

3. Serial 및 String 클래스

아두이노는 C/C++ 언어를 기반으로 하고 있다. 스케치는 `setup`과 `loop` 함수를 기반으로 구성되며 간단한 스케치의 경우 C 스타일로 함수 중심으로 작성되는 것이 일반적인 반면 아두이노에 연결된 주변 장치 제어를 위한 라이브러리들은 C++ 스타일로 객체 중심으로 작성되는 것이 일반적이다. 또한 대부분의 스케치에서 사용하는 직렬 통신과 문자열을 다루기 위해 아두이노는 `Serial`과 `String` 클래스를 기본 요소로 포함하고 있다. 한 가지 유의할 점은 아두이노 우노에 사용되는 ATmega328 마이크로컨트롤러의 경우 하나의 하드웨어 기반 시리얼 연결만을 지원한다는 점이다. 즉, 시리얼 연결을 위한 클래스에서 생성될 수 있는 객체는 오직 하나 뿐이므로 클래스와 객체의 구별이 모호하다는 점이다. 실제로 시리얼 연결을 위한 클래스의 이름은 '`Serial_`'이며 그 유일한 객체가 '`Serial`'로 미리 정의되어 있어 일반적으로 `Serial`을 클래스이면서 객체로 간주하여 설명하고 있다. 반면 문자열은 스케치 내에서 두 개 이상이 사용되는 경우가 일반적이므로 `String` 클래스의 객체는 사용자가 직접 지정하여 사용한다. 아두이노 우노의 경우 `Serial_`과 `Serial`을 구별해야 할 필요는 없으므로 이 책에서도 `Serial`을 클래스 이름이면서 동시에 객체의 이름으로 혼용하여 사용한다.

1. Serial 클래스

`Serial`은 아두이노와 컴퓨터 및 기타 주변 장치 사이의 시리얼 통신을 담당하는 클래스이다. 시리얼 통신은 RS-232C 프로토콜의 정의 중 RXD (receive data 또는 RX), TXD (transmit data 또는 TX), GND (ground) 세 개의 핀만을 사용하여 통신을 수행하는 방식으로 흔히 USART (Universal Synchronous Asynchronous Receiver Transmitter) 또는 UART라고 불린다. 그림 1은 UART 시리얼 통신을 이용하기 위해 컴퓨터와 아두이노 보드를 연결한 예로 송신과 수신 선이 서로 교차하여 연결되어 있는 점에 유의하여야 한다. 또 한 가지 유념해야 할 것은 그림 1에서 컴퓨터와 아두이노를 직접 연결하는 것처럼 그려져 있지만 컴퓨터의 D-SUB 연결은 RS-232C 프로토콜을 사용하므로 USART 프로토콜을 사용하는 아두이노와 직접 연결은 불가능하며 변환기가 필요하다는 점이다.

