İSİM UZAYLARI VE NUMARALANDIRMA

İsim Uzayları

İsim uzayları (namespace), değişkenleri, metotları ve sınıfları bir isim altında toplayabileceğimiz bir tanımlamadır. Birden fazla kütüphane kullanırken isim çakışmalarını önlemek için kullanılır. Böylece aynı kimlikli sınıf, değişken ve fonksiyon birden fazla isim uzayında bulunabilir.

Örneğin, `fonksiyon()` kimlikli bir fonksiyonu olan bir kod yazıyor olabilirsiniz ve aynı fonksiyon aynı kimlikle başka bir başlıkta da mevcut olabilir. Bu durumda derleyicinin, kodunuz içinde hangi `fonksiyon()` fonksiyona atıfta bulunduğunuzu bilmesinin bir yolu yoktur. İsim uzayları, bu zorluğun üstesinden gelmek için tasarlanmıştır ve farklı kütüphanelerde bulunan aynı kimliğe sahip benzer fonksiyonları, sınıfları, değişkenleri vb. ayırt etmek için ek bilgi olarak kullanılır. İsim uzaylarına en iyi örnek, C++ standart kütüphanesidir (`std`). Bu nedenle, bir C++ programı yazarken genellikle `namespace std` kullanarak yönergeyi ekleriz;

İsim Uzayı Tanımlama

Bir isim uzayının tanımı, aşağıdaki gibi `namespace` anahtar sözcüğüyle tanımlanır;

```
namespace  isim-uzayı-kimliği
{
    // kod;
    // -değişken tanımlama (int a,b;)
    // -fonksiyon tanımlama (void topla();)
    // -sınıf tanımlama (class Kisi{ /* ... ?/ };)
}
```

Burada isim uzayına ilişkin bloğun noktalı virgül ile bitmediği gözden kaçırılmamalıdır. Çünkü isim uzayına ait bloğun görevi tanımlamaları gruplamaktır. İsin uzayının içindeki bir öğeye erişmek için kapsam çözümleme işleci (scope resolution operator) kullanılır. Ayrıca `using namespace` yönergesini (directive) kullanarak isim uzaylarını ile kapsam çözümleme işlecinin önek olarak eklenmesini de önleyebilirsiniz.

```
#include <iostream>
using namespace std; // std:: kullanmamak için

namespace birinci // birinci kimlikli isim uzayı
{
    int tamsayi=100; // Birinci isim uzayındaki tamsayi değişkeni
    void fonksiyon() {
        cout << "Birinci isim uzayındaki fonksiyon." << endl;
    }
}

namespace ikinci // ikinci kimlikli isim uzayı
{
    int tamsayi=200; // ikinci isim uzayındaki tamsayi değişkeni
    void fonksiyon() {
        cout << "İkinci isim uzayındaki fonksiyon." << endl;
    }
}

using namespace birinci;  //birinci:: kullanmamak için
int main () {
  fonksiyon(); // birinci isim uzayındaki fonksiyon çağrılır.
  cout << tamsayi << endl;

  ikinci::fonksiyon();
```

```
  cout << ikinci::tamsayi << endl;
}
/* Programın Çıktısı:
Birinci isim uzayındaki fonksiyon.
100
İkinci isim uzayındaki fonksiyon.
200

...Program finished with exit code 0
*/
```

İsim uzayları, aşağıdaki şekilde iç içe (nested) de yerleştirilebilir;

```
#include <iostream>
using namespace std; // std:: kullanmamak için

namespace birinci // birinci kimlikli isim uzayı
{
    int tamsayi=100; // Birinci isim uzayındaki tamsayi değişkeni
    void fonksiyon() {
        cout << "Birinci isim uzayındaki fonksiyon." << endl;
    }
    namespace ikinci // ikinci kimlikli isim uzayı
    {
        int tamsayi=200; // ikinci isim uzayındaki tamsayi değişkeni
        void fonksiyon() {
            cout << "İkinci isim uzayındaki fonksiyon." << endl;
        }
    }
}

using namespace birinci::ikinci;
int main () {
  birinci::fonksiyon(); // birinci isim uzayındaki fonksiyon çağrılır.
  cout << birinci::tamsayi << endl;

  fonksiyon(); // ikinci isim uzayındaki fonksiyon çağrılır.
  cout << tamsayi << endl;
}
/* Program çalıştığında:
Birinci isim uzayındaki fonksiyon.
100
İkinci isim uzayındaki fonksiyon.
200

...Program finished with exit code 0
*/
```

Bağımlı Argüman Arama

Açık bir ad alanı niteleyicisi olmadan bir işlevi çağırırken, derleyici, o işlevin parametre türlerinden biri de o ad alanındaysa, bir ad alanı içindeki bir işlevi çağırmayı seçebilir. Buna bağımlı argüman arama (argument dependent lookup- ADL) denir;

```
namespace Test
{
   int func(int i);
   class AClass {...};
   int func2(const AClass &data);
}
func(5); //Hata alınır. Hangi İsim Uzayından olduğu belli değil.
Test::AClass data;
```

```
Func2(data); //Hata alınmaz. Kullandığı argümana göre isim uzayı belli.
```

Bu durumda isim uzaylarının öğelerin başında kullanılması yerinde olacaktır.

