

2. C 언어 연습

아두이노는 마이크로컨트롤러를 이용하여 만들어진 하드웨어 보드와 아두이노 보드에 프로그램을 작성하고 업로드 할 수 있는 소프트웨어인 아두이노 IDE를 통틀어 이르는 말이다. 이 중 아두이노 소프트웨어는 다양한 운영체제에서 동작할 수 있도록 Java를 사용하여 만들어져 있다. 반면 아두이노를 위한 프로그램인 스케치(sketch)는 C/C++을 기반으로 하고 있다. 아두이노는 비전공자들이 간단한 제어 장치를 만들 수 있도록 하기 위해 만들어진 만큼 큰 프로그램을 작성하는 경우는 그리 많지 않다. 또한 아두이노는 다양한 주변장치들을 손쉽게 사용할 수 있도록 라이브러리가 공개되어 있으므로 라이브러리 사용을 위한 간단한 코드만으로도 다양한 프로그램을 작성할 수 있다. 하지만 기본적인 프로그램 작성을 위해서는 C 언어의 기본 문법을 숙지하고 있어야 하며 다양한 라이브러리를 활용하거나 자신만의 라이브러리를 작성하여 배포하고자 한다면 C++ 언어의 클래스 개념에 대하여 이해하고 있어야 한다. 이 장에서는 아두이노를 위한 프로그램 작성에 필요한 C/C++ 언어의 기본적인 내용들을 살펴본다. 이 책은 C/C++ 학습을 위한 책은 아니므로 흔히 사용되는 문법 위주로 개괄적으로 살펴볼 것이므로 보다 자세한 내용은 C/C++ 관련 책을 참고하여야 한다.

1. C/C++ 언어 연습을 위한 환경

아두이노 프로그램에는 시리얼 모니터(serial monitor) 프로그램이 포함되어 있다. 시리얼 모니터는 컴퓨터와 아두이노가 연결된 직렬 통신 포트를 통해 주고받는 데이터를 확인할 수 있도록 기능을 제공하므로, 컴퓨터에서 아두이노로 문자열을 전달할 수 있고 아두이노에서 직렬 포트를 통해 전송된 문자열을 확인할 수 있다. 특히 아두이노로부터의 직렬 데이터 출력은 프로그램 실행 단계별로 메시지 출력을 통해 디버깅 용도로 흔히 사용되므로 일반적인 프로그래밍에서 콘솔(console)과 유사한 용도로 활용할 수 있다. 따라서 이 장에서의 모든 프로그램은 시리얼 모니터를 통해 그 실행 결과를 확인하도록 한다. 시리얼 모니터는 아두이노 프로그램 툴바의 가장 오른쪽에 있는 버튼을 누르거나 ‘두고 → 시리얼 모니터’ 메뉴 또는 ‘Ctrl + Shift + M’ 단축키를 통해 실행시킬 수 있다.