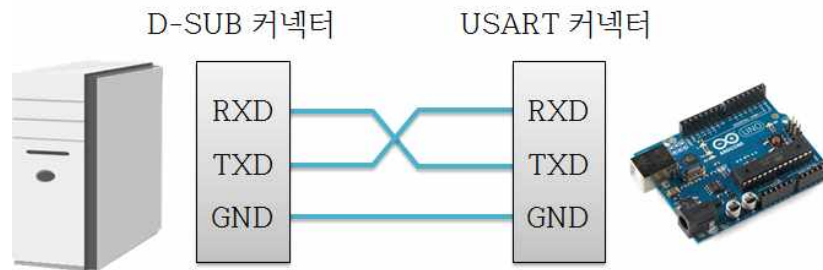


그림 1. 컴퓨터와 마이크로컨트롤러의 UART 연결¹⁾

아두이노 우노에서 시리얼 통신을 위한 핀은 디지털 0번 (RX) 핀과 1번 (TX) 핀으로 이들 핀을 통해 이루어지는 UART 시리얼 통신을 담당하는 클래스가 Serial 클래스이다. 특히 아두이노가 컴퓨터와 USB를 통해 연결된 경우 시리얼 통신을 통해 송수신되는 데이터는 아두이노의 시리얼 모니터를 통해 확인할 수 있다. 시리얼 모니터는 글자 그대로 시리얼 연결을 통해 송수신되는 데이터를 모니터링 하는 프로그램으로 기본적으로 아두이노를 컴퓨터와 연결하였을 때 장치 관리자에 나타나는 COM 포트를 모니터링 한다. 2장에서 이미 시리얼 모니터를 콘솔처럼 사용하여 출력을 확인하는 용도로 사용할 수 있음을 살펴보았다. 툴바의 가장 오른쪽에 있는 버튼을 누르거나 ‘두고 → 시리얼 모니터’ 메뉴 또는 ‘Ctrl + Shift + M’ 단축키를 통해 시리얼 모니터를 실행시켜 보자. 문자열 송수신을 위해 사용할 수 있는 영역 이외에도 아래쪽에 두 개의 콤보 박스를 확인할 수 있다. 첫 번째 콤보 박스는 문자열 전송시에 문자열 끝에 추가할 문자를 선택할 수 있도록 해준다. 디폴트값으로 추가 문자는 없는 것으로 설정되어 있지만 필요한 경우 Newline (ASCII Code 10, 또는 Line Feed), Carriage Return (ASCII Code 13) 또는 두 문자 모두를 추가할 수 있다. 두 번째 콤보 박스는 데이터 변조 속도를 설정하는 콤보 박스로 디폴트값은 9600보율(baud rate)로 설정되어 있다. 시리얼 통신의 경우 송신측과 수신측이 동일한 속도로 설정되어야만 정확한 데이터 전달이 가능하므로 반드시 아두이노와 연결된 장치의 보율을 일치시켜야 한다. 또한 문자열의 끝을 표시하기 위해 개행문자를 사용하는 경우가 많으므로 필요에 따라 추가 문자를 선택하면 된다.

