivi Internet of Things). Ma è anche molto utilizzato nella scuole, soprattutto nei paesi del Nord Europa, per avvicinare i bambini alla programmazione. Arduino ha infatti il pregio di essere estremamente semplice da programmare, e avere tante applicazioni pratiche capaci di stimolare l'interesse dei neofiti. In Italia non è ancora molto diffuso a questo scopo, ma abbiamo pensato di proporvi alcuni progetti interessanti

con cui avvicinare alla programmazione i vostri figli e tutti gli amici che vorrebbero cominciare a scrivere codice ma si annoiano con i corsi teorici tradizionali. Del resto, una volta molti ragazzi diventavano programmatori perché attratti dalla possibilità di inventare un videogioco: oggi l'internet of things e la possibilità di costruire oggetti intelligenti, unendo la programmazione al bricolage, possono fungere da motivazione per avvicinarsi alla programmazione. Chi vuole procurarsi un Arduino originale può trovarlo su Amazon: per le scuole, la soluzione migliore è rappresentata da Farnell (https://it.farnell.com/arduino/a000066/arduino-uno-evaluation-board/dp/2075382), che permette pagamenti tramite il sistema MEPA contattando il servizio clienti per una quotazione personalizzata. In alternativa, un docente può semplicemente mettere dei fondi di tasca propria (o farli raccogliere dal rappresentante dei genitori) e comprare dei cloni di Arduino Uno su AliExpress (https://it.aliexpress.com/w/wholesale-arduino-uno-r3.html): costano mediamente 2,80 euro l'uno, quindi se si hanno 20 studenti bastano 56 euro per garantire a tutti un Arduino Uno. Tra l'altro, gli esempi che proponiamo possono essere realizzati anche usando un clone di Arduino Nano, che costa appena 1,70 euro. Così se uno studente brucia un paio di schede mandandole in corto circuito non è un grave danno economico. Se poi non si vuole dare una scheda a ciascuno studente, ma dividere la classe in gruppi di lavoro, è possibile permettere la sperimentazione sul sito TinkerCAD: https://www.tinkercad.com/circuits. Si tratta di un simulatore completamente gratuito, che permette di disegnare un circuito con la classica breadboard virtuale e testare il codice. È quindi perfetto per permettere agli studenti di provare sia il circuito che il codice del programma prima di caricarlo sul un vero Arduino e vedere come funziona. Chiaramente, oltre ad Arduino sono necessari anche alcuni componenti, come LED, sensori, buzzer, eccetera: i vari siti che abbiamo citato (Amazon, Farnell, AliExpress) offrono tutto il materiale di cui si può avere bisogno. Ci si può chiedere perché puntare su

Arduino, piuttosto che su PLC Siemens che costano decine di euro ciascuno: il fatto è che Arduino costa molto meno, ha un'ottima documentazione, e se si compra la scheda originale invece dei cloni si ha già una certificazione CE (http://web.archive.org/web/20170320180212/https://www.arduino.cc/en/Main/warranty). Questo significa che gli studenti che imparano a usare Arduino a scuola potranno anche utilizzarlo in seguito nella loro carriera lavorativa. Per quanto i PLC siano al momento più diffusi nei macchinari industriali, infatti, Arduino è certamente il modo migliore per iniziare, e molte aziende stanno cominciando a puntare soprattutto sui modelli più piccoli (Arduino Nano e Micro) per le applicazioni IoT. E se si deve cominciare a imparare la programmazione, è sempre meglio bruciare un piccolo Arduino da 2 euro, piuttosto che un PLC da 200 euro.



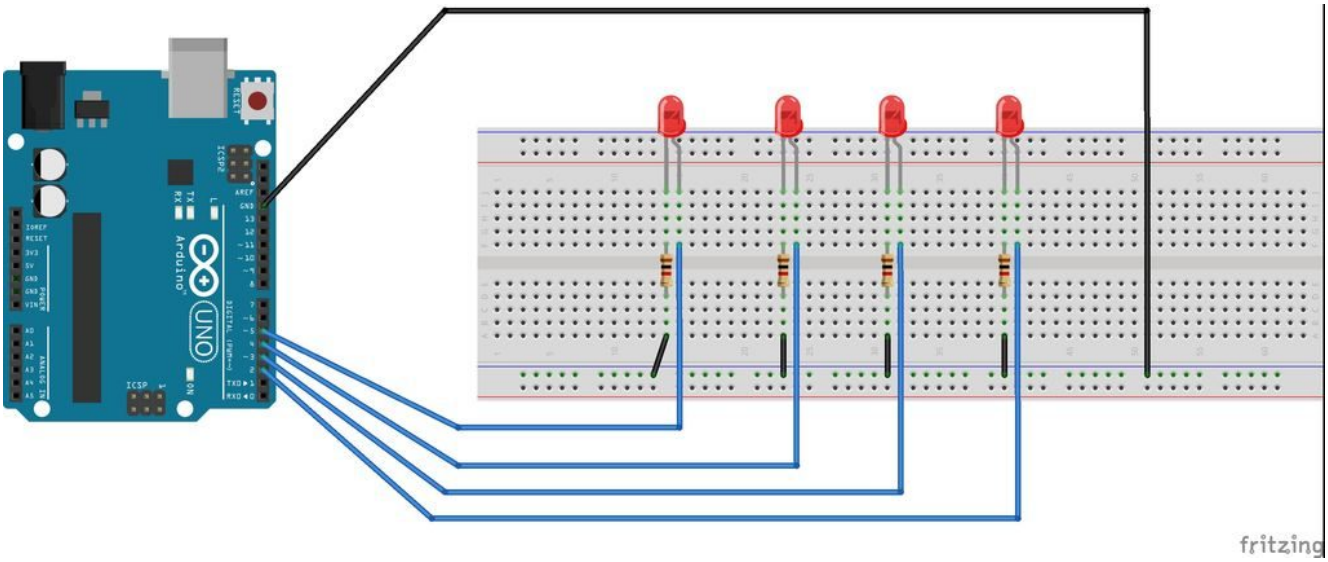
Un semplice Arduino UNO con un LED che può essere acceso o spento con un programma

Ottenuto un Arduino, lo si può programmare con l'Arduino IDE, che si può recuperare dalla pagina https://www.arduino.cc/en/Main/Software. È importante non confondersi con l'Arduino Web Editor, che è un'altra cosa.

