Explicit Template Arguments

Suppose you wish to pass `3.5` to the `print(int)` version of the template. You may **explicitly instantiate** the function by specifying the type of template parameter inside angle brackets, when calling the function, like this:

```
print<int>(3.5);
```

Even though you pass a **`double`** as the function argument, the function is **instantiated** with the generic parameter **`T`** replaced by type **`int`**. So, this call truncates the fractional part of the argument before it prints the number, rather than generating an overloaded **`print(double)`** function.

To fix your **`addem()`** problem, you can just add an extra **template parameter** for the return type:

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

Call the function by providing an explicit template argument: `addem<double>(2, 3.5);`. Here, the template parameter **`RET`** is replaced with **`double`**. That's a little awkward, but is the only way to handle this problem prior to C++11.