



Branch: master

Find file

Copy path

Cplusplus_Ders_Notlari / any_sinifi.md



necatiergin Rename any.md to any_sinifi.md d8e89e8 Dec 13, 2019

1 contributor

321 lines (264 sloc) | 12.9 KB

Raw

Blame

History



std::any Sınıfı

Öyle bir nesne olsun ki istediğimiz herhangi türden bir değeri tutabilsin. İsteddiğimiz zaman nesnemizin tuttuğu değeri herhangi türden bir değer olarak değiştirebilelim. C++17 standartları ile dile eklenen `std::any` sınıfı işte bu işe yarıyor.

C++ dilinin sağladığı en önemli avantajlardan biri tür güvenliği (`type safety`). Yazdığımız kodlarda değer taşıyacak nesnelerimizi bildirirken onların türlerini de belirtiyoruz. Derleyici program bu bildirimlerden edindiği bilgi ile nesne üzerinde hangi işlemlerin yapılabileceğini derleme zamanında biliyor ve kodu buna göre kontrol ediyor. C++ dilinde değer taşıyan nesnelerin türleri programın çalışma zamanında hiçbir şekilde değişmiyor.

`std::any` sınıfı herhangi bir türden değer tutabilirken bir değer türü (`value type`) olarak tür güvenliği de sağlıyor. `any` bir sınıf şablonu değil. Bir `any` nesnesi oluşturduğumuzda onun hangi türden bir değer tutacağını belirtmemiz gerekmiyor. `any` türünden bir nesne herhangi bir türden değeri tutabilirken sahip olduğu değer türünü de biliyor. Peki bu nasıl mümkün oluyor? Yani nasıl oluyor da bir nesne herhangi türden bir değeri saklayabiliyor? Bunun sırrı `any` nesnesinin tuttuğu değer yanısıra bu değere ilişkin `typeid` değerini de (`typeid`) tutuyor olması.

`any` sınıfının tanımı `any` isimli başlık dosyasında:

```
namespace std {
    class any {
        //...
    };
}
```

std::any nesnelerini oluşturmak

Bir `any` sınıf nesnesi belirli türden bir değeri tutacak durumda ya da boş olarak yani bir değer tutmayan durumda hayata getirilebilir:

```
#include <string>
#include <any>
#include <bitset>
#include <vector>

int main()
{
    using namespace std::literals;

    std::any a1{ 12 }; //int
    std::any a2 = 4.5; //double
    std::any a3{ "necati" }; //const char *
    std::any a4{ "necati"s }; //std::string
    std::any a5{ std::bitset<16>{} }; //std::bitset<16>
    std::any a6{ std::vector<int>{1, 3, 5} }; //std::vector<int>
    std::any b1; //boş
    std::any b2{}; //boş
    std::any b3 = {}; //boş
}
```

`any` sınıf nesnesinin kurucu işlevine gönderilen argümandan farklı türden bir değeri tutabilmesi için kurucu işlevin ilk parametresine standart `<utility>` başlık dosyasından tanımlanan `in_place_type<>` argümanının gönderilmesi gerekiyor. `any` tarafından tutulacak nesnenin kurucu işlevine birden fazla değer gönderilmesi durumunda da yine `in_place_type<>` çağrısındaki ilk argüman olmalı:

```
#include <any>
#include <string>
#include <complex>
#include <set>

int main()
{
    using namespace std;
    any a1{ "alican" }; //const char *
    any a2{ in_place_type<string>, "necati" }; //string
    any a3{ in_place_type<complex<double>>, 4.5, 1.2 }; //complex<double>
    auto fc = [](int a, int b) {return std::abs(a) < std::abs(b); };
    any a4{ in_place_type<set<int, decltype(fc)>>, {1, 4, 5}, fc }; //set<int>
    //...
}
```

