********** NECATİ ERGİN C++ **********

-----enum class-----

```
enum class Color {white, Red,Black};
 int main()
 {
          Color color=Color::Black;
```

------ L value referans sematiği ------

```
int x=20;
int &r = x; //burda & declaration, operator değil
r=40;
//pointer ve referans arasında performans farkı yok.
//c++' da elemanları referans olan dizi oluşturulamaz.
```

------ expression ------

```
the value category of an expression
//nesne alanlar L value expression
int x=10 ;
int *ptr =&x;
x, ptr, *ptr //C' de L value
&x, x+4, ++x //C' de R value
```

------ value category in c++ ------

```
L value
PR value (pure R value)
X value  (Expiring value)

//Nesnelerin isimleri olan ifadeler her zaman L value expression' dır.
int x, int x[], a[2]...
++x, --x
a > 1 ? x : y =10;

//ifadelerle oluşturulanlar R value
x+4 ...
x++, x-- // c de l value
```

	C	c++
++x	r	L
-xx	R	L
x++	R	R
(x , y)	R	L
a > 1 ? x : y	R	L
atama	R	L
foo()		r
&foo()		L

```
int x=10;
int &r = x++; //geçersiz x++ r val.
int &r = ++x; //geçerli

const int x=10;
int &r1 = x; //geçersiz
const int &r2 = x; // geçerli


void swap(int &a, int &b)
{
	int temp=a;
	a=b;
	b=temp;
}


int a[5]={1,2,3,4,5};
int (*ptr)[5] = &a //*ptr demek a demek
//referans ile....
int (&ra)[5]=a;s
```

	pointer	referans
default initi.	int *p;	-
x=10, y=20	ptr=&x, ptr=&y	doğuştan const//not rebandleble
NULL	NULL pointer var	NULL pointer yok
array	int *p[10]	-
x=10, y=20	int *p=&x; int **ptr=&p	- (type deduction haricinde yok)

------- type deduction (tür çıkarımı) -------

-- auto --

```
//derleme zamanını etkiler
//auto, decltype,decltype(auto), template, lambda expression

auto x= "orkun"; //const char[6]

//auto da referans ve const düşler
const int x=10; , auto y= x; //int y=x olur;

//auto' da & referans kullanıldıgında const düşmez
const int x= 10;
auto& r=x; // r=const int

int a[10]{};
auto x = a; // array decey

int a[]{1,2,3,4,5,6};
auto &x = a; //int[6];
int(&x)[6] = a;


auto x= "asdf"; //x const char*

auto &x = "ezgi"; //const char[5]
const char (&y)[5] = "ezgi";


int fonc(int);
int (&fptr)(int) = func; //func referance
auto &fptr=func; //aynı

const char *func(const char*, const char*);//func declaration
const char *(*fp)(const char*, const char*) = &func;
auto fp = &func; //aynı


//universal reference
//forwarding reference
int x{};
auto &&rand = x; // l value normalde && pr value
a) ilk defer veren ifade L value expression ise auto karşılıgı olan tür sol taraf referansı
a) ilk defer veren ifade R value expression ise auto karşılıgı olan türün kendisi
        //normalde c++ dilinde referansa refereans olmaz,
```

ancak tür çıkarımı yapılan bağlamlarda tür çıkarımı
sonucunda referansa referans oluşabilikmektedir.
bu durumda derleyici "referance collapsing kuralını uygular.

-- decltype --

-- A --

decltype operatorünün operandı bir isim formunda ise
örnk: decltype(x), decltype(ptr->x), decltype(a.x)

```
int a[10] {};
decltype(a) c[20] // int c[20][10];
```

```
const int x= 56;
decltype(x) y; // hatali, const a ilk deger verilmeli
```

```
int x = 10;
decltype(x + 5)y; //y nin türü int
```

```
int x=10;
int y=20;
int *ptr = &x;
decltype(*ptr) z = y; //int &
```

```
int x=10;
int &r = x;
decltype(r) // type -> int &
```

```
int a[10]{};
decltype(a) // type -> int [10]
```

```
const int b[3] {1,2,3};
decltype(b) // type -> const int [3]
```

```
int x=50;
```

```
int &r = x;
decltype(r) t; // type -> int& ; ---> syntax error


int a[10]{};
decltype(a) k; type -> int k[10]


int a[10]{};
decltype(a) c[20]; type -> int c[20][10]


const int x = 50;
decltype() y; //syntax error type -> const x; //initial
```

