
13. 센서

마이크로컨트롤러는 외부 환경의 다양한 데이터를 받아들이고 이를 처리하여 외부 환경과 상호작용 할 수 있는 간단한 제어 장치를 구성하기 위해 흔히 사용되며, 데이터 입력을 위해서는 다양한 센서가 사용된다. 조도가 일정 수준 이하로 떨어지는 경우 자동으로 조명을 켜기 위해서는 광센서가 사용되고, 온도가 일정 수준 이하로 떨어지는 경우 난방 장치를 가동하기 위해서는 온도센서가 사용되는 등이 그 전형적인 예에 해당한다. 마이크로컨트롤러는 디지털 및 아날로그 입력을 받아들일 수 있으므로 디지털 또는 아날로그 출력을 내는 센서들을 사용할 수 있지만 디지털 출력을 내는 센서들은 아날로그 신호를 디지털 신호로 변환하는 ADC (Analog to Digital Converter)를 내장하고 있는 경우가 대부분이다. 대부분의 마이크로컨트롤러에는 이미 ADC가 포함되어 있으므로 마이크로컨트롤러와 함께 사용되는 센서들은 아날로그 출력을 내는 것이 일반적이다. 이 장에서는 흔히 사용되는 온도 센서와 조도 센서 사용 방법을 살펴보고 센서를 활용하여 외부 환경과 상호작용하는 제어 장치를 구성하는 방법을 알아본다.

1. 광센서

빛의 양에 따라 특성이 변화하는 소자에는 포토레지스터(photo resistor), 포토다이오드(photo diode), 포토트랜지스터(photo transistor) 등이 있다. 이 중 흔히 사용되는 소자는 빛의 양에 따라 저항값이 변하는 포토레지스터, 일반적으로 광센서라 불리는 소자로 조도센서, 광전도셀, 포토셀 등으로도 불린다. 광센서는 CdS 광센서가 주로 사용되며, 이는 카드뮴(Cd)과 황(S)을 화합하여 만들어지는 황화카드뮴 결정에 금속 다리를 결합하여 만들어진 진다.



그림 1. CdS 광센서

광센서는 가시광선이 없는 어두운 곳에서는 높은 저항값을 가져 전류가 흐르지 않다가 가시광선이 닿으면 저항값이 낮아져서 전류가 흐르는 특성을 갖는다. 즉, 광량에 반비례하는 저항값을 가진다. 광센서의 저항값은 최소 수 $k\Omega$ 에서 최대 수십 $M\Omega$ 까지 변하며 그림 2는 광량과 그에 따른 저항값의 변화를 나타낸 특성 곡선이다. 광센서는 가격이 저렴하여 간단하게 활용할 수 있지만 반응속도가 느리고 광량에 따른 출력 특성이 정밀하지 않으므로 광량을 정밀하게 측정하고자 하는 경우에는 포토다이오드나 포토트랜지스터를 사용하여야 한다.

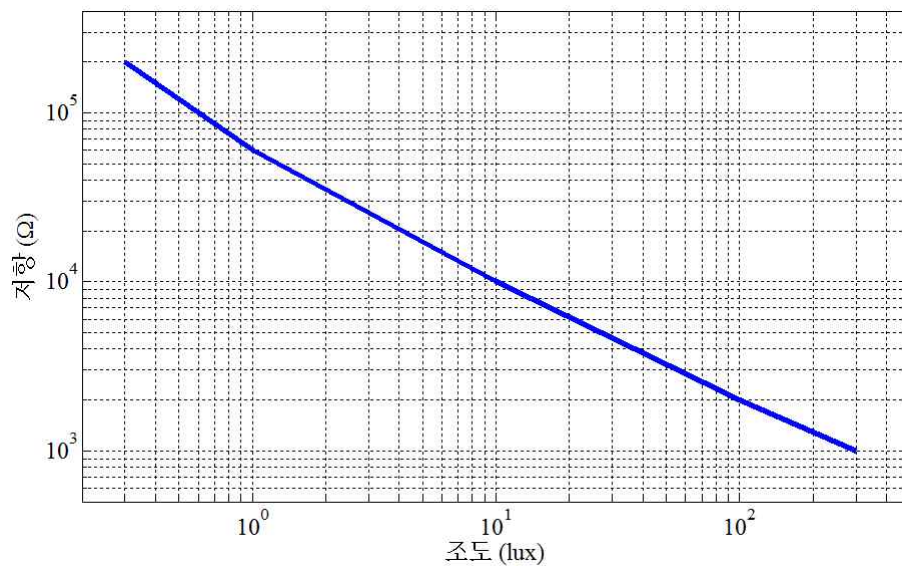


그림 2. 포토레지스터 특성 곡선

100kΩ 저항과 광센서를 그림 3에서와 같이 아날로그 2번 A2에 연결해 보자.

