Capitolo 4

Libreria per la generazione di un codice Morse

Ora che conoscete abbastanza bene l'IDE Arduino e sapete come accendere i led potete affrontare un progetto più significativo. In questo capitolo verrà sviluppato un generatore di codice Morse che legge un testo inviato alla porta seriale e lo trasmette in output sotto forma di segnali luminosi che accendono e spengono un led.
La realizzazione di questo progetto permette di approfondire la conoscenza della trasmissione seriale tra Arduino e computer. L'approfondimento riguarda anche il funzionamento della stessa IDE di sviluppo, in quanto si vedrà come utilizzare librerie esistenti e come strutturare progetti di grandi dimensioni nelle proprie librerie di programmi. Al termine di questo capitolo sarete in grado di realizzare una libreria che potrà essere pubblicata su Internet.

In questo capitolo

Cosa serve

- Una scheda Arduino, per esempio un modello Arduino Uno, Duemilanove o Diecimila.
- Un cavo USB per collegare la scheda Arduino al computer.
- Un led.
- Un altoparlante o un buzzer (facoltativi).

Concetti di base del codice Morse

Il codice Morse fu inventato per convertire un testo in segnali audio (`http://en.wikipedia.org/wiki/Morse_Code` o `http://it.wikipedia.org/wiki/codice_Morse`). Da un punto di vista teorico la codifica è simile a quella che converte un insieme di caratteri nel codice ASCII. La differenza è che il codice ASCII rappresenta i caratteri in forma numerica, mentre il codice Morse è costituito da sequenza di punti e linee (chiamati in gergo *di* e *dah*). I punti hanno una durata inferiore rispetto a quella delle linee. Una lettera A è codificata dalla sequenza · – (*di dah*), mentre la lettera Z corrisponde a – – · · (*dah dah di di*).

Il codice Morse è caratterizzato anche da una temporizzazione che stabilisce la durata dei punti e delle linee e deve anche specificare la durata delle pause tra simboli differenti e tra parole. L'unità di base del codice Morse è costituita dalla durata di un punto, da cui deriva per esempio che una linea corrisponde a tre punti. Tra due simboli consecutivi la pausa deve durare quanto un punto, mentre tra due lettere la pausa è di tre punti. Tra due parole si deve avere una pausa di sette punti.

La trasmissione di un messaggio in codice Morse richiede una modalità di trasmissione di segnali caratterizzati da lunghezze differenti l'uno dall'altro. Una tecnica classica è rappresentata dall'utilizzo di segnali sonori, ma in questo progetto si impiegherà un led che verrà acceso e spento con diversi intervalli di tempo. In Marina è ancora in uso la trasmissione Morse basata su segnali luminosi.

Ora potete implementare il generatore di codice Morse!

Realizzare un generatore di codice Morse

L'elemento principale della libreria del generatore è costituito dalla classe C++ chiamata `Telegraph`. In questo paragrafo verrà definita la sua interfaccia a partire dallo schema di principio indicato di seguito.

Telegraph/Telegraph.pde

```
void setup() {
}

void loop() {
}
```

Si tratta ovviamente di un programma Arduino ridotto ai minimi termini, che non svolge alcuna operazione a eccezione della dichiarazione delle funzioni obbligatorie, ancora prive di istruzioni. Questa tecnica di sviluppo consente però di compilare il lavoro progressivamente verificando di volta in volta la presenza di errori sintattici. Salvate il programma con il nome `Telegraph`, in modo che l'IDE possa creare una cartella chiamata *Telegraph* e vi includa il file `Telegraph.pde`. Tutti i file e le directory richiesti dalla libreria saranno memorizzati nella cartella *Telegraph*.

A questo punto aprite una nuova scheda e indicate il nome di file `telegraph.h`. Proprio così, avete appena creato un file header in C; anzi, per essere precisi è un file header in C++. Di seguito è riportato l'elenco delle istruzioni di questo file.

