

# Resolución de nombres

*¿en 5 minutos?*


using std::cpp 2017

David Rodríguez Ibeas <[dribeas@bloomberg.net](mailto:dribeas@bloomberg.net)>



*Las diapositivas se pueden presentar, a velocidad considerable, en 5 minutos. Por necesidad el texto es escaso incluyen una gran carga gráfica y depende de los comentarios durante la presentación.*

*En esta versión he añadido comentarios, identificados por:*

- fondo blanco*
- texto en cursiva*
- marca gráfica* 

# Resolución de nombres

- Objetivo



- Origen



- Indicaciones



- Camino



# Tipos de resolución

(básicos)

- Unqualified

`swap(a,b)`

- Qualified

`std::swap(a,b)`

- Fully qualified

`::std::swap(a,b)`



*La resolución de un nombre calificado no es búsqueda: la expresión contiene la descripción completa del nombre..*

# Tipos de resolución

(básicos)

- Unqualified

`swap(a,b)`

- Qualified

Búsqueda termina con el primer resultado:

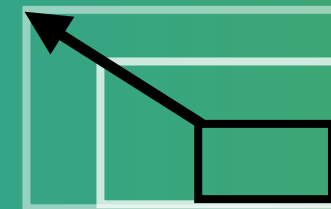
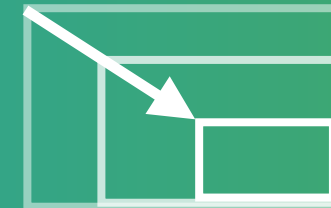
***ocultación***

- Fully qualified

`::std::swap(a,b)`

# Búsqueda

- C++      Top-Down
- Resolución Bottom-Up
- Ámbitos dentro (anidados)
- Resolución hacia afuera



# Ámbitos



- namespace
- clases
- función
- bloque

```
namespace A {  
    namespace B {  
        void f(bool c) {  
            if (c) {  
                // ...  
            }  
        }  
    }  
}
```

# Ámbitos

- namespace

```
namespace C {
```

- clases

```
}  
namespace A {
```

- función

```
namespace B {
```

```
void f(bool c) {
```

```
if (c) { X }
```

- bloque



*El namespace C es visible, pero sus contenidos (en negro) no lo son (búsqueda va hacia afuera). El código por debajo del nombre buscado tampoco es visible*



# Ámbitos

- namespace

- **clases**

- función



*Cuando la búsqueda mira el contexto de una clase continua por las bases. La búsqueda es en paralelo, no hay ocultación entre bases.*

```
class B1 {
```

```
};
```

```
class B2 : BB {
```

```
};
```

```
class D : B1, B2 {
```

```
class InnerB { ... }
```

```
class InnerD : InnerB {
```

# Ámbitos

- namespace


- **clases**



*Las funciones definidas dentro de la clase se consideran como definidas inmediatamente después de la llave de cierre. De esta forma la búsqueda hacia arriba encuentra todos los miembros de la clase.*

```
class C {  
    int f() {  
        return ++value;  
    }  
    int value;  
}
```

```
int C::f() {  
    return ++value;  
}
```



# Indicaciones

```
namespace A {  
    @class C { ... };  
}
```

```
namespace B {  
    void f() {
```



*Cuando el objetivo es un nombre que no está en el camino que el compilador sigue, ¿cómo encontrar ese nombre?*



```
}
```

```
}
```

# Indicaciones

- aliases

- directiva *using*

```
namespace A {  
    class C { ... };  
}
```

```
namespace B {  
    void f() {
```

```
        A:C c;  
    }
```



*Calificando el nombre cambiamos la resolución al nombre **A**, que sí está en el camino del compilador.*

