Item 7) Distinguish between () and {} when creating objects.

일반적인 초기화의 방법

C++11에서 객체를 초기화 하는 방법은 여러가지 방법이 있다

```
int x(0); // initializer is in parentheses
int y = 0; // initializer follows "="
int z{0}; //initializer is in braces

int z = {0}; // initializer uses "=" and braces
```

📌 일반적인 경우 컴파일러는 equal-sign-plus-braces syntax는 braces-only version과 같게 취급한다.

C++에서는 `initialization`과 `assignment`를 구별하는 것은 매우매우매우 중요한 topic이다.

```
Widget w1; // call default constructor
Widget w2 = w1; // not an assignment; calls copy constructor
w1 = w2; // an assignment; calls copy operator =
```

modern c++에서는 다양한 초기화 문법이 있지만 다양한 STL을 모두 커버하기에는 힘들다. 이런 이유에서 C++11에서는 `uniform initialization`이라는 개념을 소개한다. 이런 개념을 구체화 한 방법은 `Braced initialization`이다.

```
std::vector<int> v{1, 3, 5}; // v's initial content is 1, 3, 5
std::vector<int> v2{};
```

비정적 멤버 변수의 기본 초기화

C++11 이전에는 멤버 변수를 클래스 선언 내에서 초기화할 수 없었지만, C++11부터는 클래스 내에서 기본값을 지정할 수 있다. 이때 사용할 수 있는 초기화 방식은 `{}` 또른 `=`이며, `()`는 사용할 수 없다.

```
class Widget {
    // 멤버 변수의 기본 초기화
private:
    int x{ 0 }; // 중괄호를 사용한 초기화: OK, x의 기본값은 0
    int y = 0;  // 등호를 사용한 초기화: OK, y의 기본값은 0
    int z(0);   // 괄호를 사용한 초기화: Error!
};
```

복사할 수 없는 객체의 초기화

일부 객체, 특히 복사할 수 없는 객체(`std::atomic`)은 `{}` 또는 `()`를 사용하여 초기화할 수 있지만, `=`를 사용한 초기화는 불가능하다.

```
std::atomic<int> ai1{ 0 }; // 중괄호 초기화: OK
std::atomic<int> ai2(0);   // 괄호 초기화: OK
std::atomic<int> ai3 = 0;  // 복사 초기화: Error!
```

결론

`{}`를 사용하는 braced initialization은 모든 상황에서 일관되게 사용할 수 있는 초기화 방식이다. 이는 다른 초기화 방법보다 유연하며, 복사할 수 없는 객체뿐만 아니라 기본 자료형, 클래스 멤버 초기화에도 사용가능하다.

---

braced initialization의 특징

특징1. narrow conversion

braced initialization의 특징은 implicit한 narrow conversion을 허용하지 않는다는 것이다.

```
double x, y, z;

int sum1{x + y + z}; // sum of doubles may not be expressible as int
```

그러나 `=`나 `()`를 사용한 초기화 방법은 narrow conversion을 허용한다.

```
int sum2(x + y + z); // okay

int sum3 = x + y + z; // okay
```

특징2 : vexing parse

정의 : 언어의 문법적 애매함으로 발생하는 문제를 의미하며 주로 생성자에 의해 발생한다.

자 만약 Widget클래스의 객체를 하나 생성한다고 가정을 해보자

```
Widget w1(10); // call Widget ctor with argument 10
Widget w2(); // most vexing parse!!!
Widget w3{}; // calls Widget ctor with no args
```

먼저 `()`를 사용하게 된다면, 특정 argument를 전달하며 생성자를 호출하게 된다, 그러나 w2의 경우를 본다면 컴파일러가 객체의 생성자를 호출하는 것인지, function을 호출하는 것인지 모르는 경우가 있다.

그러므로 만약 default constructor를 호출하고 싶은 경우에는 무조건 `{}`를 사용하는 습관을 들이자!!

---

braced initialization의 단점

당연하겠지만, 모든 상황에서 braced initialization을 사용하는 것이 좋은 습관은 아니다.
특히 클래스 내에서 `std::initializer_lists`를 사용해서 constructor overload를 사용하게 된다면 고려해야 할 점이 있다.

생성자를 호출하는 경우, 만약 `std::initializer_list`가 포함되지 않는 경우에는 `{}`와 `()`는 같은 의미를 가진다.

```
class Widget {
public:
    Widget(int i, bool b);   // 생성자: std::initializer_list가 아닌 파라미터
    Widget(int i, double d); // 생성자: std::initializer_list가 아닌 파라미터
    …
};

Widget w1(10, true);  // 첫 번째 생성자 호출 (int, bool)
Widget w2{10, true};  // 첫 번째 생성자 호출 (int, bool)
Widget w3(10, 5.0);   // 두 번째 생성자 호출 (int, double)
Widget w4{10, 5.0};   // 두 번째 생성자 호출 (int, double)
```

그러나 단 하나의 생성자라도 `std::initializer_list`가 포함되는 경우에 `{}`를 사용해 생성자를 호출하게 된다면 `std::initializer_list`를 우선적으로 오버로딩 하게 된다.

```
class Widget {
public:
    Widget(int i, bool b);   // 생성자: std::initializer_list가 아닌 파라미터
    Widget(int i, double d); // 생성자: std::initializer_list가 아닌 파라미터
    Widget(std::initializer_list<long double> il); // added
};
```

```
Widget w1(10, true);  // 괄호를 사용, 첫 번째 생성자 호출 (int, bool)
Widget w2{10, true};  // 중괄호를 사용, std::initializer_list 생성자 호출
                      // (10과 true가 long double로 변환됨)

Widget w3(10, 5.0);   // 괄호를 사용, 두 번째 생성자 호출 (int, double)
Widget w4{10, 5.0};   // 중괄호를 사용, std::initializer_list 생성자 호출
                      // (10과 5.0이 long double로 변환됨)
```

특히 narrowing conversion이 발생할 수 있기 때문에 class를 생성하는 경우에는 주의가 필요하다.

`{}`를 사용한 생성자가 하이제킹이 되지 않는 경우는 argument가 `std::initializer_list<template>`으로 타입 변환이 가능하지 않는 경우이다.

```
class Widget {
public:
    Widget(int i, bool b);                // 이전과 동일한 생성자
    Widget(int i, double d);              // 이전과 동일한 생성자
    // std::initializer_list의 요소 타입이 이제 std::string으로 변경
    Widget(std::initializer_list<std::string> il);
    …  // 암시적 변환 함수 없음
};

Widget w1(10, true);  // 괄호를 사용, 여전히 첫 번째 생성자 호출 (int, bool)
Widget w2{10, true};  // 중괄호를 사용, 이제 첫 번째 생성자 호출 (int, bool)
Widget w3(10, 5.0);   // 괄호를 사용, 여전히 두 번째 생성자 호출 (int, double)
Widget w4{10, 5.0};   // 중괄호를 사용, 이제 두 번째 생성자 호출 (int, double)
```

📌 주의점

- 위의 내용을 바탕으로 주의할 점은 `std::vector`를 생성하는 경우이다.

```
std::vector<int> v1(10,20); -> non-std::initializer_list (10개의 element를 20의 value로 초기화 한다)
std::vector<int> v2{10,20} -> std::initializer_list를 사용하는 방식이다 element 10, 20으로 초기화 되는 경우이다.
```