일일 업무 사항 정리

작성자	제품팀 이민성 인턴
업무 일시	20220919 ~ 20220923

세부 사항

1. 업무 내역 요약 정리

목표 내역	Done & Plan
1 주차) C 언어 개요 2 주차) 연산자/제어문 3 주차) 배열/포인터 4 주차) 함수 5 주차) 구조체/공용체 6 주차) 전처리기	C 언어의 기본적인 개념에 대해 살펴보았습니다. **1. C언어의 개론적인 이야기** - C언어는 프로그래밍 언어이다. 프로그래밍 언어 컴파일러 컴파일 기계어 어셈블리어 고급어 - C언어의 역사와 특징 - C언어의 장점 **2. C 프로그램의 완성과정** **3. Hello World!** - C언어의 기본단위인 '함수'의 이해 - 예제 Hello.c에서의 함수는 어디에? - 함수 내에 존재하는 문장의 끝에는 세미콜론 문자 ; 을 붙여준다. - 표준 라이브러리와 printf 함수 - 현 시점에 어울리는 '헤더파일 선언의 필요성'에 대한 설명 - return은 함수의 종료와 값의 전달(반환)이라는 두 가지 의미를 지닌다. **4. 주석이 들어가야 완성된 프로그램** - 주석의 필요성 - 블록 단위 주석 - 행 단위 주석 - Hello.c에 주석 추가하기 - 주석처리에 있어서의 주의점 **5. printf 함수의 기본적인 이해** - printf 함수를 이용한 정수의 출력과 서식문자

2. 내용 세부 (업무 세부 내역 정리 및 기타 사항 정리)

C 언어 개요

1. C언어의 개론적인 이야기

- C언어는 프로그래밍 언어이다.

프로그래밍 언어: 사람과 컴파일러가 이해할 수 있는 약속된 형태의 언어

컴파일러: 프로그래밍 언어로 작성한 프로그램을 컴퓨터가 이해할 수 있도록 기계어로 번역하는 역할

컴파일: 번역하는 일 자체

기계어: 0 과 1(2 진법)로 CPU가 직접 해독하고 실행할 수 있는 비트 단위로 쓰인 컴퓨터 언어

어셈블리어: 기계어와 일대일 대응이 되는 컴퓨터 프로그래밍의 저급 언어 (기계어에서 숫자를 의미 있는 단어로 바꿔서 사람들이 이해하기 쉽게 만든 언어)

고급어: 사람들이 이해하기 편하도록 만들어진 프로그래밍 언어 (ex : C, C++, Java, C#)

- C언어의 역사와 특징

C언어는 1971 년경에 UNIX라는 운영체제의 개발을 위해 Dennis Ritchie와 Ken Thompson이 함께 설계한 범용적인 고급언어이다. 하지만 그 탄생의 배경은 훨씬 이전부터 시작되었다. ALGOL 60(1960 년)을 시작으로 CPL(1963 년), BCPL(1969 년), B언어(1970 년)에 이르기까지 그 기원을 두고 있으며, 그 후에 탄생한 것이 C언어이다. 기존의 UNIX라는 운영체제는 '어셈블리 언어'라는 저급언어로 만들어졌기 때문에 하드웨어의 의존도가 높았다. (이를 가리켜 '이식성이 낮다'라고 표현한다.)쉽게 말해서, CPU의 종류가 바뀌면 프로그램을 다시 작성해야만 했다. 예를 들어 인텔 CPU를 기반으로 구현된 프로그램은 AMD기반의 시스템에서 동작하지 않았다. 때문에 똑같은 기능의 프로그램이라고 CPU의 종류에 따라서 별도로 구현해야만 했던 것이다.

