✳ Namespace:

• İçinde isim koyabileceğimiz scope

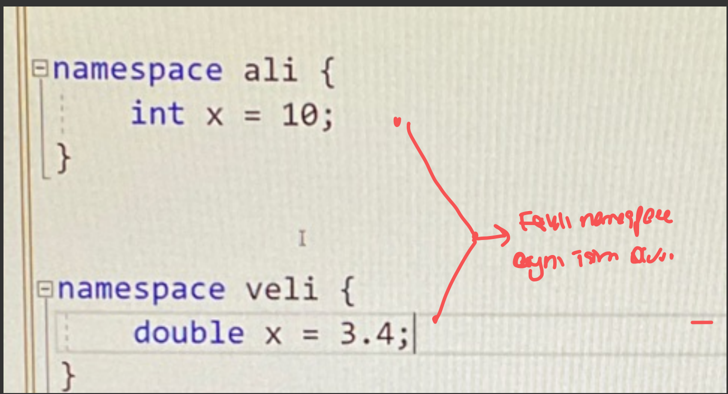
✳ C'de
- File
- Block
- Function Prototype
- Function

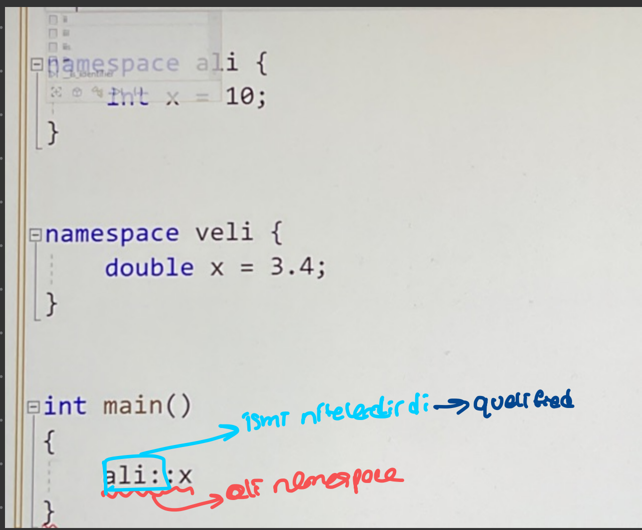
C++'da
- Namespace Scope
- Class Scope
- Block scope
- Function Prototype Scope
- Function scope

→ Namespace oluşturma: namespace nec {

} → sonunda noktalı virgül olmaz (terminator)

→ Nec namespace

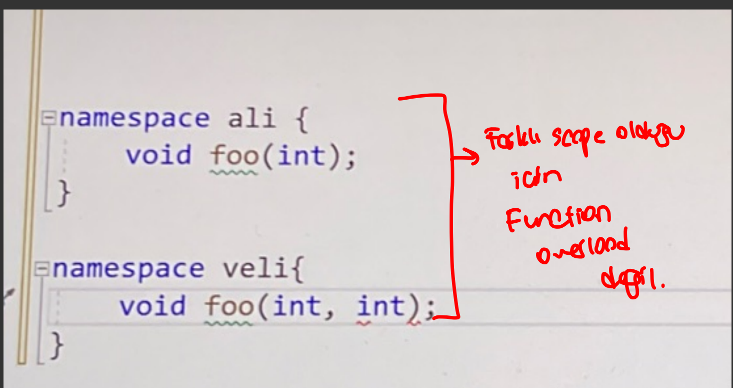




• Fonksiyon bloğu içinde namespace oluşturamayız !! → Global namespace'te tanımlanır.

• Başlık dosyasından gelenler hep std namespace içindedir.

• Namespace'in access kontrolü yok!! → private → protected gibi



```
namespace nec {

    namespace ali {

        namespace veli {
            int x;
        }
    }

}

int main()
{
    nec::ali::veli::x

}
```

→ nested namespace

```
namespace nec {
    int a, b, c;
}

namespace nec {
    int x, y;
}
```

x ve y de buraya dahil olur.

- Compiler bu iki tanımı birleştiriyor → kümülatif
- Böylelikle birden fazla header dosyası tek bir namespace'e eklenti yapabilir

→ Qualified olarak bildirimleri bune şöyle geçiriyor:

- using declaration
- using namespace directive (declaration)
- ADL = Argument Dependent Look-up

→ Using Declaration:

```
namespace ali {
    int x, y, z;
}

int main()
{
    using ali::x;

    x = 10;
    ali::x = 20;
}
```

→ Burada başka int x tanımlayamaz!