Telegraph/telegraph.h

```
#ifndef __TELEGRAPH_H__
#define __TELEGRAPH_H__

class Telegraph {
public:
  Telegraph(const int output_pin, const int dit_length);
  void send_message(const char* message);

private:
  void dit();
  void dah();
  void output_code(const char* code);
  void output_symbol(const int length);

  int _output_pin;
  int _dit_length;
  int _dah_length;
};

#endif
```

Ricordate sempre che la programmazione orientata agli oggetti non è più una esclusiva delle CPU di grandi capacità! In questo caso il programma descrive l'interfaccia della classe `Telegraph` da impiegare nel vostro primo progetto importante, a condizione ovviamente che siate interessati alla trasmissione di informazioni in codice Morse.

La prima riga del programma imposta un meccanismo di esclusione della doppia inclusione; in altre parole, il corpo del file header definisce una macro di preprocessore denominata `__TELEGRAPH_H__`. Il corpo del file (che contiene la definizione della macro) è incluso in una istruzione `#ifndef`, pertanto le sue istruzioni vengono compilate solo se la macro non è stata ancora definita. In questo modo potete includere l'header più volte nel progetto e garantire che le istruzioni dell'header saranno compilate una sola volta.

L'interfaccia della classe `Telegraph` è caratterizzata da una parte pubblica cui hanno accesso gli utenti della classe e da una parte privata che può essere utilizzata solo dai membri della classe. Nella parte pubblica sono presenti due elementi: un costruttore per la creazione di nuovi oggetti `Telegraph` e un metodo chiamato `send_message()` che invia un messaggio tramite l'emissione di un segnale in codice Morse. Nelle applicazioni del vostro progetto potrete utilizzare questa classe C++ come segue:

```
Telegraph telegraph(13, 200);
telegraph.send_message("Hello, world!");
```

Nella prima riga viene creato un nuovo oggetto `Telegraph` che comunica con il pin 13 ed emette "punti" che hanno una durata di 200 millisecondi. La seconda istruzione trasmette il messaggio "Hello, world!" in codice Morse. Questa soluzione permette di inviare un messaggio qualsiasi e di modificare facilmente l'impostazione che riguarda il pin di trasmissione e la durata di un punto.
Dopo aver definito l'interfaccia della classe potete implementare il programma in base alle indicazioni del prossimo paragrafo.

Perfezionare l'interfaccia del generatore

La dichiarazione delle interfacce è una fase importante dello sviluppo di un programma, almeno quanto lo è la sua implementazione. Create una nuova scheda, immettete il nome di file `telegraph.cpp` e riportate le istruzioni che seguono.

> **NOTA**
>
> Le versioni meno recenti di Arduino presentano un bug che non consente la creazione di un nuovo file in questo modo, dato che l'IDE segnala la presenza di un file che ha lo stesso nome. Per sapere come risolvere il problema consultate la pagina web all'indirizzo `http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1251245246`.

Telegraph/telegraph.cpp

```
#include <ctype.h>
#include <WProgram.h>
#include "telegraph.h"

char* LETTERS[] = {
  ".-", "-...", "-.-.", "-..", ".",     // A-E
  "..-.", "--.", "....", "..", ".---",  // F-J
  "-.-", ".-..", "--", "-.", "---",     // K-O
  ".--.", "--.-", ".-.", "...", "-",    // P-T
  "..-", "...-", ".--", "-..-", "-.--", // U-Y
  "--.."                                // Z
};

char* DIGITS[] = {
  "-----", ".----", "..---", "...--", // 0-3
  "....-", ".....", "-....", "--...", // 4-7
  "---..", "----."                    // 8-9
};
```

Analogamente alla maggior parte dei programmi in C++, anche questo importa in primo luogo alcune librerie. Dato che più avanti verranno impiegate funzioni particolari, per esempio `toupper()`, il programma include `ctype.h` cui si aggiunge `telegraph.h` per rendere disponibile la dichiarazione della classe illustrata nel paragrafo precedente e le sue funzioni. A cosa serve però la libreria `WProgram.h`?