-- B --

decltype operatorünün operandı bir isim formunda değil ise
örnk: decltype(x+5), decltype(*ptr), decltype((x))

//bu durumda decltype karşılığı elde edilen tür parantez içindeki value category' sine bağlı

a) eğer ifade PR value expression ise decltype yerine gelen tür taraf T
b) eğer ifade L value expression ise decltype yerine gelen tür T&
c) eğer ifade X value expression ise decltype yerine gelen tür T&&

```
int x = 10;
decltype(x + 5) y; // pr value expression type -> int


int x = 10;
int y = 20;
int *ptr = &x;
decltype(*ptr) z = y; // *ptr l valu expression z type -> int&


int x = 10;
int y = 20;
decltype(x) a = y; // a  pr value type -> int
decltype((x)) b = y; // b l value type int&
++b; // y = 21 olur
++a; // y =20 olur


int x = 10;
```

```
int y = 20;
decltype(++x) z = y; // z l val. type -> int&
++z; // y =21
cout<<x; //x =10
```

unevaluated context: Bazu durumlarda yazılan kodda bir ifade olmasına karşı derileyici dilin kuralların göre o ifade icin bir işlem kodu üretmez.
ornk: auto y = sizeof(x++);
decltype(x++) y ;

```
int x = 10;
auto p = &x; // p type -> int*
auto *ptr = &x; ptr type -> int*
        --------  -------
```

------ constexpr ------

```
const int x = 10;
cons int z = foo();
int a[x];
int b[z]; // syntax error
constexpr int y = foo(); //syntax error
constexpr int c = x * 5; //geçerli


//sabit ifadeler compile time da hesaplanır
constexpr int square(int x, int y)
{
        return x * x + y * y;
}
int main()
{
        constexpr int a = 5;
        constexpr int b = 7;
        constexpr int c = square(a, b);
        // c nin 74 oldugu compile time da hesaplanır
}

const int x = 100;
const int y = 200;
int a[sum_square(x,y,1)]{}; //geçerli
```

==========

---- function overloading ----

```
void foo(int = 5, int); // hahtalı
void foo(int, int = 5, double = 7); // gecerli
default degerin sağındakiler de default olmalı



void foo(int = 1, int = 2, int z = 3);
int main()
{
	foo(10,20,40);
	foo(10,20);
	foo(10);
	foo();
}
void foo(int x, int y, int z)
{
	std::cout << "x = " << x << "y = " << y <<"z = "<<z<<std::endl;
}




void foo(const char *p = "error");
int main()
{
	foo();
}
void foo(const char *p )
{

	std::cout << p<<std::endl;
}




inf foo(int x = 10);
int func(int y = foo());
```

```
int main()
{
	func(); // func(foo(10))
}
```

```
int foo()
{
	static int x = 0;
	std::cout << "foo cagırıldı\n";
	return ++x;
}
void func(int x = foo())
{
	std::cout << "x = "<<x<<std::endl;
}
int main()
{
	func();
	func();
	func();
}
```

```
  output
foo cagırıldı
x = 1
foo cagırıldı
x = 2
foo cagırıldı
x = 3
```

```
int foo(int a, int b = a); // hatali
```

```
void func(int, int , int =20);
void func(int, int = 10 , int );  // gecerli
```

```
void func(int, int *ptr = nullptr);
func(12); varsayılan null ptr
func(12,&x);
```

```
void print_date(int day = -1, int mon = -1, int year = -1)
{
	if (year == -1) {
	std::time_t timer;
	std::time(&timer);
	auto p = std::localtime(&timer);
	year = p->tm_year + 1900;
	if(mon == -1) {
	mon = p->tm_mon + 1;
	if(day == -1) {
		day = p-> tm_mday;
	}
	}
	}
	printf("%02d-%02d-%d\n",day, mon, year);
}
int main()
{
	print_date();
	print_date(18);//day
	print_date(18,5);//day,mon
}
```

```
int func(int)
int func(int, double)
int func(double)
int func(int *)
4 overloading

void func(int x);
void func(const int x);
rederation


void func(int *p);
void func(int * const p);
//function overloading degil

void func(int *p);
void func(int const * p);
//function overloading


typedef int itype;
using itype = int; //aynı
```

```
tpdef int (*fptr)(int);
using fptr = int (*)(int);


void func(int &);
void func(const int &);
overloading


void func(int *);
void func(const int *);
overloading

void func(int &);
void func(const int &);
void func(int &&);
3 overloading


void func(int[]);
void func(int *);
rederation


function overload resolation
        a) gecerli
        b) gecersiz
        - no match
        - ambiguitiy //uygun olan birden fazla secenek olması

1) candidate functions
2) viable functions
        - parametre sayısı arguman sayısına eşit olucak
        - parametre türü ile argümanın türü gecerli bi dönüşüm olucak


void func(bool);
void func(void *);
int main()
{
        int x{};
        func(&x);
}
//2 func da viable


argümandan parametreye yapılan dönüşüm
```

