Tekrar: → Formatter Interface

- ON/OFF flagler: Booleandır. setf ile fmtflags türünden maske geçilir.
-

```cpp
using namespace std;

auto n = cout.width(20);
```

n → ilk değeri return edildi

cout.width(20) → width set edildi.

-

```cpp
using namespace std;

cout.setf(ios::left, ios::adjustfield);
auto c = cout.fill('+');   // → boşluk yerine +
cout << (int)c << "\n";
cout.width(40);

cout << 345 << "murat";
```

-

```cpp
int main()
{
    using namespace std;

    cout << typeid(streamsize).name() << "\n";

    cout.setf(ios::fixed, ios::floatfield);
    cout.precision(12);

    cout << 736245.8723457834958234;   // Fixed'ten 12 basamak yazdırdı!
}
```

- Precision, noktadan sonra kaç eleman alınacağını set eder. Ancak print edilmesini garantilemez. Çünkü print edilmesi Fixed / Scientific olmasına da bağlı !!

- Bir ostream objesinin format state'i başka bir ostream nesnesine kopyalanabilir. Bunun için copyfmt fonksiyonu kullanılır.

```cpp
int main()
{
    using namespace std;

    cout.setf(ios::boolalpha | ios::uppercase | ios::showbase);
    cout.setf(ios::hex, ios::basefield);

    cout << 47802 << ' ' << (3 > 5) << '\n';

    ostringstream oss;

    oss.copyfmt(cout);

    oss << 57054 << ' ' << true;

    cout << oss.str() << '\n';
```

*** RAII idiom ile cout nesnesinin özelliklerini set edip, sonra ilk haline döndürme:** → mülakatta sorulabilir.

```cpp
#include <sstream>


class fmt {
public:
	fmt(std::ostream& os) : m_os{os}, m_flags{os.flags()} {}
		~fmt()
	{
		m_os.flags(m_flags);
	}
private:
	std::ostream& m_os;
	std::ios::fmtflags m_flags;
};


int main()
{
	using namespace std;

	{	// → eski haline çevirmek için, scope içine aldı
		fmt fm{ cout };
		cout.setf(ios::boolalpha | ios::uppercase | ios::showbase);
		cout.setf(ios::hex, ios::basefield);
		cout << 47802 << " " << 57054 << '\n';
	}

	cout << 47802 << " " << 57054 << '\n';
```

*** Ostream manipulators:**

```cpp
std::ostream& Endl(std::ostream& os)
{
	os.put('\n');
	os.flush();
	return os;
}


int main()
{
	using namespace std;

	cout << 12 << Endl << 34 << Endl;
```

→ Örneğin endl bir stream manipulator. Örneğin orijinalinden farkı, orijinal endl bir function template!

* Parametre almayan manipülatörler, ios başlık dosyasında

```cpp
std::ostream& sl(std::ostream& os)
{
	os << "\n*******************************************\n";
	os.flush();
	return os;
}


int main()
{
	using namespace std;

	cout << 12 << sl << 34 << sl << 4.5 << sl << "alican" << sl;

}
```

```cpp
std::ostream& BoolAlpha(std::ostream& os)
{
    os.setf(std::ios::boolalpha);
    return os;
}

std::ostream& NoBoolAlpha(std::ostream& os)
{
    os.unsetf(std::ios::boolalpha);
    return os;
}


int main()
{
    using namespace std;

    cout << (10 > 5) << BoolAlpha << (10 > 5) << NoBoolAlpha << '\n';
    cout << 57054 << Hex << 57054 << '\n';
```

* Parametre Alan Manipulatörler: → iomanip kütüphanesinde

- setw
- setprecision
- set fill gibi manipulatörler.
- setiosflag
- resetiosflags

→ Custom Parameterized Manipulators:

```cpp
class nl {  → Bunun için bir yöntem: NL bir class olsun
public:
    nl(int val) : mc{val} {}
    friend std::ostream& operator<<(std::ostream& os, const nl& nx)
    {
        for (int i = 0; i < nx.mc; ++i) {
            os.put('\n');
        }
        return os;
    }
private:
    int mc;
};

int main()
{
    using namespace std;

    int x = 35;
    double dval = 56.12;
    string name{ "eray" };

    cout << x << nl(5) << dval << nl(7) << name;
```

Operator << insertion olsun bu şekilde.

Istenilen sayıda newline

*Std:: quoted manipulatörü: •Çift tırnak içinde yazılmasını sağlar.

```cpp
int main()
{
    std::string str{ "samet yazici" };

    std::cout << std::quoted(str) << " " << std::quoted("musa sertkaya") << '\n';
}
```

→ kullanım alternatifleri

*Custom Inserter Functions:

→ Array inserter:

```cpp
#include "nutility.h"
#include <array>


template <typename T, std::size_t size>
std::ostream& operator<<(std::ostream& os, const std::array<T, size>& ar)
{
    os << '[';
    for (std::size_t i{}; i < ar.size() - 1; ++i) {
        os << ar[i] << ", ";
    }

    return os << ar.back() << ']';

}

int main()
{
    using namespace std;

    array a{ 2, 6, 8, 9 };

    cout << a << '\n';
}
```

*String stream Sınıfları: Kullanım amacı: okuma / yazma işlemlerini bellek üzerinde yapmak. (sprintf / sscanf)

include <sstream>

→Os tringstream:

```cpp
#include <sstream>


int main()
{
    using namespace std;
    int ival = 1243;
    char c{ ':' };
    double dval{ 30.7456 };

    auto s = (ostringstream{} << ival << c << dval).str();

    cout << s << "\n";
}
```

geçici osstringstream sınıfının bellek alanından siler.

ostringstream türünü.

.str() önemli.

Ayrıca manipüle edilebilir.

→Unutma:

Os tringstream kendi kendini flushlamaz, içinde tuttuğu değerler, akışı yapılmadığı / nesne yok olmadığı sürece silinmez. Aynı hafıza / bufferda durmaya devam eder.

```cpp
int main()
{
    using namespace std;

    Irand rand{ 0, 7000 };

    ostringstream oss;

    for (int i = 0; i < 10; ++i) {
        oss << '\"' << rand() << '\"';
        cout << oss.str() << '\n';
        oss.str("");
    }

}
```

hem set
hem get

.str() fonksiyonuna argüman geçmezsek eğer, ostringstream sınıfının buffer'ını temizler,

Alternatif olarak, local değişken ({} arasında, scope içinde) olarak kullanmak,

* SON 30 DAKİKADA SES - GÖRÜNTÜ KAYNIYOR *

→ istringstream Sınıfı:

```cpp
int main()
{
    using namespace std;

    istringstream iss("emre eray necati hasan safa samet");
    string name;

    while (iss >> name) {
        cout << name << "\n";
    }

}
```

→ cstring de olabilir.

→ getline global fonksiyonu: → getline'da boşluklar dahil satır alır!

```cpp
using namespace std;

string sline;

cout << "bir yazi girin: ";
getline(cin, sline);

istringstream is{ sline };

string word;

while (is >> word) {
    cout << word << '\n';
}
```

→ bu şekilde de istringstream oluşturulabilir.

→ Aynı şekilde bir cümlenin da kaydedilir.

```
{
    using namespace std;

    string sline;

    cout << "bir yazi girin: ";
    getline(cin, sline);

    istringstream is{ sline };

    string word;
    set<string> myset;

    while (is >> word) {
        myset.insert(word);
    }

    for (const auto& s : myset)
        cout << s << ' ';


}
```