Finora non è stato necessario occuparsi della provenienza di costanti quali `HIGH`, `LOW` oppure `OUTPUT`. Queste costanti sono definite in diversi file header messi a disposizione dall'IDE Arduino e che potete trovare nella directory *hardware/cores/arduino* della stessa IDE. Esaminando la libreria `WProgram.h` vedete che nel file `wiring.h` sono contenute tutte le costanti impiegate finora e molte altre ancora. Questa libreria dichiara molte macro e gran parte delle funzioni principali di Arduino.
Fintanto che modificate programmi scritti a partire da zero non dovete preoccuparvi di includere file header standard, in quanto l'IDE svolge questa operazione dietro le quinte. Non appena iniziate a elaborare progetti più complessi che includono codice C++ vero e proprio, nasce l'esigenza di gestire l'inclusione di tutte le librerie. Dovete in altre parole importare tutte le librerie richieste dal programma, come quelle delle costanti di Arduino.
Dopo aver importato i file header il programma determina due array di tipo stringa chiamati `LETTERS` e `DIGITS`. Questi array contengono il codice Morse di lettere e cifre che useremo più avanti per convertire il testo del messaggio in codice Morse. Prima di effettuare questa operazione il programma definisce il costruttore che deve creare e inizializzare i nuovi oggetti `Telegraph`.

Telegraph/telegraph.cpp

```
Telegraph::Telegraph(const int output_pin, const int dit_length) {
  _output_pin = output_pin;
  _dit_length = dit_length;
  _dah_length = dit_length * 3;
  pinMode(_output_pin, OUTPUT);
}
```

Il costruttore richiede due argomenti: il numero del pin ove inviare il codice Morse e la lunghezza di un punto espresso in millisecondi. Il programma memorizza questi valori in variabili istanza corrispondenti, calcola la durata di una linea e imposta il pin di comunicazione come pin di output.
Avete probabilmente notato che le variabili istanza di tipo privato hanno un nome che inizia con un underscore (_), una convenzione che non è peraltro imposta dal linguaggio C++ o dall'IDE Arduino.

Output dei simboli del codice Morse

Al termine delle inizializzazioni il programma può iniziare a trasmettere in output i simboli del codice Morse. La leggibilità del codice è facilitata dalla presenza di svariati metodi helper.

Telegraph/telegraph.cpp

```
void Telegraph::output_code(const char* code) {
  for (int i = 0; i < strlen(code); i++) {
```

```
  if (code[i] == '.')
    dit();
  else
    dah();
  }
}

void Telegraph::dit() {
  Serial.print(".");
  output_symbol(_dit_length);
}

void Telegraph::dah() {
  Serial.print("-");
  output_symbol(_dah_length);
}

void Telegraph::output_symbol(const int length) {
  digitalWrite(_output_pin, HIGH);
  delay(length);
  digitalWrite(_output_pin, LOW);
}
```

La funzione `output_code()` accetta una sequenza in codice Morse costituita da punti e linee e la converte in chiamate delle funzioni `dit()` e `dah()`. I metodi `dit()` e `dah()` inviano un punto e una linea alla porta seriale e lasciano il resto dell'elaborazione al metodo `output_symbol()`, a cui passano la lunghezza del simbolo in codice Morse da trasmettere. La funzione `output_symbol()` imposta il pin di output nello stato `HIGH` per tutta la lunghezza del simbolo, dopodiché viene ripristinato lo stato `LOW`. Le operazioni vengono effettuate in base allo schema di temporizzazione del codice Morse; manca solo l'implementazione del metodo `send_message()`, riportata di seguito.

Telegraph/telegraph.cpp

```
Riga 1  void Telegraph::send_message(const char* message) {
    -     for (int i = 0; i < strlen(message); i++) {
    -       const char current_char = toupper(message[i]);
    -       if (isalpha(current_char)) {
    5         output_code(LETTERS[current_char - 'A']);
    -         delay(_dah_length);
    -       } else if (isdigit(current_char)) {
    -         output_code(DIGITS[current_char - '0']);
    -         delay(_dah_length);
   10       } else if (current_char == ' ') {
    -           Serial.print(" ");
    -           delay(_dit_length * 7);
    -       }
    -     }
   15     Serial.println();
    -   }
```