# Indicaciones

- aliases

```
typedef X Y;    using Y = X;  
namespace N2 = N1;
```

- directiva  
*using*

```
using namespace std;
```

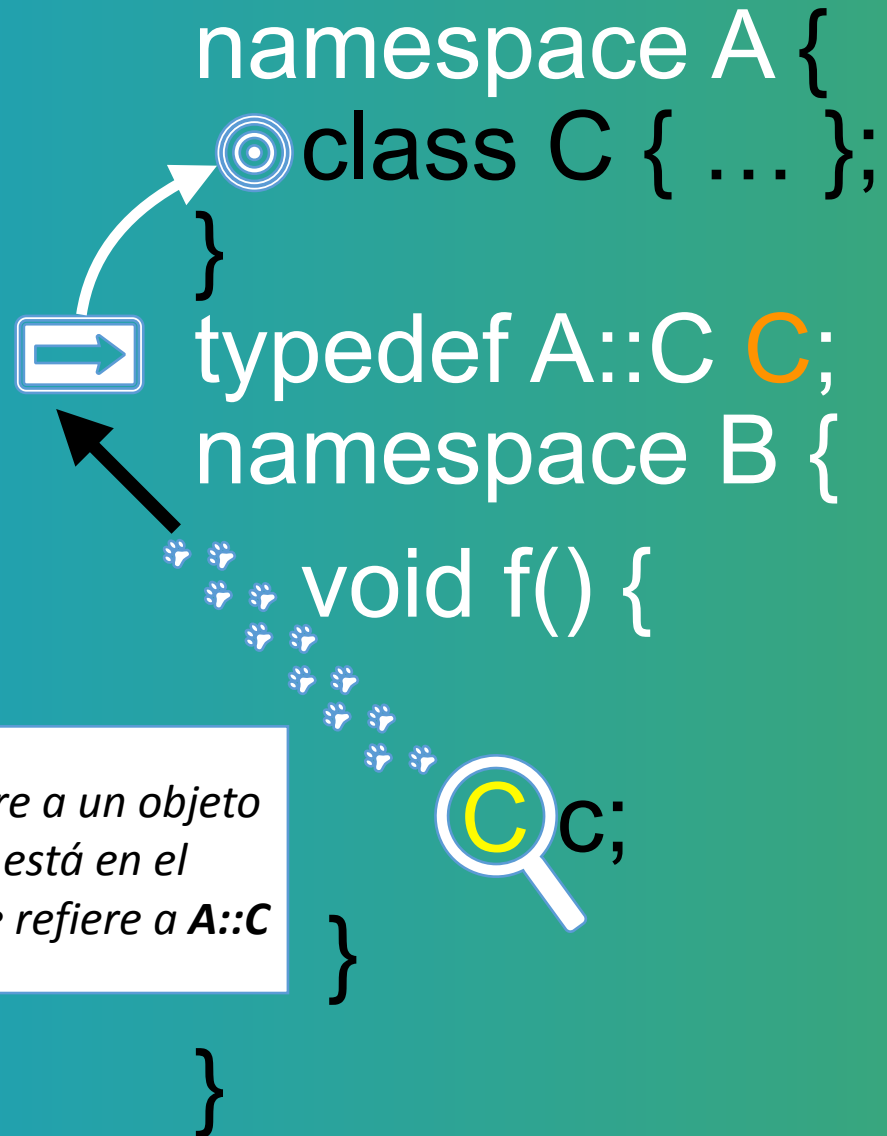
- declaración  
*using*

```
using std::swap;
```

# Indicaciones

- aliases

- directiva *using*



Los aliases proveen un nuevo nombre a un objeto existente. En este caso, el nombre **C** está en el camino que sigue el compilador y se refiere a **A::C**