`std::make_any<>` yardımcı işlevi

`any` türünden bir nesne oluşturmanın bir başka yolu da `make_any<>` yardımcı fabrika işlevini kullanmak. Burada `any` nesnesinin tutacağı değerin türü şablon tür argümanı olarak seçildiğinden `in_place_type<>` yardımcısının kullanılması gerekmiyor:

```
#include <any>
#include <string>
#include <complex>

int main()
{
    using namespace std;

    auto a1{ make_any<string>(10, 'X') };
    auto a2{ make_any<complex<double>>(1.2, 4.5) };
    //...
}
```

std::any nesneleri için bellek ihtiyacı

Bir `any` sınıf nesnesi tarafından tutulacak değerin bellek gereksinimi (`storage`) 1 byte da olabilir 5000 byte da. `any` nesnesi sahip olacağı değeri tutmak için heap alanında bir bellek bloğu edinebilir. Bu konuda derleyiciler istedikleri gibi kod üretebiliyorlar. Derleyiciler tipik olarak doğrudan `any` nesnesi içinde bir bellek alanını görece olarak küçük nesnelerin tutulması amaçlı kullanıyorlar. (C++17 standartları da böyle bir gerçekleştirimi öneriyor.) Eğer `any` tarafından saklanacak değer bu bellek alanına sığıyor ise değer bu alanda tutuluyor. Bu tekniğe "küçük tampon optimizasyonu" (`small buffer optimization SBO`) deniyor. Saklanacak nesne bu bellek alanına sığmıyor ise `heap` alanından bir bellek bloğu elde ediliyor. Aşağıda programı kendi derleyiciniz ile derleyerek çalıştırın ve `any` nesneleri için `sizeof` değerinin ne olduğunu görün:

```
#include <any>
#include <iostream>

int main()
{
    std::cout << "sizeof(any) : " << sizeof(std::any) << '\n';
}
```

Benim farklı derleyiciler ile yaptığım testlerin sonucu şöyle oldu:

GCC 8.1	16
Clang 7.0.0	32
MSVC 2017 15.7.0 32-bit	40
MSVC 2017 15.7.0 64-bit	64

std::any nesnesinin değerini değiştirmek

Sınıfın atama operatör işlevi ya da `emplace<>` işlev şablonu ile bir `any` sınıf nesnesinin değeri değiştirilebilir. Aşağıdaki kodu inceleyin:

```

#include <any>
#include <string>
#include <vector>
#include <set>

class Nec
{
    int mx, my;
public:
    Nec(int x, int y) : mx{ x }, my{ y } { }
    //...
};

int main()
{
    using namespace std;

    auto fcmp = [](int x, int y) {return abs(x) < abs(y); };
    any a;

    a = 12; //int
    a = Nec{ 10, 20 }; //Nec
    a.emplace<Nec>(2, 5); //Nec
    a.emplace<string>(10, 'A'); //string
    a.emplace<vector<int>>(100); //vector<int>
    a.emplace<set<int, decltype(fcmp)>>({ 1, 5, -4, -6, 3 }, fcmp);
}

```

std::any nesnesini boşaltmak

Bir değer tutan `any` nesnesini boşaltmak için sınıfın `reset` isimli işlevi çağrılabilir:

```
a.reset();
```

Bu çağrı ile `any` türünden `a` değişkeni eğer boş değil ise `a` değişkeninin tuttuğu nesnenin hayatı sonlandırılıyor. Bu işlemden sonra `a` değişkeni boş durumda. `any` nesnesini boşaltmanın bir başka yolu da ona varsayılan kurucu işlev (`default constructor`) ile oluşturulmuş bir geçici nesneyi atamak:

```
a = std::any{};
```

Atama aşağıdaki gibi de yapılabilir:

```
a = {};
```

std::any nesnesinin boş olup olmadığını sınamak

Bir `any` nesnesinin boş olup olmadığı yani bir değer tutup tutmadığı sınıfın `has_value` isimli üye işleviyle sınanabilir. (`boost::any` sınıfında olan `empty` üye işlevi yerine `has_value` isminin tercih edilmiş olması ilginç.)

```
bool has_value()const noexcept;
```

