→ Regex Anlatımı son 52. Dakikada Bitiyor! Izlemeden Devam Edemezsiniz!

Nifty Counter / Schwarz Counter Idiom:

- Farklı header dosyaların, statik ömürlü değişkenin hayata geliş sırası belli değil (Static initialization fiasco)
- Fakat bazı nesnelerin hayata gelme garantisi olmalı: cout cerr gibi!

```
class Myclass {

public:
    Myclass()
    {
        std::cout << "hello from myclass\n";
    }

    ~Myclass()
    {
        std::cout << "hello from destructor\n";
    }
};


Myclass m;

int main()
{

}
```

Sınıfların constructor / destructorlarında bile olsa, daha önce hayata gelmiş.

#Wikibooks - C++ Idioms Oku

Ajan:

```
#ifndef STREAM_H
#define STREAM_H

struct Stream {
    Stream();
    ~Stream();
};

extern Stream& scout; // global stream object

struct StreamInitializer {
    StreamInitializer();
    ~StreamInitializer();
};

static StreamInitializer stream_initializer;


#endif // STREAM_H
```

bu bildirim yerine structtan önce de yazabilirim!

bu header'ı include eden her dosya için ayrı bir stream instance olur!

```cpp
// Stream.cpp
#include "stream.h"
#include <new>          // placement new
#include <type_traits> // aligned_storage

static int nifty_counter; // zero initialized at load time

static typename std::aligned_storage<sizeof(Stream), alignof (Stream)>::type
stream_buf; // memory for the scout object


Stream& scout = reinterpret_cast<Stream&> (stream_buf);

Stream::Stream()
{
    // initialize things
}

Stream::~Stream()
{
    // clean-up
}

StreamInitializer::StreamInitializer()
{
    if (nifty_counter++ == 0)
        new (&scout) Stream(); // placement new
}
```

Bir Dosyayı Stream'e Vermenin En Kolay Yolu:

```cpp
using namespace std;


int main()
{
    ifstream ifs{ "tor.txt" };
    //:

    cout << ifs.rdbuf();
```

pointer döndürür!

→ Dosya kopyalarak için de kullanılır!

* Conditionally Explicit Constructor (Cpp 20):

↳ Constructor'ın explicit olması = örtülü dönüşüme izin vermemesi!

```
class Myclass {
public:
    explicit Myclass(int, int);
};

int main()
{
    Myclass m = { 345, 845 };

    m = { 23, 56 };
}
```

explicit olduğu için conversion constructor çağrılan artık legal değil!

* Cpp 20:

```
class Myclass {
public:
    explicit(true) Myclass(int);
};
```

Compile time da evaluate edilebilen boolean bir değer

eğer true ise ctor explicit : Eğer false ise ctor explicit değil X

```
template <typename T>
class Myclass {
public:
    explicit(std::is_integral_v<T>)Myclass(T);

};

int main()
{
    //Myclass<int> m1 = 12;
    Myclass<double> m2 = 1.2;
}
```

integral türler için explicit.

bu syntax hatası

bu legal

*Bu yapıyı en çok class wrapperlarında kullanırız!

```cpp
class Nec {
public:
    Nec();
    Nec(int);
};

template <typename T>
class Wrapper {
public:
    Wrapper();
    template <typename U>
    Wrapper(U);
private:
    T mx;
};

int main()
{
    Wrapper<Nec> x = 5;
}
```

Eğer Nec'e dönüştürülüyorsa, Wrapper<Nec>'e de dönüşür!

```cpp
class Nec {
public:
    Nec();
    explicit Nec(int);
};

template <typename T>
class Wrapper {
public:
    Wrapper();

    template <typename U>
    Wrapper(U);
private:
    T mx;
};

int main()
{
    //Nec mynec = 5;
    Wrapper<Nec> x = 5;
}
```

Fakat bu explicit olmasına rağmen çağrı legal!

⇒Uyumluluk var!

*Çünkü wrapper'ın ctor'u explicit değil!

⇒

```cpp
class Nec {
public:
    Nec();
    explicit Nec(int);
};

template <typename T>
class Wrapper {
public:
    Wrapper();

    template <typename U>
    explicit(!std::is_convertible_v<U, T>)Wrapper(U);

private:
    T mx;
};

int main()
{
    Wrapper<Nec> m1 = 5;
}
```