- variadic conversion // en düşük dönüşüm
	void func(...);

-user difened conversion(programcı tanımlı dönüşüm)

- standard conversion
	a) exact match
	array decay
	const conversion
	function to pointer conversion

	---

	void func(long double);
	void func(char);
	int main()
	{
	func(2.4L);
	func('A');//c de int c++ da char
	}
	---

	void func(const int*);
	int main()
	{
	int x{12};
	func(&x); // int * ==> const int*
	}
	---

	*array decay ile yapılan da exact match
		void func(int *); ....
		int a[]{1,2};
		func(a);
	---

	void func(int (*)(int)); //function pointer
	int foo(int);
	int main()
	{
		func(foo);// &foo
	}

	b) promotion
	- integral promotion
	char ==> int
	signed char ==> int
	short ==> int

```
	bool ==> int
	unsigned short ==> int

	float ==> double
	c) conversion
```

```
void func(int &)
{
	std::cout<<"func(int &)\n";
}
void func(const int &)
{
	std::cout<<"func(const int &)\n";
}
//int func(int x);
int main()
{
	int x = 10;
	const int y{2};
	func(y); // int to const olmadıgından int & vaible
	func(x); // 2 func da vaible ama tercih const int&
}
//output
	func(const int &)
	func(int &)
```

```
void func(Myclass &);
void func(Myclass &&); //overloading


void func(int &)
{
	std::cout<<"func(int &)\n";
}
void func(int &&)
{
	std::cout<<"func(int &&)\n";
}
int main()
{
```

```
	int x {14}; //l value
	func(x); //func(int &)
	func(250); // func(int &&)
}
```

```
void func(const T&);
void func(T &&);
// L value ile cagırılırsa const T&
// R value ile cagırılırsa ikisi de vaible olur
	fakat && üstünlüğü var


void func(const int &)
{
	std::cout<<"func(const int &)\n";
}
void func(int &&)
{
	std::cout<<"func(int &&)\n";
}
int main()
{
	func(250); //func(int &&)
}
```

* Parametre pointe degilse constluk imza farklılıgı olusturmaz.

```
	int g(int);
	int g(const int); //function overloading yok
```

parametre sayısı 1' den fazla ise fonksiyonun
secilebilmesi icin :

	a) en az bir parametre digerlerinden üstün gelicek
	b) diger parametreler daha kötü olmayacak

```
void func(int);   //1
void func(double);     //2
void func(long);          //3
```

```
void func(bool);     //4

void foo()
{
   int x = 10;

   func('A'); // 1 promotion
   func(2.3F); // 2 promotion
   func(4u);  //  ambiguitiy
   func(10 > 5); // 4
   func(&x); // sadece 4 vaible
   func(nullptr); // no match
}
```

```
void func(void *);  //1
void func(bool); //2

void foo()
{
   double x = 1.0;

   func(0); // ambiguitiy
   func(nullptr); // 1 secilir, 2 vaible degil
   func(&x); // ikisi de vaible ama 1 cagırlı
   func(x); // 2
}
```

```
void func(char *p);  //1
void func(const char *p); //2

void foo()
{
   char str[] = "Herb Sutter";
   const char cstr[] = "Stephan Lavavej";
   char *const p1 = str;
   const char *p2 = str;
   func(nullptr); // ambiguitiy
   func("Bjarne Stroustrup"); //array decay ile const char*, 2
   func(str); // 1
   func(cstr); // 2
```

```
    func(p1); // 1
    func(p2); // 2
}

========================================
```

