

Chapter 104. 프로그램의 기본 구조

이전 장에서 아트멜 스튜디오를 설치하고 프로그램을 작성하여 이를 다운로드하고 실행시키는 방법을 살펴보았다. 본격적인 프로그램 작성에 앞서, 프로그램을 작성할 준비를 마쳤는지 확인해 보자. 확인하여야 할 사항들이 많으니 하나씩 꼼꼼하게 확인하도록 한다.

- 아두이노 IDE를 설치하였는가? 아두이노 IDE를 사용하여 프로그램을 작성하는 경우는 없겠지만 아트멜 스튜디오의 아두이노 애드온이 아두이노 IDE를 사용하므로 설치하여야 한다.
- 아트멜 스튜디오 6를 설치하였는가? 아두이노 스타일과 C 스타일 프로그램 작성을 위해 아트멜 스튜디오 6를 사용하므로 설치하여야 한다.
- Arduino Ide for Atmel Studio를 설치하였는가? 아트멜 스튜디오는 아두이노 스타일 프로그램 작성을 지원하지 않으므로 아두이노 스타일의 프로그램을 작성하기 위해서는 애드온인 Arduino Ide for Atmel Studio를 설치하여야 한다.
- 아두이노 우노 (또는 호환) 보드가 준비되었는가? 이 책에서 사용하는 마이크로컨트롤러는 ATmega328로 아두이노 우노 보드에서 사용된 마이크로컨트롤러이다. 작성한 프로그램은 기본적으로 아두이노 우노 보드에 업로드 하여 테스트 할 것이므로 아두이노 우노 (또는 호환) 보드의 구입을 추천한다.
- 브레드 보드에 기본 회로를 구성하였는가? 프로그램 테스트를 위해서는 아두이노 우노 보드를 사용하겠지만 아두이노 우노 보드는 학습용 보드에 가까워 테스트에서는 필요로 하지 않는 부품들이 다수 포함되어 있는 것이 사실이다. 좀 더 간단한 제어 장치를 만들고자 한다면 간단히 브레드 보드에 회로를 구성하여 사용하는 것도 가능하며 이를 통해 마이크로컨트롤러에 대한 이해를 높일 수 있으므로 한번쯤은 브레드보드에 구성해 보기를 추천한다.
- ISP 장치가 준비되었는가? 프로그램을 ATmega328에 업로드 하기 위해서는 ISP 방식을 기본으로 사용하므로 ISP 장치는 반드시 필요하다.
- USB-시리얼 변환 장치가 준비되었는가? USB-시리얼 변환 장치는 두 가지 용도가 있다. 한 가지는 시리얼 방식의 프로그램 업로드를 위해 사용된다. 아두이노 우노 보드를 사용하는 경우에는 보드 내에 USB-시리얼 변환 장치가 포함되어 있으므로 별도로 준비할 필요는 없지만 브레드 보드에 구성한 기본 회로를 사용한다면 프로그램 업로드를 위해 필요하다. 프로그램 업로드에는 기본적으로 ISP 방식을 사용하지만 시리얼 방식은 아두이노의 특

징 중 하나이므로 브레드 보드에 구성한 기본 회로를 사용하고자 한다면 역시 하나쯤 구입하기를 추천한다. 다른 한 가지는 컴퓨터와의 시리얼 통신을 위해 사용된다. 이 역시 아두이노 우노를 사용한다면 필요하지 않으며 기본 회로를 사용하는 경우에만 필요하다.

- ATmega328의 퓨즈를 설정하였는가? ATmega328은 기본적으로 내부 1MHz 클록을 사용하도록 설정되어 있지만 아두이노 우노와 호환을 위해 이 책에서는 외부 16MHz 클록을 사용한다. ATmega328 칩을 새로 구입 하였다면 외부 클록을 사용하도록 퓨즈를 설정하여야 한다. 퓨즈 설정은 아두이노의 부트로더를 구움으로써도 가능하고 아트멜 스튜디오에서 퓨즈 값만을 별도로 설정하여서도 가능하다.

1. C 스타일 프로그램의 구조

모든 준비가 끝났다면 먼저 C 스타일의 프로그램부터 살펴보자. 사용할 프로그램은 코드 1로 이전 장에서 사용한 'Hello World'와 동일하다.