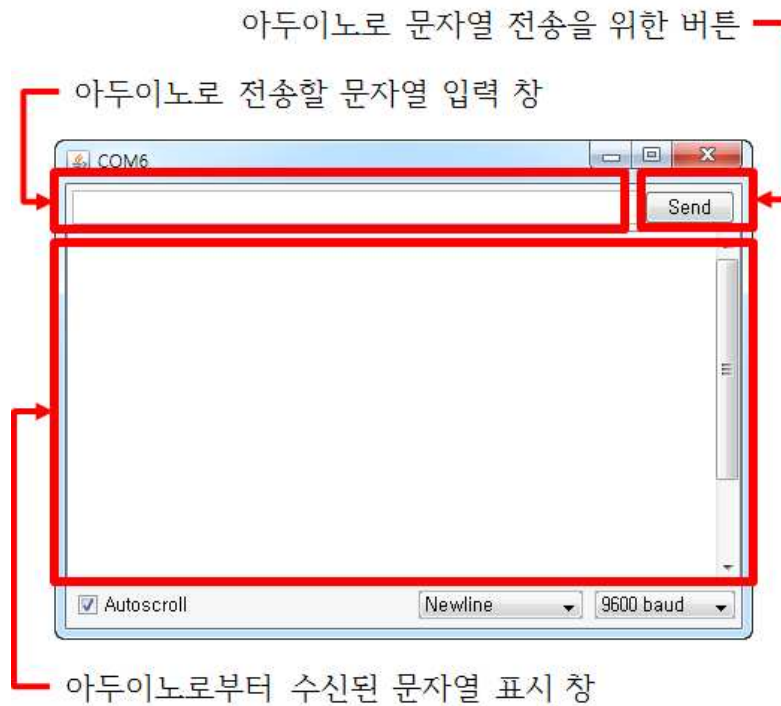


그림 1. 시리얼 모니터 프로그램 실행 화면

시리얼 모니터를 콘솔 용도로 사용하기 위해서 아두이노 스케치에서는 줄바꿈이 없는 출력 함수인 `Serial.print` 함수와 줄바꿈이 있는 출력 함수인 `Serial.println` 함수를 사용한다. 두 함수는 문자열뿐만 아니라 숫자 역시 문자열로 변환하여 출력할 수 있다. 문자열의 조작이 필요한 경우에는 `String` 클래스를 사용한다. `Serial` 및 `String` 클래스에 대한 보다 자세한 내용은 다음 장에서 다룰 것이며 각 클래스에서 사용 가능한 멤버 함수는 부록의 멤버 함수 목록을 참고하면 된다.

C/C++ 언어 연습을 위한 기본 코드는 코드 1과 같다. 코드 1에서 밑줄로 표시된 부분은 스케치 실행을 위한 기본 코드 및 시리얼 모니터 초기화를 위한 담당하는 코드로 이 장에서의 모든 스케치에서 공통으로 사용되는 부분이다.

코드 1. C/C++ 언어 연습을 위한 기본 코드

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

```

void loop()                // 반복 실행 함수
{
    Serial.println("String");    // 문자열 출력

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

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

    while(true):            // 1회 실행 후 대기
}

```

2. 데이터 타입

아두이노는 C/C++ 언어를 기반으로 하고 있으므로 사용할 수 있는 데이터 타입 역시 C/C++ 에서와 동일하다. 다만 아두이노 우노에 사용하는 ATmega328 마이크로컨트롤러가 8비트 마이크로컨트롤러이므로 일반적으로 컴퓨터에서 사용하는 인텔 CPU에 비해 각 유형이 필요로 하는 메모리 바이트 수는 작거나 같다. 표 1은 아두이노에서 사용 가능한 데이터 타입 및 아두이노 우노에서 필요로 하는 바이트 수를 나타낸 것이다.

데이터 타입	크기 (byte)	설명	비고
boolean	1	논리형	true 또는 false
char	1	문자형	
byte	1	부호 없는 정수형	
int	2	정수형	
word	2	부호 없는 정수형	
long	4	정수형	
short	2	정수형	
float	4	단정도 실수형	
double	4	배정도 실수형	

표 1. 데이터 타입

3. 연산자

아두이노에서 사용 가능한 연산자 역시 C/C++에서와 동일한 산술 연산자, 비교 연산자, 논리 연산자 그리고 비트 연산자를 사용할 수 있다.

연산자	의미	사용 예	비고
+	더하기	<code>a = b + 3;</code>	
-	빼기	<code>a = b - 3;</code>	
*	곱하기	<code>a = b * 3;</code>	
/	나누기	<code>a = b / 3;</code>	
%	나머지	<code>a = b % 3;</code>	정수형만 가능
=	대입		비교 연산자 '=='와 구별됨

표 2. 산술 연산자

연산자	의미	수학식
<code>x > y</code>	크다	$x > y$
<code>x >= y</code>	크거나 같다	$x \geq y$
<code>x < y</code>	작다	$x < y$
<code>x <= y</code>	작거나 같다	$x \leq y$
<code>x == y</code>	같다	$x = y$
<code>x != y</code>	다르다	$x \neq y$