L’Arduino IDE è un semplice programma gratuito e open source installabile su tutti i sistemi operativi, che permette di scrivere il codice dei propri programmi e caricarlo su una qualsiasi scheda Arduino (o derivate).

1 Una scala di led che si accendono in sequenza

Per cominciare, realizziamo un progetto semplice: una serie di led (Light Emissing Diode, piccole lampadine per circuiti elettrici a basso consumo) che si accenda a seconda della posizione di un potenziometro. In poche parole, avremo una rotella che si potrà girare per accendere da 1 a 5 lampadine. Un potenziometro è infatti una resistenza variabile, tipicamente una rotella od una leva che possiamo spostare per modificare la resistenza (come quelle con cui si cambia il volume sugli impianti stereo). Arduino è in grado di leggere un valore numerico in funzione della resistenza: il valore è compreso tra 0 e 1023, e noi possiamo scrivere un programma che riconosce questo numero ed in base alla sua posizione nell’intervallo 0-1023 accende 5 led.



Con una breadboard è facile collegare tanti led senza bisogno di saldare nulla

Per esempio, significa che se il potenziometro è a 0 tutti i led saranno spenti, se è a 1023 saranno tutti accesi, e se è a 200 soltanto due led saranno accesi. Per poter leggere il numero, il potenziometro va collegato ad uno dei pin analogici, mentre i led devono essere collegati a dei pin digitali, così potremo tenerli accesi o spenti. Il pin di sinistra del potenziometro va collegato al GND di Arduino, il pin centrale va connesso ad uno dei pin analogici, ed il pin di destra va connesso ai 5V di Arduino. Il pin lungo dei led va collegato ad uno dei pin digitali di Arduino, mentre il pin corto di ogni led va connesso al GND di Arduino. Il codice, che possiamo scrivere direttamente nell'Arduino IDE, è il seguente:

Prima di tutto definiamo alcune variabili: sono delle semplici parole a cui viene associato un valore di qualche tipo. Per esempio, una variabile di tipo **int** contiene un numero intero (mentre i numeri con la virgola sono di tipo **double**). La variabile chiamata **potPin** è quella che indica il pin (analogico) di Arduino cui abbiamo collegato il potenziometro: nell'esempio, si tratta del pin numero 2. Similmente, le varie variabili **ledPin** indicano i pin (digitali) di Arduino cui abbiamo collegato i led: ce ne sono 5 perché ciascuna di esse indica il pin di uno dei 5 led del nostro progetto. In pratica, abbiamo collegato ad Arduino 5 diversi led, dal pin 3 al 7. Il nome delle variabili è chiaramente a nostra discrezione, avremmo potuto chiamarle "arancia", "mela", e "banana", e si dichiarano sempre nella forma TIPO NOME = VALORE. L'assegnazione del primo valore non è fondamentale, ma è buona norma, e in questo caso si parla di "inizializzazione". Ora, definiamo la nostra prima funzione:

La funzione setup è una delle due funzioni standard di arduino (l'altra è loop). Tutto il codice contenuto dentro questa funzione verrà eseguito una sola volta, ovvero all'avvio di arduino (appena viene acceso). Una funzione, di fatto, non è

altro che un blocco di codice, un po’ come un capitolo all’interno di un libro. Ce ne sono molte già pronte all’uso, il cui codice è già stato scritto da qualcun altro, ma se vogliamo possiamo anche scrivere noi delle funzioni. Quando scriviamo una funzione dobbiamo indicare il suo tipo (void è un tipo molto comune, perché significa che la funzione fa qualcosa ma poi non fornisce un risultato che possa essere registrato dentro una variabile) ed il suo nome, cominciando poi a scrivere il suo codice con la parentesi graffa aperta. I tipi delle funzioni sono gli stessi delle variabili, perché sono tipi di dato generici. Per esempio, una funzione dichiarata come **int** potrà fornire un risultato che è un numero intero. Ma per ora ci bastano le funzioni che svolgono delle operazioni senza però fornire un risultato in termini numerici. Per esempio, nella funzione **setup** del nostro programma cominciamo a scrivere questo codice:

Qui stiamo usando la funzione (che non dobbiamo scrivere noi, è integrata in Arduino) chiamata **pinMode**, ed è anch’essa una funzione **void**, cioè una che non fornisce un risultato numerico ma si limita a fare cose. La funzione ci permette di abilitare dei ping per un segnale elettrico di output. I pin digitali di Arduino possono infatti avere due diverse modalità di funzionamento: INPUT ed OUTPUT. Siccome i nostri pin sono collegati a dei led, saranno ovviamente pin di output, perché non ci servono per fornire ad Arduino dei dati, ma piuttosto per fornire una informazione da Arduino ai led stessi (cioè, dobbiamo dire ai LED se debbano rimanere accesi o spenti). Quindi utilizzando la funzione **pinMode** possiamo impostare la modalità OUTPUT a tutti i vari pin dei led connessi ad Arduino. Il bello delle funzioni è proprio questo: quando una funzione è stata scritta, poi può essere chiamata ogni volta che vogliamo, eventualmente anche con degli argomenti (cioè informazioni utili alla procedura desiderata, come in questo caso il numero del pin e la modalità). Questo ci risparmia di dover scrivere ogni volta tutto il codice, scrivendo quindi

solo una riga. Esistono molte funzioni già presenti in Arduino, quindi non abbiamo bisogno di scriverle di nostro pugno, possiamo direttamente chiamarle.