Questa funzione trasmette un carattere del messaggio alla volta, all'interno di un ciclo di istruzioni. Alla riga 3 il carattere iniziale è convertito in una lettera maiuscola, dato che le minuscole non sono riconosciute dal codice Morse (questo è il motivo per cui non è possibile implementare un client per chat da trasmettere in codice Morse). A questo punto il programma verifica se il carattere attuale è costituito da una lettera ricavata dalla funzione `isalpha()` del linguaggio C. In questo caso si utilizza la funzione per stabilire la rappresentazione in codice Morse memorizzata nell'array `LETTERS`. Per eseguire questa operazione si adotta un vecchio stratagemma: nella tabella ASCII tutte le lettere (e le cifre) compaiono in modo sequenziale, ovvero si ha A=65, B=66 e così via. Il carattere attuale può pertanto essere trasformato nell'indice dell'array `LETTERS` sottraendo 65 (`'A'`) dal valore del codice ASCII. Il valore corrente del codice Morse è poi trasferito al metodo `output_symbol()` e si ritarda in questo modo il programma per un tempo pari alla durata di una linea.

L'algoritmo prevede un funzionamento analogo per la trasmissione di cifre. Occorre semplicemente ricavare l'indice della array `DIGITS` e non dell'array `LETTERS` e poi sottrarre il valore ASCII del carattere 0.

Alla riga 10 il programma verifica la presenza di un carattere vuoto; in questo caso si deve inviare alla porta seriale un carattere vuoto e attendere per un tempo pari a sette punti. Tutti gli altri caratteri vengono ignorati dal programma, che può elaborare solo lettere, cifre e caratteri vuoti. Al termine dell'esecuzione del metodo il programma invia alla porta seriale un carattere newline per identificare la fine del messaggio.

Installare e utilizzare la classe Telegraph

La classe `Telegraph` è veramente completa e può essere impiegata per realizzare altri esempi di programma. Questa considerazione è importante per due motivi: potete verificare le funzionalità del codice presente nella libreria e gli utenti della classe possono ricavarne informazioni utili per studiarne l'utilizzo.

L'IDE Arduino ricerca le librerie in due posizioni differenti, ovvero nella cartella globale *libraries* relativa alla directory di installazione dell'IDE e nella directory locale dei progetti dell'utente dell'applicazione. In fase di sviluppo conviene utilizzare la directory dei progetti, la cui posizione può essere individuata tra le preferenze dell'IDE, come si può osservare nella Figura 4.1. Create una nuova directory di nome *libraries* nella directory locale dei progetti.

Per rendere disponibile la classe `Telegraph` dovete creare una sottocartella *Telegraph* nella directory *libraries*, poi copiate in questa cartella i file `telegraph.h` e `telegraph.cpp`. Non copiate il file `Telegraph.pde`. Riavviate l'IDE.

Ora potete provare la madre di tutti i messaggi: "Hello, world!". Create un nuovo progetto denominato `HelloWorld` e digitate le istruzioni che seguono.

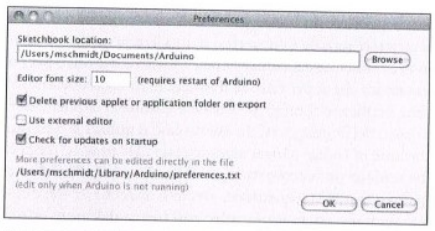

Figura 4.1 Individuate nelle preferenze la directory in cui memorizzare i progetti.

Telegraph/examples/HelloWorld/HelloWorld.pde

```
#include "telegraph.h"
const unsigned int OUTPUT_PIN = 13;
const unsigned int DIT_LENGTH = 200;

Telegraph telegraph(OUTPUT_PIN, DIT_LENGTH);

void setup() {}

void loop() {
  telegraph.send_message("Hello, world!");
  delay(5000);
}
```

Questo programma trasmette ogni cinque secondi la stringa "Hello, world!" in codice Morse. Le istruzioni includono la definizione della classe `Telegraph` e delle costanti che riguardano il pin ove è collegato il led e la durata di un punto. A questo punto il programma crea un oggetto globale `Telegraph` e una funzione vuota `setup()`. Nella funzione `loop()` si chiama ogni cinque secondi il metodo `send_message()` sull'istanza `Telegraph`.