표 3. 비교 연산자

논리 연산자	의미	사용 예
<code>&&</code>	AND	<code>(a > 100) && (a < 200)</code>
<code> </code>	OR	<code>(a == 100) (b == 200)</code>
<code>!</code>	NOT	<code>!(a < 100)</code>

표 4. 논리 연산자

구분	연산자	종류	결과
비트 논리 연산자	a & b	비트 AND	a와 b의 비트 단위 AND
	a b	비트 OR	a와 b의 비트 단위 OR
	a ^ b	비트 XOR	a와 b의 비트 단위 XOR
	~a	비트 NOT	a의 비트 단위 NOT (1의 보수)
비트 이동 연산자	a << n	왼쪽으로 이동	a를 n비트 왼쪽으로 이동하고 오른쪽은 0으로 채움
	a >> n	오른쪽으로 이동	a를 n비트 오른쪽으로 이동하고 왼쪽은 0으로 채움

표 5. 비트 연산자

이외에도 기본 연산자들을 결합하여 축약한 형태인 복합 연산자들도 사용 가능하다. 이 중 스케치에서는 일반적인 프로그램에서와 달리 비트 연산자가 많이 사용된다. 마이크로컨트롤러 프로그래밍에서는 레지스터를 조작하는 작업이 많으며 레지스터는 비트별로 그 의미가 지정되어 있는 경우가 많으므로 비트 연산을 통해 데이터를 비트 단위로 조작해야 하는 경우를 흔히 볼 수 있다. 코드 2는 바이트 단위의 데이터가 주어진 경우 비트별로 값을 읽어내고, 이를 조작하는 예를 보여준다. 한 가지 주의할 점은 C/C++의 경우 이진수를 표시하기 위한 별도의 표기법을 제공하지는 않지만 스케치에서는 '0b'로 접두어로 사용하는 경우 이진수를 표시할 수 있다는 점이다. 단, 이진수 표현은 8비트 크기의 데이터 표현만을 위해 사용할 수 있다.

코드 2. 비트 연산
<pre> void setup() { Serial.begin(9600); // 컴퓨터와의 시리얼 통신 초기화 } void loop() { byte data = 0b11110101; // 데이터 // 각 위치의 비트값 확인 Serial.print("Bit Extraction : "); </pre>

```
for(int i = 0; i < 8; i++){
    byte mask = 1 << i;
    byte result = (data & mask) >> i;
    Serial.print(result);
}
Serial.println();

// 상위 4비트를 1로 설정
Serial.print("Bit Set      : ");
for(int i = 4; i < 8; i++){
    byte mask = 1 << i;
    data = data | mask;
}
Serial.println(data, BIN);

// 하위 4비트를 반전
Serial.print("Bit Inversion : ");
for(int i = 0; i < 4; i++){
    byte mask = 1 << i;
    data = data ^ mask;
}
Serial.println(data, BIN);

// 하위 4비트를 0으로 설정
Serial.print("Bit Clear    : ");
for(int i = 0; i < 4; i++){
    byte mask = ~(1 << i);
    data = data & mask;
}
Serial.println(data, BIN);

while(true);
}
```

