The specialization mechanism has changed with the evolution of the language. We look at the current form as mandated by the C++ Standard, then look at two older forms supported by older compilers.

**Third Generation Specialization (ISO/ANSI C++ Standard)**

After C++ experimented with the approaches described later, the C++ Standard settled upon this approach:

1. For a given function name, you can have a non-template function, a template function, and an explicit specialization template function.

2. The prototype and definition for an explicit specialization should be preceded by `template <>` and should mention the specialized type by name.

3. A specialization overrides the regular template, and a non-template function overrides both.

Here's how prototypes for swapping type `job` structures would look for these three forms:

```cpp
// non-template function prototype
void Swap(job &, job &);

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

// explicit specialization for the job type
template <> void Swap<job>(job &, job &);
```

As mentioned previously, if more than one of these prototypes is present, the compiler chooses the non-template version over explicit specializations and template versions, and it chooses an explicit specialization over a version generated from a template.

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