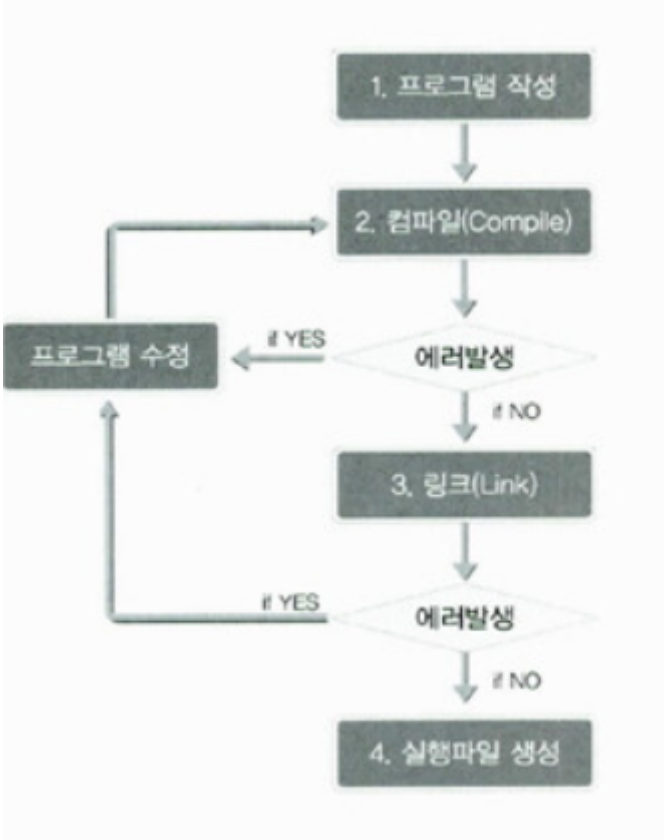
이러한 단점의 해결을 위해서, 더불어 활용에 제약이 많이 따르는 어셈블리 언어의 대체를 위해서, 어셈블리

언어의 저급 언어적 특징을 지니면서도 이식성도 좋고, 더불어 익히기도 쉬운 언어가 필요했는데, C언어는 이러한 요구조건을 모두 만족하였다. 결과적으로, C언어의 개발로 인해서 UNIX라는 운영체제의 90%이상이 C언어로 대체되었고 이를 시작으로 C언어의 인기는 더욱 더 높아져갔다.

- C언어의 장점

1. C언어는 절차지향적 특성을 지닌다. 때문에 익숙해지는데 오랜 시간이 걸리지 않는다.
여기서 말하는 절차지향에는 '정해진 순서의 실행흐름'을 중시한다는 의미가 담겨있다. 즉, 절차지향 프로그래밍에서 중심이 되는 것은 '순서'이다.
2. C언어로 작성된 프로그램은 이식성이 좋다.
C언어로 작성된 프로그램은 CPU의 종류에 상관없이 실행이 가능하고, 운영체제의 차이에도 덜 민감하다.
3. C언어로 구현된 프로그램은 좋은 성능을 보인다.
사용하는 메모리의 양이 상대적으로 적고, 속도를 저하시키는 요소들을 최소화한 언어이다.

2. C 프로그램의 완성과정



1 단계: 프로그램의 작성
2 단계: 작성한 프로그램의 컴파일
3 단계: 컴파일 된 결과물의 링크 (보통 컴파일의 과정에서 함께 진행)
4 단계: 실행파일 생성

3. Hello World!

- C언어의 기본단위인 '함수'의 이해

"C언어는 함수로 시작해서 함수로 끝난다"

C언어로 프로그램을 작성한다는 것은 '함수를 만들고, 만든 함수들의 실행순서를 결정하는 것'이기 때문이다.

$$y = 3x + 4$$

위 함수의 x에 2 를 대입하면 y는 10 이다. C언어에서는 x에 삽입되는 값을 '입력'이라 하고, 그 결과로 얻게 되는 y의 값을 '출력'이라 한다. 그리고 적절한 입력과 그에 따른 출력이 존재하는 것을 가리켜 '함수(function)'라 한다.