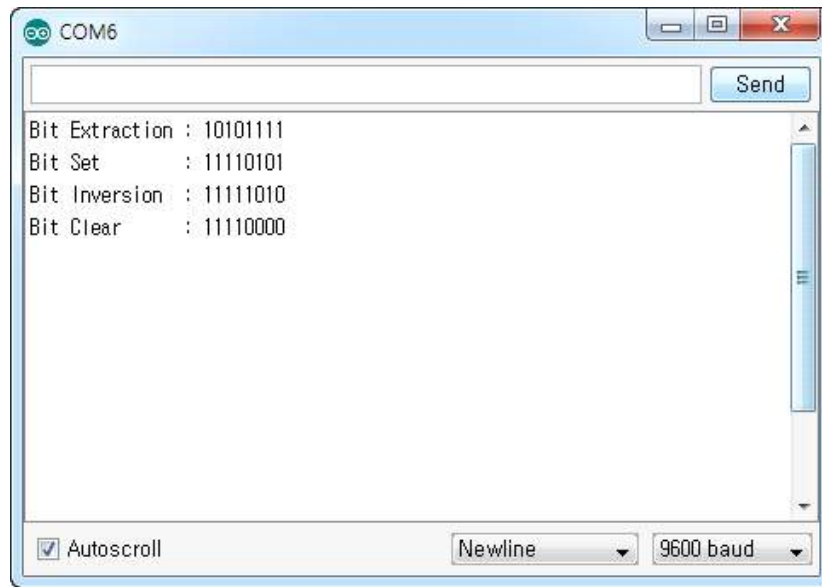


그림 2. 코드 2 실행 결과

4. 조건문과 반복문

C/C++ 언어로 작성한 코드는 위에서 아래로 순차적으로 실행됨을 기본으로 한다. 프로그램 실행 중 실행 순서를 바꾸어야 할 필요가 있는 경우 실행 흐름을 바꾸기 위해 필요한 문장에 조건문과 반복문이 있다. 조건문은 주어진 조건에 따라 특정 블록을 실행하도록 하기 위해 사용되며 if-else 문과 switch-case 문이 조건문에 포함된다. 반복문은 특정 블록을 0회 이상 반복해서 실행하도록 하기 위해 사용되며 while 문, do-while 문, for 문이 반복문에 포함된다.

if-else 문은 boolean 형식으로 주어지는 조건에 따라 조건이 참(true)인 경우와 거짓(false)인 경우의 최대 두 개의 흐름으로 나눌 수 있다. 반면 switch-case 문은 int 형식으로 조건이 주어지므로 3개 이상의 흐름으로 나누는 것이 가능하다. 코드 3은 주어진 두 개의 값 A와 B를 비교하여 두 값의 대소에 따라 다른 메시지를 출력하는 예이다.

코드 3. if-else 문

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

```
void loop()
{
    int A = 20, B = 30;

    if(A > B){
        Serial.println("A is greater than B.");
    }else if(A < B){
        Serial.println("B is greater than A.");
    }else{
        Serial.println("A is equal to B.");
    }

    while(true);
}
```

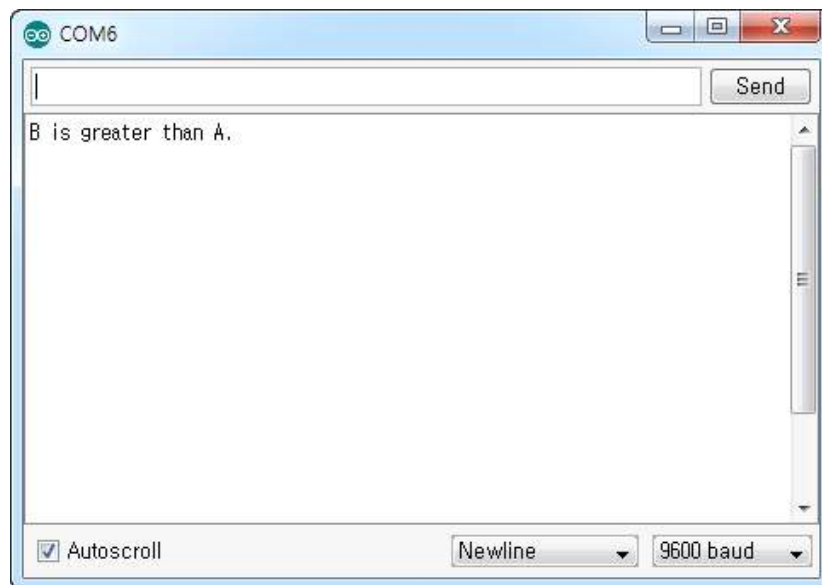


그림 3. 코드 3 실행 결과

코드 3에서와 같이 if-else 문을 반복해서 사용하여 3개 이상의 흐름으로 나누어지는 경우 정확한 조건을 파악하기 어려운 경우가 있다. 이런 경우에는 switch-case 문을 사용할 수 있다. 하지만 코드 3의 경우에는 '크다'에 해당하는 정확한 정수 조건을 파악하기 어려우므로 if-else 문을 사용하는 것이 바람직하다. 물론 차이를 두 수의 차이가 -1, 0, 1의 세 가지로