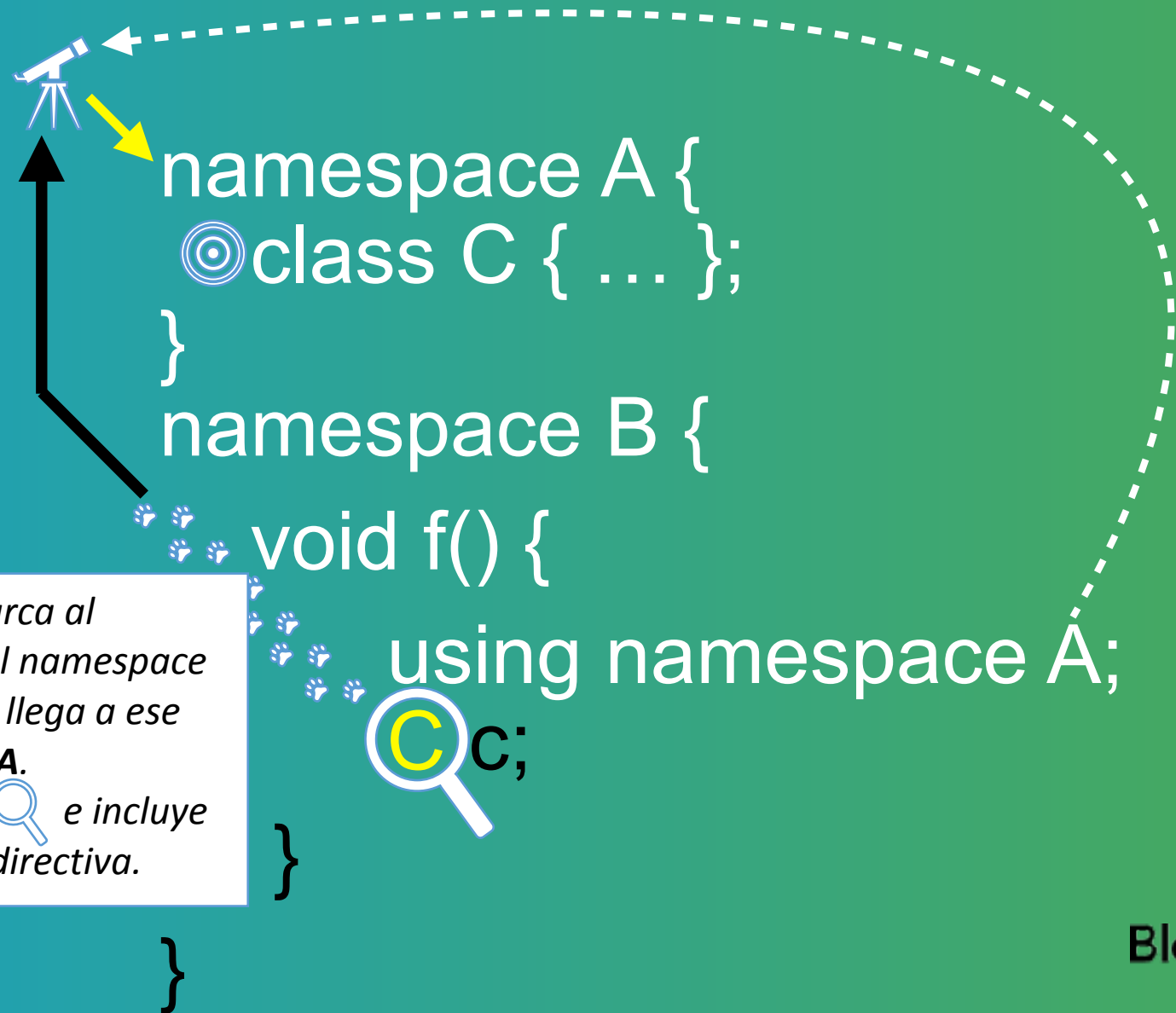
# Indicaciones

- aliases

- directiva  
*using*



Las directivas **using** añaden una marca al namespace común a la directiva y el namespace mencionado. Cuando el compilador llega a ese namespace incluye el contenido de **A**. La búsqueda es hacia arriba desde 🔍 e incluye nombres declarados después de la directiva.



# Indicaciones

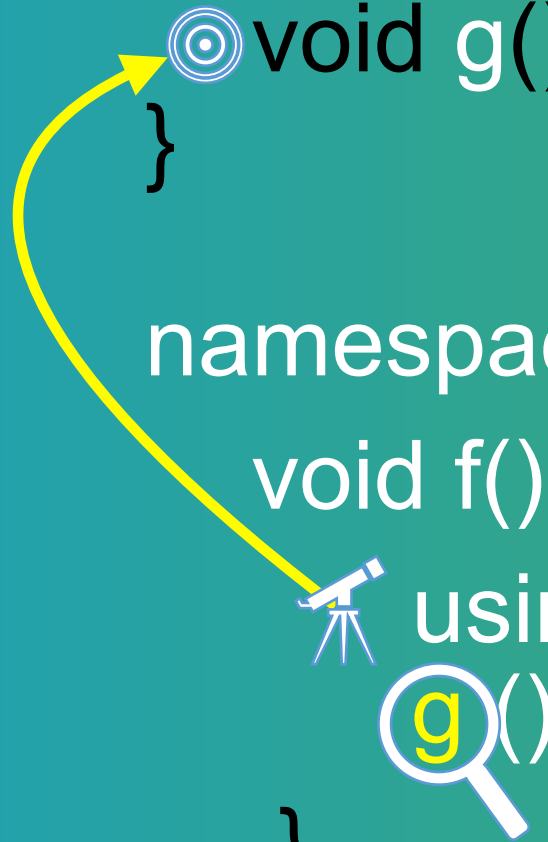
- aliases

- directiva *using*

- declaración *using*

```
namespace A {  
    void g();  
}
```

```
namespace B {  
    void f() {  
        using ::A::g;  
    }  
}
```



Las declaraciones **using** hacen el nombre re-declarado visible en un nuevo ámbito. Sólo los nombres presentes antes de la declaración **using** son visibles, la búsqueda de los nombres que se re-declaran se hace desde la declaración **using**, hacia arriba.