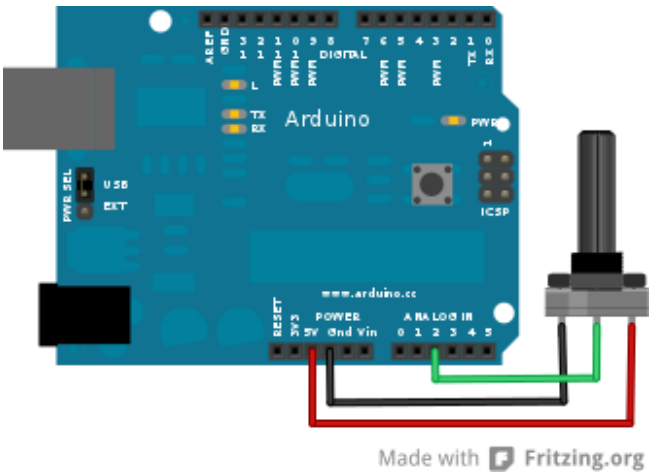
Il nostro programma è molto semplice, quindi possiamo chiudere qui la funzione setup, utilizzando una parentesi graffa chiusa. Riassumendo, la nostra funzione **setup**, eseguita automaticamente una sola volta all'accensione di Arduino, si limita a chiamare a sua volta la funzione **pinMode** per impostare i pin dei LED come output invece che input.

La funzione loop

Ora cominciamo a scrivere l'altra funzione standard di Arduino, chiamata loop.

Il codice contenuto dentro questa funzione verrà ripetuto all'infinito, finché Arduino non verrà spento. Quindi, è il codice che fa effettivamente qualcosa.

Definiamo una nuova variabile chiamata **val**, il cui valore iniziale sia 0 (per convenzione si indica sempre un valore iniziale pari a 0 per tutte le variabili, ma non è obbligatorio). Poi assegniamole il valore fornito dalla funzione **analogRead**. Questa è la funzione che legge il segnale del pin analogico che indichiamo: nel nostro caso abbiamo indicato il pin analogico cui è connesso il potenziometro, quindi la variabile val conterrà il numero, compreso tra 0 e 1023, relativo alla posizione del potenziometro.



Collegare un potenziometro a Arduino è facile

Per il momento, abbiamo solo letto la posizione del potenziometro, quindi sappiamo come è posizionata la rotella (per esempio, 0 sarà tutto a sinistra, 1023 tutto a destra, 500 circa a metà).

Ora la variabile **val** contiene un numero compreso tra 0 e 1023, ma per noi questo è scomodo: abbiamo 5 led, quindi ci farebbe comodo ricondurre il numero registrato dal potenziometro ad un intervallo compreso tra 0 e 5. Possiamo farlo chiamando la funzione **map**, che si occupa proprio di mappare (cioè tradurre) una variabile da un intervallo ad un altro. Nel caso specifico, chiediamo alla funzione map di mappare la variabile val dall'intervallo che da va 0 a 1023 sull'intervallo che va da 0 a 5. Il risultato di questa traduzione verrà memorizzato nella variabile **mappedval**, che abbiamo appena creato (con la riga int mappedval). Vale a dire che invece di avere un numero compreso tra 0 e 1023, per indicare la posizione della rotella, ne avremo uno compreso tra 0 e 5. Questa cosa si può fare perché la funzione map (che è sempre integrata in Arduino) non è una **void**, ma una **int**, e quindi ci fornisce un risultato numerico sotto forma di numero intero. Questo risultato può essere assegnato a una variabile semplicemente con il simbolo di uguaglianza.

Ora dobbiamo cominciare ad accendere i vari led: ovviamente, la loro accensione dipenderà dal valore appena ottenuto, che è memorizzato nella variabile **mappedval**. Secondo la logica del nostro progetto, il primo led si accenderà se il valore è superiore a 0, il secondo si accenderà se il valore è superiore ad 1, e così via (ricordiamo che lavoriamo con numeri interi, quindi non esistono valori come 0,3 o 1,5, vengono automaticamente arrotondati al numero intero più vicino). Possiamo ottenere questo risultato con un blocco if: i blocchi sono piccole porzioni di codice che hanno una qualche forma di controllo per la loro esecuzione. Per esempio i blocchi if (che in inglese significa il condizionale "se") sono porzioni di codice che vengono eseguite solo se una certa condizione è valida, mentre in caso contrario vengono completamente ignorate. Scrivendo **if (mappedval>0)**, tutto il codice che indichiamo dopo la parentesi graffa viene eseguito soltanto se la variabile **mappedval** ha un valore maggiore di 0.

Il codice che vogliamo eseguire in tale situazione è ovviamente molto semplice: vogliamo solo accendere il primo led. Per accendere un led con Arduino basta dare un valore alto (HIGH, ovvero 5 Volt) al pin cui il led è collegato, in questo caso ledPin1. Il valore può essere assegnato al pin chiamando la funzione **digitalWrite**.

Analogico o digitale?
La differenza fondamentale tra i due tipi di pin presenti su Arduino è che il primo può gestire segnali analogici, ed il secondo segnali digitali. Ma cosa significa questo, in breve? In pratica, un segnale analogico può avere tutta una serie di valori che, solitamente, spaziano dallo zero all'uno (ad esempio 0,4). Un segnale digitale, invece, può essere solamente 0 oppure 1. Questo non significa che uno sia meglio dell'altro: dipende tutto da ciò che dobbiamo fare. Se

vogliamo semplicemente avere (o dare) una informazione del tipo si/no un pin digitale è perfetto. Se, invece, vogliamo utilizzare una diversa scala di valori, ci conviene utilizzare un segnale analogico. Quest'ultimo è, dunque, il migliore per l'uso di sensori ambientali: esistono, comunque, dei sensori digitali, anche se sono più rari. Per quanto riguarda Arduino, i pin digitali sono 12, mentre quelli analogici sono 6. Inoltre è necessario inizializzare solo i pin digitali (scrivendo nel programma la riga pinMode(10, OUTPUT); per specificare che il pin 10 deve essere usato in output).
Chiudiamo il blocco if e continuiamo con il codice:

Uno dei lati interessanti di un blocco if è che quando lo terminiamo, con la parentesi graffa, possiamo indicare un codice alternativo con la parola **else** (che significa "altrimenti"). Quindi, se la la condizione specificata tra parentesi tonde (cioè mappedval>0) è vera, viene eseguito il codice compreso tra le prime parentesi graffe. Altrimenti, se tale condizione non è vera, viene eseguito il codice presente tra le parentesi graffe immediatamente dopo la parola **else**.