나오도록 정규화 시켜 사용하면 switch-case 문을 사용할 수도 있으며 이는 연습 문제를 참고하도록 한다. switch-case 문은 입력 값이 한정된 범위의 정수로 주어지는 경우에 적합하다. 코드 4는 변수 A의 값이 네 종류의 과일 중 한 가지로 1에서 4 사이의 값으로 주어지는 경우이다.

코드 4. switch-case 문

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

void loop()
{
    typedef enum { APPLE, PEACH, MELON, BANANA } Fruits;
    Fruits A = PEACH;           // APPLE부터 정수 0의 값이 주어진다.

    switch(A){
        case 0: // case APPLE:      // 0과 APPLE은 동일한 값을 가진다.
            Serial.println("Apple");
            break;
        case 1: // case PEACH:
            Serial.println("Peach");
            break;
        case 2: // case MELON:
            Serial.println("Melon");
            break;
        case 3: // case BANANA:
            Serial.println("Banana");
            break;
    }

    while(true);
}
```

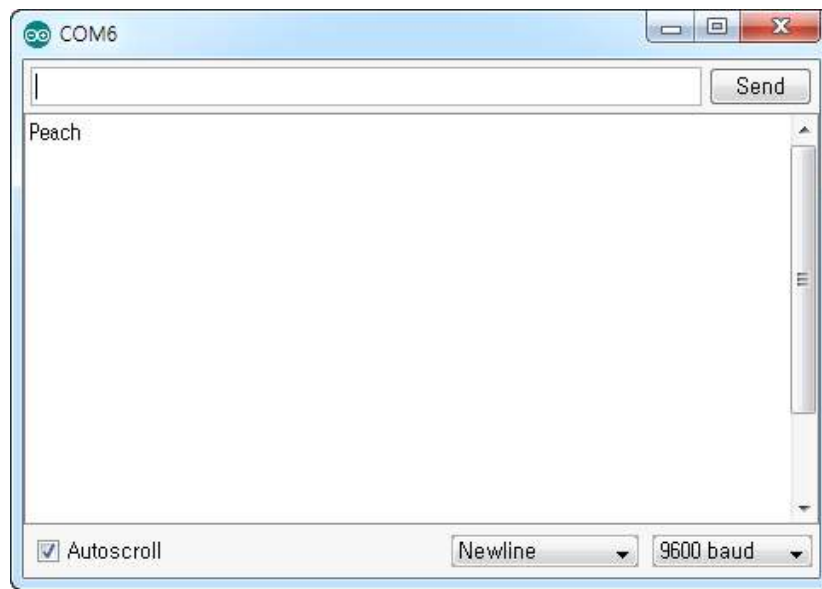


그림 4. 코드 4 실행 결과

반복문에는 주어진 블록의 코드를 1회 이상 반복하는 do-while 문장과 0회 이상 반복하는 while 및 for 문장이 포함된다. 이 중 for 문장의 활용성이 가장 많이 사용되지만 경우에 따라서는 while 문이나 do-while 문으로 보다 간단하게 표현 가능한 경우가 있으므로 세 가지 반복 문장의 차이를 알고 있어야 한다. 코드 5는 1부터 100까지의 정수 중 홀수의 합을 구하는 코드를 세 가지 반복문을 사용하여 작성한 예이다.

코드 5. 반복문

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

void loop()
{
    int sum, i;

    // for 문 사용
    sum = 0;
    for(i = 1; i < 100; i += 2){
```

```
        sum += i;
    }
    Serial.print("for      : ");
    Serial.println(sum);

    // while 문 사용
    sum = 0;
    i = 1;
    while(i < 100){
        sum += i;
        i += 2;
    }
    Serial.print("while    : ");
    Serial.println(sum);

    // do-while 문 사용
    sum = 0;
    i = 1;
    do{
        sum += i;
        i += 2;
    }while(i < 100);
    Serial.print("do-while : ");
    Serial.println(sum);

    while(true);
}
```

