The two equivalent forms of the same declaration are shown below:

```
double (*f(long long))(int x);         // classic return-type syntax
auto f(long long) -> double (*)(int);  // trailing return-type syntax
```

Note that both syntactic forms of the same declaration may appear together within the same scope. Note also that not all functions that can be represented in terms of the trailing syntax have a convenient equivalent representation in the classic one:

```
#include <utility>  // declval

template <typename A, typename B>
auto foo(A a, B b) -> decltype(a.foo(b));
    // trailing return-type syntax

template <typename A, typename B>
decltype(std::declval<A&>().foo(std::declval<B&>())) foo(A a, B b);
    // classic return-type syntax using C++11's std::declval
```

In the example above, we were essentially forced to use the C++11 Standard Library template `std::declval`[3] to express our intent with the classic return-type syntax.

Use Cases

Function template whose return type depends on a parameter type

Declaring a **function template** whose return type depends on the types of one or more of its parameters is not uncommon in **generic programming**. For example, consider a mathematical function that linearly interpolates between two values of possibly different types:

```
template <typename A, typename B, typename F>
auto linearInterpolation(const A& a, const B& b, const F& factor)
    -> decltype(a + factor * (b - a))
{
    return a + factor * (b - a);
}
```

The return type of `linearInterpolation` is the type of **expression** inside the ***decltype*** *specifier*, which is identical to the expression returned in the body of the function. Hence, this interface necessarily supports any set of input types for which `a + factor * (b - a)` is valid, including types such as mathematical vectors, matrices, or **expression templates**. As an added benefit, the presence of the expression in the function's declaration enables **expression SFINAE**, which is typically desirable for **generic** ~~template functions~~ (see Section 1.1."**decltype**" on page 25).

[3] **cpprefd**