코드 1. HelloWorld

```
#define F_CPU 16000000L
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
    DDRB = 0x20;

    while(1)
    {
        PORTB = 0x00;    // 끄기
        _delay_ms(1000);
        PORTB = 0x20;    // 켜기
        _delay_ms(1000);
    }

    return 1;
}
```

ATmega328을 위한 코드는 C 언어로 작성한다. 코드 1은 크게 전처리 문장(`#include`, `#define`)과 `main` 함수의 두 부분으로 나누어 볼 수 있다. 포함된 (`include`) 헤더 파일 중 '`io.h`' 파일에는 ATmega328 프로그래밍에 사용되는 레지스터 이름들이 정의되어 있고 '`delay.h`' 파일에는 시간 지연을 위한 `_delay_ms` 함수가 정의되어 있다. `#define` 문장은 ATmega328이 동작할 클록 주파수를 정의하는 문장으로 `_delay_ms` 함수를 사용하기 위해 필요하다.

`main` 함수는 `while`문을 중심으로 `while`문 이전의 초기화 부분, `while`문 그리고 `while`문 이후 `return`의 세 부분으로 이루어져 있다. C 언어에서 가장 먼저 실행되는 함수는 `main` 함수이므로 프로그램이 실행될 때 가장 먼저 실행되는 부분은 초기화 블록이다. 초기화 블록에서는 단어 의미 그대로 변수의 초기화나 레지스터 설정 등을 수행한다. `while`문의 조건 부분에는 '1'이 들어있다. C 언어에서 영(`zero`)이 아닌 모든 정수는 참(`true`)으로 간주되므로 `while(1)` 문장은 무한 루프를 형성하며 이를 '메인 루프' 또는 '이벤트 루프'라고 부른다. 마이크로컨트롤러의 프로그램은 일반적으로 전원이 꺼질 때까지 종료하지 않으며 메인 루프에서 주변 장치로부터 데이터를 받아 이를 처리하고 출력하는 일을 반복한다. 무한 루프가 존재한다면 마지막 `return`문은 언제 실행될까? `return` 문장은 결코 실행되지 못한다. 컴퓨터에서 동작하는 프로그램의 경우 실행 결과를 운영체제에 알려주기 위해 `main` 함수에서 특정한 값을 반환하지만 마이크로컨트롤러에는 운영체제가 없을 뿐만이 아니라 결코 종료하지 않는 단 하나의 프로그램만이 존재하므로 값을 반환하는 경우는 없다. 하지만 아트멜 스튜디오의 컴파일러는 `main` 함수가 정수형(`int`) 값을 반환하도록 하고 있으므로 `return`문을 사용하지 않는 경우 경고 메시지가 출력될 수 있으므로 C 프로그래밍의 관례에 따라 `return`문을 사용하고 있을 뿐이다. 그림 1은 C 스타일로 작성한 마이크로컨트롤러를 위한 프로그램의 구조를 요약한 것이다.

