

# 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

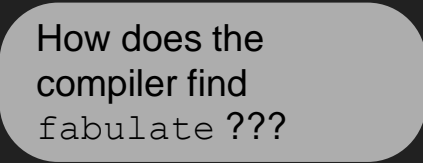
int main()
{
    return fabulate( detail::A{} );
    // Returns 42
}
```

# 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
}
```



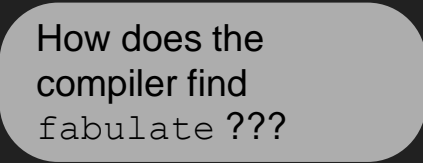
How does the  
compiler find  
fabulate ???

# 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 ???

A diagram consisting of two white arrows. One arrow starts from the `detail::A{}` argument in the `return fabulate( detail::A{} )` call in the `main` function and points to the `detail` namespace definition. The second arrow starts from the `return 42;` line inside the `fabulate` function and points to the `detail` namespace definition.

# 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
}
```

The diagram illustrates the resolution of the function call `fabulate` in `main`. A white arrow originates from the `fabulate` call in `main` and points to the `fabulate` function definition inside the `detail` namespace. A second white arrow points from the `detail` namespace label to the `fabulate` function definition, indicating that the compiler searches for the function in the namespace where the argument type `A` is defined.

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);  
}
```

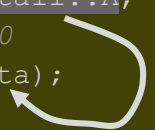
# Example: ADL in a Function Template

```
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

```
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 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!

```
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{} );
}
```

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



LIVE



CORE C++

# ADL AS A CUSTOMIZATION POINT

June 28th SHALOM CRAIMER | EXPLAIN ADL | SHOW HOW TO USE FOR CUSTOMIZATION OF TEMPLATES

end