1) 이미지 출처 : <http://arduino.cc>

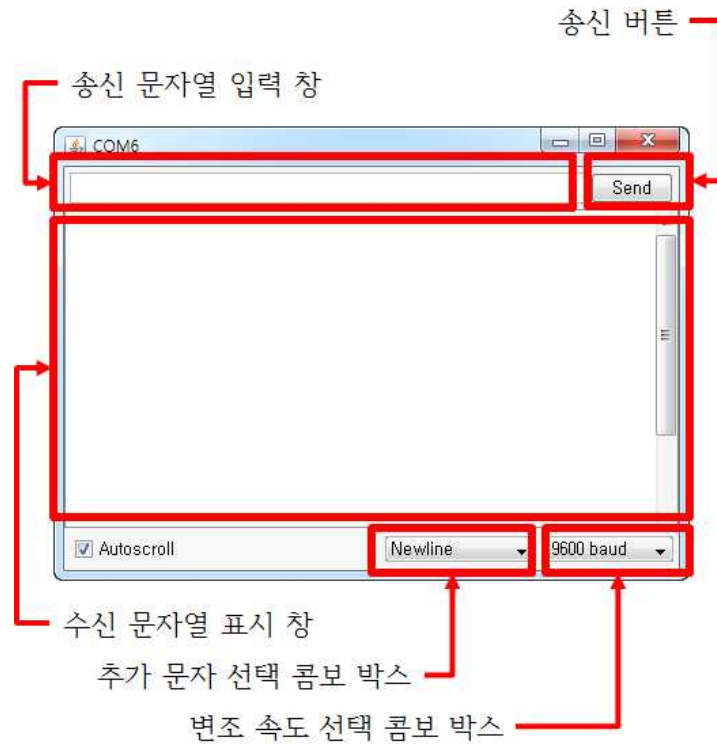


그림 2. 시리얼 모니터 프로그램

먼저 Serial 클래스를 통해 컴퓨터로 데이터를 전송하는 방법을 살펴보자. 데이터 전송에 사용할 수 있는 멤버 함수에는 문자열 기반의 `print` 및 `println` 함수와 이진 데이터 기반의 `write` 함수가 있다. `println` 함수는 전송되는 문자열 끝에 개행문자를 추가하는 점을 제외하면 `print` 함수와 동일하다. `print` 함수는 인자로 주어지는 내용을 문자열로 변환하여 출력하므로 문자나 문자열뿐만 아니라 정수와 실수 역시 문자열로 출력할 수 있다. 반면 `write` 함수는 바이트 단위로 데이터를 출력한다. 예를 들어 정수 65를 `print` 함수로 출력하는 경우 시리얼 모니터에는 문자열 '65'가 출력되는 반면 `write` 함수로 출력하는 경우 시리얼 모니터에는 65에 해당하는 ASCII 문자인 'A'가 출력된다.

```
size_t print(value, format)
```

value : 출력값 (char, char 배열, String, 정수, 실수 등)

format : 출력 형식

반환값 : 직렬 포트로 출력된 바이트 수

```
size_t write(uint8_t ch)
    ch : 출력할 바이트 단위 데이터
    반환값 : 출력한 바이트 수
```

코드 1은 print 및 write 함수를 사용하여 다양한 유형의 데이터를 시리얼 모니터로 출력하는 예이다.

코드 1. print 함수와 write 함수

```
void setup() {
    Serial.begin(9600);                // 직렬 포트 초기화
}

void loop() {
    Serial.print("String  : ");
    Serial.println("Test String");    // 문자열 출력

    Serial.print("Char   : ");
    Serial.println('c');              // 문자 출력

    Serial.print("Integer : ");
    Serial.println(123);              // 정수 출력

    Serial.print("Float  : ");
    Serial.println(3.14);             // 실수 출력

    byte data = 65;
    Serial.println();
    Serial.print("With print : ");
    Serial.println(data);             // print 함수

    Serial.print("With write : ");
    Serial.write(data);              // write 함수
```

```
while(true);
}
```

