Report Laboratorio 1 del corso Tecnologie per IoT

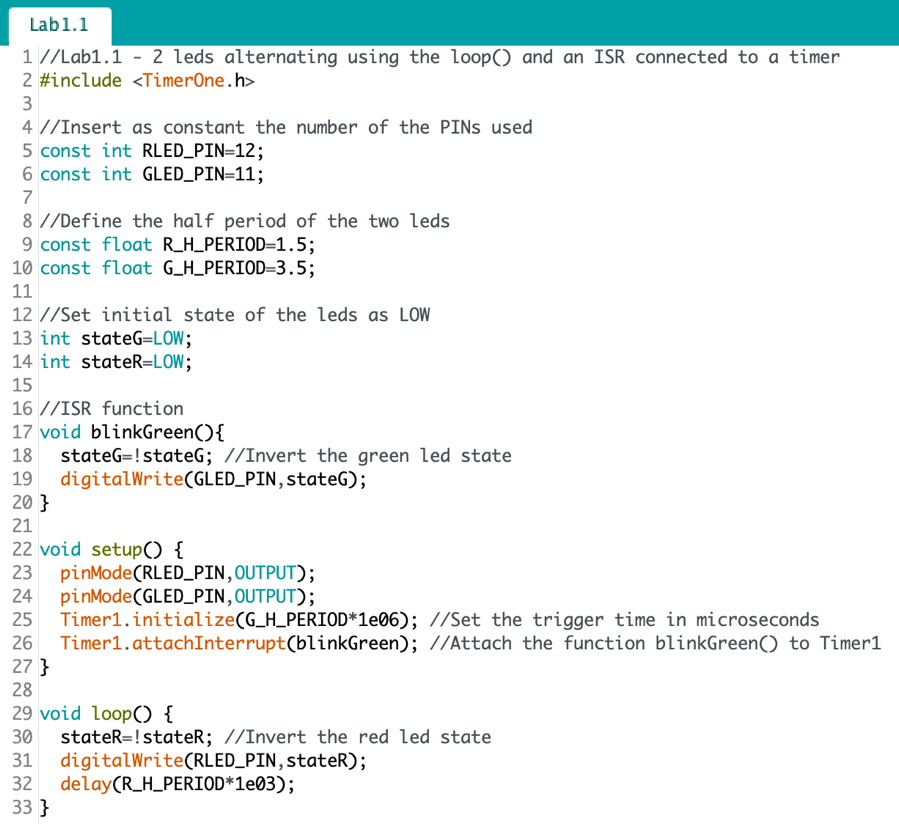
Il primo laboratorio del corso “Tecnologie per IoT” è suddiviso in sei esercizi, basati sull’utilizzo della scheda Arduino Yun e del linguaggio di programmazione Arduino. Lo scopo generale di questo laboratorio era quello di permetterci di prendere confidenza con la scheda elettronica, con i vari sensori e con la scrittura di semplice codice.

Il report è suddiviso in sei parti, ognuna relativa al corrispettivo esercizio, nei quali verrà analizzata l’implementazione del codice.

Esercizio 1

Il primo esercizio ha lo scopo di farci utilizzare la scheda Arduino per controllare l’accensione e lo spegnimento di due led. Entrambi i led saranno collegati in serie con una resistenza da 180Ω e collegati ad Arduino tramite due differenti pin digitali. Il periodo di oscillazione dei due led deve essere indipendente dunque abbiamo scelto due periodi primi tra loro ovvero 7 e 3 secondi.

In particolare, il led rosso deve essere controllato dalla funzione loop e da un delay, mentre il led verde deve essere modificato tramite una ISR che viene chiamata allo scadere di un timer preimpostato.