---- classes ----

data members
- non-static data members
- static data members

```
class A {
        int m; //non-static member
        static int y; //static member
};


class B {
        int a;
        double z;
}; // B nin sizeof' u 4 + 8
// empty class sizeof 1 byte


//instance/object sınıf türünden nesneye denir.
//B x; // instantiate -> olusan nesne
```

-- scopes in C
- -block scope
- -file scope
- -function prototype scope
- -function scope

-- scopes in c++
- -block scope
- -namespace scope
- -class scope

-function prototype scope
-function scope

scope resolution // cözünürlük op.
- ::x // unary
- data::x //binary
a) namespace
b) class

// sınıfın üye fonksiyonları rederation olamaz

```
class Myclass {
void func(int); //function member
void func(int); // syntax error
}
```

// class Myclass; //bildirimi
// class a {}; //defination

x.h

```
#ifndef DATE_H
#define DATE_H
class Date {
public:
void set(int d, int m, int y);
};
#endif
```

x.cpp

```
#include "date.h"
void Date::set(int d, int m, int y)
{
}
```

```
class A {
public:
    void foo();
private:
```

```
        int y;
};
void foo(int x)
{

}
void A::foo()
{
        foo(12) //syntax error
}
//isim arama (name look up)
        - ilk bulunduğu scope da aranır bulunmazsa
        class scope a geçer.

//yukarıdaki örnekte foo(12) deki fooo A classındaki
        parametresiz fonksiyondur.

//global foo yu cağırmak için
        ::foo(12);

//class ın her hangi member functionında private erişilebilir
        A global;
        A::foo()
        {
        global.y = 5;

        A loc;
        loc.y=10;
        }


void func(Data *p); // nutator, set function
void foo(const Data *p); // accessor, get function


class B {
public:
        void func(); //nutator
        void foo()const; //accessor //const member function
private:
 int mx;
}
void B::func(){}
void B::foo()const {}

non-static member functions
        a)non-const
        b)const
```

```
void B::func()
{
	mx =20;
}
void B::foo()const
{
	mx = 20; //syntax error
	func(); //syntax error const T* türünden T* a dönüşüm yok

	B newobj;
	newobj.mx=20; //gecerli
}
```

=> const üye işlevlerde sınıfın non-static
veri elemanları değiştirilemez.//syntax error

=> const üye işlevlerde sınıfın non-const üye fonksiyonarı cağırılamaz

=> const sınıf nesneleri için sınıfın non-const
üye fonksiyonarı çağırılamaz

---> mutable: const üye fonksiyonlar içinde degerinin
degisitirilmesine izin verir

```
class Fighter {
public:
	int get_age()const;
	mutable int debug_call_count{};
	int debug_call_count_1{};

};

int Fighter::get_age()const
{
	++debug_call_count;
	++debug_call_count_1{}; //syntax error

}
```

*** this ***

-this is keyword
-this is a pointer
-PR value expression

```
class S {
public:
	void func();
	void foo();
private:
	int mx;
};

void S::func()
{
	std::cout << "this: " << this;
}

void S::foo()
{
	mx = 10;  //unqualified
	S::mx = 20; //qualified
	this->mx = 30;
	// 3 deger de aynı
}

int main()
{
	S obj;
	std::cout <<"&obj = "<< &obj; //obj.func() aynı adres
	obj.func();

	foo();
	S::foo();
	this->foo() // anlamsal olarak aynı
}
```

---> qualified name (nitelenmiş isim), bir ismi çözünürlük operatorunun operandı olarak kullanılması.

---> *this, hangi nesne icin cagırıldıysa o nesnenin kendisine erisir.

```
        S newobj;
        newobj.func(); // *this = newobj
```

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

void g1(Myclass *p);
void g2(Myclass &p);
void g3(Myclass p);

void Myclass::func()
{
        g1(this);
        g2(*this);
        g3(*this);
}
```

```
class Myclass {
public:
        Myclass &func();
        void foo();
        Myclass &f()const;
}

Myclass& Myclass::func()
{
        //....
        return *this;
}
Myclass& Myclass::f()const
{
        return *this; // syntax error
}

int main()
{
```

```
        Myclass cl;

        cl.func();
        cl.foo(); // 2 satır asagıdaki ile aynı
        cl.func().foo(); //ilk func cagırılır
}
```

--> sınıf türünden const fonk, this döndüremez