İsim Uzaylarını Genişletme

Ad alanlarının kullanışlı bir özelliği de onları genişletebilmenizdir ya da yeni üyeler ekleyebilmenizdir;

```
namespace Test
{
    void fonk1() {}
}
//arada başka kodlar yer alabilir
namespace Test
{
    void fonk2() {}
}
```

Using Anahtar Kelimesi

1. `namespace` anahtar sözcüğüyle birleştirildiğinde bir `using` yönergesi yazarsınız;

```
namespace Test
{
    void func() {}
    void func2() {}
}
//...
Test::func2(); //kapsam çözümleme işleci kullanarak istediğimiz üyeyi kullanabiliriz
using namespace Test; //Test isim uzayını projemize dahil ediyoruz.
//Artık kapsam çözümleme işlecini kullanmamıza gerek kalmaz.
func();
func2();
```

2. Tüm isim uzayı yerine, isim uzayındaki seçili üyeleri projeye dahil etmek mümkündür;

```
using Test::func2;
func2(); //Başarılı bir şekilde projeye dahil edilmiştir.
func(); // Bu fonksiyon projeye dahil edilmemiştir.
```

3. Özellikle başlık (header) dosyalarında `using namespace` kullanmak çoğu durumda kötü görülür. Bu yapılırsa, isim uzayı dahil edilmiş başlığı içeren her dosya projeye aktarılır. Bu da çatışmalara yol açabilir. Örneğin; `AAA.h` Dosyamız;

```
namespace AAA
{
    class C;
}
```

   `BBB.h` Dosyamız;

```
namespace BBB
{
    class C;
}
```

   `CCC.h` Dosyamız;

```
using namespace AAA;
```

   `main.c` Dosyamız;

```
#include "BBB.h"
#include "CCC.h"
using namespace AAA;
C c; // Hata: BBB::C mi? AAA::C mi?
```

Numaralandırma Sınıfı

C++ dilinde kapsamlı numaralandırma (scopped enumaration) olarak da adlandırılan numaralandırma sınıfı (enumaration class), bir grup tamsayılardan oluşan (integral) değişmezden (literal) oluşan numaralandırılmış kullanıcı tanımlı bir veri tipidir. Bir bütünün parçalarını tamsayılar olarak ifade eden değişmezlere kullanıcı tanımlı adlar atamak istediğinizde kullanışlıdır.

```
enum class numaralandırmakimliği {
   adlandırılmış-tamsayı1 [=DEĞER1],
   adlandırılmış-tamsayı2 [=DEĞER2],
   ...
};
```

Sözde kodda `DEĞER1`, `DEĞER2` olarak belirtilenler aslında tamsayılara verilen isimlerdir. Yani tamsayı değişmezlerdir (integer literal). Bu değişmezler büyük harfle yazılırlar ve belirtilmedikçe ilkinin değeri `0`, ikincisinin değeri `1`, ... şeklinde ilerler. Örnek;

```
enum class Durum {
    OK = 0,     // 0
    Hata = 1,   // 1
    Uyarı = 2   // 2
};
```

Bu numaralandırma sınıfı öğeleri bir kod içerisinde yine kapsam çözümleme işleci (scope resolution operator) ile erişilir. İstenirse tamsayılardan oluşan numaralandırma başka bir tamsayı veri tipinde (`char`, `unsigned`, `short`, ...) de oluşturulabilir. Bu durumda numaralandırma sınıfı aşağıdaki şekilde tanımlanır.

```
enum class numaralandırmakimliği: farklı-bir-tamsayı-veritipi {
   adlandırılmış-tamsayı1 [=DEĞER1],
   adlandırılmış-tamsayı2 [=DEĞER2],
   ...
};
```

Örnek;

```
    enum class cinsiyetKodu:char {
        ERKEK = 'E',
        KADIN = 'K',
        BELIRSIZ = 'B'
    };
```

Farklı bir tiple numaralandırma sınıfı oluşturulmuş ise `static_cast` ile tip dönüşümü yapılmalıdır. Aşağıda buna ilişkin bir örnek verilmiştir.

```
#include <iostream>
using namespace std;
int main() {

    enum class cinsiyetKodu:char {
        ERKEK = 'E',
        KADIN = 'K',
        BELIRSIZ = 'B'
    };

    struct ogrenci {
        unsigned yas;
        char cinsiyet;
        float kilo;
        unsigned boy;
    } ogrenci1;

    ogrenci1.yas=19;
```

```
    ogrenci1.cinsiyet=static_cast<char>(cinsiyetKodu::ERKEK);
    ogrenci1.kilo=75.5;
    ogrenci1.boy=180;

    cout << "Öğrenci1'in:" << endl << "yaşı:" << ogrenci1.yas
         << ", cinsiyeti: " <<ogrenci1.cinsiyet
         << ", kilosu:" << ogrenci1.kilo
         << ", boyu:" << ogrenci1.boy << endl;

    struct ogrenci ogrenci2={25,static_cast<char>(cinsiyetKodu::KADIN),55.0,165};
    cout << "Öğrenci 2'nin:" << endl << "yaşı:" << ogrenci2.yas
         << ", cinsiyeti: " << ogrenci2.cinsiyet
         << ", kilosu:" << ogrenci2.kilo
         << ", boyu:" << ogrenci2.boy << endl;
}
/*Program Çıktısı:
Öğrenci1'in:
yaşı:19, cinsiyeti: E, kilosu:75.5, boyu:180
Öğrenci 2'nin:
yaşı:25, cinsiyeti: K, kilosu:55, boyu:165

...Program finished with exit code 0
*/
```