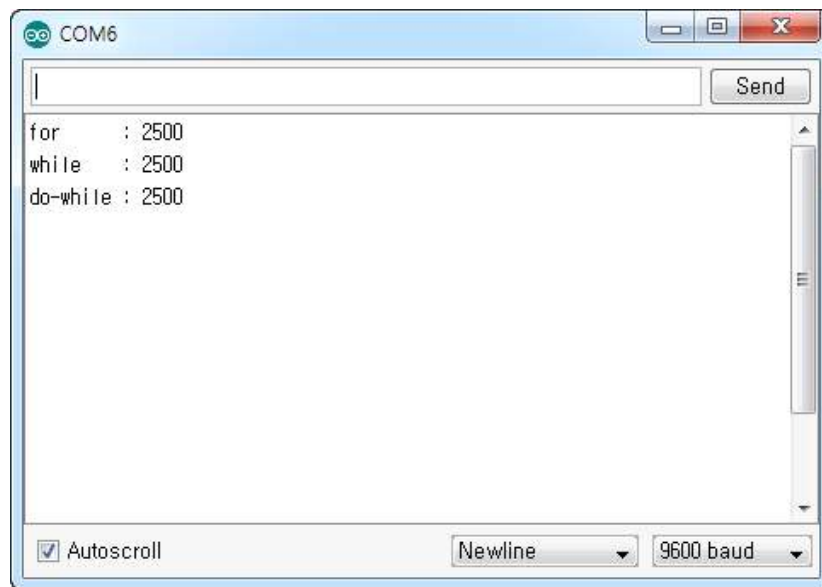


그림 5. 코드 5 실행 결과

5. 함수

C 언어는 함수 중심의 언어이다. 스케치의 경우 `setup` 함수와 `loop` 함수를 반드시 정의하여야 하며 `loop` 함수 내에서 처리해야 하는 내용이 많은 경우 추상적인 기능 분류에 따라 여러 개의 함수로 하향식으로 나누어 작성하고 함수 내에서 이들 함수를 호출하여 사용한다. C 언어에서의 함수는 주어진 입력에 대하여 특정 연산을 수행하고 그 결과를 한 개 이하의 값으로 되돌려 줄 수 있다. C 언어에서의 함수는 선언, 정의, 호출의 세 부분으로 구성되며 함수의 선언이나 정의가 호출 이전에 반드시 나와야 한다. 하지만 스케치의 경우 정의나 선언이 호출 이전에 오지 않아도 오류가 발생하지 않으므로 함수 선언의 번거로움을 피할 수 있다. 코드 6은 1에서 10 사이의 정수에서 소수를 찾아 출력하는 예로 소수 여부를 boolean 값으로 반환하는 함수를 사용하고 있다.

코드 6. 함수를 이용한 소수 판별

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

```
void loop()
{
    for(int i = 1; i <= 10; i++){
        if(is_prime_number(i))
            Serial.println(i);
    }

    while(true);
}

// n이 소수이면 true를, 소수가 아니면 false를 반환
boolean is_prime_number(int n)
{
    int i;
    if(n < 2) return false;

    for(i = 2; i < n; i++){
        if(n % i == 0) break;
    }

    if(i == n) return true;
    else return false;
}
```