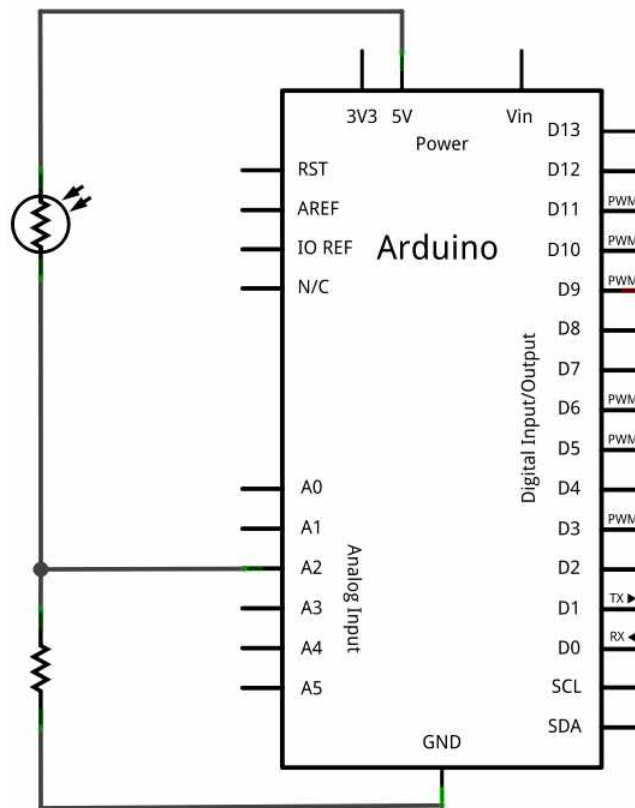


그림 3. 광센서 연결 회로도

그림 3은 낮은 광량에서는 큰 저항값을 가져 광센서에 높은 전압이 인가되고, 높은 광량에서는 낮은 저항값을 가져 광센서에 높은 전압이 인가된다. 즉, 광량에 비례하는 전압이 A2핀으로 입력된다. 스케치 1은 A2핀으로 입력되는 아날로그 값을 시리얼 모니터로 출력하는 예이다.

스케치 1. 광센서

```
void setup(){
    Serial.begin(9600);          // 시리얼 통신 초기화
}

void loop(){
    int val = analogRead(A2);    // 아날로그 입력 2번

    Serial.println(val);         // 센서 값 출력
```

```
delay(1000);           // 1초에 한 번 출력  
}
```

스케치 1을 업로드 하고 광센서를 손으로 가려서 광량을 줄이면 작은값이 출력되는 것을 확인할 수 있다. 단, 광센서는 정확성이 높지 않으며 주변 상황이나 소자에 따라 출력되는 값이 달라질 수 있다.