La compilazione di questo programma implica anche la compilazione automatica della libreria `Telegraph`. Un eventuale errore di sintassi nella libreria verrà pertanto segnalato in questa fase di compilazione. Correggete gli errori assicurandovi di modificare i file originali del codice sorgente. Dopo aver corretto gli errori copiate nuovamente i file nella cartella *libraries*; non dimenticare di riavviare l'IDE.

La conversione di una stringa statica in codice Morse è un'operazione interessante ma non sarebbe meglio fare in modo che il programma elaborasse stringhe arbitrarie? Per ottenere questo risultato dovete studiare un esempio di programma più sofisticato del precedente. Questa volta si scriverà un codice che legge messaggi dalla porta seriale e li riporta verso un'istanza `Telegraph`. Create un nuovo progetto di nome `MorseCodeGenerator` e digitate le istruzioni qui riportate.

Telegraph/examples/MorseCodeGenerator/MorseCodeGenerator.pde

```
#include "telegraph.h"

const unsigned int OUTPUT_PIN = 13;
const unsigned int DIT_LENGTH = 200;
const unsigned int MAX_MESSAGE_LEN = 128;
const unsigned int BAUD_RATE = 9600;
const int LINE_FEED = 13;

char message_text[MAX_MESSAGE_LEN];
int index = 0;

Telegraph telegraph(OUTPUT_PIN, DIT_LENGTH);

void setup() {
  Serial.begin(BAUD_RATE);
}

void loop() {
  if (Serial.available() > 0) {
    int current_char = Serial.read();
    if (current_char == LINE_FEED || index == MAX_MESSAGE_LEN - 1) {
      message_text[index] = 0;
      index = 0;
      telegraph.send_message(message_text);
    } else {
      message_text[index++] = current_char;
    }
  }
}
```

Anche in questo caso si include il file header della classe `Telegraph` e come al solito si definiscono alcune costanti: `OUTPUT_PIN` stabilisce il pin di collegamento del led, mentre `DIT_LENGTH` contiene la durata di un punto espressa in millisecondi. La costante `LINE_FEED` è impostata con il valore ASCII che corrisponde al carattere linefeed; questo carattere è necessario per stabilire la fine del messaggio da trasmettere. Infine, la costante `MAX_MESSAGE_LEN` stabilisce la lunghezza massima dei messaggi che il programma consente di trasmettere in codice Morse.

A questo punto si definiscono tre variabili globali: `message_text` è il buffer di caratteri da riempire con i dati ricevuti dalla porta seriale, `index` tiene traccia della posizione attuale nel buffer e `telegraph` è l'oggetto `Telegraph` da impiegare per convertire il messaggio in "segnali luminosi". La funzione `setup()` inizializza la porta seriale, mentre `loop()` verifica l'arrivo di nuovi dati chiamando il metodo `Serial.available()`. Il programma legge il byte successivo di dati, se disponibile, e verifica se si tratta di un carattere linefeed oppure dell'ultimo byte da inserire nel buffer di caratteri. In entrambi i casi si imposta l'ultimo byte di `message_text` con il valore 0, dato che in C/C++ le stringhe sono terminate dal carattere null.

Ora dovete compilare e caricare il programma. Aprite *Serial Monitor* e selezionate *Carriage return* come terminazione di riga nel menu a comparsa visibile nella parte inferiore della finestra. Questa opzione permette a *Serial Monitor* di aggiungere automaticamente un carattere newline a ogni riga inviata ad Arduino. Digitate un messaggio qualsiasi, per esempio il vostro nome, e osservate come Arduino converte il messaggio in segnali luminosi.
L'incapsulamento della logica del codice Morse nella classe `Telegraph` comporta il vantaggio che il programma principale è breve e conciso. La creazione di software per dispositivi miniaturizzati non significa che si debba rinunciare ai vantaggi offerti dalla programmazione orientata agli oggetti.
Non resta che aggiungere alcuni dettagli meno significativi per trasformare il progetto in una libreria di prima classe, in base alle indicazioni del prossimo paragrafo.