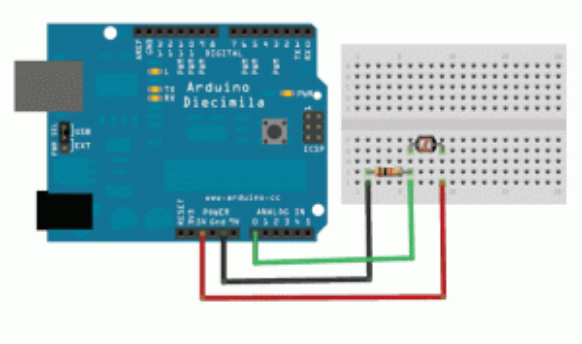
Naturalmente, noi vogliamo che se la condizione del ciclo if non è vera il led sia spento. Per farlo basta scrivere il valore basso (LOW, pari a 0 Volt) sul pin del led. Poi possiamo chiudere anche la parentesi graffa di else.

Si ripete per gli altri led

Il codice per il primo led è scritto, occupiamoci ora del secondo:

Similmente, costruiamo un ciclo if-else per il secondo led, identificato dal pin digitale ledPin2, che deve essere acceso (HIGH) solo se la variabile **mappedval** è maggiore di 1, e spento (LOW) in tutte le altre occasioni.

Si ripete lo stesso tipo di ciclo adattandolo agli altri tre led, così anch'essi possono essere accesi e spenti a seconda della posizione della rotella del potenziometro. Intuitivo, vero? Poi si può chiudere la funzione loop, che è quella di cui stavamo scrivendo il codice finora, con una parentesi graffa. Il programma è completo.



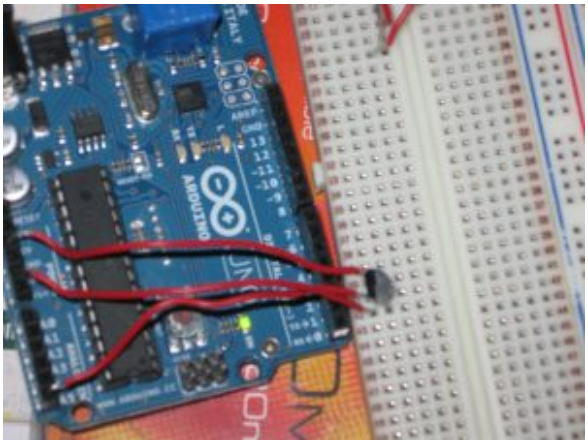
Il potenziometro può essere facilmente sostituito con una fotoresistenza

Volendo, si può anche sostituire il potenziometro con una fotoresistenza: sono dei semplici sensori di luminosità. Una fotoresistenza ha due pin perfettamente identici: uno va collegato al pin 5V di Arduino, l'altro al pin analogico 2. Quello collegato al pin analogico 2 va anche collegato, con una semplice resistenza da 10kOhm, al pin GND di Arduino. Similmente, si può sostituire il potenziometro con qualsiasi altro sensore capace di dare un segnale analogico, per esempio anche un microfono, con il quale si otterrebbe un visualizzatore di volume audio (i led si accenderebbero sempre di più man mano che si parla più forte al microfono). Serve solo un po' di fantasia.

2 Utilizzare un sensore LM35

per misurare la temperatura

Il potenziometro dell’esempio precedente è un sensore, perché permette di fornire dati ad Arduino. Esistono diversi altri sensori, che misurano informazioni relative all’ambiente: temperatura, pressione ed umidità, per esempio. Uno dei sensori più semplici da utilizzare è chiamato LM35, e misura la temperatura di una stanza. E possiamo utilizzare Arduino per leggerla, inviando il valore esatto al nostro computer attraverso il cavo USB. Cerchiamo innanzitutto di capire come funziona il sensore di temperatura LM35: contrariamente a quanto si potrebbe pensare, il sensore non ci fornisce direttamente la temperatura. Ci fornisce il solito valore compreso tra 0 e 1023, e questo valore è proporzionale alla temperatura. Con una semplice formula matematica siamo in grado di risalire alla temperatura corretta con una precisione di mezzo grado: è sufficiente moltiplicare il numero fornito dall’LM35 per circa 0,5 (esattamente per 500/1023=**0,488**, come indicato dai produttori del sensore) per ottenere la temperatura in gradi Celsius. È poi ovviamente possibile applicarle una serie di trasformazioni per cambiare scala: se per qualche motivo la volessimo in scala assoluta (i cosiddetti gradi Kelvin del sistema internazionale di misure) basterà sommare al valore ottenuto il numero 273,15. Il sensore andrà collegato ad Arduino nel seguente modo: guardando la scritta sul lato piatto, il pin a sinistra va collegato alla alimentazione a 5V, quello centrale va collegato al pin analogico 1 di Arduino, ed infine quello più a destra va collegato con il pin GND di Arduino. Se preferiamo rendere impermeabile il sensore, in modo da poterlo anche mettere a contatto con liquidi, basta ricoprirlo con della pellicola trasparente per alimenti.



Per collegare un sensore di temperatura a Arduino bastano tre cavi e una breadboard

Tenete comunque presente che, anche se il sensore può sopportare temperature che variano dai -55 °C ai 140 °C, potrebbe non essere una buona idea sottoporlo a situazioni critiche. Ad esempio, non immergetelo nella pentola dell'acqua per vedere quando bolle. Potete però appoggiarlo ad un cubetto di ghiaccio per vedere quando fonde. Il codice è relativamente semplice:

Cominciamo anche questo programma dichiarando due variabili: una è un **numero intero**, si chiama **pin**, e rappresenta il pin analogico di Arduino cui abbiamo collegato il sensore LM35, nel nostro caso il pin numero 1. L'altra è di tipo **double** perché si tratta di un **numero con virgola**, si chiama temp e rappresenterà la temperatura rilevata dal sensore (per ora la impostiamo uguale a 0.0, non dimentichiamo che con Arduino si utilizza la notazione internazionale che prevede il punto al posto della virgola per le cifre decimali). Cominciamo subito la funzione setup:

La funzione setup (eseguita una volta sola all'avvio di Arduino) non fa altro che avviare la comunicazione tramite porta seriale, con una velocità (bitrate) di 9600 baud, che è lo standard più comune. La porta seriale ovviamente è la porta

USB (Universal Serial Bus), tramite la quale potremo poi leggere i messaggi di Arduino con un apposito terminale dal nostro computer (il Monitor seriale dell'IDE di Arduino), al quale abbiamo collegato Arduino tramite il cavo USB.