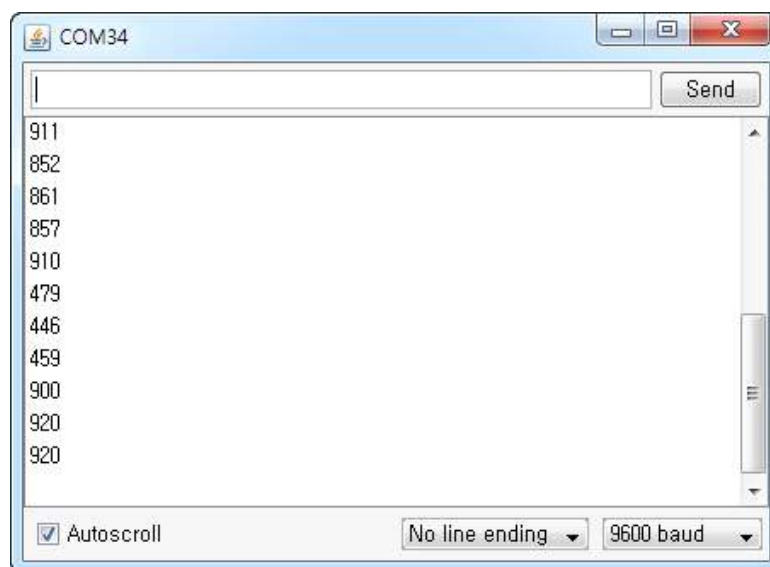


그림 4. 스케치 1 실행 결과

조도 센서와 LED를 사용하여 어두워지면 자동으로 조명이 켜지는 자동 점등 시스템을 구성해 보자. 조도 센서는 그림 13과 같이 연결하고 LED는 디지털 13번 핀에 연결된 LED를 사용한다. LED가 켜지는 조도의 기준 값은 500이하로 설정하였으며 이는 스케치 1의 테스트를 통해 결정하도록 한다. 스케치 2는 현재 조도와 LED 상태를 시리얼 모니터로 출력하는 스케치의 예이다.

스케치 2. 광센서로 LED 제어

```
int THRESHOLD = 500;           // LED를 켜는 최대 광량  
int led_pin = 13;             // LED 연결 핀
```

```
void setup(){
  Serial.begin(9600);           // 시리얼 통신 초기화
  pinMode(led_pin, OUTPUT);
}

void loop(){
  int val = analogRead(A2);     // 아날로그 입력 2번

  Serial.print(String(val) + " : LED is ");
  if(val < THRESHOLD){
    digitalWrite(led_pin, HIGH); // LED 켜기
    Serial.println("ON !!");
  }
  else{
    digitalWrite(led_pin, LOW);  // LED 끄기
    Serial.println("off...");
  }

  delay(1000);
}
```

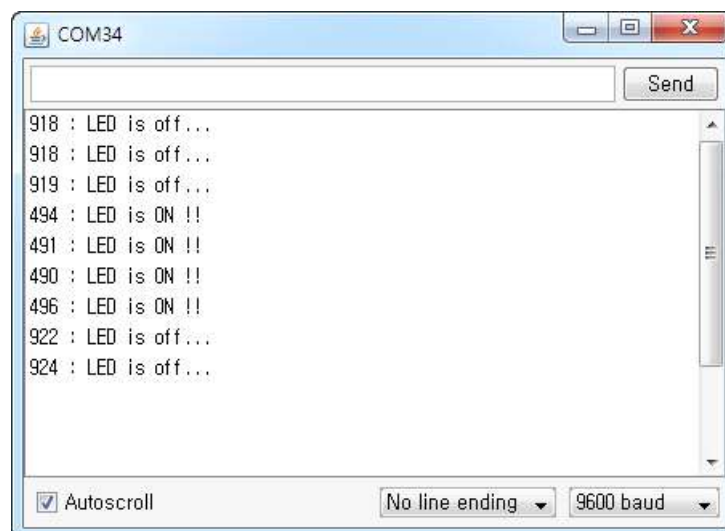


그림 5. 스케치 2 실행 결과

센서 값은 주변의 전기적 잡음, 정전기 등으로 인해 부정확한 값이 반환되는 경우가 종종 있다. 이처럼 잡음에 영향을 받는 경우에는 한 번 센서값을 읽어 들이기 보다는 여러 번 센서값을 읽고 이를 평균하여 사용하는 것이 바람직하다. 이를 구간 평균(moving average)라고 하며 이전 $n-1$ 개 값과 현재값, 총 n 개의 값을 평균하여 사용하게 된다. 이를 위해서는 크기 n 인 배열을 정의하고 이전 값들을 저장한 후 사용하면 된다. 단, 시작 후 $n-1$ 번까지는 n 개의 데이터 값이 존재하지 않으므로 정확한 평균을 얻을 수는 없다.

스케치 3은 크기 n 인 배열을 원형 큐(circular queue)로 사용하여 n 개의 값을 저장하고 이를 평균하여 사용하는 스케치의 예이다. 스케치 1이나 2에 비해 값을 평균하여 사용하므로 급격하게 변하지는 않음을 알 수 있다. 하지만 처음 출력되는 $n-1$ 개의 값은 초기값 0으로 인해 실제 값보다 작은 값이 나온다는 점에 주의해야 한다.

