definition using specific types rather than one that is a generic description.

The addition of the explicit instantiation led to the new syntax of using `template` and `template <>` prefixes in declarations to distinguish between the explicit instantiation and the explicit specialization. Often, the cost of doing more is more syntax rules. The following fragment summarizes these concepts:

```cpp
...  
template <class Any>
void Swap (Any &, Any &); // template prototype

template <> void Swap<int>(job &, job &); // explicit specialization for job
int main(void)
{
    template void Swap<char>(char &, char &); // explicit instantiation for char
    short a, b;
    ...
    Swap(a,b); // implicit template instantiation for short
    job n, m;
    ...
    Swap(n, m); // use explicit specialization for job
    char g, h;
    ...
    Swap(g, h); // use explicit template instantiation for char
    ...
}
```

When the compiler reaches the explicit instantiation for `char`, it uses the template definition to generate a `char` version of `Swap()`. When the compiler reaches the function call `Swap(a,b)`, it generates a `short` version of `Swap()`. When the compiler reaches `Swap(n,m)`, it uses the separate definition (the explicit specialization) provided for the `job` type. When the compiler reaches `Swap(g,h)`, it uses the template specialization it already generated when it processed the explicit instantiation.

**Which Function?**