Aşağıdaki koda bakalım:

```
#include <any>
#include <iostream>

int main()
{
    using namespace std;
    cout.setf(ios::boolalpha);

    any a;
    cout << a.has_value() << '\n'; //false
    a = 45;
    cout << a.has_value() << '\n'; //true
    a.reset();
    cout << a.has_value() << '\n'; //false
    a = false;
    cout << a.has_value() << '\n'; //true
    a = {};
    cout << a.has_value() << '\n'; //false
}
```

std::any_cast<> işlev şablonu

any sınıf nesnesinin tuttuğu değere erişmenin tek yolu onu global any_cast<> işleviyle tuttuğu değerin türüne dönüştürmek. any_cast<> işleviyle, any sınıf nesnesi bir değer türüne bir referans türüne ya da bir pointer türüne dönüştürülebilir:

```
#include <any>
#include <string>
#include <iostream>

int main()
{
    using namespace std;

    string name{ "kaan aslan" };
    any a{ name };
    string s{ any_cast<string>(a) };
    s = "oguz karan";
    cout << any_cast<string>(a) << "\n"; //kaan aslan
    string& rs{any_cast<string &>(a) };
    rs = "necati ergin";
    cout << any_cast<string>(a) << "\n"; //necati ergin
    const string& crs{ any_cast<string &>(a) }
    crs = "ali serce"; //gecersiz
}
```

std::bad_any_cast

any_cast<> ile yapılan dönüşüm başarısız olursa yani dönüşümdeki hedef tür any nesnesinin tuttuğu tür ile aynı değilse bad_any_cast sınıfı türünden bir hata nesnesi gönderilir:

```
#include <any>
#include <iostream>

int main()
{
    using namespace std;

    any a{ 12 };

    try {
        int ival = any_cast<int>(a);
        cout << "ival = " << ival << '\n';
        a = 3.4;
        ival = any_cast<int>(a);
        cout << "ival = " << ival << '\n';
    }
    catch (const std::bad_any_cast& ex) {
        cout << "hata yakalandi: " << ex.what() << '\n';
    }
}
```

Burada gönderilen `bad_any_cast` sınıfı için türetme hiyerarşisi şöyle:

```
std::exception
  ^
std::bad_cast
  ^
std::bad_any_cast
```

`any_cast<>` dönüştürme işlevini kullanarak `any` tarafından tutulan nesneye gösterici (pointer) semantiği ile de erişilebilir. Ancak bu durumda şablon argümanı olarak kullanılan tür tutulan nesnenin türü değil ise bir hata nesnesi gönderilmez (`exception throw` edilmez), `nullptr` değeri elde edilir:

```
#include <any>
#include <iostream>

int main()
{
    using namespace std;

    any a{ 12 };

    if (int *p = any_cast<int>(&a))
        cout << *p << "\n";
    else
        cout << "tutulan nesne int türden değil\n";
    //...
}
```

tutulan nesnenin türünü öğrenmek

Sınıfın `type` isimli üye işlevi ile `any` tarafından tutulmakta olan nesnenin türü öğrenilebilir:

```
const std::type_info& type() const noexcept;
```

İşlevin geri dönüş değeri `any` nesnesinin tuttuğu değer tür bilgisini taşıyan `type_info` nesnesi. Eğer `any` nesnesi boş ise `type` işlevinin geri dönüş değeri `typeid(void)` olur. Bu işlevle erişilen `type_info` nesnesi