Ritocchi finali

Una delle funzioni più interessanti dell'IDE Arduino riguarda l'uso dei colori per distinguere la sintassi dei comandi. I nomi delle classi, i nomi delle funzioni, le variabili e altro ancora sono visualizzati con colori differenti nella finestra dell'editor. In questo modo il codice sorgente è più semplice da leggere ed è possibile anche colorare la sintassi delle librerie. Dovete semplicemente aggiungere al progetto un file chiamato `keywords.txt`.

Telegraph/keywords.txt

```
# Sintassi a colori della libreria Telegraph

Telegraph       KEYWORD1
send_message    KEYWORD2
```

La riga che inizia con un carattere # è di commento e sarà ignorata dal compilatore. Le righe successive contengono il nome di uno dei membri della libreria e il tipo di membro. Separate i due parametri con un carattere di tabulazione. Le classi sono di tipo `KEYWORD1`, mentre le funzioni sono di tipo `KEYWORD2`. Per quanto riguarda le costanti impostate il tipo `LITERAL1`.
Attivate la sintassi a colori per la libreria `Telegraph` copiando il file `keywords.txt` nella cartella *libraries* e riavviate l'IDE. A questo punto il nome della classe `Telegraph` compare in arancione, mentre `send_message()` è in colore marrone.
Prima di terminare la pubblicazione della libreria dovete effettuare le operazioni descritte di seguito.

- Memorizzate gli esempi di programma in una cartella chiamata *examples* e copiate questa cartella nella directory *libraries*. Ogni programma deve avere una cartella in questa directory.

- Stabilite i termini della licenza d'uso del progetto e trascriveteli in un file chiamato `LICENSE`. All'indirizzo `http://www.opensource.org/` trovate molte informazioni al riguardo e licenze di tipo standard. Ricordate che questa opzione consente di aumentare la fiducia dei potenziali utenti del progetto.
- Aggiungete istruzioni per l'installazione e altra documentazione di progetto. In genere gli utenti si aspettano di trovare la documentazione in un file chiamato `README` e cercano le istruzioni di installazione nel file `INSTALL`. Provate a installare la libreria utilizzando sistemi operativi diversi tra loro e per ogni sistema fornite adeguate istruzioni di installazione.

Al termine di queste operazioni la cartella della libreria dovrebbe avere un aspetto simile a quello mostrato nella Figura 4.2.

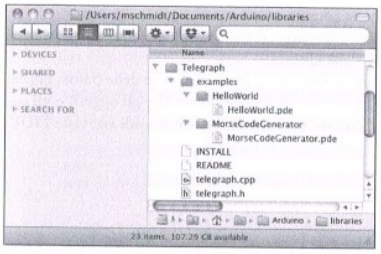

Figura 4.2 Esempio tipico del contenuto di una libreria Arduino.

Ora non rimane che creare un archivio ZIP che contenga tutti i file del progetto Arduino. La maggior parte dei sistemi operativi richiede semplicemente di fare clic con il tasto destro del mouse sulla directory in *Esplora risorse*, nel *Finder* o in un'altra funzionalità analoga a queste e trasformare la directory in un archivio ZIP. I sistemi Linux e Mac consentono di creare un archivio ZIP anche utilizzando una delle istruzioni da riga di comando qui riportate:

```
maik> zip -r Telegraph Telegraph
maik> tar cfvz Telegraph.tar.gz Telegraph
```

La prima istruzione crea un file chiamato `Telegraph.zip`, mentre la seconda crea l'archivio `Telegraph.tar.gz`. Entrambi i formati sono molto diffusi ed è sempre bene mettere a disposizione degli utenti entrambi i formati compressi.
La creazione di una libreria Arduino è abbastanza semplice, anche se dovete eseguire più di una operazione manuale. In definitiva, non avete scusanti: ogni volta che pensate di aver realizzato qualcosa di interessante dovete renderlo pubblico.