- Bildirilen ismi bildirimi yapılan scope'a enjekte edilmiş oluyor
  - ↓
  - o namespace'te tanımlanmış sayılacak.

```cpp
namespace ali {
    int x, y, z;
}

namespace veli {
    using ali::x;
}

int main()
{
    veli::x = 10;
}
```

Ali'deki x bu namespace'e enjekte edildi

Artık veli'nin'de x'i var.

```cpp
int g = 10;

namespace nec {
    int x = 4;
}

namespace ali {
    using ::g;
    using nec::x;
    int y = 10;
}

int main()
{
    ali::g = 10;
    ali::x = 10;
    ali::y = 120;
}
```

Statik ömürlü olurlar.

global değişkeni g

```cpp
#include <iostream>

int func()
{
    std::cout << "func is called!\n";
    return 5;
}
namespace nec {
    int x = func();
}
int main()
{
    std::cout << "main starts\n";
}
```

global değişken

Aslında bu da global namespace içinde tanımlanmış bir değişken. 1 numaralı yargı geçerli.

Sonra 2 olur.

*Fakat her zaman ilk fonksiyon main'dir derlemiyor.

→ Using Namespace Direktifi:

- Bir namespace'te ne var ne yok kullanmak için bu bildirimin de kapsamı vardır.

- Bildirime konu isim alanı içindeki isimleri bildirildiği isim alanına enjekte ETMEZ!!

```
namespace ali {
    int a, b, c;
}

int main()
{
    using namespace ali;

    a = 5;
    b = 7;
    c = 10;
}
```

Artık bu fonksiyon scope'unda tanımlı.

Bunlar legal.

```
#include <iostream>

namespace neco {
    int x, y;
}

using namespace neco;

int x;

int main()
{
    //x = 5;
}
```

Bu bildirim legal. Çünkü using namespace → enjekte etmez

Fakat bu ambiguity

```
namespace ali {
    void foo();
}

namespace veli {
    int foo = 5;
}

using namespace ali;
using namespace veli;

int main()
{
    foo();
    foo = 5;
    foo;
}
```

hepsi ambiguity

→ ADL: Argument Dependent Lookup: Koenig Lookup

```
namespace ali {

    class A {

    };

    void func(A);
}


int main()
{
    ali::A ax;

    func(ax);
}
```

→ Fonksiyona, bir namespace içindeki bir tür ile ilgili argüman gönderiyorsak, fonksiyonun ismini nitelemesek bile, ilgili namespace'te de arar

↓

hem yerde de arar ilgili namespace'te de

```
#include <iostream>


int main()
{
    //std::cout << "merhaba dunya";
    operator<<(std::cout, "merhaba dunya");
}
```

ADL olduğu için overload fonksiyon std:: yazmaya gerek yok

→ ADL olmasa (std::cout) olduğu için overload olamazdı.

```
namespace nec {
    class X {
        //...
    };

    void func(X)
    {
        std::cout << "nec::func\n";
    }
}

void func(nec::X)
{
    std::cout << "global func\n";
}

int main()
{
    nec::X nx;

    func(nx);
}
```

→ bu da ambiguity

```cpp
namespace nec {
    class X {

    };

    void func(X);
}


int main()
{
    int func(int);
    nec::X nx;

    func(nx);
}
```

→ Information hiding hatası

```cpp
#include <iostream>
#include <vector>


class Myclass {
public:
    friend void func(Myclass);
};


int main()
{
    Myclass m;

    func(m);
}
```

→ Friend bildirimi olsa bile, func bulunamayacaktı

→ Fakat şu an ADL devreye girdi ve buldu

→ Unnamed Namespace:

• C'deki global static variable ile aynı.

```cpp
//
namespace {
    int x = 10;
    void foo();
    class Myclass {

    };
}
```

→ Çünkü C++'da global static ile gizlemek deprecated.

→ Namespace Alias:

- C'de yalnızca typedef ile ters eş isim tanımlarken, C++'a usingle'de yapılır hale geldi.

- Çünkü template kullanımına uygun

```
typedef int (*fcmp)(const void*, const void*);

using fcmp = int (*)(const void*, const void*);
```

- Bir namespace nested ise, bunu yazmak zahmetli olabilir.

```
#include <iostream>


namespace ali {
    namespace veli {
        namespace necati {
            int x = 5;
        }
    }
}

namespace project = ali::veli::necati;

int main()
{
    project::x = 5;

}
```

namespace alias

- Version kontrolünde kullanılıyor.

- Fakat namespace isimleri de çakışabilir.