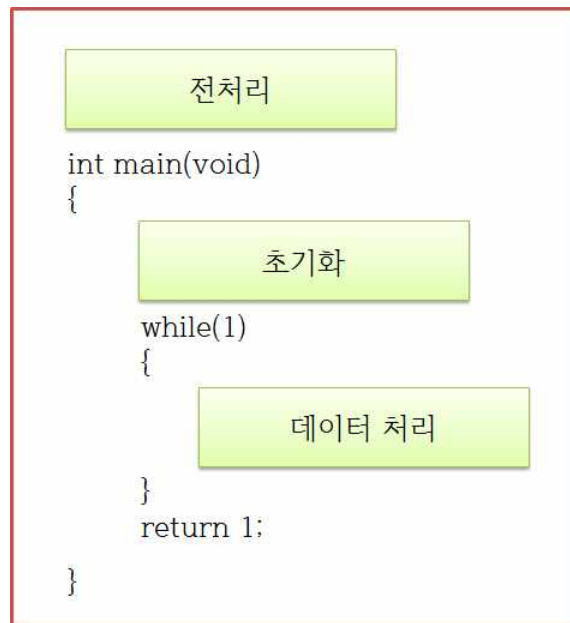


그림 1. C 스타일 프로그램의 구조

코드 1을 좀 더 자세히 살펴보자. 코드 1에는 두 개의 레지스터 DDRB와 PORTB가 사용되었다. 코드에서 알 수 있듯이 레지스터에는 일반 변수와 마찬가지로 값이 할당되고 있지만 별도로 정의되어 있지는 않다. 레지스터들은 'io.h' 파일에 정의되어 있으므로 코드에 'io.h' 파일을 포함시키고 있음은 앞에서도 설명하였다. 마이크로컨트롤러에서 주변 장치와의 모든 데이터의 입출력은 레지스터를 거치게 된다. ATmega328에는 B, C, D 세 개의 포트가 데이터 입출력을 위해 사용될 수 있으며 각 포트는 전용의 레지스터와 연결되어 있다. 코드 1에서는 이 중 'Port B'의 핀들 중 하나에 연결된 LED를 깜빡이는 작업을 수행한다. 포트의 핀들은 데이터의 입력과 출력을 위해 사용될 수 있지만 동시에 사용될 수는 없다. 따라서 포트를 사용하기 이전에 입력 또는 출력으로 사용할 것임을 명시해 주어야 하며 이를 위해 사용하는 레지스터가 DDRx (Data Direction Register, x는 B, C, 또는 D) 레지스터이다. DDRx 레지스터는 8비트 레지스터로 핀 별로 입력과 출력을 개별적으로 지정하는 것이 가능하다. DDRx 레지스터의 구조는 그림 2와 같다.

비트	7	6	5	4	3	2	1	0
비트 이름	DDx7	DDx6	DDx5	DDx4	DDx3	DDx2	DDx1	DDx0
읽기/쓰기	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
초깃값	0	0	0	0	0	0	0	0

(1은 출력, 0은 입력 상태)

그림 2. DDRx 레지스터 구조

DDRx 레지스터의 초깃값은 0으로 모든 핀은 입력으로 설정되어 있다. 따라서 LED를 켜는 출력으로 사용하기 위해서는 해당 핀에 할당된 레지스터의 비트를 1로 설정해 주어야 한다. 예를 들어 DDRB에 0xF0의 값을 대입하였다면 포트 B의 상위 4비트(비트 4~7)는 출력으로 사용하고 하위 4비트(비트 0~3)는 입력으로 사용한다는 의미이다. 코드 1에서는 DDRB = 0x20; 으로 설정되어 있으므로 비트 5만을 출력으로 사용하고 나머지는 입력으로 사용한다는 의미이다. 물론 비트 5에 해당하는 핀에 연결된 LED를 제외하고는 아무런 주변 장치가 연결되어 있지 않으므로 아무런 의미 없는 값이 입출력될 수 있다.

이처럼 마이크로컨트롤러 프로그래밍에서는 비트 단위의 조작이 많이 사용되므로 일반적으로 사용하는 10진수보다는 비트 단위의 계산이 간편한 16진수를 많이 사용한다. 16진수 숫자는 숫자 앞에 '0x' 또는 '0X'를 붙여 10진수와 구별한다. 비트별 의미를 보다 분명하게 구별하기 위해서는 2진수 표현법도 사용되며 2진수의 경우 '0b' 또는 '0B'를 숫자 앞에 붙여 나타낸다. 한 가지 주의할 점은 일반적인 C 프로그래밍에서도 '0x' 또는 '0X'를 사용하여 16진수를 구별하는 것은 동일하지만 2진수를 구별하기 위한 표기법은 없다는 점이다. 마이크로컨트롤러 프로그래밍에서도 2진수 표기법을 지원하기는 하지만 2진수는 0과 1로만 구성되어 있고 길이가 길어져 사용하기가 쉽지만은 않으므로 16진수 표기법을 사용하기를 추천한다.

```
DDRB = 0x20;
DDRB = 0b00100000;
```

입출력의 방향이 결정되면 데이터 출력은 PORTx (x는 B, C, 또는 D) 레지스터에 데이터를 대입함으로써 이루어진다. PORTx 레지스터 역시 핀별로 0또는 1의 값을 출력하는 것이 가능하며 1은 HIGH (또는 VCC) 값을 0은 LOW (또는 GND) 값을 의미한다. 아두이노 우노의 디지털 13번 핀(PB5)에 연결된 LED의 회로도는 그림 3과 같다. 즉, HIGH 값이 출력되면 LED가 켜지고 LOW 값이 출력되면 LED는 꺼진다.