All’inizio del nostro codice abbiamo definito alcune costanti e alcuni parametri che ci sarebbero serviti successivamente. Nello specifico, abbiamo definito due costanti di tipo intero che rappresentano i due differenti pin utilizzati nel collegamento dei led (11 per il led rosso e il 12 per il led verde) e due costanti di tipo float per rappresentare i semi-periodi di lampeggiamento dei due led. All’interno della funzione setup abbiamo settato i due pin digitali come output, in quanto è la scheda Arduino a mandare i segnali di controllo ai led, e abbiamo inoltre inizializzato il timer che deve effettuare una Interrupt ogni semiperiodo del led verde (il semiperiodo viene passato al timer in microsecondi). Successivamente abbiamo passato al timer la funzione da eseguire alla scadenza del tempo impostato. La funzione blinkGreen viene infatti chiamata all’effettuarsi della Interrupt e semplicemente inverte lo stato del led verde e successivamente esegue la funzione digitalWrite per scrivere sul pin dedicato al led verde il nuovo state. Analogamente all’interno del loop viene invertito lo stato del led rosso, viene scritto sul pin il nuovo valore e viene effettuata una funzione di delay di lunghezza pari al semiperiodo del led rosso.

Utilizzando questo codice il comportamento dei due led sarà analogo ma il risultato è stato ottenuto tramite due processi differenti.

Esercizio 2

Questo esercizio è nuovamente basato sull’accensione e spegnimento di due led ma in questo caso è necessario l’utilizzo del monito seriale per poter richiedere lo stato di uno dei due led attraverso l’invio di caratteri all’Arduino. Il programma dovrà quindi leggere se sulla porta seriale ci sono eventuali caratteri di input da parte dell’utente e in caso affermativo dovrà capire se il comando è valido e quindi stampare a schermo lo stato e del led, oppure se il carattere letto non è valido e quindi restituire un errore.

