ADL as a customization point

What I wish My Mentors Had Told Me

What is ADL?

- Argument-Dependant Lookup is when the compiler, at compile time, decides where to look for functions, according to the arguments passed to that function.

```
func(x);
```

You might expect the compiler to look for `func` in:

- The current namespace
- Namespaces enclosing this namespace

What is ADL?

```
func(x);
```

Thanks to ADL, the compiler will also search in the namespace in which the type of `x` was declared.

Example: Simple ADL

```
namespace detail
{
   class A { };

   static int fabulate(A const a)
   {
      return 42;
   }
} // namespace detail

int main()
{
    return fabulate( detail::A{} );
    // Returns 42
}
```

Example: Simple ADL

```
namespace detail
{
   class A { };

   static int fabulate(A const a)
   {
      return 42;
   }
} // namespace detail

int main()
{
    return fabulate( detail::A{} )
    // Returns 42
}
```

How does the compiler find `fabulate` ???

Example: Simple ADL

```
namespace detail
{
   class A { };

   static int fabulate(A const a)
   {
      return 42;
   }
} // namespace detail

int main()
{
    return fabulate( detail::A{} )
    // Returns 42
}
```

How does the compiler find `fabulate` ???

Example: Simple ADL

```
namespace detail
{
   class A { };

   static int fabulate(A const a)
   {
      return 42;
   }
} // namespace detail

int main()
{
    return fabulate( detail::A{} )
    // Returns 42
}
```

How does the compiler find `fabulate` ???

Example: Simple ADL

```
namespace detail
{
   class A { };

   static int fabulate(A const a)
   {
      return 42;
   }
} // namespace detail

int main()
{
    return fabulate( detail::A{} )
    // Returns 42
}
```

How does the compiler find `fabulate` ???

Example: ADL in a Function Template

```
namespace detail {
   class A { };

   static int fabulate(A const a) { return 42; }
}

namespace rocket {
   class A { };

   static int fabulate(A const a) { return 51; }
}
```

Example: ADL in a Function Template

```
namespace detail {
   class A { };

   static int fabulate(A const a) { return 42; }
}

namespace rocket {
   class A { };

   static int fabulate(A const a) { return 51; }
}
```

```
template <typename T, size_t N>
static int zap(std::array<T,N> const src)
{
    return fabulate(src[0]) * src.size();
}
```

Example: ADL in a Function Template

```
namespace detail {
   class A { };

   static int fabulate(A const a) { return 42; }
}

namespace rocket {
   class A { };

   static int fabulate(A const a) { return 51; }
}
```

```
template <typename T, size_t N>
static int zap(std::array<T,N> const src)
{
    return fabulate(src[0]) * src.size();
}
```

A

```
int main()
{
    std::array<detail::A, 10> data;
    // Returns 420
    return zap(data);
}
```

B

```
int main()
{
    std::array<rocket::A, 10> data;
    // Returns 510
    return zap(data);
}
```

Example: ADL in a Function Template

```cpp
namespace detail {
   class A { };

   static int fabulate(A const a) { return 42; }
}

namespace rocket {
   class A { };

   static int fabulate(A const a) { return 51; }
}
```

```cpp
template <typename T, size_t N>
static int zap(std::array<T,N> const src)
{
    return fabulate(src[0]) * src.size();
}
```

A

```cpp
int main()
{
    std::array<detail::A, 10> data;
    // Returns 420
    return zap(data);
}
```

B

```cpp
int main()
{
    std::array<rocket::A, 10> data;
    // Returns 510
    return zap(data);
}
```

Example: ADL in a Function Template

```cpp
namespace detail {
   class A { };

   static int fabulate(A const a) { return 42; }
}

namespace rocket {
   class A { };

   static int fabulate(A const a) { return 51; }
}
```

T = detail::A

```cpp
template <typename T, size_t N>
static int zap(std::array<T,N> const src)
{
    return fabulate(src[0]) * src.size();
}
```

A

```cpp
int main()
{
    std::array<detail::A, 10> data;
    // Returns 420
    return zap(data);
}
```

B

```cpp
int main()
{
    std::array<rocket::A, 10> data;
    // Returns 510
    return zap(data);
}
```

Example: ADL in a Function Template

```
namespace detail {
   class A { };

   static int fabulate(A const a) { return 42; }
}

namespace rocket {
   class A { };

   static int fabulate(A const a) { return 51; }
}
```

```
T = detail::A
```

```
template <typename T, size_t N>
static int zap(std::array<T,N> const src)
{
    return fabulate(src[0]) * src.size();
}
```

A

```
int main()
{
    std::array<detail::A, 10> data;
    // Returns 420
    return zap(data);
}
```

B

```
int main()
{
    std::array<rocket::A, 10> data;
    // Returns 510
    return zap(data);
}
```

Example: ADL in a Function Template

```
namespace detail {
   class A { };

   static int fabulate(A const a) { return 42; }
}

namespace rocket {
   class A { };

   static int fabulate(A const a) { return 51; }
}
```

T = detail::A

```
template <typename T, size_t N>
static int zap(std::array<T,N> const src)
{
    return fabulate(src[0]) * src.size();
}
```

A

```
int main()
{
    std::array<detail::A, 10> data;
    // Returns 420
    return zap(data);
}
```

B

```
int main()
{
    std::array<rocket::A, 10> data;
    // Returns 510
    return zap(data);
}
```

Example: ADL for fun!

```
template<class InputIt, class T>
T accumulate(InputIt first, InputIt last, T init)
{
    for (; first != last; ++first) {
        init = init + *first;
    }
    return init;
}
```

```
namespace detail {
   class A { };

   static JSON::string operator+(JSON::string out, A const a) { /* ... */ }
}

int main()
{
   array<detail::A, 10> data;
   JSON::string json = accumulate( begin(data), end(data), JSON::string{} );
}
```

LIVE

CORE C++

ADL AS A CUSTOMIZATION POINT

June 28th

SHALOM CRAIMER | EXPLAIN ADL | SHOW HOW TO USE FOR CUSTOMIZATION OF TEMPLATES end