```
//     Alcuin Montessori School
//     Adolescent Program
//     Air Morse
//     (C) Copyright 2010.  All Rights Reserved.

#include <LiquidCrystal.h>

// ----- Team Display -----
//

// LED functions:

// Green light is IO 9
// Red 1 (next to green light) is IO 8
// Red 2 is IO 7
// Red 3 is IO 6

#define GREEN 9
#define RED1 8
#define RED2 7
#define RED3 6

void displayDih()
   {
   digitalWrite(GREEN, HIGH) ;
   digitalWrite(RED1,  LOW ) ;
   digitalWrite(RED2,  HIGH) ;
   digitalWrite(RED3,  LOW ) ;
   }

void displayDah()
   {
   digitalWrite(GREEN, HIGH) ;
   digitalWrite(RED1,  HIGH) ;
   digitalWrite(RED2,  HIGH) ;
   digitalWrite(RED3,  HIGH) ;
   }

void displaySpace()
   {
   digitalWrite(GREEN, HIGH) ;
   digitalWrite(RED1,  LOW ) ;
   digitalWrite(RED2,  LOW ) ;
   digitalWrite(RED3,  LOW ) ;
   }

void displayClear()
   {
   digitalWrite(GREEN, LOW) ;
   digitalWrite(RED1,  LOW) ;
   digitalWrite(RED2,  LOW) ;
   digitalWrite(RED3,  LOW) ;
   }


// LCD functions:

LiquidCrystal lcd (12, 11, 5, 4, 3, 2);

int row = 0 ;
int col = 0 ;

void displayChar(char c)
   {
   int i ;

   if(c==8)
```

```
      return displayBackspace() ;

   col = col + 1 ;
   if(col>15)
      {
      col = 0 ;
      row = row + 1 ;
      if(row>1)
         row = 0 ;
      for(i=0 ; i<16 ; i++)
         {
         lcd.setCursor(i,row) ;
         lcd.print(" ") ;
         }
      }
   lcd.setCursor(col,row) ;
   lcd.print(c) ;
   }

void displayBackspace()
   {
   lcd.setCursor(col,row) ;
   lcd.print(' ') ;

   col = col - 1;
   if(col<0)
      {
      col = 15 ;
      row = row -1 ;
      if(row<0)
         row = 1 ;
      }
   lcd.setCursor(col,row) ;
   }

// Initialize function:

void displayInit()
   {
   // set LED pin directions
   pinMode (GREEN, OUTPUT) ;
   pinMode (RED1,  OUTPUT) ;
   pinMode (RED2,  OUTPUT) ;
   pinMode (RED3,  OUTPUT) ;

   // start LEDs off
   displayClear() ;

   // setup LCD display
   lcd.begin(16, 2);
   lcd.cursor() ;
   lcd.clear() ;
   lcd.print("AMSAP Air Morse");
   row = 1 ;
   col = 0 ;
   }

// ----- Team Decode -----
//

char decode(char morse[])
   {
   if(0==strcmp(morse,".-"   )) return 'A' ;
   if(0==strcmp(morse,"-..." )) return 'B' ;
   if(0==strcmp(morse,"-.-." )) return 'C' ;
   if(0==strcmp(morse,"-.."  )) return 'D' ;
   if(0==strcmp(morse,"."    )) return 'E' ;
```

```
   if(0==strcmp(morse,"..-." )) return 'F' ;
   if(0==strcmp(morse,"--."  )) return 'G' ;
   if(0==strcmp(morse,"...." )) return 'H' ;
   if(0==strcmp(morse,".."   )) return 'I' ;
   if(0==strcmp(morse,".---" )) return 'J' ;
   if(0==strcmp(morse,"-.-"  )) return 'K' ;
   if(0==strcmp(morse,".-.." )) return 'L' ;
   if(0==strcmp(morse,"--"   )) return 'M' ;
   if(0==strcmp(morse,"-."   )) return 'N' ;
   if(0==strcmp(morse,"---"  )) return 'O' ;
   if(0==strcmp(morse,".--." )) return 'P' ;
   if(0==strcmp(morse,"--.-" )) return 'Q' ;
   if(0==strcmp(morse,".-."  )) return 'R' ;
   if(0==strcmp(morse,"..."  )) return 'S' ;
   if(0==strcmp(morse,"-"    )) return 'T' ;
   if(0==strcmp(morse,"..-"  )) return 'U' ;
   if(0==strcmp(morse,"...-" )) return 'V' ;
   if(0==strcmp(morse,".--"  )) return 'W' ;
   if(0==strcmp(morse,"-..-" )) return 'X' ;
   if(0==strcmp(morse,"-.--" )) return 'Y' ;
   if(0==strcmp(morse,"--.." )) return 'Z' ;

   if(0==strcmp(morse,"-----")) return '0' ;
   if(0==strcmp(morse,".----")) return '1' ;