Immagine che contiene screenshot

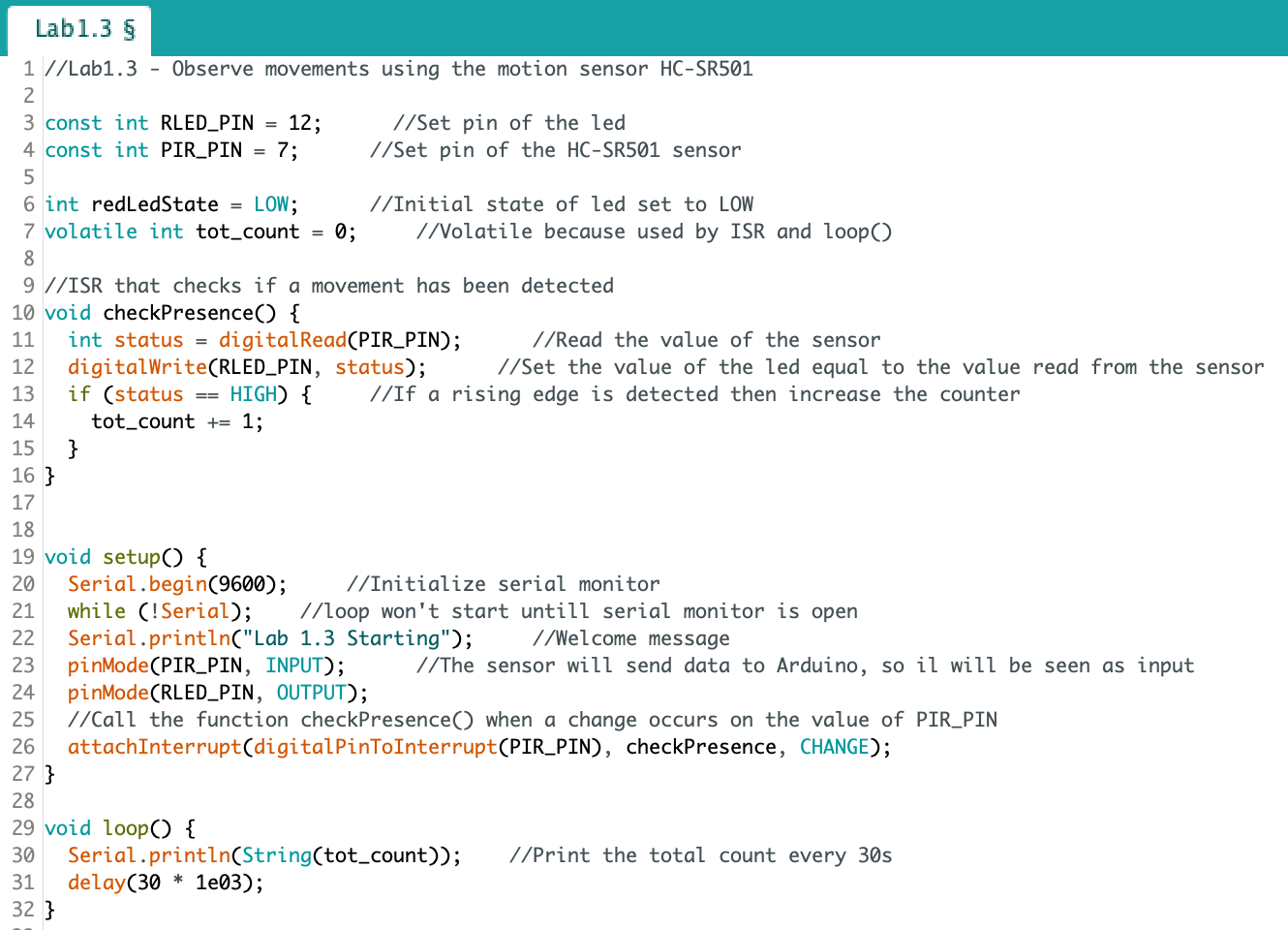
Descrizione generata automaticamente

Buona parte del codice scritto è uguale all’esercizio precedente e la spiegazione verrà omessa. Ciò che è stato aggiunto è principalmente la funzione serialPrintStatus che viene chiamata nel loop, e dunque avrà un comportamento sincrono rispetto al funzionamento del led rosso. Nella funzione come prima cosa viene controllato se è presente qualche tipo di dato all’interno del buffer del collegamento seriale. In caso negativo la funzione terminerà e la sua presenza sarà completamente trasparente. Nel caso opposto, ovvero sono presente dei dati nel buffer, da quest’ultimo viene letto un byte (dato che i nostri comandi sono basati su caratteri di dimensione 1 byte) usufruendo della funzione Serial.read(). Nel caso il valore ritornato sia il codice ASCII della lettera ‘r’, allora verrà stampato sul monitor seriale lo stato del led rosso (nel nostro caso il valore viene convertito in stringa per facilitare la concatenazione e la scrittura più chiara del risultato). Analogamente se il valore ritornato corrisponde al codice ASCII della lettera ‘g’ allora verrà stampato lo stato del led verde. Nel caso il valore letto non corrisponde a nessun comando, un errore viene stampato a schermo. È importante notare che in questo esercizio lo stato dei led verde è definiti come ‘volatile’ in quanto la variabile di stato è utilizzata sia dal loop che dalla ISR e bisogna dunque evitare che si violi il comportamento Read-After-Write sul registro di questa variabile.

Nel setup l’unica differenza rispetto all’esercizio 1 è che definiamo e inizializziamo la connessione seriale. Il parametro 9600 passato a Serial.begin serve per settare il baud-rate della comunicazione. Infine, alla riga successiva impostiamo una while in modo da aspettare che venga aperto il monito seriale dall’utente prima di fare eseguire il resto del programma.

Esercizio 3

Questo esercizio ha lo scopo di farci lavorare con il sensore HC-SR501 per rilevare movimenti nelle vicinanze del sensore stesso. Nello specifico, lo scopo è quello di accendere un led al momento del rilevamento di movimento e contare il numero di movimenti rilevati a partire dell’avvio del programma, stampando sul monitor seriale il totale ogni 30 secondi. Per interfacciare il sensore alla scheda Arduino è necessario utilizzare tre pin. Il primo pin del sensore sarà collegato al GRD, un’altro ai 5V di alimentazione e l’ultimo ad un pin GPIO che supporti le Interrupt (nel nostro caso abbiamo utilizzato il pin 7).