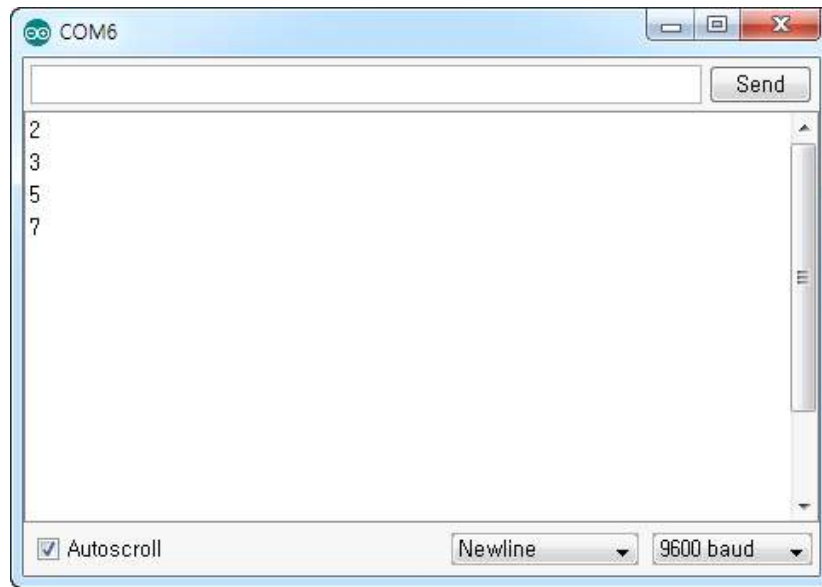


그림 6. 코드 6 실행 결과

6. 클래스

C 언어는 함수 중심의 절차적 프로그래밍 언어인 반면 C++ 언어는 객체 중심의 객체지향 프로그래밍 언어이다. C++ 언어에서 객체는 클래스로 표현되며 클래스의 선언은 헤더 파일(*.h)에, 클래스의 정의는 소스 파일(*.cpp)에 기술되는 것이 일반적이다. 아두이노 설치 디렉터리 아래의 'libraries' 디렉터리에는 라이브러리 별로 별도의 디렉터리가 만들어져 있고 그 아래에는 디렉터리 이름과 동일한 이름의 헤더 파일과 소스 파일이 존재하는 경우가 대부분이다. 예를 들어 기본 라이브러리 중 하나인 소프트웨어 시리얼 라이브러리 디렉터리인 SoftwareSerial 디렉터리 아래에는 SoftwareSerial.h 파일과 SoftwareSerial.cpp 파일을 찾아볼 수 있다. 간단한 라이브러리의 경우 소스 파일 없이 헤더 파일만 존재하는 경우도 있다. 간단한 제어 장치 구성을 위해서는 setup 함수와 loop 함수에 몇 개의 사용자 정의 함수로 충분한 경우가 대부분이지만 연결된 주변 장치를 제어하기 위해서는 라이브러리를 사용하여야 하는 경우가 많으므로 클래스의 구조와 사용법에 익숙해질 필요가 있다.

클래스는 객체의 속성을 저장하기 위한 멤버 변수와 객체의 동작을 표현하는 멤버 함수로 구성된다. 정수를 표현하기 위한 INTEGER라는 클래스를 가정해 보자. 속성으로는 정수값을 저장하기 위한 정수 변수를 멤버 변수로 가질 수 있다.

```
int num;
```

정수값과 관련된 동작은 목적에 따라 다르지만 특정 값으로 나누어지는지의 여부를 알려주는 함수를 멤버 함수로 가질 수 있다.

```
boolean is_dividable_by(int n);
```

이들을 멤버 변수와 멤버 함수로 가지는 클래스는 다음과 같이 정의될 수 있다. 멤버 함수의 정의는 일반적으로 별도의 소스 파일로 작성하지만 이 예에서는 헤더 파일에 함께 작성하였다.

```
class INTEGER{
private:
    int num;
public:
    INTEGER(int n) { num = n; }    // 생성자
    boolean is_dividable_by(int n){
        if(num % n == 0) return true;
        else return false;
    }
}
```

클래스가 정의되면 사용하기 이전에 객체를 생성하여야 하며 이는 일반적인 변수 생성과 동일하다. 단, 초기화를 위해 생성 시에 변수 `num`의 값을 지정하도록 하였다. 생성된 객체에서 변수와 함수는 객체 이름에 도트 연산자 `'.'`를 사용하여 접근할 수 있다. 코드 7은 `INTEGER` 클래스를 이용하여 숫자 n 이 주어졌을 때 1부터 n 까지의 정수 i 중에서 n 을 i 로 나누었을 경우 나머지가 0인 i 의 목록을 구하는 코드의 예이다.