- 함수의 정의: 만들어진 함수, 실행이 가능한 함수를 일컬음
- 함수의 호출: 함수의 실행을 명령하는 행위
- 인자의 전달: 함수의 실행을 명령할 때 전달하는 입력 값

- 예제 Hello.c에서의 함수는 어디에?

정해진 순서에 의해서 진행되는 함수의 호출이 바로 프로그램의 흐름이 되는 것이다. 그렇다면 제일 먼저 호출되는 함수는 무엇일까? 바로 main이라는 이름의 함수이다. 프로그램이 실행되자마자 컴퓨터는 main이라는 이름의 함수를 찾아서 호출을 한다. 따라서 C언어로 구현된 모든 프로그램에는 main이라는 이름의 함수가 정의되어야 한다.

int main(void)

출력형태 함수이름(입력형태)

위는 출력의 형태가 int이고 입력의 형태가 void인 main이라는 이름의 함수이다. 이렇듯 함수의 특성(입력형태, 출력형태, 함수이름)만 정의했다고 해서 함수가 되는 것은 아니다. 함수는 기능을 지녀야 한다. C언어에서 그 기능은 중괄호 안에 표현이 된다. 즉 main 함수의 기능은 다음과 같다.

```c
#include <stdio.h>
int main(void)
{
    printf("Hello World! \n");
    return 0;
}
```

위의 중괄호 내부는 두 개의 문장으로 이뤄져 있는데, 이 문장들은 순차적으로 실행이 된다. 즉, 중괄호 내의 존재하는 문장은 그 수에 상관없이 위에서 아래로 순차적으로 실행이 된다. 그리고 이렇게 함수의 기능을 정의하고 있는 영역을 가리켜 '함수의 몸체(body)'라 한다.

```c
출력형태 함수이름(입력형태)
int main(void)
{
   함수의 몸체
}
```

이제 비로소 예제 Hello.c가 하나의 함수로 구성되어 있음이 눈에 보일 것이다. 그리고 '출력의 형태'라는 표현은 함수임을 강조할 때 사용이 되며, 실제로는 '반환형(return type)'이라는 표현이 주로 사용된다.

- 함수 내에 존재하는 문장의 끝에는 세미콜론 문자 ; 을 붙여준다.

이제 함수의 내부를 살펴볼 차례이다. main 함수의 몸체는 다음 두 문장으로 이뤄져 있다.

```
printf("Hello World! \n");
return 0;
```

그런데 각 문장의 끝에는 세미콜론 문자 ; 가 붙어있다. 이렇듯 C언어는 문장의 끝을 표현하기 위해서 세미콜론을 사용한다. 그런데 모든 문장에 세미콜론이 붙는 것은 아니다. 이후에 공부할 조건문이나 반복문과 같은 컨트롤 문장에는 세미콜론이 붙지 않는다.

- 표준 라이브러리와 printf 함수

main 함수의 몸체에는 다음 문장이 존재한다. 이것은 우리가 처음 보는 함수의 호출문이다.

```
printf("Hello World! \n");
```

함수의 호출문에서 소괄호 안에는 함수호출 시 전달할 인자정보를 표현한다. 그리고 C언어는 큰따옴표를 이용해서 문자열을 표현한다. 즉, 위 문장은 문자열 "Hello World! ₩n"을 인자로 전달하면서 printf라는 이름의 함수를 호출하는 문장이다. 그런데 가만히 생각해 보면, 우리는 printf라는 이름의 함수를 만든 적이 없다. 그렇다면 만들지도 않은 함수를 어떻게 호출할 수 있었던 것일까?

printf 함수는 함수호출 시 전달되는 문자열을 모니터에 출력하는 기능을 지닌다. 이러한 printf 함수는 직접 만들지 않아도 호출이 가능한, 기본적으로 제공되는 함수이다. 즉, 누구나 가져다 쓸 수 있도록 이미 만들어져 있는 함수이다. 이렇듯 기본적으로 제공되는 함수를 가리켜 '표준함수'라 하고, 표준함수들의 모임을 가리켜 '표준 라이브러리'라 한다. C언어는 많은 수의 표준함수를 제공한다. 따라서 프로그램을 구현할 때 필요한 모든 함수를 직접 만들지 않아도 된다. 필요한 함수가 표준함수로 정의되어 있다면, 이를 사용해서 프로그램을 완성하는 것이 더 바람직하다.