Dopo aver definito le costanti di riferimento per il numero dei pin e aver inizializzato lo stato del led a LOW e il totale dei movimenti rilevati a zero (la variabile è volatile perchè verrà utilizzata sia dal loop che dalla ISR del sensore), nel setup abbiamo inizializzato la comunicazione con il monitor seriale (in modo del tutto analogo a quanto fatto nell’esercizio precedente), lanciato un messaggio di introduzione sul monitor seriale e successivamente abbiamo settato il pin del led come output mentre il pin del sensore come input in quanto quest’ultimo invia segnali alla scheda quando un movimento viene rilevato. Infine abbiamo dovuto gestire i segnali di Interrupt che sarebbero arrivati dal sensore e infatti usando la funzione attachInterrupt abbiamo imposto di chiamare la funzione checkPresence qualora un qualsiasi cambiamento di stato fosse avvenuto sul pin di riferimento al sensore. Abbiamo scelto di chiamare la funzione ad ogni cambiamento in modo da poter facilmente impostare lo stato del led cosa che sarebbe stata decisamente più complessa nel caso avessimo usato l’opzione ‘RISING’. All’interno della funzione checkPresence, salviamo l’attuale valore ricevuto dal sensore e impostiamo lo stato del led con il medesimo valore. Infatti se rileviamo un voltaggio HIGH dal sensore significa che un movimento è stato rilevato e conseguentemente il led dovrà accendersi, nel caso il valore del sensore sarà LOW allora il sensore non sta rilevando nessun movimento e il led dovrà spegnersi. Come ultimo comando eseguiamo un controllo per verificare se ci troviamo in una situazione di ‘rising edge’ o ‘falling edge’. Infatti dato che la ISR verrà richiamata per ogni variazione di segnale, è nostro compito verificare che ad ogni ‘rising edge’ corrisponda un aumento del conteggio. All’interno del loop viene ciclicamente stampato sul monitor seriale il totale dei movimenti rilevati seguito da un delay di 30 secondi.

Esercizio 4

L'obiettivo di questa parte del laboratorio è quello di utilizzare un motore a corrente continua (nel nostro caso il motore DFR0332) la cui velocità di rotazione può essere modificata tramite appositi comandi trasmessi ad Arduino tramite seriale. La particolarità di questo esercizio è l’utilizzo di un componente che funziona tramite PWM (pulse-width modulation) ovvero la cui intensità può essere modificata utilizzando un pin digitale modificando il duty cycle del segnale. E’ importante notare che sebbene nel codice si utilizzi la funzione analogWrite, il pin utilizzato sarà un pin digitale con la particolarità di supportare la PWM (nel nostro caso abbiamo utilizzato il pin 6).