```
class A {
public:
        void func();
        void func()const; //function overloading
}
```

*** constructor - destructor ***

special member functions
- -default constructor
- -destructor
- -copy constructor
- -move constructor
- -copy assignment
- -move assignment

constructor
- a) ismi sınıfın ismi
- b) global fonk. olamaz
- c) non-static
- d) geri dönüş kavramı yok
- e) const üye fonk. olamaz
- f) public, private, protected olabilir
- g) overload edilebilir

destructor
- a) sınıf ismiyle aynı başında ~
- b) const üye fonk. olamaz
- c) public, private, protected olabilir
- d) non-static
- e) global fonk. olamaz
- f) overload edilemez
- g) parametre degiskeni yok

l) geri dönüş kavramı yok

--> default constructor: parametre degiskeni olmayan,
tüm parametreleridefault alan constructor
-> default initial edildiklerinde sınıfın default constructor
fonksiyonu cagırılır.
Myclass n; // default constructor cagırlır

```
class Myclass {
public:
  Myclass();
  ~Myclass();
};

Myclass::Myclass()
{
  std::cout<< this <<" adresinde bir Myclass nesenesi hayata geldi\n";
}
Myclass::~Myclass()
{
  std::cout<< this <<" adresinde nesne hayata veda ediyor\n";

}

Myclass g;

int main()
{
  std::cout<< "main basladı\n";
  std::cout<< "main sona erdi\n";
}
// output
        0x558c00d4e151 adresinde bir Myclass nesenesi hayata geldi
        main basladı
        main sona erdi
        0x558c00d4e151 adresinde nesne hayata veda ediyor
```

-> main calısmadan önce nesne hayata gelir
-> program sonlanmadan hemen önce destructor calısır//mainden sonra

```
class A;
class B;
class C;

A a;
B b;
C c;
// ilk önce a nesnesinin constructorı calısır daha sonra sırayla b ve c
// destructor tam tersi calısır. ilk c nesnesi sonra sırasıyla b ve a
```

--> ilk hayata gelen nesne, hayata en son vede eder.

```
void func()
{
        std::cout<<"func cagırıldı\n";
        Myclass x; //otomatik ömürlü
        std::cout<<"func calısıyor\n";
}
int main()
{
        func();
}
//  output
        func cagırıldı
        0x7fff09c4ac07 adresinde bir Myclass nesenesi hayata geldi
        func calısıyor
        0x7fff09c4ac07 adresinde nesne hayata veda ediyor
```

```
class Myclass {
public:
  Myclass();
  ~Myclass();
};

Myclass::Myclass()
{
  std::cout<< this <<" adresinde bir Myclass nesenesi hayata geldi\n";
}
Myclass::~Myclass()
{
  std::cout<< this <<" adresinde nesne hayata veda ediyor\n";
```

```
}

void func()
{
        static Myclass m;

}
int main()
{
  std::cout<< "main basladı\n";
  func();
  func();
  func();
  func();
  func();
  std::cout<< "main sona erdi\n";
}
//  output
        main basladı
        0x5562e17cd159 adresinde bir Myclass nesenesi hayata geldi
        main sona erdi
        0x5562e17cd159 adresinde nesne hayata veda ediyor


constructor initializer list/member initializer list/MIL
-> default constructor

class Myclass {
public:
  Myclass();
  void print()const;
private:
  int x, y;
};

Myclass::Myclass() : x(0), y(3) //constructor initializer list
// veya y{3} x{}
{
  std::cout<< "Myclass default ctor\n";
}

void Myclass::print()const
{
        std::cout<< "x = "<<x<<"\ny = "<<y;
}
int main()
{
```

```
  Myclass x;
  x.print();
}
        //output
        Myclass default ctor
        x = 0
        y = 3
```

```
Myclass::Myclass : x{x}, y{y}
        -> : sonraki ilk x ve ilk y class scope da aranır
```

```
class A {
public:
        A(int);
private:
        int mx, my;
};

A::A(int a) : my{a}, mx{ my/5 }
```

```
=> Bildirimdeki sraya göre nesneler hayata gelir.
        yani ilk mx dah sonra my. Bu sebeple my cop
        degerde oldugundan tanımsız davranış oluşur.
```

```
class B
{
public:
        B(int);
};

B::B(int x)
{
        std::cout<<x;
}

int main()
{
        B a(12); //direct init
        B b{17} //direct list init
        // daraltıcı dönüşüm error
```