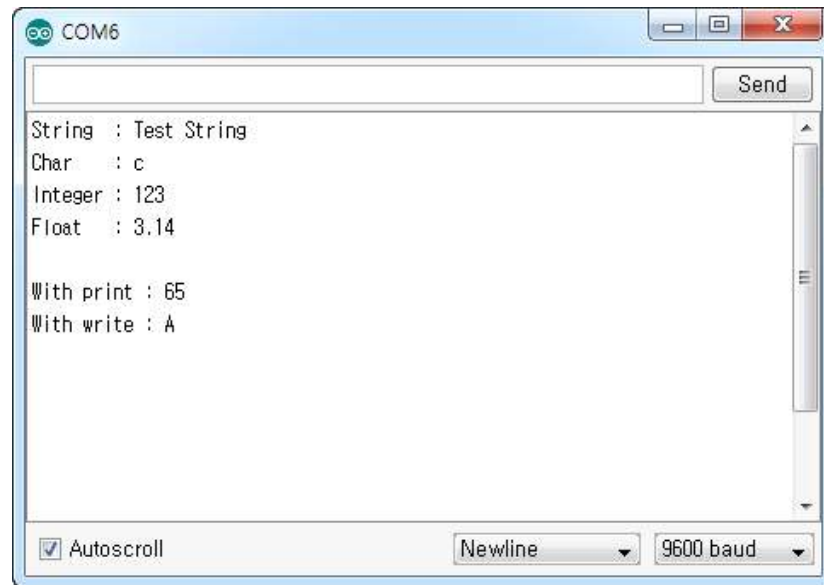


그림 2. 코드 1 실행 결과

숫자를 print 함수로 출력하는 경우에는 두 번째 매개변수를 사용할 수 있으며 두 번째 매개변수는 정수를 출력하는 경우에는 진법을, 실수를 출력하는 경우에는 소수점 이하 자릿수를 지정하기 위해 사용된다. 코드 2는 동일한 값의 정수를 이진수, 십진수 및 십육진수로 출력하는 방법과 실수의 소수점 이하 자릿수를 지정하는 방법을 보여준다. 정수의 경우 디폴트값은 십진수이며 실수의 경우 디폴트값은 소수점 이하 두 자리이다. 진법 지정을 위해서는 미리 정의된 상수값인 BIN (Binary), DEC (Decimal), HEX (Hexadecimal)를 수용할 수 있다.

코드 2. 정수의 진법 변환 및 실수의 소수점 이하 자릿수 지정

```
void setup() {
    Serial.begin(9600);    // 직렬 포트 초기화
}

void loop() {
    int n = 123;
```

```
float f = 3.1415927;

Serial.println(n, BIN);           // 이진수
Serial.println(n, DEC);           // 십진수. 디폴트값
Serial.println(n, HEX);           // 십육진수

Serial.println();
Serial.println(f);                 // 디폴트값으로 소수점 이하 두 자리
Serial.println(f, 5);             // 소수점 이하 다섯 자리

while(true);
}
```

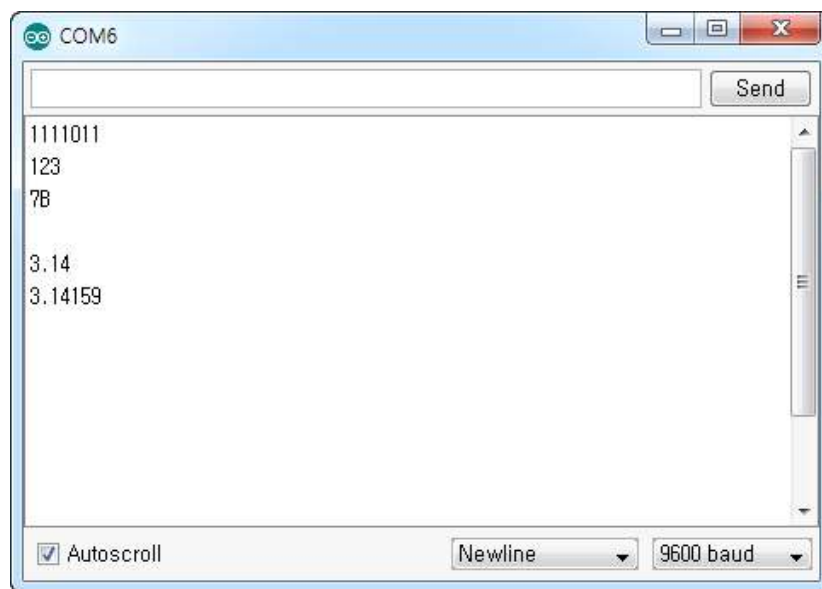


그림 3. 코드 2 실행 결과

UART 시리얼 통신은 양방향 통신이므로 시리얼 모니터를 포함하여 디지털 0번과 1번을 통해 연결된 시리얼 장치로 데이터를 전송하는 함수(print, write)와 더불어 수신을 위한 함수들도 마련되어 있다. 수신을 위한 함수에는 수신된 함수가 존재하는지의 여부를 알아보는 available, 수신된 데이터를 바이트 단위로 읽어오는 read와 peek 함수가 있다. 수신된 데이터는 수신 버퍼에 저장되며 available 함수는 버퍼에 읽지 않고 저장되어 있는 데이터의 바이트 수를 반환한다. read 함수의 경우 읽어온 데이터를 버퍼에서 삭제하는 반면 peek 함수는