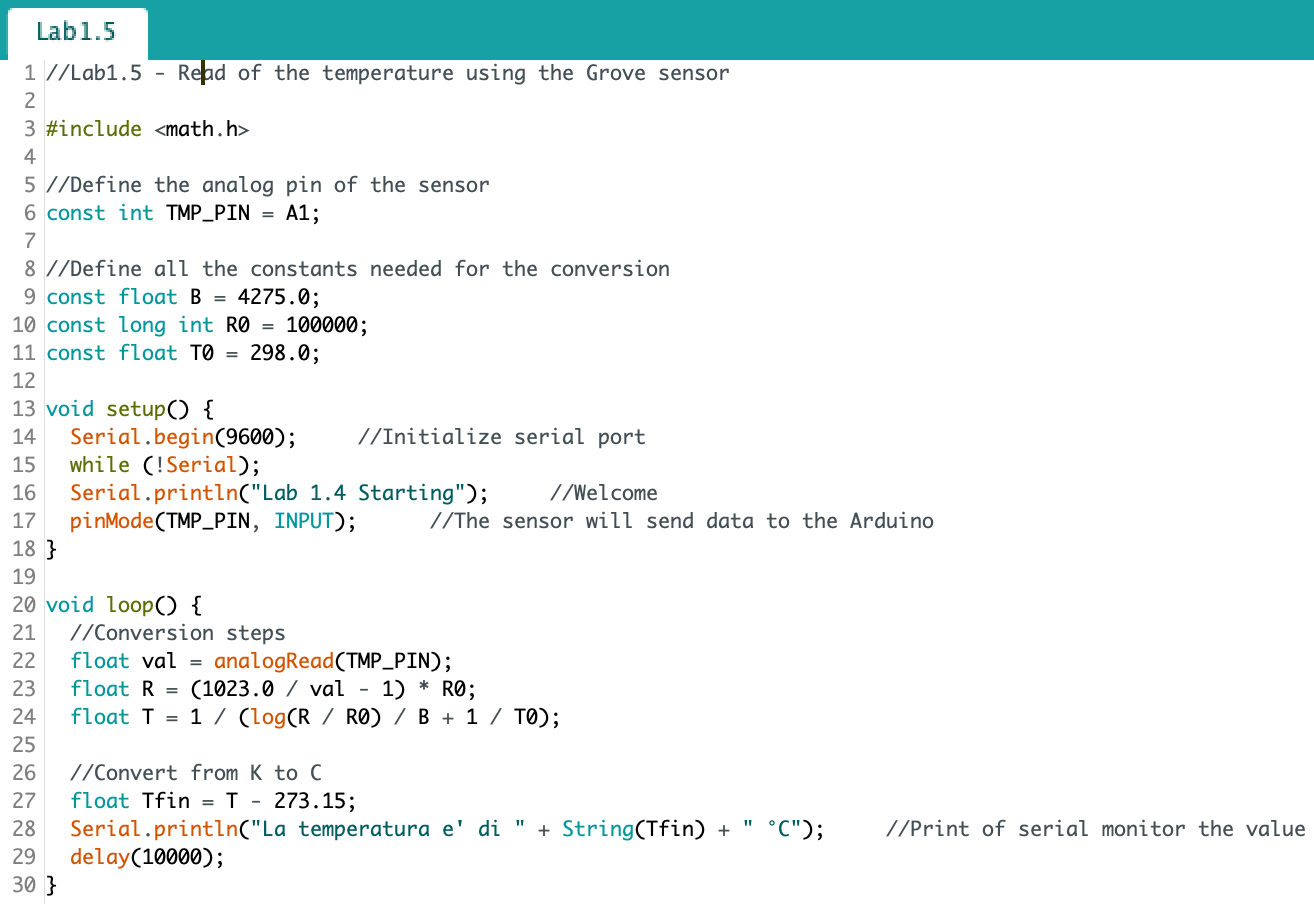
Come prima cosa definiamo le costanti relative al pin del motore e dello step di incremento che utilizzeremo nella funzione changeSpeed (noi abbiamo scelto 25.5 in modo da ottenere 10 step di incremento) e settiamo la variabile della velocità a zero in modo da evitare che il motore inizi a ruotare all’avvio.

Dopo le inizializzazioni viste precedentemente della comunicazione seriale e dei vari pin, abbiamo usato le analogWrite per impostare la velocità di partenza del motore. All’interno del loop abbiamo semplicemente una chiamata alla funzione changeSpeed che gestirà le varie situazioni.

