



# Namespaces

Bachelor of Science - École polytechnique

[gael.thomas@inria.fr](mailto:gael.thomas@inria.fr)

# Key concepts

- A namespace avoids name collisions
  - `namespace id { .. };`
  - Isolate the symbols defined between the braces
  - Accessed with `id::`
- Importing a name space with `using namespace id;`
- We can alias a namespace with `namespace a = b;`
- A class can acts as a namespace with the `static` keyword

# Namespace

- New concept introduced in C++ to avoid name collision
  - `namespace id { .. };`
  - Isolate the symbols defined between the braces
  - Accessed with `id::`

```
namespace game {  
    struct monster_t {  
        const char* name;  
        int health;  
    };  
};  
  
int main(int argc, char* argv[]) {  
    game::monster_t m { "Pikachu", 42 };  
}
```

# Namespace

- We can implement a method of a namespace where we want

```
namespace game {  
    struct monster_t {  
        const char* name;  
        int health;  
  
        monster_t();  
        void print();  
    };  
  
    monster_t::monster_t()  
        : name { nullptr }, health { 0 } { }  
};  
  
void game::monster_t::print() {  
    printf("( %s, %d)\n", name, health);  
}
```

# Importing a namespace

- We can import a namespace with **using namespace id;**

```
namespace game {  
    struct monster_t {  
        const char* name;  
        int health;  
  
        void print();  
    };  
};  
  
using namespace game;  
  
void monster_t::print() {  
    printf("(%s, %d)\n", name, health);  
}
```

# Namespace aliasing

- We can alias a namespace with `namespace a = b;`

```
namespace x {
    namespace y {
        struct monster_t {
            const char* name;
            int health;
        };
    };
};

namespace z = x::y;

int main(int argc, char* argv[]) {
    x::y::monster_t m1 { "Pikachu", 42 };
    z::monster_t m2 { "Blastoise", 83 };

    return 0;
}
```

# static methods and fields

- A class can act as a namespace with the `static` keyword
  - A static method belongs to a class but does not have a receiver
  - A static field belongs to a class but is a global variable

```
struct point_t {  
    static int nb_points; // nb_points is global  
    int x;  
  
    point_t(int x) : x { x } {  
        nb_points++;  
    }  
  
    static void print_nb_points() {  
        std::cout << nb_points << std::endl;  
        // no access to x since we don't have a "this" parameter  
    }  
};  
  
// nb_points also has to be declared as a global variable  
int point_t::nb_points = 0;
```

# static methods and fields

- A class can act as a namespace with the `static` keyword
  - A static method belongs to a class but does not have a receiver
  - A static field belongs to a class but is a global variable

```
struct point_t {  
    static int nb_points; // nb_points is global  
    int x;  
    ...  
};  
  
...  
  
void f() {  
    // point_t acts now as a kind of namespace  
    // for the field nb_points and the method display()  
    point_t::display();  
}
```



# Key concepts

- A namespace avoids name collisions
  - `namespace id { .. };`
  - Isolate the symbols defined inside the braces
  - Accessed with `id::`
- Importing a name space with `using namespace id;`
- We can alias a namespace with `namespace a = b;`
- A class can acts as a namespace with the `static` keyword