버퍼에서 삭제하지 않는 차이가 있다.

```
int available(void)
```

반환값 : 직렬 포트 수신 데이터 버퍼에 저장된 데이터의 바이트 수

```
int peek(void)
```

반환값 : 직렬 통신 수신 버퍼의 첫 번째 바이트 데이터 또는 -1

```
int read(void)
```

반환값 : 직렬 통신 수신 버퍼의 첫 번째 문자 또는 -1

코드 3은 시리얼 모니터에서 입력한 문자를 아두이노에서 수신하고 다시 이를 시리얼 모니터로 재전송하여 출력하는 예이다. 송신 문자열 입력 창에 문자열을 입력하고 전송 버튼을 누르면 한 글자 단위로 재전송되어 돌아오는 것(에코 백, echo back)을 확인할 수 있다.

코드 3. 시리얼 모니터에서 전송된 데이터의 에코 백

```
void setup() {  
    Serial.begin(9600);    // 직렬 포트 초기화  
}  
  
void loop() {  
    if(Serial.available() > 0){  
        byte data = Serial.read();  
  
        Serial.print("Echo back : ");  
        Serial.write(data);  
        Serial.print(" ");  
        Serial.println(data);  
    }  
}
```

```
}
```

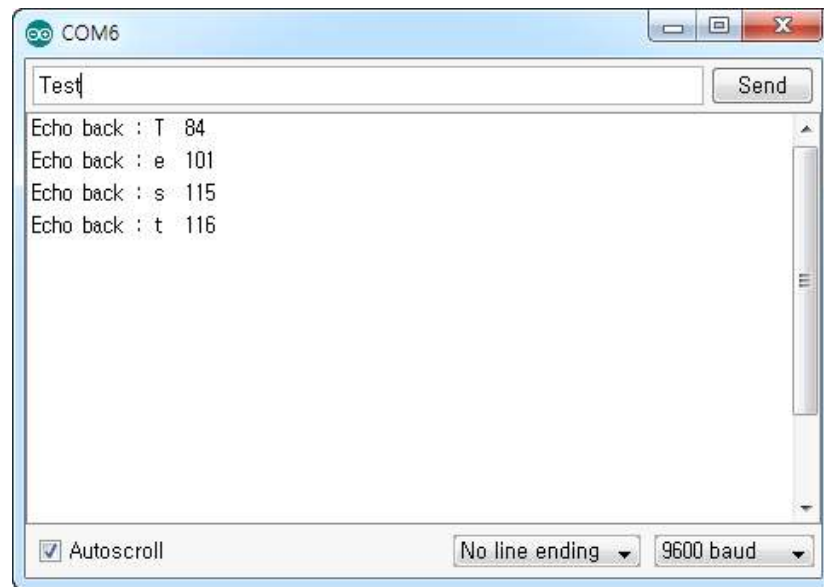


그림 4. 코드 3 실행 결과

2. String 클래스

String 클래스는 문자열을 다루기 위한 클래스이다. C에서는 문자열을 다루기 위한 별도의 데이터 형이 존재하지 않으므로 문자 배열로 문자열을 처리한다. 문자 배열로 문자열을 다루는 경우 적은 메모리만을 필요로 하는 장점은 있지만 번거롭고 많은 주의를 필요로 한다. 반면 C++에서는 문자열을 위한 전용 클래스를 제공하고 있으며 아두이노 역시 문자열을 위한 전용 클래스인 String 클래스를 제공하고 있다. String 클래스를 사용하면 문자열은 물론 숫자도 문자열로 쉽게 변환하여 새로운 문자열로 만들 수 있다. 하지만 실수의 경우에는 문자열로 변환할 수 없으므로 주의하여야 한다. 코드 4는 문자열을 생성하고 기존의 문자열 또는 다른 유형의 데이터를 이용하여 새로운 문자열을 생성하는 방법을 보여준다. 문자열의 출력을 위해서는 Serial 클래스의 print 함수를 사용할 수 있다.