Quando changeSpeed viene chiamata, viene controllato se ci sono byte disponibili da leggere nel buffer in caso negativo la funzione ritornerà e nulla verrà eseguito. In caso contrario verrà letto un byte dal buffer in modo da leggere un carattere, se il valore letto dal buffer è il corrispettivo del codice ASCII del carattere ‘+’ allora viene effettuato un controllo per verificare che la velocità sia minore di 255 ovvero la velocità massima, e in questo caso si aumenterà il valore della velocità di uno step e verrà inviato il valore al motore tramite analogWrite. In modo analogo si verificherà se il carattere nel buffer è il carattere ‘-’ e si verificherà che la velocità non sia già a zero. Nel caso questo non si verifichi allora la velocità verrà ridotta di uno step. In caso di caratteri non validi o in caso di tentativi di aumentare oltre la velocità massima o ridurre oltre la velocità minima, un errore verrà stampato sul monitor seriale.

Esercizio 5

In questo esercizio abbiamo usato Arduino e un sensore di temperatura (Grove) per rilevare la temperatura ambientale. La principale difficoltà di questo esercizio è la conversione dei valori ricevuti dal sensore in una temperatura in gradi Celsius. Il valore della temperatura deve nuovamente essere stampato sul monitor seriale. Il sensore in questione funziona tramite un termistore, la cui resistenza varia al variare della temperatura. Il sensore è basato su un partitore di tensione. Utilizzando queste informazioni unite alla relazione non lineare che lega la resistenza alla temperatura (che è possibile leggere sul datasheet del sensore), è possibile ricavare la temperatura in gradi Kelvin che potrà essere facilmente convertita in gradi Celsius.Il processo dovrà essere diviso in due passaggi. Prima dovremo ricavarci il valore della resistenza utilizzando la formula inversa del partitore di tensione e successivamente usare la definizione del parametro B per ottenere la temperatura. Bisogna inoltre tenere conto che la scheda Arduino utilizza un convertitore analogico-digitale con una risoluzione a 10 bit, quindi nelle formule di conversione, il valore massimo di tensione 5V dovrà essere scritto come 1023.