```
        B c = 23; //copy init

}
```

ODR -> One definition Rule
- varlıkların bildirimi birden çok olabilir,
- tanımı tek olmalıdır

aşağıdaki varlılardan fazla kaynak dosyada tanımlanması durumunda
- eger tüm tanımlar token-by-token aynı ise ODR ihlal edilmiş olmaz
- a) Class defination
- b) Inline functions
- c) Inline variables (c++17)

--- inline ---
- baslık dosyasından tanımlanan
- ODR ihlal etmeyen
- bildirim veya tanımda inline yeterli
- sınıf tanımı içinde tanımlanırsa inline anatar

kelimesine gerek yok

```
class A {
public:
void set() //inline
{
        ...
}


};
```

delete function

```
void func() = delete;
//bildirimi yapılmış kabul edilir

class A {
public:
        A()  = default;
        void func(int) = delete;
};
```

--- type cast operators ---

type conversion // type-cast ile aynı degil
	implicit (örütlü-otomatik-kapalı) type conversion //derleyici tarafından yapılan dönüşüm
	char c1 = 10; char c2 = 15;
	c1 + c2; //implicit type
	explicit type conversion

type cast
	- static_cast
	- const_cast
	- reinterpret_cast
	- dynamic_cast

```
static_cast<int>(deval)

const int x = 10;
char *p = reinterpret_cast<char *>(const_cast<int *>(&x));

reinterpret_cast<char *>(myptr)
(char *)mptr
```

```
enum Color {White, Red, Green};
int main()
{
	Color mycolor{Red};
	int ival = mycolor//gecerli
	mycolor = static_cast<Color>(ival);
}
```

```
enum class Color {White, Red, Green};
int main()
{
	Color mycolor{class::Red};
	int ival = mycolor// gecersiz
	int ival = static_cast<int>(mycolor);
}
```

-> void* türünde hem static_cast hem de reinterpret_cast kullanılabilir

```
//C nin derleyicisine göre derler
extern "C" void f1(int);
extern "C" void f2(int);
extern "C" void f3(int);

//veya

extern "C"{
extern "C" void f1(int);
extern "C" void f2(int);
extern "C" void f3(int);
}
```

.h

```
#ifdef __cplusplus
extern "C" {
#endif
	int a(int);
	int b(int);
	int c(double,int);
#ifdef __cplusplus
}
#endif

#endif
```

```
class Myclass {
//Myclass(){}
private:
	int a;
	double dval;
}
int main()
{
	Myclass x; //default ctor
	Myclass y{}; //default ctor -zero init
	//eger kurucu fonk. yazılmamışsa y zero init
	//eğer kurucu varsa iki degerde de zero init olmaz
}
```

Eger bir sınıf nesnesi degerini (kopyalama yoluyla) aynı türünden
baska sınıf nesnesinden alarak hayata gelecek ise sınıfın
"copy ctor" denilen çzel ibr kurucu islevi cagırılır.

```
//T sınıf
T x;
T y = x;
//T y(x);
//T y{x};
//auto y = z;
//auto y {z};
//auto y (z);

void func(T x);
T y;
func(y);
```

Derleyicinin yazdıgı copy ctor (cc)

- sınıfın public
- non-static
- inline

```
class Myclass {
public:
    Myclass(const Myclass &other) : ax(other.ax), bx(other.bx), cx(other.cx)
    //copy ctor
private:
    A ax;
    B bx;
    C cx;
    //...
}
Myclass x = y;
```

```
class Myclass {
public:
  Myclass(int val = 0) : mx{val}
  {
```

```
        std::cout << "myclass default ctor this = "<<this<<"\n";
  }
  Myclass(const Myclass &other) : mx(other.mx)
  {
        std::cout << "myclass copy ctor this = "<<this<<" &other : "<<&other<<"\n";
  }
  ~Myclass()
  {
        std::cout << "myclass destructor this = "<<this<<"\n";

  }
private:
  int mx;
};

int main()
{
  Myclass a(10);
  Myclass b = a;

  std::cout << "&a = "<< &a <<"\n";
  std::cout << "&b = "<< &b <<"\n";
}
        //output
myclass default ctor this = 0x7ffcc2340bf0
myclass copy ctor this = 0x7ffcc2340bf4 &other : 0x7ffcc2340bf0
&a = 0x7ffcc2340bf0
&b = 0x7ffcc2340bf4
myclass destructor this = 0x7ffcc2340bf4
myclass destructor this = 0x7ffcc2340bf0
```

- copy assignment function -