- 문자열에 삽입된 ₩n

  큰따옴표로 표현되는 문자열 안에는 특수한 의미의 문자를 삽입할 수 있다. 앞서 보인 문자열에 삽입된 문자 ₩n도 그러한 특수문자 중 하나이다. 이러한 특수문자의 공식명칭은 이스케이프 시퀀스인데, 이스케이프 시퀀스 중 하나인 ₩n은 줄을 바꾸라는(개행 하라는) 의미로 사용이 된다.

- 현 시점에 어울리는 '헤더파일 선언의 필요성'에 대한 설명

Printf와 같은 표준함수의 호출을 위해서는 printf 함수와 관련 있는 '헤더파일 선언'이라는 것을 해야 한다. 앞서 보인 예제의 헤더파일 선언은 다음과 같다.

```
#include <stdio.h>
```

이는 stdio.h라는 확장자가 .h로 끝나는 헤더파일을 포함하라는 의미의 선언이다. "헤더파일 stdio.h에는 printf 함수의 호출에 필요한 정보가 존재한다. 따라서 이 파일의 정보를 포함하는 헤더파일 선언문이 삽입되어야 한다." 표준 라이브러리에는 다양한 표준함수가 존대하기 때문에, 헤더파일도 다양하게 존재한다. 그래서 필요에 따라서 여러 개의 헤더파일 선언문을 삽입하기도 한다. 이제 마무리 차원에서 헤더파일 stdio.h와 관련해서 다음 두 가지 내용을 정리하고자 한다.

- Printf 함수의 호출을 위해서는 stdio.h를 대상으로 헤더파일 선언을 해야 한다.
- 헤더파일의 선언은 소스파일의 맨 앞부분, main 함수 정의 이전에 와야 한다.

- return은 함수의 종료와 값의 전달(반환)이라는 두 가지 의미를 지닌다.

이제 마지막으로 앞서 보인 main 함수의 다음 두 번째 문장에 대해서 설명하겠다.

```
return 0;
```

이를 가리켜 return문이라 하는데, 이 문장이 지니는 두 가지 의미는 다음과 같다.

- 함수를 호출한 영역으로 값을 전달(반환)
- 현재 실행중인 함수의 종료

따라서 예제 Hello.c에서는 이 문장을 실행하면서 main 함수를 호출한 영역으로 0 을 전달한다. 그리고는 함수를 빠져나온다. 그런데 main 함수의 종료는 프로그램의 종료로 이어지기 때문에 결국 프로그램이 종료가 된다. 우리는 앞으로 main이 아닌 다른 이름의 함수도 정의하게 된다. 그리고 그 함수에서도 위의 문장을 사용할 것이다. 값의 전달과 함수의 종료를 목적으로 말이다. 참고로 0 이 아닌 1 을 전달하려면 다음과 같이 return문을 구성하면 된다.

```
return 1;
```

- main 함수의 return문

  main 함수의 마지막에서 0 을 전달(반환)하는 이유는 무엇일까? 이 값은 main 함수를 호출한 영역으로 전달된다. 그런데 main 함수는 프로그램이 시작되면 자동으로 호출되는 함수이다. 그리고 호출의 주제는 windows나 linux와 같은 운영체제이다. 따라서 0 은 운영체제에게 전달된다. 그리고 그 값은 프로그램의 종료상태를 알리는 용도로 사용된다. 보통 0 은 정상적인 종료의 상황에서 전달하는 값이다. 반면 비정상적인 상황으로 인해서 종료될 때에는 일반적으로 0 이 아닌 값을 전달한다.