La funzione loop, invece, (che è quella eseguita ripetutamente) legge il valore fornito dal sensore chiamando la funzione **analogRead**, che abbiamo già visto, moltiplicando immediatamente questo numero per il valore correttivo, ovvero 500/1023, assegnando il risultato dell'operazione alla variabile temp. In poche parole, abbiamo chiesto ad Arduino di calcolare il risultato di una semplice equazione: il numero fornito dal sensore viene moltiplicato per 500 e diviso per 1023, ed il risultato diventa il valore della variabile temp. Se al posto delle funzione **analogRead** avessimo scritto X ed al posto di temp avessimo scritto Y, l'avreste riconosciuta subito come una banale equazione di primo grado.

Ora possiamo scrivere dei messaggi al computer, il quale li riceverà sulla porta USB. I messaggi sono ovviamente dei testi (che in programmazione sono chiamati **stringhe**, e tipicamente delimitati dalle virgolette "), e possono essere inviati con la funzione **Serial.print**. Ogni messaggio è costituito da una riga, che termina quando chiamiamo la funzione **Serial.println()**. Possiamo quindi scrivere un messaggio composto dalle parole **Stanza1:** seguite dal valore della temperatura memorizzato nella variabile **temp** (che è un numero, ma viene automaticamente tradotto da Arduino in una stringa di testo) ed infine dal simbolo dei gradi Celsius. Il messaggio viene terminato con l'indicazione del termine della riga (**ln** in **println** significa **line new**, ovvero nuova riga).

L'ultima riga di codice della funzione loop dice ad Arduino di attendere 1 secondo (1000 millesimi di secondo, si ragiona sempre in millesimi di secondo) prima di eseguire nuovamente la funzione: questo significa che la temperatura verrà letta

una volta al secondo. Possiamo cambiare questo valore se vogliamo che la temperatura sia letta più spesso o più raramente.

I pin del sensore LM35 sono facili da riconoscere: guardando il lato piatto, da sinistra abbiamo il pin 5V (positivo), quello analogico, e il pin GND (negativo)

Dopo avere caricato lo sketch su Arduino, noterete che il piccolo led TX della scheda lampeggia. A questo punto si può cliccare sul menù **Strumenti/Monitor seriale** dell'Arduino IDE: apparirà una nuova finestra in stile prompt dei comandi con una serie di scritte del tipo "Stanza 1: 20.00°C", una riga per ogni secondo.

Leggere, modificare, e

scrivere i PDF

Tutti hanno bisogno di realizzare o modificare documenti in formato PDF: è tipicamente una delle funzioni più richieste all'interno di una applicazione qualsiasi. Soprattutto per quanto riguarda il desktop, mercato in cui i clienti principali sono aziende e pubblica amministrazione, che ovviamente utilizzano i computer per produrre documenti digitali proprio in formato PDF. Questo perché il Portable Document Format inventato da Adobe nel 1993, e le cui specifiche sono open source e libere da qualsiasi royalty, è lo standard universale ormai accettato da qualsiasi sistema operativo e su qualsiasi dispositivo per la trasmissione di documenti. Proprio perché il formato è utilizzabile gratuitamente da chiunque in lettura e scrittura, è stato inserito in praticamente qualsiasi programma ed è così conosciuto dal grande pubblico. Ormai chiunque sa cosa sia un PDF, e qualsiasi utente vorrà poter archiviare informazioni in questo formato. È quindi fondamentale essere in grado di scrivere programmi che possano lavorare con i PDF, altrimenti si resterà sempre un passo indietro. Il problema è che il formato PDF è abbastanza complicato da gestire, ed è quindi decisamente poco pratico realizzare un proprio sistema per leggere e scrivere questi file. Bisogna basarsi su delle apposite librerie, e ne esistono varie, anche se purtroppo spesso non sono ben documentate come l'importanza dell'argomento richiederebbe, e chi si avvicina al tema rischia di non sapere da dove iniziare. Per questo motivo, abbiamo deciso di presentarvi un metodo per leggere e uno per creare PDF multipagina, con le principali caratteristiche dei PDF/A.

Come programma di esempio, abbiamo realizzato una interfaccia grafica per gli OCR Tesseract e Cuneiform, capace di funzionare sia su Windows che su GNU/Linux e MacOSX. Per motivi di spazio e di pertinenza, non presenteremo tutto il

codice del programma ma soltanto le parti relative alla manipolazione dei PDF. Trovate comunque il link all'intero codice sorgente alla fine dell'articolo.

Le librerie Qt, sulle quali si basa non soltanto l'interfaccia grafica multipiattaforma del nostro programma di esempio, ma anche lo strumento di scrittura dei PDF, sono rilasciate con due licenze libere e una commerciale. Le due licenze libere sono GNU GPL e GNU LGPL: in entrambe i casi sono completamente gratuite, la differenza è che la prima richiede la pubblicazione dei programmi basati sulle Qt con la stessa licenza (quindi si deve fornire il codice sorgente), mentre la LGPL permette di utilizzare le librerie pur non distribuendo il codice sorgente del proprio programma. L'opzione GPL è valida per tutte le applicazioni che verranno rilasciate come software libero da programmatori amatoriali, mentre la LGPL è più indicata per aziende che non vogliono rilasciare il programma come free software. La licenza commerciale serve solo nel caso si voglia modificare il codice sorgente delle librerie Qt stesse senza pubblicare il codice delle modifiche.

