```
            ret = biggest;
            biggest = element;
        }
        else if (ret < element) {
            ret = element;
        }
    });

    return ret;
}
```

The declarations of second and ret use the placeholder **auto** (see Section 2.1."**auto** Variables" on page 195) to deduce the variables' types from their respective initializers. The return type of secondBiggest is also declared **auto** and is deduced from the type of ret (see Section 3.2. "**auto** Return" on page 1182). The generic lambda being passed to std::for_each uses the C++14 init-capture (see Section 2.2."Lambda Captures" on page 986) to initialize biggest to the largest value known so far. Because the lambda is declared **mutable**, it can update biggest each time a larger element is encountered. The ret variable is also captured — by reference — and is updated with the previous biggest value when a new biggest value is encountered. Note that, at the point where ret appears in the lambda capture, its type has already been deduced. When for_each invokes the function-call operator, the type of the **auto** parameter, element, is conveniently deduced to be the element type for the input range and is thus the same reference type as ret except with an added **const** qualifier.

Constraints on deduced parameters

A generic lambda can accept any mix of **auto** and non**auto** parameters:

```
void g1()
{
    auto y1 = [](auto& a, int b, auto c) { a += b * c; };

    int    i = 5;
    double d = 1;

    y1(i, 2, 2);   // i is now 9.
    y1(d, 3, 0.5); // d is now 2.5.
}
```

If the **auto** placeholder in a generic lambda parameter is part of a type declaration that forms a potentially cv-qualified reference, pointer, pointer-to-member, pointer-to-function, or reference-to-function type, then the allowable arguments will be restricted accordingly:

```
struct C1 { double d_i; };
double f1(int i);
```