```
struct S
{
    typedef int T;
    auto h1() -> T;  // trailing syntax for member function
    T h2();          // classical syntax for member function
};

auto S::h1() -> T { /*...*/ }  // equivalent to S::T S::h1() { /.../ }
T    S::h2()      { /*...*/ }  // Error, T is unknown in this context.
```

The same advantage would apply to a nonmember function[1] defined outside of the namespace in which it is declared:

```
namespace N
{
    typedef int T;
    auto h3() -> T;  // trailing syntax for free function
    T h4();          // classical syntax for free function
}

auto N::h3() -> T { /*...*/ }  // equivalent to N::T N::h3() { /.../ }
T    N::h4()      { /*...*/ }  // Error, T is unknown in this context.
```

Finally, since the syntactic element to be provided after the arrow token is a separate type unto itself, return types involving pointers to functions are somewhat simplified. Suppose, for example, we want to describe a **higher-order function**, `f`, that takes as its argument a **`long long`** and returns a pointer to a function that takes an **`int`** and returns a **`double`**[2]:

```
// [function(long long) returning]
//     [pointer to] [function(int ~~x~~) returning] double   f;
//     [pointer to] [function(int ~~x~~) returning] double   f(long long);
//                  [function(int ~~x~~) returning] double*  f(long long);
//                                              double (*f(long long))(int ~~x~~);
```

Using the alternate trailing syntax, we can conveniently break the declaration of `f` into two parts: (1) the declaration of the function's signature, **`auto f(long long)`**, and (2) that of the return type, say, `R` for now:

```
// [pointer to] [function (int) returning] double   R;
//              [function (int) returning] double*  R;
//                                         double (*R)(int);
```

[1] A **`static`** member function of a **`struct`** can be a viable alternative implementation to a free function declared within a namespace; see **lakos20**, section 1.4, "Header Files," pp. 190–201, especially Figure 1-37c on p. 199, and section 2.4.9, "Only Classes, `structs`, and Free Operators at Package-Namespace Scope," pp. 312–321, especially Figure 2-23 on p. 316.

[2] Coauthor John Lakos first used the shown verbose declaration notation while teaching Advanced Design and Programming Using C++ at Columbia University (1991–1997).