```
        Myclass a, b;
        a = b;
        //a.operator=(b);
```

Derleyicinin yazdıgı copy assignment function

- sınıfın public
- non-static
- inline

```
class Myclass {
public:
```

```
	Myclass(const Myclass &other) : ax(other.ax), bx(other.bx), cx(other.cx)//copy ctor
	Myclass& operator = (const Myclass &other)
	{
	ax = other.ax;
	bx = other.bx;
	cx = other.cx

	return *this;
	}
private:
	A ax;
	B bx;
	C cx;
	//...
}


class A {
//..
public:
	void func();
};
int main()
{
	A x, y;
	(x=y).func();
	x.operator=(y).func();
}//geri dönüs degeri kendisine atama yapılan nesne




class A {
public:
	A& operator = (const A &other)
	{
	ax = other.ax;
	bx = other.bx;

	return *this;
	}
private:
	X ax;
	Y bx;
}
m1 = m2;
m1.operator=(m2); //aynı
```

```
Sentence s{"aaaaa"};
Sentence z{"bbbbb"};
z = s;
z.print(); //(x = s).print(); //iki ifadenin aynısı


z = s;
x.operator=(s); // yukardaki ifadenin aynısı
```

-- move semantics --

move ctor
move assignment

```
Class Myclass{};

void func(const Myclass &) //const sol taraf referansı
void func(myclass &&r) //sağ taraf referansı
	func(L val); // L value
	func(R val); // ikisi de vaible ama ikinci func cagırılır


class Myclass {
public:
	Myclass(const Myclass &other); //copy ctor
	Myclass( Myclass &&other); // move ctor
}


class A{
public:
	A(); //default ctor
	~A(); //destructor
	A(const A&); //copy ctor
	A(A &&); //move ctor
	A& operator=(const A&); //copy assignment
	A& operator=(A&&); //move assignment
};
```

move();
	- geri dönüs degeri R value
	- A ax; // class
	func(std::move(ax)); // ax -> r value expression olur

Myclass a(x); //copy ctor
Myclass a(move(x)); //move ctor

x = move(y); move assignment, R value
x = y; // copy assignment

class Myclass {};

void func(Myclass && x)
{
	x // value category L value
	// data category Myclass &&

	std::move(x) // R value expression
}

// bir fonksiyonun parametresi neden sağ taraf referansı olur?
	- taşımak için

void func (const Myclass &r)
{
	Myclass x(r); //copy ctor
}

void func (Myclass &&r)
{
	Myclass x(std::move(r)); //move ctor
}

//derleyicinin yazdıgı
// Rulo of Zero

```
Myclass(Myclass &&other) : ax(std::move(other.ax)), ....
{

}
Myclass& operator=(Myclass && other)
{
        ax = std::move(other.ax);
        bx = std::move(other.bx);
}
```

------ special member can be:

- not declared (bildirilmemis)

```
        class A{
        public:
        A{int};
        };
```

- implicitly declared
    - defaulted // derleyici tarafından yazılması
    - deleted

```
        class B{
        public:
        // derleyici B() = delete; yapar
        private:
        const int x ; //initial edilmeli
        };
        int main()
        {
        B bx; // deleted hatası verir
        }
        ////////////////////////////////////////////

        class A {
        public:
        A(int);
        };
        class B{
        A ax;
        };
        int main()
        {
        b x; // hata
```

```
        //A nın default ctoru olmadıgından derleyici deleted hatası verir
        }

        ////////////////////////////////////////////////

        PS: private tanımlanan ctor başka sınıfta cagırıldıgında deleted hatası verir

        class A {
        A();
        };
        class B{
        A ax;
        };
        int main()
        {
        b x; // hata
        }



        -user declared
        user declared (defined)
        defaulted
        deleted

        Myclass();
        Myclass() = default; //derleyici yazacak
        Myclass() = delete;
-------




        --- copy elision ---

 - temporary object (gecici nesne)

void func(const Myclass &);
int main ()
{
        Myclass m{12};
        func(m); // kullanılmasa bile hayatı devam eder
        // veya
        func(Myclass{ 12 }) // temporary obj.
        // ömrü ifadenin yürütülmesi süresince
}
```