Il programma è scritto in C++ con le librerie multipiattaforma Qt, delle quali ci serviremo per scrivere i PDF usando le funzioni della classe QPDFWriter. Per la lettura dei PDF, invece, utilizzeremo la libreria libera e open source Poppler, che si integra perfettamente con le librerie Qt.

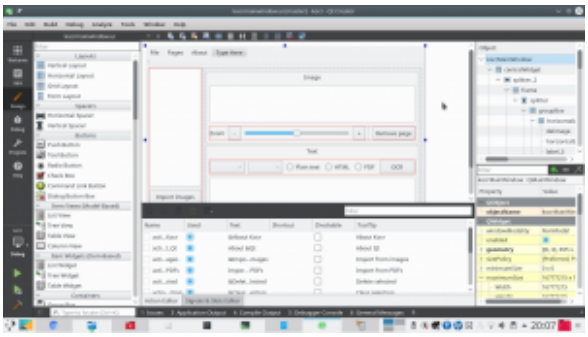
Come funziona il formato PDF?

Il formato PDF è uno standard ufficiale dal 2007, declinato in una serie di sottoformati: A,X,E,H,UA, a seconda dei vari utilizzi che se ne vogliono fare. Quello che si segue

solitamente è il PDF/A, progettato per l'archiviazione dei documenti anche a lungo termine: è pensato per integrare tutti i componenti necessari. Prima che si stabilisse questo standard, infatti, i PDF non erano davvero adatti a conservare e trasmettere documenti, perché mancavano spesso alcuni componenti fondamentali. Per esempio, se un PDF veniva visualizzato su un computer nel quale non erano installati i font con cui sul PC originale era stato scritto il testo, tutta l'impaginazione saltava. Ora, invece, i font possono essere integrati, assieme ad eventuali altri oggetti, così è possibile visualizzare correttamente un PDF/A su qualsiasi dispositivo, a prescindere dal suo sistema operativo. Questo significa che ogni PDF moderno è di fatto un po' più grande di quanto lo sarebbe stato un PDF degli anni '90, perché porta al suo interno i vari font, ma questo non è un problema considerando che il costo dello spazio dei dischi rigidi diminuisce continuamente e un paio di kilobyte in più in un file non si notano nemmeno.

Il formato PDF nasce da un formato precedente che è tutt'ora in uso e che si chiama PostScript. PostScript è di fatto un linguaggio di programmazione che permette di descrivere delle pagine: i file PS sono dei semplici file di testo che contengono una serie di istruzioni per il disegno di una pagina, con le sue immagini e il testo. Si tratta di un linguaggio che va interpretato, quindi la sua elaborazione richiede una buona quantità di risorse e di tempo. Un file PDF, invece, è di fatto una sorta di PS già interpretato, il che permette di risparmiare tempo. Per fare un esempio, in un file PS si troveranno molte condizioni "if" e cicli "loop", e si tratta di istruzioni che consumano molte risorse quando vanno interpretate. Nei PDF, invece, viene direttamente inserito il risultato dei vari cicli, così da risparmiare tempo durante la visualizzazione. Quello che è importante capire è che il formato PDF è progettato per la stampa, è pensato per essere facilmente visualizzato e stampato allo stesso modo su qualsiasi dispositivo. Insomma, una funzione di

sola lettura. Non è affatto progettato per permettere la continua modifica dei file. Ciò non significa che sia proibito, i file PDF possono ovviamente essere modificati come qualsiasi altro file, ma la modifica può essere molto complicata da fare in certi casi proprio perché le informazioni vengono memorizzate puntando a massimizzare l'efficienza della lettura, non della scrittura o della modifica. Per esempio, i testi vengono memorizzati una riga alla volta, e non in blocchi di paragrafi o colonne, come invece risulterebbe comodo per modificarli successivamente. Un'altra differenza importante è che nei PDF ogni pagina è un elemento a se stante, mentre nei PostScript le pagine sono legate e condividono alcune caratteristiche (come le dimensioni).



Utilizzando l'IDE gratuito QtCreator è molto facile anche disegnare l'interfaccia grafica multipiattaforma

Includere Poppler

Cominciamo subito col nostro programma di esempio. Le librerie necessarie possono essere incluse nell'intestazione del codice come da prassi del C++. Quelle che servono per la gestione dei PDF sono le seguenti:

Per scrivere i PDF utilizzeremo infatti la libreria QpdfWriter, che si trova nella stessa cartella di tutte le altre librerie Qt, e che quindi viene trovata in automatico dall'IDE. Per la lettura dei PDF, invece, useremo Poppler, che va installata a parte. Qui le cose cambiano un po', perché mentre in GNU/Linux esiste un percorso standard nel quale installare le librerie, e quindi si può facilmente trovare poppler nella cartella **poppler/qt5/**, su Windows questo non esiste. Quindi, sfruttando gli **ifdefine** forniti dalle librerie Qt, possiamo distinguere la posizione dei file che contengono la libreria Poppler a seconda del fatto che il sistema sia Windows (**Q_OS_WIN**) o GNU/Linux (**Q_OS_LINUX**). La posizione delle librerie per Windows potrà essere stabilita nel file di progetto, che vedremo più avanti. Possiamo ora cominciare a vedere il codice: non lo vedremo tutto, solo le parti fondamentali per la gestione dei PDF.

Entro breve, le librerie Qt integreranno direttamente una classe per la lettura dei PDF, chiamata QPDFDocument, senza quindi la necessità di usare Poppler. Al momento tale classe non è ancora considerata stabile, quindi abbiamo deciso di presentare questo articolo basandoci ancora su Poppler. Quando il rilascio di QtPdf sarà ufficiale, la presenteremo in nuovo articolo.

La prima funzione che implementiamo dovrà permettere l'importazione dei PDF. Infatti, vogliamo permettere agli utenti di importare dei PDF scansionati, in modo da poter eseguire su di essi l'OCR e ricavare il testo.

Chiamando la funzione **getOpenFileNames** di **QFileDialog** si visualizza una finestra standard per consentire all'utente la selezione di più file, il cui percorso completo viene inserito

in una lista di stringhe che chiamiamo **files**.