I progetti illustrati finora hanno comunicato con l'esterno tramite led (output) e pulsanti (input). Nel prossimo capitolo si vedrà come impiegare dispositivi di input più sofisticati, per esempio sensori a ultrasuoni. Studierete anche la possibilità di visualizzare i dati trasmessi da Arduino tramite programmi eseguiti dal computer.

Cosa fare se non funziona?

L'IDE Arduino considera fondamentale la denominazione di file e directory ed è stata ideata per realizzare progetti, non librerie. Per questo motivo dovete effettuare alcune operazioni manuali sui file per organizzare l'archivio di una libreria. Nella Figura 4.2 potete osservare l'aspetto finale della directory di una libreria Arduino. Se avete installato più di una versione dell'IDE Arduino ricordate di verificare che state impiegando la corretta cartella *libraries*.
Tenete presente che dovete spesso riavviare l'IDE e comunque dovete farlo ogni volta che modificate uno dei file della libreria.
Se la sintassi a colori non funziona dovete controllare che il file delle parole chiave sia denominato `keywords.txt`. Verificate con molta attenzione che gli oggetti e gli indicatori di tipo siano separati da un carattere di tabulazione, quindi riavviate l'IDE.

Esercizi

- Il codice Morse supporta non solo le lettere ma anche i numeri. Definisce inoltre simboli di uso comune, per esempio le virgole. Modificate la classe `Telegraph` in modo che il programma possa interpretare l'intero codice Morse.
- L'accensione dei led produce un effetto interessante ma il codice Morse è in genere associato alla ricezione di segnali audio. Sostituite il led con un altoparlante piezo, di basso costo e semplice da utilizzare. La Figura 4.3 mostra il collegamento di un altoparlante alla scheda Arduino. Gli altoparlanti hanno un pin di massa e un pin di segnale: collegate tra loro i pin di massa dell'altoparlante e di Arduino, mentre il pin di segnale dell'altoparlante deve essere collegato al pin 13 della scheda. Sostituite il metodo `output_symbol()` con le istruzioni qui riportate:

```
void Telegraph::output_symbol(const int length) {
  const int frequency = 131;
  tone(_output_pin, frequency, length);
}
```

  Questo metodo invia un'onda quadra all'altoparlante e riproduce un tono a frequenza di 131 Hz. L'esempio di progetto "Melody" incluso nell'IDE Arduino spiega come suonare note particolari con un altoparlante piezo.
- Modificate il progetto della libreria in modo da supportare dispositivi di output differenti tra loro. Potete per esempio impostare un oggetto `Output-Device` del costruttore `Telegraph` e derivare un oggetto `LedDevice` e `SpeakerDevice` da `OutputDevice`. Osservate il codice che segue.

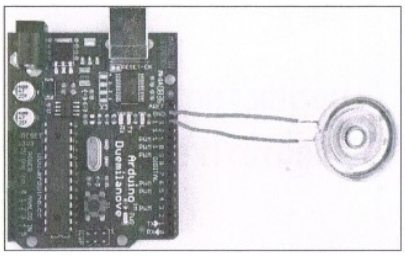

Figura 4.3 Collegare un altoparlante alla scheda Arduino è facile.

```
class OutputDevice {
  public:
  virtual void output_symbol(const int length);
};

class Led : public OutputDevice {
  public:
  void output_symbol(const int length) {
    // ...
  }
};

class Speaker : public OutputDevice {
  public:
  void output_symbol(const int length) {
    // ...
  }
};
```

Potete utilizzare queste classi come segue:

```
Led led;
Speaker speaker;
OutputDevice* led_device = &led;
OutputDevice* speaker_device = &speaker;

led_device->output_symbol(200);
speaker_device->output_symbol(200);
```

Il resto del programma è lasciato come esercizio da svolgere per conto vostro.

- Imparate il codice Morse. Fate trasmettere da un vostro amico alcuni messaggi che vengano visualizzati nel terminale seriale e provate a interpretare il testo del messaggio che state leggendo. Questo esercizio non è necessario per studiare lo sviluppo dei programmi Arduino ma può essere molto divertente!