코드 7. 클래스를 이용하여 나누어 떨어지는 수 구하기

```
// 클래스 선언 및 정의
class INTEGER{
private:
    int num;                // 멤버 변수
public:
    INTEGER(int n) { num = n; }    // 생성자
    boolean is_dividable_by(int n){    // 멤버 함수
        if(num % n == 0) return true;
        else return false;
    }
};

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

void loop()
{
    // 12의 정수값을 가지는 INTEGER 객체 생성
    INTEGER N(12);

    for(int i = 1; i <= 12; i++){
        if(N.is_dividable_by(i))    // 멤버 함수 호출
            Serial.println(i);
    }

    while(true);
}
```

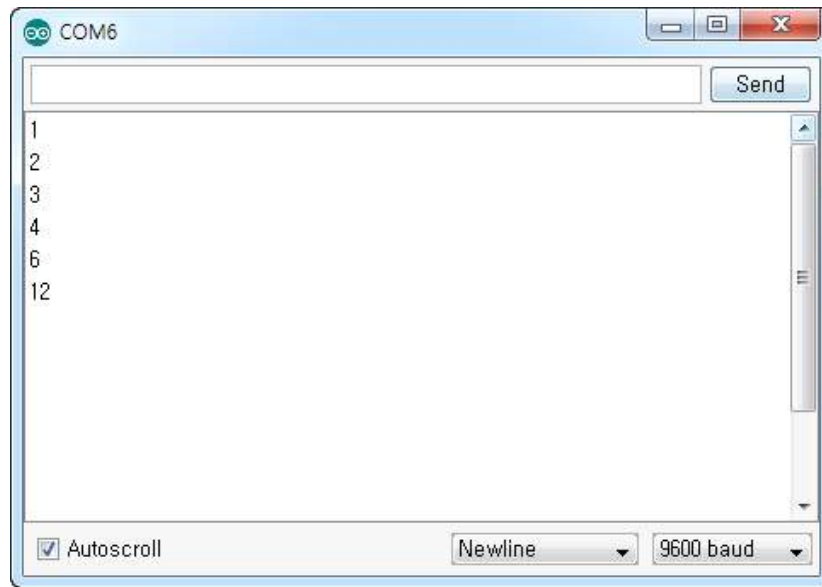


그림 7. 코드 7 실행 결과

코드 7의 경우 아주 단순화된 예로 실제 객체지향 프로그래밍에서 클래스의 작성은 코드 7의 경우처럼 간단하지는 않다. 코드 7을 통해 클래스의 기본 구조를 파악하고 멤버 변수와 멤버 함수를 파악할 수 있으며 멤버 변수를 호출하는 방법을 파악하는 것으로 충분하다. 한 가지 주의해야 할 점은 코드 7의 경우 클래스 선언 및 정의 내용이 별개의 파일로 작성되지 않았다는 점이다. 일반적으로 라이브러리는 별개의 디렉터리에 저장된 헤더 파일과 소스 파일로 구성된다는 사실을 SoftwareSerial 라이브러리의 예에서 살펴보았다. 이처럼 클래스가 별개의 파일로 작성된 경우에는 스케치에서 해당 헤더 파일을 포함시켜 주어야 한다. 예를 들어 코드 7에서 INTEGER 클래스가 별도의 헤더 파일로 작성되었다면 그 파일을 포함시켜 주어야 하며 라이브러리의 이름은 클래스의 이름과 동일하게 주어지는 것이 일반적이다.

```
#include "INTEGER.h"
```

INTEGER.h 파일의 내용은 코드 7에서의 내용과 동일하다.

코드 8. INTEGER.h

```
// 클래스 선언 및 정의
```

```
class INTEGER{
private:
    int num;                // 멤버 변수
public:
    INTEGER(int n) { num = n; }    // 생성자
    boolean is_dividable_by(int n){    // 멤버 함수
        if(num % n == 0) return true;
        else return false;
    }
};
```