Possiamo anche creare una MessageBox per chiede conferma all'utente, così se dovesse avere scelto i file per sbaglio potrà annullare il procedimento prima di cominciare a lavorare sui file (operazione che può richiedere del tempo).

Banalmente, se il pulsante premuto dall'utente è **Cancel**, allora interrompiamo la funzione. Altrimenti, con un semplice ciclo for scorriamo tutti gli elementi della lista di file, passandoli uno alla volta a un funzione che si occuperà di estrarre le pagine dal PDF e aggiungerle alla lista delle pagine su cui lavorare.

Aprire un PDF in lettura

Abbiamo chiamato la funzione che opera effettivamente l'estrazione delle pagine da un PDF, **addpdftolist**.

Questa funzione comincia controllando che il file che ha ricevuto come argomento **pdfin** sia esistente (**QfileInfo.exists** controlla che il file esista e non sia vuoto).

Ora abbiamo bisogno di una cartella temporanea, nella quale inserire tutte le immagini che estrarremo dalle pagine del PDF. La libreria **QTemporaryDir** si occupa proprio di creare una cartella temporanea a prescindere dal sistema operativo. Possiamo memorizzare il percorso di tale cartella in una stringa che chiamiamo **tmpdir**. Dobbiamo anche specificare che la cartella non va sottoposta all'auto rimozione, altrimenti il programma cancellerà la cartella automaticamente al termine di questa funzione, mentre noi ne avremo ancora bisogno in altre funzioni. La cancellazione di tale cartella potrà essere fatta manualmente alla chiusura definitiva del programma.

Siamo finalmente pronti per leggere il PDF. Basta creare un oggetto di tipo **Poppler::Document**, usando la funzione load che permette per l'appunto la lettura di un file PDF. Se il PDF non conteneva un documento valido, conviene terminare la funzione con l'istruzione return per evitare problemi.

Il documento potrebbe avere più pagine, quindi utilizziamo un ciclo for per leggerle tutte una alla volta.

Ogni pagina può essere estratta usando un oggetto **Poppler::Page**, e con l'apposita funzione **page** di un documento. Se la pagina è invalida, il ciclo si ferma.

La pagina può poi essere renderizzata in una immagine, rappresentata dall'oggetto **QImage**, secondo una carta risoluzione orizzontale e verticale (che di solito coincidono, ma non sempre).

Poppler permette di leggere tutti i componenti di un PDF, annotazioni incluse

Nel nostro programma, consideriamo tale risoluzione pari a 300 dpi, e l'abbiamo inserita in una apposita variabile all'inizio del programma chiamata per l'appunto **dpi**.

Per salvare l'immagine della pagina basta chiamare la funzione **save** dell'immagine. Tuttavia, prima dobbiamo decidere il nome del file: sarà ovviamente composto dal percorso della cartella

temporanea più il nome **tmppage** seguito dal numero progressivo della pagina e l'estensione **tiff**. Il numero della pagina viene scritto con 4 cifre, giustificando con lo 0. Quindi, la pagina 1 sarà **tmppage0001**, mentre la pagina 23 sarà **tmppage0023**, e la 145 sarà **tmppage0145**. In questo modo siamo sicuri di non confondere mai l'ordine delle pagine.

È bene ricordarsi di eliminare l'oggetto pagina e il documento, per liberare la memoria (lavorando ad alte risoluzioni è facile che venga richiesta molta RAM per svolgere queste operazioni).

Avremmo potuto inserire direttamente ogni immagine estratta nell'elenco delle immagini che vogliamo passare all'OCR, ma possiamo anche semplicemente leggere il contenuto della cartella temporanea cercando tutti i file che contiene e, scorrendoli uno ad uno, passare il loro percorso alla funzione **addimagestolist**. È infatti questa la funzione che si occuperà di inserire le singole immagini nella lista. Chiamando la funzione soltanto dopo l'estrazione delle pagine, le immagini appariranno nell'interfaccia grafica tutte assieme e l'utente capirà che la procedura è terminata.

La funzione in questione è molto semplice: viene creato un nuovo elemento del qlistwidget (l'oggetto che nell'interfaccia grafica del nostro programma funge da elenco delle pagine). All'elemento viene assegnata una icona, che proviene dal file stesso e che quindi costituirà la sua anteprima. L'elemento viene infine aggiunto all'oggetto presente nell'interfaccia grafica (**ui**).

Il file di progetto

Per consentire al compilatore di trovare la libreria Poppler basta inserire nel file di progetto (.pro) le seguenti righe:

In questo modo il compilatore saprà che su Windows i file .h si troveranno nella cartella **include/poppler-qt5** del codice sorgente, mentre la libreria compilata sarà nella cartella **lib**.

Fusione dei PDF

Dopo avere eseguito l'OCR sulle varie pagine, si ottengono da Tesseract tanti PDF quante sono per l'appunto le pagine del documento. Ciò significa che dovremo riunirle manualmente, fondendo assieme tutti i vari file in un unico PDF. Per farlo, prima di tutto decidiamo il nome di un file temporaneo nel quale riunire tutti i PDF:

Lo facciamo sfruttando lo stesso meccanismo che abbiamo usato per la cartella temporanea, ma con la libreria **QTemporaryFile**. Ovviamente, il file dovrà avere estensione **pdf**, e il suo nome è contenuto nella variabile **tmpfilename**.

Per scrivere sul PDF temporaneo, basta creare un nuovo oggetto di tipo QpdfWriter associato al file e un oggetto **Qpainter** associato al **pdfWriter**. Il QPainter è il disegnatore che si occuperà di, per l'appunto, disegnare il contenuto del PDF secondo le nostre indicazioni.

Le varie pagine, cioè i pdf da riunire, si trovano nella stringa **allpages** separati dal simbolo **|**. Con un semplice ciclo for possiamo prendere un pdf alla volta, inserendo il suo nome nella stringa **inp**.

Se quello su cui stiamo lavorando non è il primo dei file da unire (quindi il contatore delle pagine **i** è maggiore di 0), allora possiamo inserire una interruzione di pagina nel PDF finale con la funzione **newPage**. Questo ci permette di unire i vari file dedicando una nuova pagina a ciascuno.

