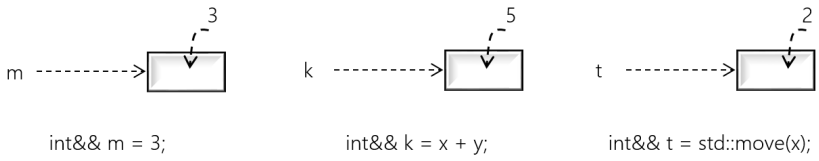
```
int x = 2, y = 3;
int&& m = 3;              // int m = 3;과 같은 효과
int&& k = x + y;          // int k = x + y;와 같은 효과
int&& t = std::move(x)    // t는 x의 별명이 아니고 int t = x;와 같은 효과
```



[그림 1.36] 우측값 레퍼런스의 정의

마지막 문장에서 &를 두 개 명시해서 우측값 레퍼런스로 t를 정의했기 때문에 t는 x의 별명이 아니고 x의 값인 2를 저장하는 일반 변수와 같다. std::move 함수는 [리스트 1.10]의 예제에서 설명한다.

좌측값 레퍼런스는 기존 변수에 대한 별명이고 따로 메모리 방이 생성되지 않지만 우측값 레퍼런스는 임시값을 변수화시키는 것이기 때문에 메모리 방이 생성되고 이름도 부여된다. 또한, 우측값 레퍼런스의 유효 범위는 레퍼런스가 정의된 블록 내가 되는 것도 일반 변수의 유효 범위와 마찬가지다.

그렇다면 일반 변수로 정의하면 될 것을 무용지물 같은 우측값 레퍼런스라는 개념을 도입해서 혼란을 주는 이유가 무엇일까? 그 이유는 [리스트 1.10]에서 찾을 수 있다. [그림 1.37]은 이 프로그램의 실행 결과를 보인 것이다.

[리스트 1.10] Syntax8 프로젝트의 RreferenceApp.cpp 파일

```
01: #include <iostream>
02:
03: void increment(int& value)
04: {
05:     std::cout << "좌측값 레퍼런스로 증가" << std::endl;
06:     ++value;
07: }
08:
09: void increment(int&& value)
10: {
11:     std::cout << "우측값 레퍼런스로 증가" << std::endl;
12:     ++value;
13: }
14:
15: int main()
16: {
17:     int a = 10, b = 20;
18:     increment(a);
```