코드 4. 문자열 생성 방법

```
void setup() {
```

```
Serial.begin(9600);    // 직렬 포트 초기화
}

void loop() {
    String str1 = "One string", str2 = "Another string";
    int n = 1234;
    float f = 3.14;
    char c = 'A';

    Serial.println(str1);           // String 출력
    Serial.println(str1 + " " + str2); // 문자열 연결

    Serial.println(String(n));      // 정수로부터 10진 문자열 생성
    Serial.println(String(n, BIN)); // 정수로부터 2진 문자열 생성
    Serial.println(String(n, HEX)); // 정수로부터 16진 문자열 생성

    // 실수의 경우 문자열로 직접 변환할 수 없다.
    // 단, Serial 클래스의 print 함수로 문자열 형식으로 출력은 가능하다.
    // Serial.println(String(f));
    Serial.println(f);

    // 다른 타입의 데이터를 연결하여 새로운 String 객체를 생성한다.
    Serial.println("String + integer : " + String(n));
    Serial.println("String + character : " + String(c));

    String str3 = "String + character : ";
    str3 += n;
    Serial.println(str3);

    while(true);
}
```

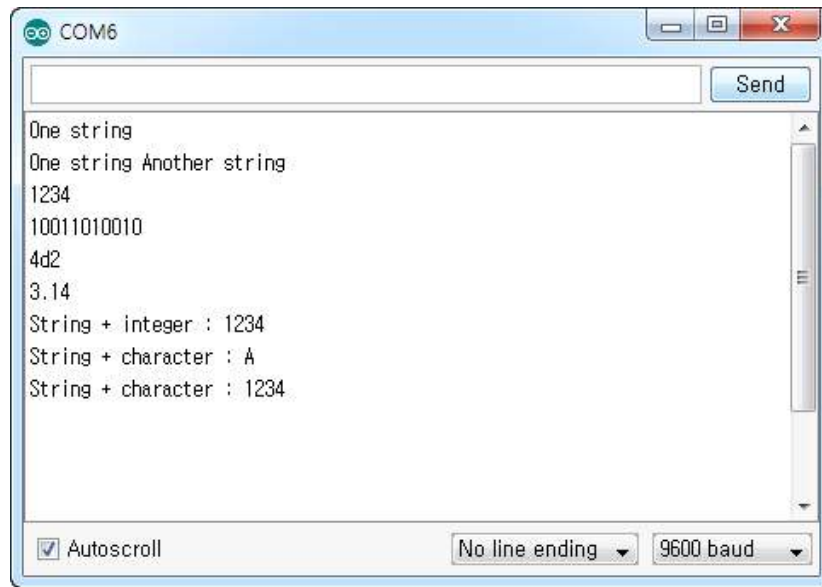


그림 5. 코드 4 실행 결과

코드 4에서 한 가지 눈여겨 볼 문장은 문자열과 정수를 연결하는 부분이다.

```
// char 배열 + String 객체 => String 객체
Serial.println("String + integer : " + String(n));
// char 배열 + 정수 => ??
Serial.println("String + integer : " + n);
// String 객체 + 정수 => String 객체
String str3 = "String + character : ";
str3 += n;
Serial.println(str3);
```

첫 번째와 세 번째 println 함수는 의도한 바와 같이 문자열에 정수값이 문자열로 변환되어 이어진다. 하지만 두 번째 println 문장의 경우 예상치 못한 결과가 나올 것이다. 이는 연결하고자 하는 두 데이터 중 하나는 String 클래스의 객체여야만 원하는 결과가 나온다는 것을 보여준다. 또한 큰 따옴표로 둘러싸여진 문자열은 String 객체가 아니라 char 배열로 다루어진다는 사실이다.