스케치 3. 원형 큐로 센서값 평균하기

```
#define N 10                                // 큐의 크기

int values[N] = {0, };                      // 큐 초기화
int index = 0;                              // 큐 내 데이터 저장 위치

void setup(){
    Serial.begin(9600);                     // 시리얼 통신 초기화
}

void loop(){
    int val = analogRead(A2);               // 아날로그 입력 2번
    values[index] = val;                    // 큐에 데이터 저장

    // 배열의 범위를 벗어나면 배열 처음으로 되돌린다.
    index++;
    if(index == N) index = 0;

    // 10개를 평균하므로 int를 사용하면 정수의 표현 범위를 벗어날 수 있으므로
    // float를 사용하여 합을 계산한다.
    float sum = 0.0;
    for(int i = 0; i < N; i++){
```

```
    sum += values[i];  
}  
  
Serial.println(sum / N);      // 센서 값 출력  
  
delay(1000);                  // 1초에 한 번 출력  
}
```

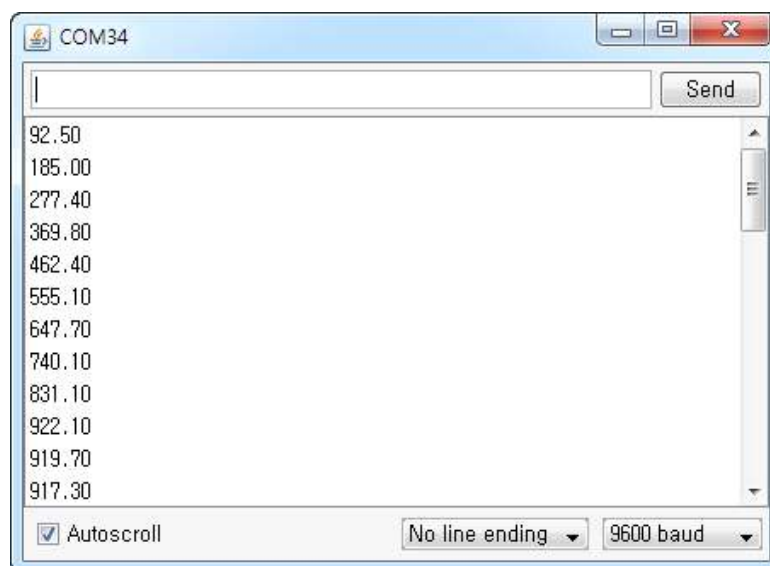


그림 6. 스케치 3 실행 결과

2. 온도 센서

온도 센서는 온도에 따라 출력되는 전압이 변하는 특성을 가지는 센서로, 그림 7은 흔히 사용되는 온도 센서 중 하나인 LM35를 나타낸다.



그림 7. LM35 온도 센서

LM35 온도 센서는 섭씨온도에 비례하여 전압이 출력되므로 사용하기 쉽다. 한 개의 전원을 사용하는 경우 LM35 온도 센서는 $2\sim 150^{\circ}\text{C}$ 범위의 온도를 측정할 수 있으며 두 개의 전원을 사용하는 경우에는 $-55\sim 150^{\circ}\text{C}$ 범위의 온도를 측정할 수 있다. 한 개의 전원을 사용하는 경우 전형적인 연결 방법은 그림 8과 같다.

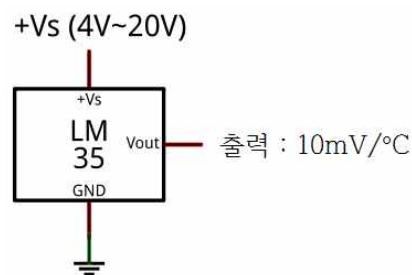
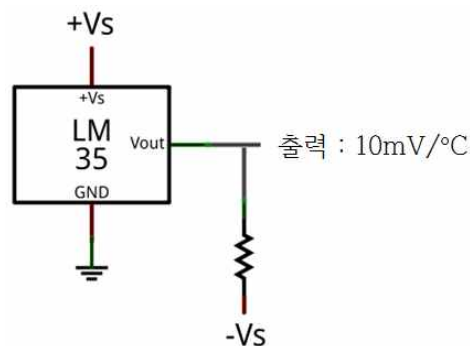


그림 8. 한 개 전원을 사용하는 경우 LM35 온도 센서 연결

그림 8의 경우 출력은 1°C 에 10mV 씩 증가한다. 두 개의 전원을 사용하는 경우 전형적인 연결 방법은 그림 9와 같다.