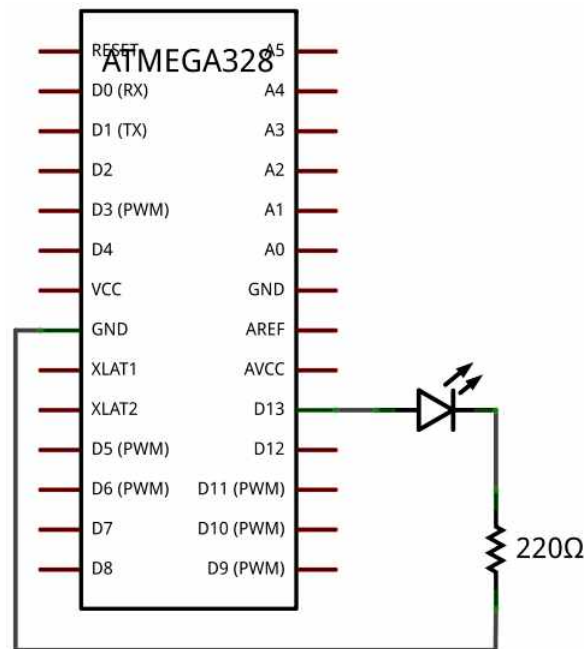


그림 3. 디지털 13번 핀, PB5에 연결된 LED 회로도

PORTx 레지스터의 구조는 그림 4와 같다.

비트	7	6	5	4	3	2	1	0
비트 이름	PORTx7	PORTx6	PORTx5	PORTx4	PORTx3	PORTx2	PORTx1	PORTx0
읽기/쓰기	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
초깃값	0	0	0	0	0	0	0	0

그림 4. PORTx 레지스터 구조

코드 1에는 사용되지 않았지만 DDRx, PORTx 레지스터와 더불어 가장 기초적인 레지스터에 PINx (Port Input, x는 B, C, 또는 D) 레지스터가 있다. DDRx는 데이터 입출력을 결정하는 레지스터로 LED를 켜거나 끄기 위해서는 출력으로 (1의 값으로) 설정하여야 한다. 입력으로 (0의 값으로) 설정된 경우는 주변 장치로부터 데이터를 읽어 들이는 경우이다. 버튼을 연결하였다고 가정하면 버튼을 누른 상태에 따라 0 또는 1의 값이 마이크로컨트롤러로 입력되며 이 값은 PINx 레지스터에 저장되고 PINx 레지스터의 값을 통해 주변 장치의 상태를 알아낼 수 있다. PINx 레지스터의 구조는 그림 5와 같다. PINx 레지스터는 다른 두 레지스터와 다르게 초깃값은 지정되어 있지 않다.

비트	7	6	5	4	3	2	1	0
비트 이름	PINx7	PINx6	PINx5	PINx4	PINx3	PINx2	PINx1	PINx0
읽기/쓰기	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
초깃값	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

그림 5. PINx 레지스터 구조

이처럼 DDRx, PORTx, PINx 레지스터를 통해 주변 장치와 데이터를 주고받을 수 있으며 이를 범용 입출력(General Purpose Input/Output, GPIO)이라고 이야기 한다.

2. 아두이노 스타일 프로그램의 구조

다음은 아두이노 스타일 프로그램의 구조를 살펴보자. 사용할 프로그램은 코드 2로, 코드 1과 마찬가지로 이전 장에서 사용한 'Hello Blink'를 사용한다.

