Item 32) Use init capture to move objects into closures.

앞서 살펴본 것 처럼, lambda의 capture는 copy와 reference capture를 지원한다.
그렇다면 copy와 reference를 지원하지 않는 `move-only object(std::unique_ptr, std::function)` 혹은, copy를 하는데 비용이 너무 많이 들어가는 object를 capture하기 위해서는 어떻게 해야할까??

이런 문제를 C++11과 C++14는 다른 방식으로 처리한다.

C++11에서는 `move-only` 객체를 람다에서 캠쳐할 수 있는 방법은 없다.
또한, 복사하는데 비용이 많이 드는 객체를 람다에 넣고 싶을 때, C++11에서는 그 객체를 복사할 수밖에 없다.

C++14에서는 `init capture` 라는 새로운 갭쳐 메커니즘을 도입해서, 객체를 람다에 (move)할 수 있는 방법을 제공한다.
이 방식은 더 많은 유연성을 제공한다.

들어가기에 앞서... what is init capture

```cpp
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;


int main(void)
{

    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);

    vector<int> myVec{1,2,3,4,5};

    auto myFun = [myVec = std::move(myVec)] () -> auto{
        for(const auto& elem: myVec)
            cout << elem << " ";
        cout << endl;
    };
    myFun();
    return (0);
}
```

기본 문법

```cpp
[variable_name = expression](parameters) {
    // lambda body
}
```

- capture clause안에 있는 `variable_name` 에 자료형을 명시적으로 작성할 필요는 없다. 컴파일러가 자동으로 추론하기 때문이다.

특징

1. init capure에서는 캡처할 변수를 초기화하는 표현식을 명시적으로 작성한다.
2. init capture를 사용하면 `std::unique_ptr` 와 같은 move-only 객체를 람다에 캡처할 수 있다. C++11에서는 이를 직접적으로 할 수 없었지만, init capture에서는 이를 명시적으로 처리할 수 있다.
3. init capture는 코드의 가독성을 높여주며, 캡처된 변수가 어떤 값으로 초기화되는지를 명확하게 알 수 있게 해준다.

---

1. the name of a data member in the closure class generated from the lambda and
2. an expression initializing that data member

Code

```cpp
// Widget 클래스 정의
class Widget { // 유용한 타입
public:
    // ...
    bool isValidated() const;
    bool isProcessed() const;
    bool isArchived() const;
private:
    // ...
};
```

```
// std::unique_ptr 생성
auto pw = std::make_unique<Widget>(); // Widget 생성

// Init Capture 사용
auto func = [pw = std::move(pw)] { // pw를 이동하여 클로저의 데이터 멤버로 초기화
    return pw->isValidated() // 클로저 내에서 사용
           && pw->isArchived(); // std::move(pw)
};
```

여기서 주목해야 할 점은 `pw = std::move(pw)` 부분이다.

`=` 왼쪽의 `pw` 는 클로저 클래스의 데이터 멤버를 정의한다. 오른쪽의 `pw` 는 람다가 정의되는 스코프 내의 로컬 변수이다.

위 표현은 "클로저 클래스 내에 `pw` 라는 데이터 멤버를 생성하고, 이를 로컨 변수 `pw` 의 이동 결과로 초기화한다"는 의미이다.

📌 일반적으로, lambda의 body안에 있는 코드는 closure class의 scope이다.

즉 정리를 하자면

- 왼쪽의 `pw` 는 클로저 클래스의 데이터 멤버로 사용된다.
- 오른쪽의 `pw` 는 원래의 로컬 변수로, `std::move` 가 적용된다
- 이후 람다 본문에서는 `pw` 를 사용하면 클래스의 데이터 멤버를 참조한다. 즉, 람다 내부에서 `pw` 를 호출하면 `std::unique_ptr<Widget>` 에 접근하게 된다.

📌 위의 람다함수의 캡처 방식을 `std::make_unique` 을 사용해서 리팩터링 할 수 있다.

```
auto func = [pw = std::make_unique<Widget>()] { // 직접 초기화
    return pw->isValidated() && pw->isArchived();
};
```

- C++11에서는 캡쳐 부분 안에서 바로 init이 불가능했다!!

그렇다면 C++11에서는 어떻게 해요??

- C++11에서는 `std::bind` 를 사용해서 위와 같은 내용을 할 수 있다. (생략)