4. 주석이 들어가야 완성된 프로그램

주석(comment)은 프로그램 내에 삽입된 메모를 뜻한다. 이는 컴파일의 대상에서 제외가 되기 때문에 주석의 유무는 프로그램의 실행결과에 영향을 미치지 않는다.

- 주석의 필요성

C언어가 사람이 이해하기 쉬운 언어라 할지라도, 그 내용을 분석하는 데는 상당한 시간이 걸린다. 단적인 예로, 다른 사람이 구현한 코드라면 100 줄이 안 되는 분량의 코드를 분석하는 데에도 적지 않은 시간이 걸린다. 그런데 그보다 놀라운 사실은 시간이 지나면 본인이 구현한 프로그램임에도 불구하고 분석하는데 상당한 시간이 걸릴 수 있다는 것이다. 이러한 문제점을 최소화하기 위한 목적으로 주석이라는 것이 존재한다. 즉, 주석을 다는 것은 프로그램을 분석하는 이들을 배려하는 수단도 되지만, 동시에 프로그램을 구현한 본인 스스로를 배려하는 수단도 된다. 주석은 선택이 아닌 필수다.

- 블록 단위 주석

C언어에서 주석을 다는 방식은 두 가지가 있는데, 그 중 하나는 주석의 시작과 끝을 명시하는 방식이다. 시작은 /* 으로 명시한다. 그리고 끝은 */ 으로 명시한다. 따라서 다음과 같이 한 줄을 주석처리 하는데도 사용이 가능하고,

```
/* 주석처리 된 문장 */
```

다음과 같이 두 줄 이상을 주석처리 하는데도 사용이 가능하다.

```
/* 주석처리 된 문장1
   주석처리 된 문장2
   주석처리 된 문장3
*/
```

- 행 단위 주석

C언어의 주석처리 그 두 번째 방식은 다음과 같다. // 뒤에 등장하는 문장은 주석으로 처리된다.

```
// 주석처리 된 문장 1
```

- Hello.c에 주석 추가하기

```c
/*
제목: Hello World 출력하기
기능: 문자열의 출력하기
파일이름: Hello.c
수정날짜: 2022.09.24
작성자: 이민성
*/

#include <stdio.h> // 헤더파일 선언

int main(void) // main 함수의 시작
{
    /*
    이 함수 내에서는 하나의 문자열을 출력한다.
    문자열은 모니터로 출력된다.
    */
    printf("Hello World! \n"); // 문자열의 출력
    return 0; // 0의 반환
}    // main 함수의 끝
```

주석은 간결하고 명료해야 한다. 위의 예와 같이 복잡하게 달린 주석은 오히려 해가 될 수 있다.

- 주석처리에 있어서의 주의점

블록 단위 주석은 중첩될 수 없다.

```c
/*
    주석처리 된 문장1
    /* 단일 행 주석처리 */
    주석처리 된 문장2
*/
```

1 행의 /*을 시작으로 주석이 시작된다. 이제 */을 만나는 곳까지 주석으로 처리되는데, 3 행에서 */을 처음 만난다. 따라서 2, 3 행이 주석으로 처리되고, 5 행의 */는 잘못 삽입된 주석의 끝으로 해석되어서 컴파일 오류가 발생한다. 그러나 다음과 같이 행 단위 주석은 블록 단위 주석의 내부에 포함될 수 있다.

```c
/*
    주석처리 된 문장1
    // 단일 행 주석처리
    주석처리 된 문장2
*/
```

프로그램을 구현하다 보면, 다양한 실행결과의 확인을 위해서 일부 코드를 주석처리 하기도 하는데, 이러한 과정에서 위와 같은 형태의 주석이 유용하게 사용되기도 한다.

5. printf 함수의 기본적인 이해

우리는 문자열의 출력을 위해서 printf 함수를 사용하였다. 그런데 이 함수는 보다 많은 기능을 제공한다. 따라서 이와 관련해서 추가로 설명을 하고자 한다.