Possiamo quindi aprire il pdf usando, come già visto, **Poppler::Document**. Con un ciclo for scorriamo le varie pagine: ciascuno dei file da unire dovrebbe contenere una sola pagina, ma è comunque più prudente usare un ciclo per non correre rischi.

Le varie TextBox

Ora dobbiamo estrarre il testo della pagina, cioè il testo che Tesseract ha inserito grazie alla funzione di OCR.

Potremmo semplicemente prelevare il testo con la funzione **text**, ma preferiamo usare **textList**. Infatti, la prima ci fornisce semplicemente tutto il testo della pagina, ma a noi questo non va bene: abbiamo bisogno di avere anche l'esatta posizione, nella pagina, di ogni parola. Per questo esiste **textList**, una lista di **Poppler::TextBox**, dei rettangoli che contengono il testo e hanno una precisa posizione e dimensione.

Con un ulteriore ciclo for possiamo scorrere tutte le **textBox** ottenendo il rettangolo (**QrectF** è un rettangolo con dimensioni float) che le rappresenta usando la funzione **boundingBox**.

Ora c'è un piccolo problema: le dimensioni e la posizione del rettangolo sono state indicate, da Poppler, con il sistema di riferimento della pagina che stiamo leggendo. Invece, il nostro pdfWriter avrà probabilmente un sistema di riferimento diverso, a causa della risoluzione. Possiamo calcolare il

rapporto orizzontale e verticale semplicemente dividendo larghezza e altezza della pagina di pdfWriter per quelle della pagina di Poppler.

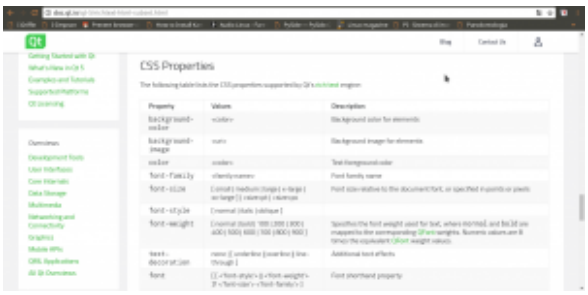
Adesso possiamo tranquillamente scrivere il testo usando il nuovo rettangolo, che abbiamo appena calcolato, come riferimento. Il testo (attributo **text** della **textBox** attuale) si aggiunge usando la funzione drawText del **painter**.

Soltanto dopo avere terminato questo ciclo for, e quindi avere scritto tutti i testi dove necessario, possiamo disegnare sulla pagina l'immagine di sfondo, con la funzione drawPixmap che si usa per inserire in un **painter** una immagine a mappa di pixel (una bitmap qualsiasi). La pixmap è ovviamente ottenuta dall'immagine che preleviamo tramite Poppler usando la già vista funzione **renderToImage**. Inserendo l'immagine dopo il testo, siamo sicuri che sarà visibile soltanto l'immagine, e il testo risulterà invisibile ma ovviamente selezionabile e ricercabile. In alternativa avremmo anche potuto scegliere il colore "trasparente" per il testo.

Ovviamente, quando abbiamo finito di leggere un file, dobbiamo eliminare il suo oggetto **document** per non occupare troppo spazio. Per quanto riguarda il PDF che stiamo scrivendo, non c'è bisogno di chiudere il file: QpdfWriter lo farà automaticamente appena la funzione termina.

Scrivere dell'HTML

C'è ancora un ultimo caso da considerare: se invece di Tesseract si vuole utilizzare l'OCR Cuneiform su Windows, purtroppo non si ottiene un PDF e nemmeno un file HOCR (cioè un HTML con la posizione delle varie parole). Si ottiene soltanto un semplice file HTML, che mantiene la formattazione ma non la posizione delle parole.



Un testo formattato può essere inserito in un PDF con QTextDocument usando la formattazione CSS delle pagine HTML (http://doc.qt.io/qt-5/richtext-html-subset.html)

Non è ottimale, ma può comunque essere utile avere un PDF che contenga il testo nella pagina, così lo si può ricercare facilmente. In questo caso, la prima cosa da fare è leggere il file html che si ottiene:

Leggendo il file come semplice testo grazie alle librerie QFile e QTextStream, possiamo inserire tutto il codice nella stringa **hocr**.

Ora, possiamo creare un nuovo documento di testo formattato, usando la libreria QTextDocument. Il contenuto del testo sarà indicato proprio dal codice **html** della stringa hocr, che quindi mantiene la formattazione. Impostiamo anche la larghezza massima del testo pari a quella della pagina di **pdfWriter**.

Come prima, dovremo calcolare la corretta dimensione con cui inserire il testo, per evitare che sia troppo piccolo o troppo grande. Siccome stavolta è solo testo, possiamo calcolare la dimensione del font con cui scriverlo usando una proporzione.

Dopo avere scelto la giusta dimensione del testo affinché

riempia tutta la pagina, possiamo inserire il testo nel painter, e quindi nel PDF, usando la funzione drawContents del QTextDocument. Il vantaggio d i questa funzione, rispetto a drawText, è che in questo modo si mantiene la formattazione e l'allineamento standard HTML.

Ovviamente, anche in questo caso si conclude la pagina inserendo sopra al testo l'immagine della pagina stessa, così il testo non sarà visibile, ma comunque ricercabile e selezionabile.

Il codice sorgente e il binario dell'esempio

Per capire come venga organizzato il codice sorgente, vi conviene controllare quello del nostro programma di esempio. Banalmente, il programma è composto da un file di progetto, un file **main.cpp** che costituisce la base dell'eseguibile, e due file (uno .h e uno .cpp) per la classe **mainwindow**, che rappresenta l'interfaccia principale del programma. Inoltre, abbiamo inserito due cartelle con il codice sorgente e il codice binario della libreria Poppler per Windows.

Trovate tutto il codice su GitHub assieme a dei pacchetti precompilati per Windows e GNU/Linux: **https://github.com/zorbaproject/kocr/releases**