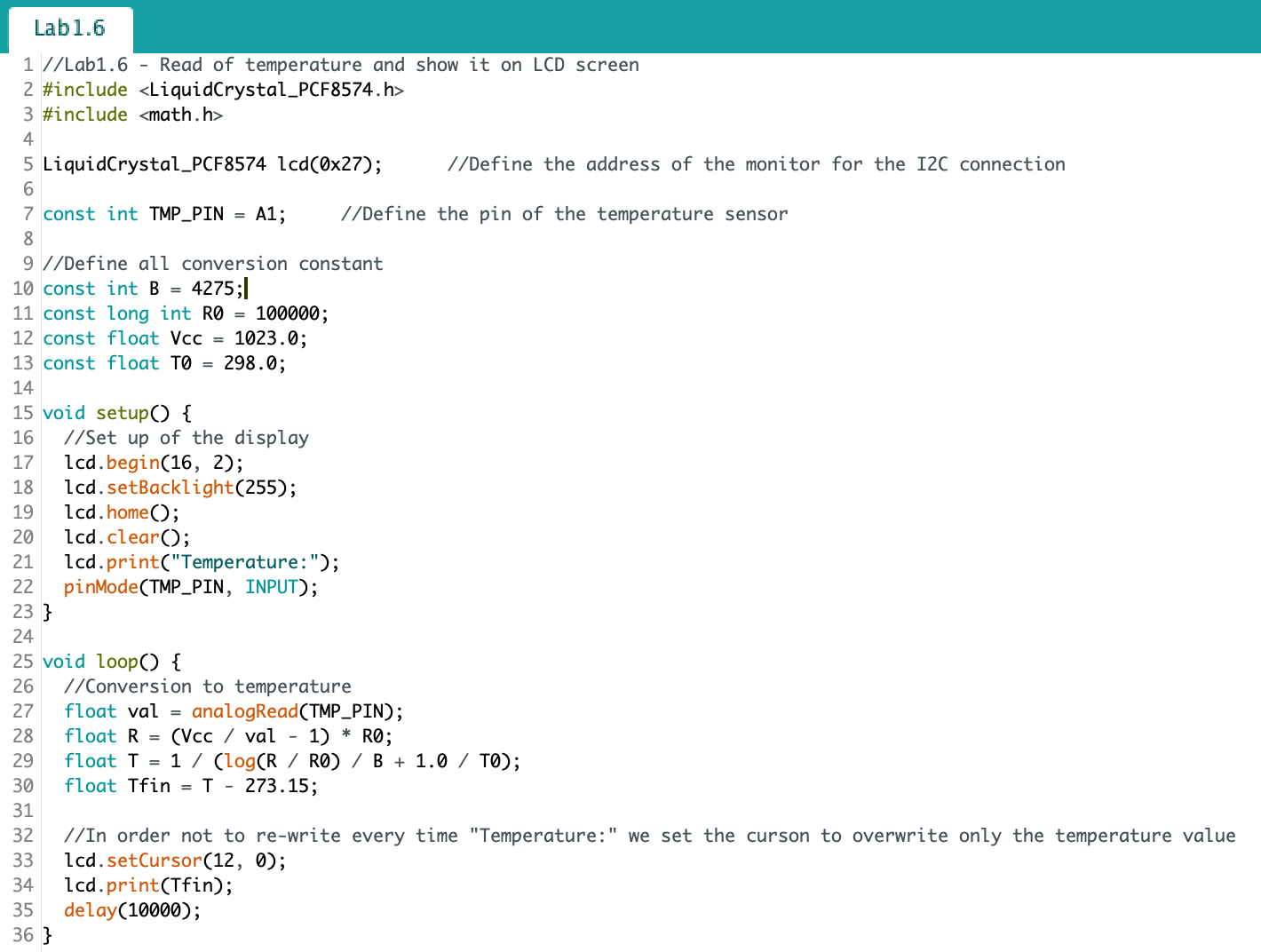


All’inizio del codice abbiamo definito il valore del pin analogico utilizzato dal sensore di temperatura e abbiamo definito come costanti tutti i parametri che serviranno nella conversione da voltaggio a temperatura. Nel setup non abbiamo fatto altro che inizializzare la comunicazione seriale, stampare un messaggio iniziale e settare il pin del del sensore come input dato che Arduino riceverà il segnale dal sensore.

Nel loop eseguiamo tre passaggi. Prima ci otteniamo il valore della resistenza del sensore tramite la formula del partitore di tensione, con il valore ottenuto possiamo ricavare il valore della resistenza in Kelvin e infine possiamo convertire da gradi Kelvin a gradi Celsius. Il passaggio finale consiste nel stampare la temperatura sul monitor seriale effettuare un delay di 10 secondi.

Esercizio 6

Questo esercizio è analogo al precedente ma è richiesto di mostrare la temperatura su un display LCD invece che tramite il monitor seriale. Il display utilizzato in questo laboratorio è il display DFRobot collegato ad Arduino tramite connessione I2C. Per interfacciare il display, abbiamo utilizzato la libreria esterna ‘LiquidCrystal\_PCF8574’.



La parte fondamentale di questo codice è la definizione dell'indirizzo del display. Infatti il protocollo di comunicazione del display è I2C è basato su master e slave, quindi il master (Arduino) deve conoscere l’indirizzo dello slave (il display). Nel nostro caso l’indirizzo è 0x27. Tutto ciò che ci resta da fare è impostare il display,. Nel setup il display viene inizializzato con 16 colonne e 2 righe, la massima luminosità, viene ripulito lo schermo da eventuali scritte e stampiamo il testo “Temperature:”. Il pin del sensore di temperatura sarà nuovamente visto come input.

All’interno del loop, oltre alle varie conversioni per ottenere la temperatura, usiamo la funzione setCursor in modo da poter riscrivere sul display solo il nuovo valore e non tutto il testo “Temperature: valore” in questo modo possiamo risparmiare l’invio di bit al display. Infine settiamo un delay di 10 secondi.