Multiple Template Arguments

Suppose you have a template function like this:

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

You can **call** the function in any of these ways:

```
string a{"hello"}, b{" world"};
cout << addem(3, 5) << endl;
cout << addem(4.5, 2.5) << endl;
cout << addem(a, b) << endl;
```

But, you **cannot** call the function like this:

```
cout << addem(3.5, 2) << endl;
```

The compiler does not know **what type to substitute** for `T` in the template. You could, however, write the template with **two template parameters**, like this:

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

Now the call `addem(3.5, 2)` uses `double` for type `T`, `int` for type `U`, and the function returns a `double`, (`5.5`) as you'd expect. However, what about that call `addem(2, 3.5);`? Now the function returns an `int`, (5) which is **not what you'd expect**. You can fix this in two ways.