String 클래스에는 문자열의 비교 및 검색 등을 위해 다양한 멤버함수가 제공되고 있다. 흔히 사용되는 멤버 함수에는 사전 순서에 따른 정렬을 위한 compareTo 함수와 두 문자의 내용

일 동일한지 비교하는 equals 및 equalsIgnoreCase 함수가 있다. equalsIgnoreCase 함수는 알파벳 비교 시에 대소문자를 구별하지 않는 것을 제외하고는 equals 함수와 동일하다. compareTo 함수는 두 개의 문자열을 비교하여 배열 순서에 따라 플러스(+), 영(zero), 또는 마이너스(-) 값을 반환한다.

```
int compareTo(const String &string2)
```

string2 : 비교 대상이 되는 문자열

반환값 : 순서상으로 string2가 먼저 나오면 양의 값, string2와 동일하면 0, string2가 뒤에 나오면 음의 값을 반환

코드 5는 다섯 개의 문자열을 비교하여 오름차순으로 정렬한 결과를 보여주는 코드의 예이다. 문자열 비교 함수의 결과로 반환되는 값의 절대값도 의미가 있지만 반환되는 값의 부호만을 사용하여 순서를 정하는 것이 일반적이다.

코드 5. 문자열 정렬

```
void setup() {
    Serial.begin(9600);    // 직렬 포트 초기화
}

void loop() {
    // 정렬할 문자열 배열
    String str[5] = {"ABC", "XYZ", "LMN", "DEF", "123"};

    // 문자열 정렬
    for(int i = 0; i < 4; i++){
        for(int j = i + 1; j < 5; j++){
            int compare = str[i].compareTo(str[j]);
            if(compare > 0){           // 오름차순으로 정렬
                String temp = str[i];
                str[i] = str[j];
                str[j] = temp;
            }
        }
    }
}
```

```
    }  
}  
  
// 정렬된 문자열 출력  
for(int i = 0; i < 5; i++){  
    Serial.println(String(i) + " : " + str[i]);  
}  
  
while(true);  
}
```

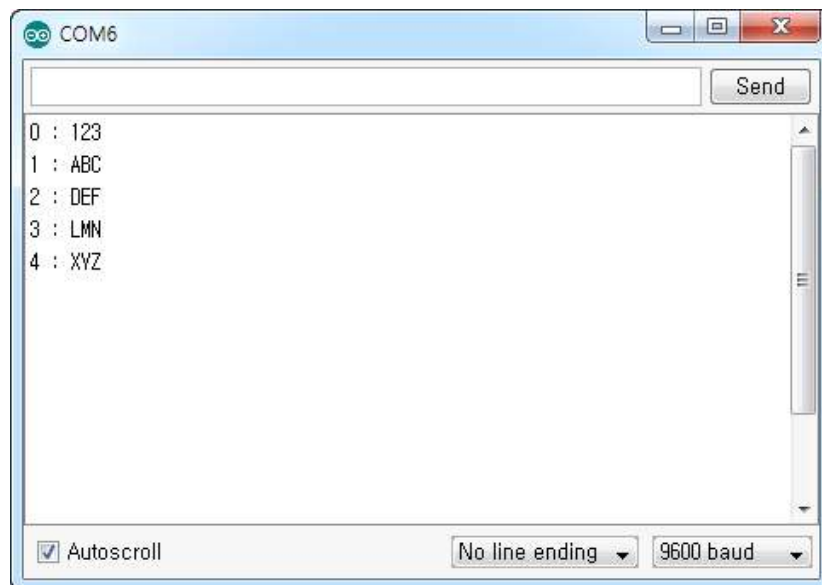


그림 6. 코드 5 실행 결과

문자열 비교 함수로도 동일한 문자열인지의 여부를 판단할 수 있지만 동일한 문자열임을 판단하기 위해 String 클래스에는 대소문자를 구별하는 equals 함수와 대소문자를 구별하지 않는 equalsIgnoreCase 함수를 제공하고 있다.

```
unsigned char equals(const String &str)
```

str : 비교 대상 문자열

반환값 : 문자열이 str과 동일한 경우 true, 아닌 경우 false를 반환

```
unsigned char equalsIgnoreCase(const String &str)
```

