1 Template Specialization

There are often advantages to treating certain combinations of template parameters as special cases. Sometimes these are performance advantages, while other times the general template may simply fail to work for a specific set of parameters. **Template specialization** allows us to define implementations for specific template parameter combinations that override the default instantiation. When the compiler encounters an instantiation matching a specialized version, it uses the specialized implementation instead of the generic one. Consider the following example:

```
#include <string>

template<class From, class To>
To convert(From x)
{
    return To(x);
}

int main()
{
    using std::string;
    double x { convert<int, double>(114) }; // x = 114.0
    char   y { convert<int, char>(114) };   // y = 'r'
    string z { convert<int, string>(114) }; // error!
}
```

The example above fails to compile because there is no viable conversion from `int` to `std::string`. While the generic template works for types with compatible constructors, some conversions require custom logic. If our project calls for an implementation, we can define one with template specialization. To declare a specialization, follow the `template` keyword with empty angle brackets `<>`, then write the specialized function declaration:

```
template<> std::string convert<int, std::string>(int x);
```

We then define our `int` → `string` algorithm within the specialization:

```
template<> std::string convert<int, std::string>(int x)
{
    std::string s {};
    do
    {
        s = std::string(1, char('0' + x % 10)) + s;
        x /= 10;
    } while (x > 0);
    return s;
}
```

The definition of a specialized template is also an instantiation:

```
// specialization.cpp

template<class T>
T max(T a, T b)
{
    return (a > b ? a : b);
}

template<> int max<int>(int a, int b)
{
    return a; // not sure why one would do this
}

int main() {}
```

When we compile and inspect the object file, we see the specialized version of `max<int>()` is instantiated:

Terminal

```
$ clang++ -std=c++23 -c specialization.cpp
$ objdump -tC specialization.o
SYMBOL TABLE:
0000000000000000 l F __TEXT,__text ltmp0
0000000000000020 l O __LD,__compact_unwind ltmp1
0000000000000000 g F __TEXT,__text int max<int>(int, int)
0000000000000018 g F __TEXT,__text _main
```