코드 2. HelloBlink

```
void setup()
{
    pinMode(13, OUTPUT);          // 13번 핀을 출력 모드로 설정
}

void loop()
{
    digitalWrite(13, HIGH);       // LED 켜기
    delay(500);
    digitalWrite(13, LOW);        // LED 끄기
    delay(500);
}
```

아두이노 스타일의 프로그램 역시 C 언어를 기반으로 하고 있다. 하지만 코드 2에는 어디에도 main 함수가 존재하지 않는다. 아두이노는 비전공자들이 쉽고 빠르게 프로그램을 작성할 수 있도록 작성해야할 코드를 최소로 하고 있으며 main 함수 역시 프로그래머가 신경 쓰지 않아도 되도록 숨겨져 있다. 없는 것이 아니다, 다만 신경 쓰지 않아도 되도록 감추어져 있을

뿐이다. 그림 1의 C 스타일의 프로그램에서 기본적으로 필요한 블록이 전처리, 초기화, 그리고 데이터 처리의 3개 블록이었다. 코드 2에서는 전처리에 해당하는 블록 역시 없다. 실제로는 전처리 블록이 존재하지만 main 함수와 마찬가지로 아두이노 IDE에서 자동으로 포함시켜서 처리해 주므로 프로그래머는 신경 쓰지 않아도 된다. 코드 2는 두 개의 함수 setup과 loop로 이루어져 있으며 setup 함수는 초기화 블록에, loop 함수는 메인 루프로 데이터 처리 블록에 해당한다. 그림 6은 아두이노 스타일로 작성한 프로그램의 구조를 요약한 것이다.

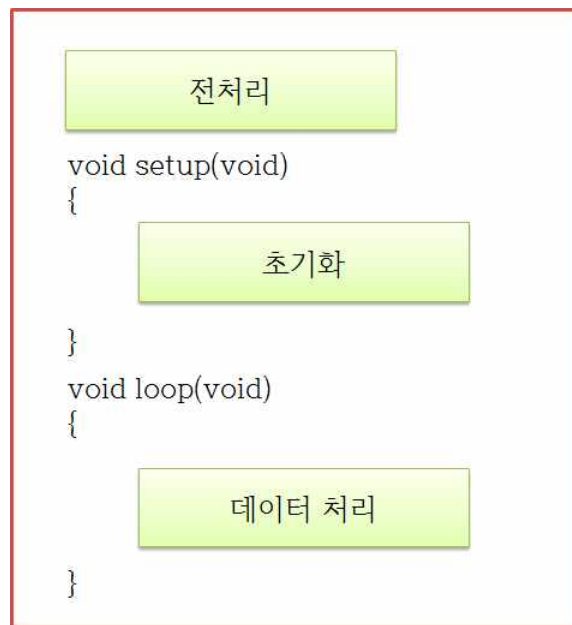


그림 6. 아두이노 스타일 프로그램의 구조

코드 2에서 pinMode 함수는 마이크로컨트롤러의 핀이 입력 또는 출력으로 사용되도록 설정하는 함수로, 코드 1에서 DDRx 레지스터에 값을 대입하는 것과 동일한 역할을 한다. 하지만 레지스터를 조작하는 것과는 사용 방식에서 차이가 있다. C 스타일에서는 8개의 핀으로 이루어지는 포트 단위로 (즉, 바이트 단위로) 데이터 입출력이 이루어지는 반면 아두이노 스타일에서는 핀 단위로 (즉, 비트 단위로) 데이터 입출력이 이루어진다. 또한 핀 단위로 입출력을 결정하므로 핀을 개별적으로 구별하기 위해 아두이노에서는 핀 별로 고유의 번호를 지정하여 사용하고 있다. 포트 B의 여섯 번째 핀인 PB5는 아두이노에서 디지털 13번 핀에 해당하는 것이 그 예이다. 하지만 잊지 말아야 한다. 아두이노 스타일이 C 스타일과 달라 보이지만 내부 동작은 동일하다. 아두이노 스타일의 코드는 결국 C 스타일의 코드로 변환된 후 컴파일 되고 기계어 파일이 생성된다.

```
void pinMode(uint8_t pin, uint8_t mode)
매개변수
    pin : 설정하고자 하는 핀 번호
    mode : INPUT, OUTPUT, INPUT_PULLUP 중 하나
반환값 : 없음
```

pinMode 함수의 매개변수인 mode는 입력이나 출력 또는 풀업 저항을 사용하는 입력 중 하나를 선택할 수 있도록 해준다. 버튼이 눌러진 경우 1, 눌러지지 않은 경우 0으로 생각하는 것이 일반적이지만 풀업 저항이 사용되면 반대로 버튼이 눌러진 경우 0, 눌러지지 않은 경우 1로 반대의 값이 입력된다.¹⁾