스케치 4. 온도 센서

```
int temp_pin = A1;           // 온도 센서 연결 핀
int input_value;             // 아날로그 입력값
float input_voltage;         // 아날로그 입력값을 전압으로 환산한 값
float temperature;           // 아날로그 입력값을 온도로 환산한 값

void setup() {
    Serial.begin(9600);
}

void loop() {
    input_value = analogRead(temp_pin);

    // 0에서 1023 사이의 값을 0V에서 5V 사이 값으로 변환한다.
    input_voltage = 5.0 * input_value / 1023.0;

    // 10mV에 1°C이므로 100을 곱해서 현재 온도를 얻는다.
    temperature = input_voltage * 100.0;

    Serial.println(temperature);

    delay(1000);
}
```



그림 11. 스케치 4 실행 결과

온도가 높아지면 LED를 통해 경고를 내는 시스템을 만들어보자. 온도가 일정 온도 이상인 경우에는 계속해서 LED를 깜빡거리게 하고, 일정 온도 미만인 경우에는 LED가 꺼진 상태에 있도록 한다. LED 점멸 속도는 0.5초로 하고 디지털 13번 핀에 연결된 LED를 사용하자.

스케치 5. 고온 경고 시스템

```
int temp_pin = A1;           // 온도 센서 연결 핀
int led_pin = 13;
int input_value;             // 아날로그 입력값
float input_voltage;         // 아날로그 입력값을 전압으로 환산한 값
float temperature;           // 아날로그 입력값을 온도로 환산한 값

boolean high_temp = false;   // 고온 상태
boolean led_state = false;   // LED 점멸 상태

unsigned long time_previous, time_current;
int INTERVAL = 500;          // 0.5 초 단위
int THRESHOLD = 29;          // 29도 이상이면 고온 상태

void setup() {
```

```
Serial.begin(9600);
pinMode(led_pin, OUTPUT);
}

void loop() {
    input_value = analogRead(temp_pin);

    // 0에서 1023 사이의 값을 0V에서 5V 사이 값으로 변환한다.
    input_voltage = 5.0 * input_value / 1023.0;

    // 10mV에 1°C이므로 100을 곱해서 현재 온도를 얻는다.
    temperature = input_voltage * 100.0;

    Serial.print(temperature);
    Serial.print(", LED ");

    if(temperature < THRESHOLD) {          // 저온 상태인 경우
        time_previous = millis();
        led_state = false;
        Serial.println("off...");
    }
    else{                                  // 고온 상태인 경우
        time_current = millis();
        if(time_current - time_previous >= INTERVAL){    // 점멸
            time_previous = time_current;
            led_state = !led_state;
            if(led_state) Serial.println("ON !!");
            else Serial.println("OFF...");
        }
    }

    digitalWrite(led_pin, led_state);

    delay(500);
}
```

```
}
```

스케치 5에서 LED가 꺼진 상태는 두 가지가 있다. 저온 상태인 경우에 LED는 꺼져 있지만 고온 상태인 경우에도 LED는 0.5초 간격으로 점멸하므로 꺼져 있는 경우가 있다. 따라서 저온 상태인 경우에는 소문자 'off'로, 고온 상태에서 LED가 꺼진 경우에는 대문자 'OFF'로 구별하여 시리얼 모니터로 출력하도록 하였다.



그림 12. 스케치 5 실행 결과

3. 센서 응용

온도 센서와 조도 센서를 이용하여 활동하기 좋은 정도를 ' 좋음, 보통, 나쁨'의 세 가지로 구별하여 텍스트 LCD에 나타내도록 해보자. 두 개의 센서에서 읽어 들인 값은 3개의 구간으로 나누고 각 센서값 구간 조합에 따라 현재 상태를 표 1과 같이 정의하였다. 각 구간에 해당하는 센서값은 임의로 정하였다.

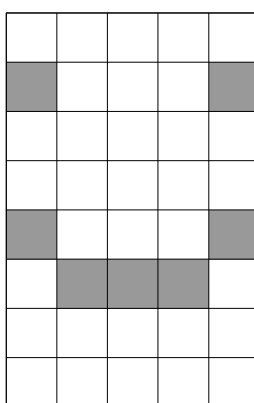
온도 조도	저온 ($27^{\circ} \geq T$)	중온 ($27^{\circ} < T \leq 29^{\circ}$)	고온 ($T > 29^{\circ}$)
낮은 조도 ($500 \geq L$)	나쁨	보통	나쁨
중간 조도 ($500 < L \leq 900$)	보통	좋음	나쁨
높은 조도 ($900 < L$)	보통	좋음	나쁨