str : 비교 대상 문자열

반환값 : 문자열이 str과 동일한 경우 true, 아닌 경우 false를 반환

코드 6은 시리얼 모니터로 전송된 문자열이 약속된 문자열 중 하나가 전송되는 경우 그에 따른 메시지를 출력하는 코드의 예이다. 약속된 문자열은 네 방향을 나타내는 “UP”, “DOWN”, “LEFT”, “RIGHT”로 하며 대소문자를 구별하지 않는 equalsIgnoreCase 함수를 사용하였다. 또한 여러 개의 글자로 이루어진 문자를 수신하여야 하므로 전송된 문자열의 끝을 표시하기 위해 시리얼 모니터에서는 ‘new line’ 옵션을 선택하여 개행 문자 ‘\n’이 함께 전송되도록 하였다.

코드 6. 문자열 비교

```
String buffer;                // 수신 문자열 저장 버퍼
boolean stringEnd;           // 문자열이 개행 문자로 끝났는지의 여부 표시

void setup() {
    Serial.begin(9600);       // 직렬 포트 초기화
    // 변수 초기화
    buffer = "";
    stringEnd = false;
}

void loop() {
    while(Serial.available() > 0){ // 수신된 문자가 있을 때
        char ch = Serial.read();   // 문자 읽음
        Serial.write(ch);
        if(ch == '\n'){            // 개행 문자를 만나면 문자열이 완성된 것으로 간주
            stringEnd = true;
            break;
        }
        buffer += ch;              // 개행 문자를 만나지 않으면 문자열을 이어붙임
    }
}
```

```
}

if(stringEnd){           // 문자열이 완성된 경우 약속된 문자열과 비교
    if(buffer.equalsIgnoreCase("UP"))
        Serial.println(" : Up");
    else if(buffer.equalsIgnoreCase("DOWN"))
        Serial.println(" : Down");
    else if(buffer.equalsIgnoreCase("RIGHT"))
        Serial.println(" : Right");
    else if(buffer.equalsIgnoreCase("LEFT"))
        Serial.println(" : Left");

    buffer = "";
    stringEnd = false;
}
}
```

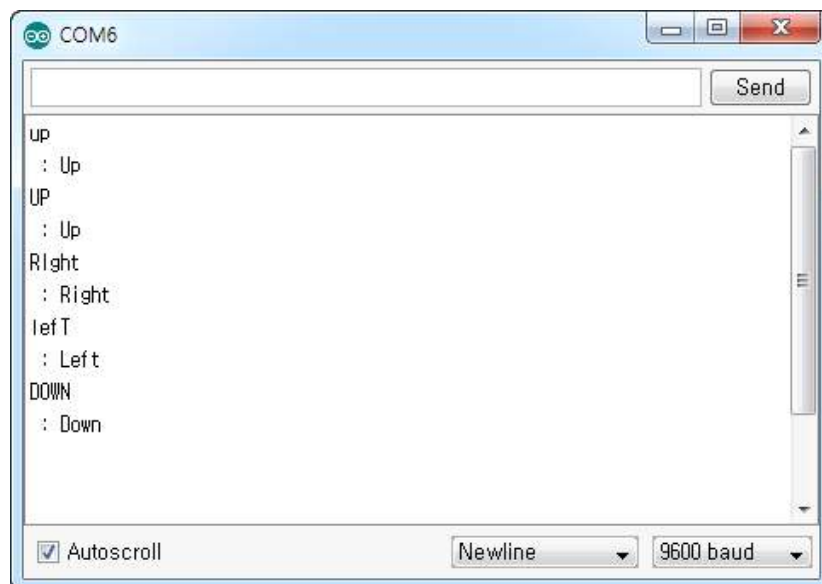


그림 7. 코드 6 실행 결과

이외에도 Serial 클래스와 String 클래스에는 여러 가지 유용한 멤버 함수들이 다수 정의되어 있으며 다른 멤버 함수들은 부록을 참고하기 바란다.