코드 2에서 digitalWrite 함수는 PORTB에 데이터를 대입하여 LED로 데이터를 출력하는 것과 동일하다. digitalWrite 함수로 출력하는 값은 PORTx에 대입하는 0이나 1의 값 중 하나로 아두이노에서는 LOW와 HIGH의 상수를 정의하여 사용하고 있다.

```
void digitalWrite(uint8_t pin, uint8_t value)
매개변수
    pin : 핀 번호
    value : HIGH 또는 LOW
반환값 : 없음
```

그림 7은 C 스타일과 아두이노 스타일 코드를 비교한 것이다. C 스타일 코드에서는 main 함수가 반드시 존재해야 하는 반면, 아두이노 스타일 코드에서는 setup과 loop 함수가 반드시 존재해야 하는 차이가 있다. 또한 몇 가지 부분에서 아두이노는 암시적으로 처리해 주는 차이가 존재하지만 기본적으로 C 스타일의 코드와 아두이노 스타일의 코드는 동일하다.

1) 풀업 저항에 대한 보다 자세한 내용은 뒤에서 다룬다.

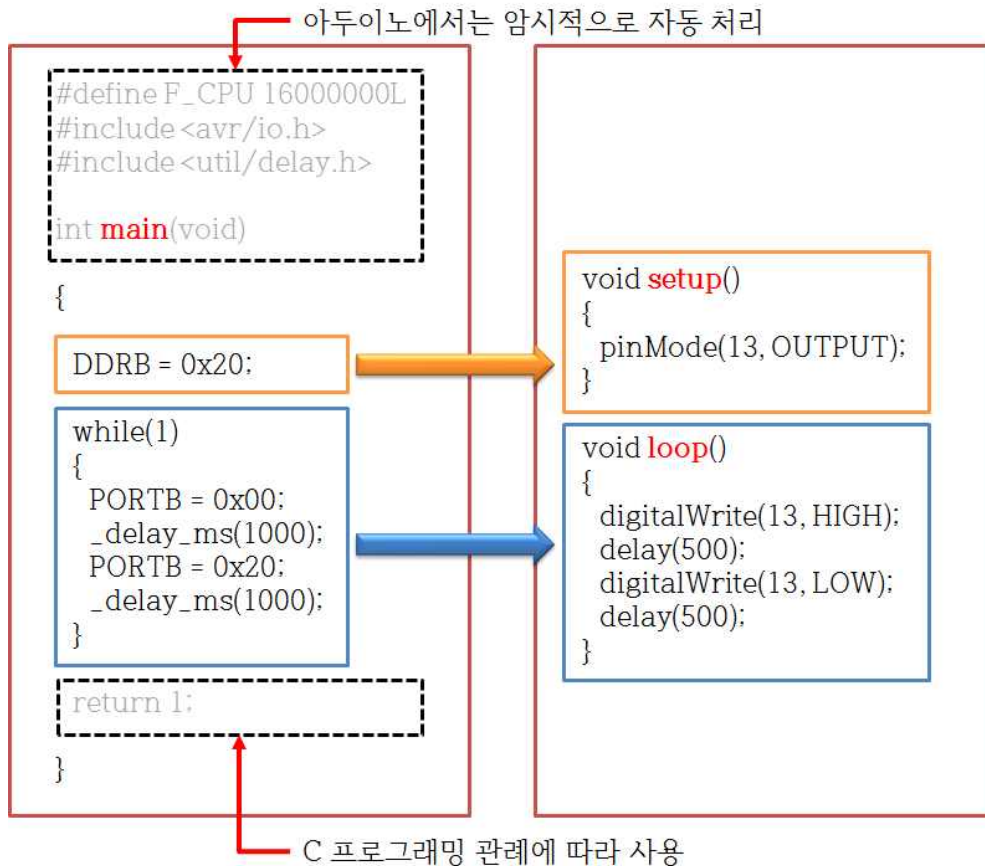


그림 7. C 스타일과 아두이노 스타일 코드 비교

아두이노 스타일 코드가 훨씬 이해하기 쉽고 간단해 보인다. 사실이다. 하지만 아두이노 스타일의 코드는 C 스타일의 코드가 하는 작업을 추상화시켜 놓은 것에 지나지 않으며 모든 ATmega328의 기능들이 추상화되어 있지는 않다. 모든 아두이노 스타일 코드들은 C 스타일 코드로 변환이 가능하지만, 모든 C 스타일의 코드가 아두이노 스타일 코드로 변환되는 것은 아니다. 또 한 가지는 아두이노 스타일 코드의 효율이 좋지 않다는 점이다. 코드 1과 코드 2는 동일한 기능을 수행하는 프로그램이지만 C 스타일 프로그램에서 생성되는 기계어 파일의 크기에 비해 아두이노 스타일의 코드에서 생성되는 기계어 파일의 크기는 더 크다.