## - Argument Dependent Lookup (*Koenig lookup*)

*La interfaz pública de un tipo incluye las funciones libres definidas en el mismo **namespace**, como operadores o puntos de extensión (**swap**)*

## - Two phase lookup

*El significado de un nombre dentro de un **template** puede depender de los argumentos del **template***

# Argument Dependent Lookup

*La interfaz pública de un tipo incluye las funciones libres definidas en el mismo **namespace**, como operadores o puntos de extensión (**swap**)*

- Resolución de “llamada a función”  $f(x)$
- Resultado de resolución normal es:

- Nada
- **Función** (no miembro, no declaración local)

➡ **Añade namespace asociados a los argumentos**

- namespace del **tipo**, **bases**, argumentos de **template**

# Argument Dependent Lookup

```
namespace A {  
    class T {...};  
    ◎ void swap(T&, T&);  
}
```

```
namespace B {  
    class C {  
        void swap(C& o) {
```

```
            swap(t, o.t);
```




```
        }  
        ::A::T t;  
    };  
};
```

# Argument Dependent Lookup

```
namespace A {  
    class T {...};  
    void swap(T&, T&);  
}
```

Función miembro,  
No ADL

```
namespace B {  
    class C {  
        void swap(C& o) {  
            swap(t, o.t);  
        }  
        ::A::T t;  
    };  
};
```



# Argument Dependent Lookup

```
namespace A {  
    class T {...};  
    void swap(T&, T&);  
}
```

```
namespace B {  
    class C {  
        void swap(C& o) {  
            using std::swap;  
            swap(t, o.t);  
        }  
        ::A::T t;  
    }  
}
```

```
namespace std {  
    template <typename T>  
    void swap(T&, T&);  
}
```

**Función  
habilita ADL**



Aunque la declaración **using** se refiere a una función distinta de la que queremos, la resolución normal encuentra una función y eso habilita ADL, encontrando **A::swap**

*La declaración no calificada de una función **friend** de una clase declara una función en el namespace que contienen a la clase. Pero esta declaración no es visible a la resolución de nombres normal. Estas declaraciones sí son visibles a ADL.*

```
namespace N {  
    class A {  
        friend void f(int x) {}  
        operator int() { return 1; }  
    };  
}  
int main() {  
    N::A a;  
    f(a);  
    //::N::f(1);  
}
```



*La declaración **friend** declara **::N::f**, pero esa declaración sólo es visible para ADL. La llamada **f(a)** usa ADL e incluye la declaración de **N::f** dentro de la clase **A** debido al argumento **a**.*

*La función no está visible para la llamada **::N::f(1)** que no usa ADL.*

*Nota: **N::f** se encuentra porque el argumento de la llamada es de tipo **N::A**, no porque **N::f** tenga un parámetro de tipo **N::A**.*

# Templates

```
string ordinal(int position);
```

```
template <typename T>  
void print(shared_ptr<T> sp,  
           int pos) {  
    std::cout << ordinal(pos) << ": ";  
    std::cout << sp[pos];  
    std::cout << '\n' << std::flush;  
}
```



# Templates

- Compilación en dos fases:
  - **definición**  
*resolución normal*
  - **instanciación**

```
string ordinal(int position);
```

```
template <typename T>  
void print(shared_ptr<T> sp,  
           int pos) {  
    std::cout << ordinal(pos) << ": ";  
    std::cout << sp[pos];  
    std::cout << '\n' << std::flush;  
}
```



# Templates

- Compilación en dos fases:

- definición

- **instanciación**  
*ADL*

```
string ordinal(int position);
```

```
template <typename T>  
void print(shared_ptr<T> sp,  
          int pos) {
```

```
    std::cout << ordinal(pos) << ": ";  
    std::cout << sp[pos];  
    std::cout << '\n' << std::flush;
```



El template es correcto sólo si:

***shared\_ptr<T>::operator[] -> U (C++17)***

***ostream << declval<U>()***

*Sólo se puede saber si es correcto tras la **instanciación**, cuando **T** es conocido. Sólo es correcto si **T** es un array de un tipo que se puede insertar a un stream.*

# Resolución de nombres

- Objetivo



- Origen



- Indicaciones



- Camino