   if(0==strcmp(morse,"..---")) return '2' ;
   if(0==strcmp(morse,"...--")) return '3' ;
   if(0==strcmp(morse,"....-")) return '4' ;
   if(0==strcmp(morse,".....")) return '5' ;
   if(0==strcmp(morse,"-....")) return '6' ;
   if(0==strcmp(morse,"--...")) return '7' ;
   if(0==strcmp(morse,"---..")) return '8' ;
   if(0==strcmp(morse,"----.")) return '9' ;

   if(0==strcmp(morse,".--." )) return ' ' ;
   if(0==strcmp(morse,"......")) return 8 ;

   // anything else
   return '?' ;
   }

// ----- Team Detect -----
//

// state variables
int state = 0 ;
int start_time = 0 ;
char capture[16] ;
int c ;

// enumerated states
#define S_INIT    0
#define S_IDLE    1
#define S_IN_DIH  2
#define S_IN_DAH  3
#define S_IN_CHAR 4

// state transition helper words
void new_state(int n_state)
   {
   start_time = millis() ;
   state = n_state ;
   }

int elapsed()
   {
   return millis() - start_time ;
   }
```

```
// utility to get current detect rate
int rate ()
   {
   int value ;
   value = analogRead(2) ;
   return value ;
   }

// utility to tell if the user is blowing
int is_blowing = 0 ;
#define BAND (5)
int blowing()
   {
   int pressure ;
   int level ;

   pressure = analogRead(0) ;
   level = analogRead(1) ;
   if(is_blowing) level = level - BAND ;
   else           level = level + BAND ;
   is_blowing = pressure > level ;
   return is_blowing ;
   }

// the finite-state machine
void detect()
  {
  // check the transitions for the current state
  switch(state)
     {
     // at startup, clear the buffer and go to idle
     case S_INIT:
        c = 0 ;
        new_state(S_IDLE) ;
        break ;

      // while idle, look for the start of blowing
      case S_IDLE:
         if(blowing())
            {
            displayDih() ;
            new_state(S_IN_DIH) ;
            }
         break ;

      // while in a DIH, look for long enough to be a DAH
      // or the end of blowing
      case S_IN_DIH:
         if(!blowing())
            {
            // end of DIH, save in buffer
            capture[c] = '.' ;
            c++ ;
            displaySpace() ;
            //displayChar('.') ;
            new_state (S_IN_CHAR) ;
            }
          if(elapsed()>rate())
            {
            displayDah() ;
            new_state (S_IN_DAH) ;
            }
         break ;

      // while in a DAH, look for the end of blowing
      case S_IN_DAH:
```

```
         if(!blowing())
            {
            // end of DAH, save in buffer
            capture[c] = '-' ;
            c++ ;
            displaySpace() ;
            //displayChar('-') ;
            new_state(S_IN_CHAR) ;
            }
         break ;

      // still in a char but not in DIH or DAH
      // look for start of a new DIH
      // or long enough to be the end of a character
      case S_IN_CHAR:
         if(blowing())
            {
            displayDih() ;
            new_state(S_IN_DIH) ;
            }
         if(elapsed()>3*rate())
            {
            // null-terminate the string
            capture[c] = 0 ;
            // clear the LEDs
            displayClear() ;
            // display the decoded character
            displayChar(decode(capture)) ;
            // clear out the buffer for more
            c = 0 ;
            new_state(S_IDLE) ;
            }
         break ;
      }
   }

// Arduino/Wiring processes:

void setup()
   {
   displayInit() ;
   }

void loop ()
  {
  detect() ;
  }
```