- printf 함수를 이용한 정수의 출력과 서식문자

printf 함수를 이용하면 문자열 이외의 데이터를 다양한 형태로 출력하는 것이 가능하다. 물론 정수를 출력하는 것도 가능하다. 그럼 이와 관련해서 다음 예제를 실행해보자.

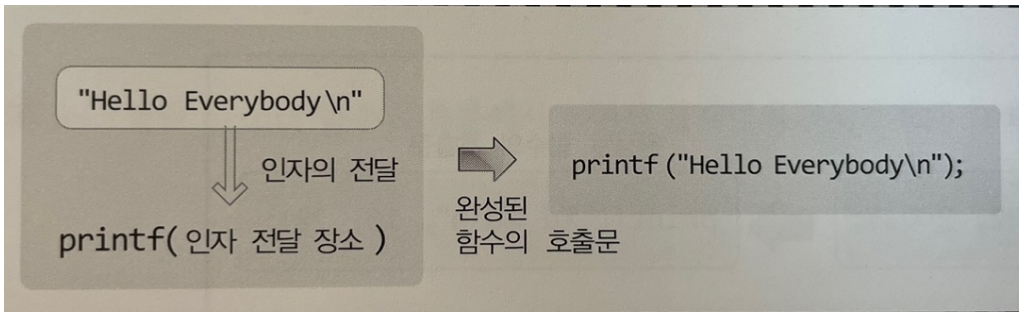
```c
#include <stdio.h>

int main(void)
{
    printf("Hello Silcroad\n");
    printf("%d\n", 1234);
    printf("%d %d\n", 10, 20);
    return 0;
}
```

```
Hello Silcroad
1234
10 20
```

실행결과를 통해서, printf 함수를 이용하여 문자열도, 정수 데이터도 출력이 가능함을 알 수 있다. 그럼 먼저 5 행을 보자. 이를 통해서 다음 사실을 알 수 있다.

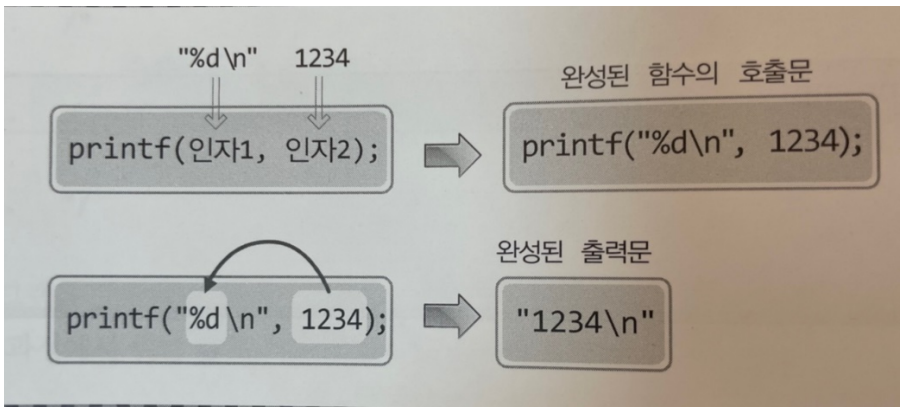
"printf 함수는 첫 번째 인자로 전달된 문자열을 출력한다."

5 행에서 이뤄진 문자열의 인자전달과 관련해서 다음과 같이 정리가 가능하다.