`type_info` sınıfının `operator==` işleviyle bir karşılaştırma işlemine sokulabilir. Aşağıdaki kodu inceleyelim:

```
#include <any>
#include <iostream>

int main()
{
    using namespace std;
    cout.setf(ios::boolalpha);

    any x;
    cout << (x.type() == typeid(void)) << '\n';
    x = 12;
    cout << (x.type() == typeid(int)) << '\n'; //true
    cout << (x.type() == typeid(double)) << '\n'; //false
}
```

std::any sınıfı ve taşıma semantiği

`any` sınıfı taşıma (`move`) semantiğini de destekliyor. Ancak taşıma semantiğinin desteklenmesi için tutulan değere ilişkin türün kopyalama semantiğini de desteklemesi gerekiyor. `unique_ptr<T>` gibi kopyalamaya kapalı ancak taşımaya açık türlerden değerler (`movable but not copyable`) `any` nesneleri tarafından tutulamazlar. Aşağıdaki kodda `string` nesnesinden `any` nesnesine ve `any` nesnesinden `string` nesnesine yapılan taşıma işlemlerini görebilirsiniz:

```
#include <any>
#include <string>
#include <iostream>

int main()
{
    using namespace std;

    string name("Dennis Ritchie");
    std::any a{ move(name) }; //name stringi a'ya taşınıyor.
    cout << any_cast<string>(a) << "\n";
    name = move(any_cast<string&>(a)); // a'daki string name'e taşınıyor
    cout << name << "\n";
}
```

std::any sınıfının kullanıldığı yerler

C++17 standartları öncesinde C++ dilinde yazılan kodlarda daha önce `void*` türünün kullanıldığı birçok yerde `any` sınıfı kullanılabilir. `void *` türünden bir gösterici (`pointer`) değişken, herhangi türünden bir nesnenin adresini tutabilir. Ancak `void*` türünden bir değişken adresini tuttuğu nesnenin türünü bilmez ve onun hayatını kontrol edemez. Ayrıca `void *` türü bir gösterici türü olduğu için "değer türü" (`value type`) semantiğine sahip değildir. `any` istenilen herhangi türden bir değeri saklayabilir. Tutulan nesnenin değeri ve türü değiştirilebilir. `any` tuttuğu nesnenin hayatını da kontrol eder ve her zaman tuttuğu nesnenin türünü bilir. Eğer tutulacak değerin hangi türlerden olabileceği biliniyorsa `any` yerine `variant` türünün kullanılması çok daha uygun olacaktır. Aşağıdaki kullanım örneği resmi öneri metninden alındı:

```
#include <string>
#include <any>
#include <list>

struct property
{
    property();
    property(const std::string &, const std::any &);

    std::string name;
    std::any value;
};

typedef std::list<property> properties;
```

Yukarıdaki kodda tanımlanan `property` türünden bir nesne hem istenilen türden bir değer saklayabilir hem de bu değere ilişkin tanımlayıcı bir yazıyı tutabilir. Böyle bir tür `GUI` uygulamalarından oyun programlarına kadar birçok yerde kullanılabilir. Bir kütüphanenin ele alacağı türleri bilmeden o türlerden değerleri tutabilmesi ve başka API 'lere bunları gönderebilmesi gereken durumlarda `any` sınıfı iyi bir seçenek oluşturabilir. Betik (`script`) dilleriyle arayüz oluşturma, betik dilleri için yazılan yorumlayıcı programlarda böyle türlere ihtiyaç artabiliyor.

`any` sınıfının tasarımında büyük ölçüde `Kelvin Henney` tarafından yazılan ve 2001 yılında `boost` kütüphanesine eklenen `boost::any` sınıfı esas alındı. `Kelvin Henney` ve `Beman Dawes` 2006 yılında ``WG21/N1939=J16/06-0009 belge numarasıyla `any` sınıfının standartlara eklenmesi önerisini sundular. Nihayet `Beman Dawes` ve `Alisdair Meredith`'in önerileriyle diğer kütüphane bileşenleriyle birlikte `any` sınıfı da `C++17` standartları ile dile eklendi. `boost::any` kütüphanesinde olmayan, `emplace` işlevi, `in_place_type_t<>` parametrelili kurucu işlev, küçük tampon optimizasyonu yapılabilmesi gibi bazı özellikler `std::any` kütüphanesine yer alıyor.