표 1. 센서 값에 따른 활동 적합 상태

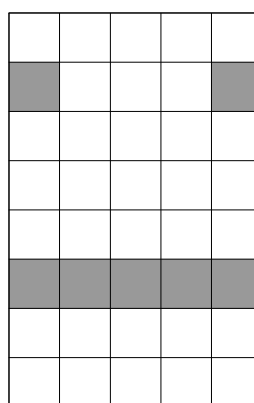
표 1의 상태 테이블은 3×3 2차원 배열로 정의하여 저장하였다.

```
byte state_table[3][3] = {
  {-1, 0, -1},
  {0, 1, -1},
  {0, 1, -1}};
```

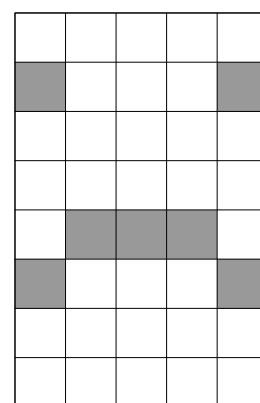
각 상태는 텍스트 LCD에 사용자 문자를 정의하여 표시하도록 하였다. 정의한 사용자 문자는 다음과 같다.



(a) 좋음



(b) 보통



(c) 나쁨

그림 13. 상태 표시 사용자 문자

```

byte condition_good[8] = {
    B00000, B10001, B00000, B00000, B10001, B01110, B00000, B00000
};
byte condition_normal[8] = {
    B00000, B10001, B00000, B00000, B00000, B11111, B00000, B00000
};
byte condition_bad[8] = {
    B00000, B10001, B00000, B00000, B01110, B10001, B00000, B00000
};

```

텍스트 LCD와 온도 센서 및 조도 센서를 연결해 보자. 텍스트 LCD 연결은 12장의 텍스트 LCD 연결을 참고하고, 온도 및 조도 센서는 그림 3과 그림 10을 참고한다.

스케치 6. 활동 적합도 안내 시스템

```

#include <LiquidCrystal.h>

// 핀 번호 (RS, E, DB4, DB5, DB6, DB7)
LiquidCrystal lcd(2, 3, 5, 6, 7, 8);          // LCD 연결

byte state_table[3][3] = {                  // 상태 테이블
    {-1, 0, -1},
    {0, 1, -1},
    {0, 1, -1}};

// 사용자 정의 문자 데이터
byte condition_good[8] = {
    B00000, B10001, B00000, B00000, B10001, B01110, B00000, B00000
};
byte condition_normal[8] = {
    B00000, B10001, B00000, B00000, B00000, B11111, B00000, B00000
};
byte condition_bad[8] = {
    B00000, B10001, B00000, B00000, B01110, B10001, B00000, B00000
};

```

```
};

void setup() {
    lcd.createChar(0, condition_good);    // 사용자 정의 문자 1 생성
    lcd.createChar(1, condition_normal);  // 사용자 정의 문자 2 생성
    lcd.createChar(2, condition_bad);     // 사용자 정의 문자 3 생성

    lcd.begin(16, 2);                    // LCD 크기 설정
    lcd.clear();                         // LCD 화면 지우기
}

void loop() {
    int light = analogRead(A2);          // 조도 센서
    int val = analogRead(A1);            // 온도 센서
    float temperature = 500.0 * val / 1023.0;

    int row, col;
    if(light <= 500) row = 0;             // 조도 구간 나눔
    else if(light <= 900) row = 1;
    else row = 2;

    if(temperature <= 27) col = 0;        // 온도 구간 나눔
    else if(temperature <= 29) col = 1;
    else col = 2;

    int state = state_table[row][col];    // 현재 상태 결정

    lcd.home();
    lcd.print("Temp.");
    lcd.print(temperature);              // 첫째 줄에 온도 출력
    lcd.setCursor(0, 1);
    lcd.print("Light");
    lcd.print(light);                   // 둘째 줄에 조도 출력

    lcd.setCursor(15, 0);
    switch(state){                      // 사용자 정의 문자 출력
```

```
case 1:  
  lcd.write(byte(0));  
  break;  
case 0:  
  lcd.write(byte(1));  
  break;  
case -1:  
  lcd.write(byte(2));  
  break;  
}  
  
delay(500);  
}
```



그림 14. 스케치 6 실행 결과