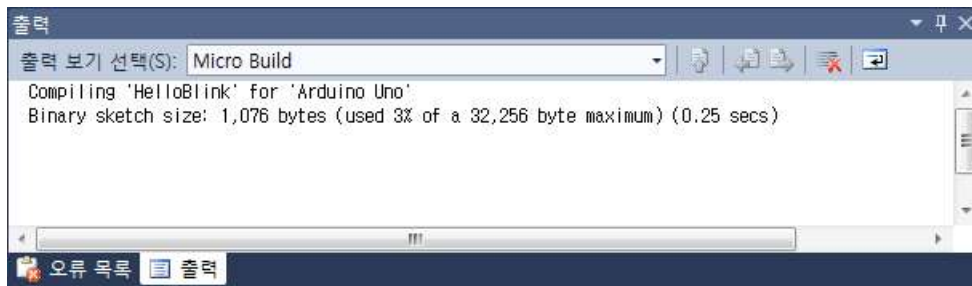


그림 8. 아두이노 스타일 코드에서 생성된 기계어 파일 크기

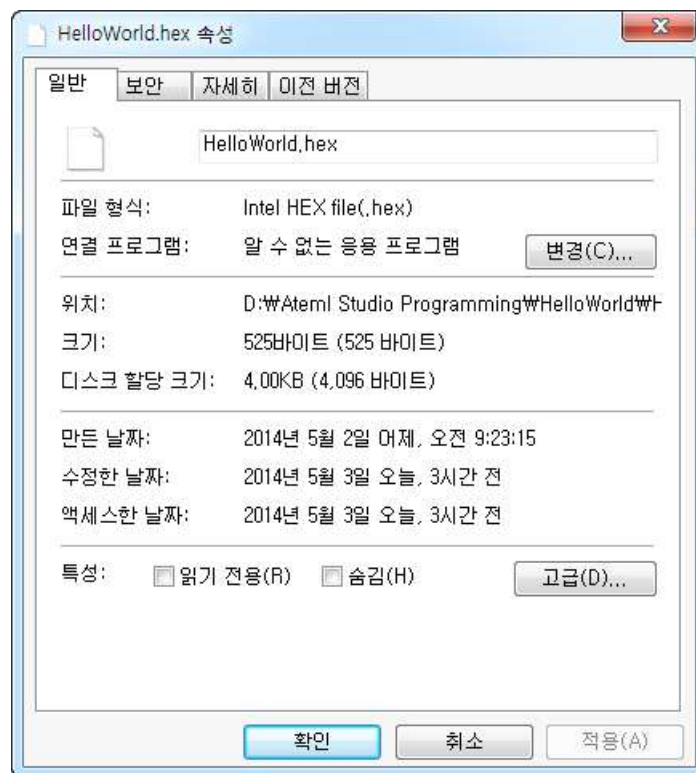


그림 9. C 스타일 코드에서 생성된 기계어 파일 크기

아두이노가 쉽고 빨리 배울 수 있는 점에서는 좋지만 아두이노의 범위 밖에도 마이크로컨트롤러를 배우기 위해 알아야 할 내용들은 많으며 컴파일 효율 또한 그지 좋지 않다. 하지만 C 스타일 프로그래밍은 모든 것을 프로그래머가 신경을 써 주어야 하며, 무엇보다 수많은 레지스터는 마이크로컨트롤러 프로그래밍에서 가장 성가시고 어려운 점이다. 어느 것을 선택할 것인가? 선택은 이 장을 읽고 있는 개개인의 몫이며 필자는 두 가지 모두를 제시하여 각각의 장단점을 알고 필요한 곳에 적절히 사용할 수 있기를 바랄 뿐이다.