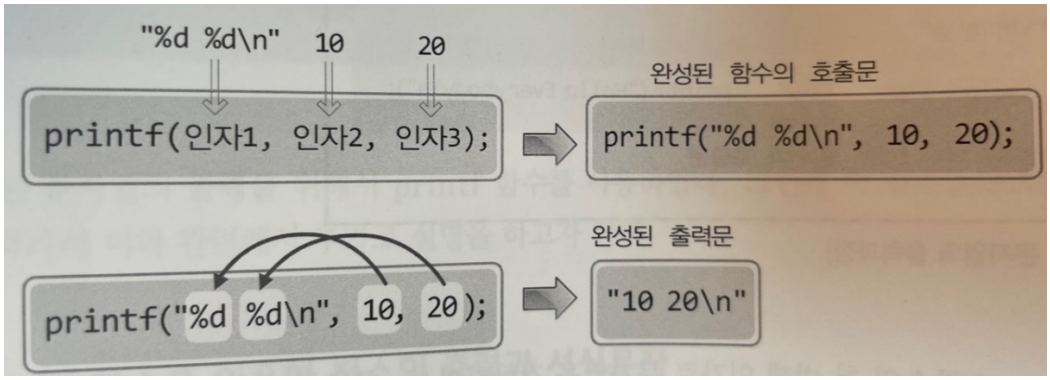


6 행도 마찬가지로 printf 함수의 첫 번째 인자로 문자열을 전달하면서, 이의 출력을 요청하는 문장이다. 그런데 문자열의 형태가 다음과 같다. "%d₩n"

문자열에서 첫 번째로 등장하는 것이 %d이다. 그리고 이러한 문자를 가리켜 서식문자라 하는데, 이는 출력의 형태를 지정하는 용도로 사용이 된다. 그런데 출력의 형태를 지정하려면 출력의 대상이 있어야 한다. 6 행을 보면, printf 함수의 호출문에 두 번째 인자가 있음을 알 수 있다. 바로 이 두 번째 인자가 출력의 대상이 된다. 그렇다면 %d가 지정하는 출력의 형태는 무엇일까? 이는 10 진수 정수형태의 출력을 의미한다. 따라서 printf 함수호출 시 전달된 두 번째 인자가 %d의 위치에서 10 진수 정수로 출력되고, 이어서 나오는 ₩n이 출력된다. 물론 ₩n은 개행을 의미하므로 별 다른 출력결과 없이 개행만 이뤄진다.



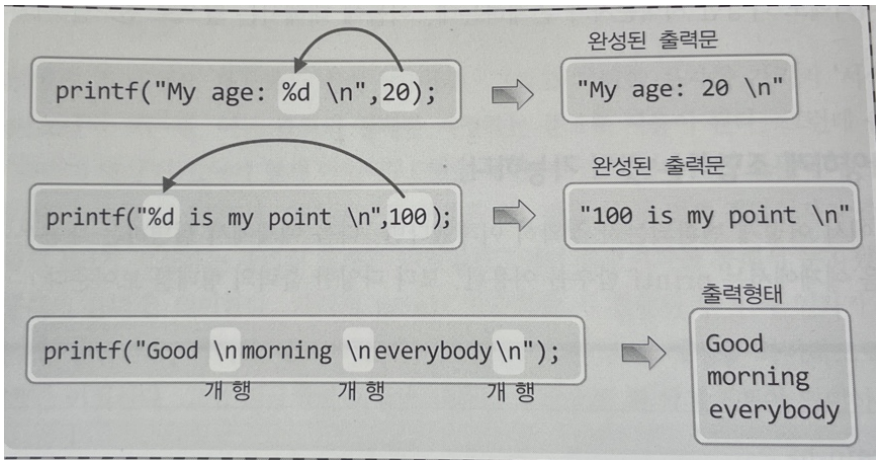
7 행과 6 행의 가장 큰 차이점은, 문자열 안에 서식문자 %d가 두 개 존재하므로 두 개의 출력대상이 두 번째, 세 번째 인자로 각각 전달되고 있다는 점이다.



- 출력의 형태를 다양하게 조합하는 것이 가능하다.

```c
#include <stdio.h>

int main(void)
{
    printf("My age: %d \n", 20);
    printf("%d is my point \n", 100);
    printf("Good \nmorning \neverybody\n");
    return 0;
}
```

```
My age: 20
100 is my point
Good
morning
everybody
```



이렇듯 서식문자의 삽입위치에는 제한이 없기 때문에, 다양하게 문자열을 조합해서 출력하는 것이 가능하다.