Inferred Return Types

Rather than having the caller explicitly instantiate the template and supply a return type, C++ 11 allows a template to **infer** the return type like this:

```
template <typename T, typename U>
auto addem(const T& a, const U& b)->decltype(a + b)
{
    return a + b;
}
```

Using `auto` as the formal template return type, and moving the **deduced** return type so it **follows** the argument list allows the compiler to replace the return type with the **declared type** of the expression `a+b`. (That's what the `decltype` keyword calculates at compile time.

In C++14 this was further simplified. We can now write the `addem` template like this. We **don't** need the trailing return type or `decltype`.

```
template <typename T, typename U>
auto addem(const T& a, const U& b)
{
    return a + b;
}
```

That doesn't mean, however, that you can write a template where only the return type differs. For instance, the following template will not compile because it can't determine whether to use `int` or `double` for the return type when called like `mybad(3, 4.5)`:

```
template <typename T, typename U>
auto mybad(const T& a, const U& b)
{
    if (a > 0) return a;
    return b;
}
```