```
class A {
public:
  A()
  {
          std::cout <<"A()\n";
  }

  ~A()
  {
          std::cout <<"~A()\n";
  }

  A(int x)
  {
          std::cout <<"A(int x) x = "<<x<<"\n";
  }

  A(const A &)
  {
          std::cout <<"copy ctor\n";
  }

};
int main()
{
  A m{12};
  std::cout<<"devam...\n";
}
//output
// A(int x) x = 12
// devam...
// ~A()

/////// farklı main()
int main()
{
  A {12};
  std::cout<<"devam...\n";
}
//output
// A(int x) x = 12
// ~A()
// devam...
```

//life extension

- eger gecici nesne, bir sag taraf ifadesi referansa baglanırsa o ifadenin yürütülmesi bitse bile ömrü devam eder
- const Myclass &r = Myclass{}; // l
- Myclass &&r = Myclass{}; // r val.
- // gecici nesnenin bağlandıgı "r" referansının scope bittiğinde ömrü sona erer

```
class A {
public:
  A()
  {
        std::cout <<"A()\n";
  }

  ~A()
  {
        std::cout <<"~A()\n";
  }

  A(int x)
  {
        std::cout <<"A(int x) x = "<<x<<"\n";
  }

  A(const A &)
  {
        std::cout <<"copy ctor\n";
  }

};
void func(A x)
{
  std::cout <<"func cagırıldı\n";
}
int main()
{
  A ax; //default ctor
  func(ax); //copy ctor
}
  //output
```

```
// A()
// copy ctor
// func cagırıldı
// ~A()
// ~A()

//////////////main()
int main()
{
  func(A()); // temporary obj.
}
        //output
//A()
//func cagırıldı
//~A()
```

RVO (return value optimization)

```
Myclass fo()
{
        return Myclass{};// temporary object
        //fonksiyonnun geri dönüş degerinintutulacağı
        bellek alanını bildiğinden, dogrudan nesneyi
        geri dönüs degerinin belek blogunda olusturur
        Dolayısıyla kopylama(copy ctor) cagırılmadı
}
int main()
{
        Myclass x = foo();
//move ctor olsa dahi cagırılmıyacaktı
}
//output
default ctor
~Myclass()

// bu iki durum mandatory copy elision
```

NRVO (Named return value optimization)

```
Myclass func()
{
        std::cout <<"func cagırıldı\n";
        Myclass mx;
        std::cout <<"func devam ediyor\n";
```

```
        return mx;
}
int main()
{
        std::cout<<"main basladı"
        Myclass x = func();
}
//
main basladı
func cagırıldı
default ctor
func devam ediyor
~Myclass()
//derleyici fonksiyonun geri dönüs degernin yazıldıgı
        yerde olusturur




        -- conversion constructor --

 class Myclass {
public:
        Myclass();
        Myclass(int);
 };
int main()
{
        int ival = 10;
        m = ival; // m = Myclass{ ival };

// derleyici sınıfın int parametreli ctor kullanılarak
//      önce geçici nesne olusturuyor
//      sınıfın kopyalayan o. işlevi
}




class A {
public:
  A()
  {
        std::cout <<"default ctor this: "<<this<<"\n";
  }
```

```
  ~A()
  {
        std::cout <<"default destructor this: "<<this<<"\n";
  }

  A(int x)
  {
        std::cout <<"A(int x) x = "<<x<<" this: "<<this<<"\n";
  }

  A(const A &)
  {
        std::cout <<"copy ctor\n";
  }

  A &operator=(const A& other)
  {
        std::cout<<"copy assignment this: "<<this<<" &other"<<&other<<"\n";
        return *this;
  }
};

int main()
{
  int ival = 10;
  A m;
  m = ival;
  std::cout<<"main devam ediyor...\n";
}
        //output
default ctor this: 0x7fffa8780682
A(int x) x = 10 this: 0x7fffa8780683
copy assignment this: 0x7fffa8780682 &other0x7fffa8780683
default destructor this: 0x7fffa8780683
main devam ediyor...
default destructor this: 0x7fffa8780682


void func(Myclass x)
//void func(Myclass &)
{

}
int main()
{
        func(12);
} // int parametleri kurucu sınıf türüne dönüştürüyor
```

// örtülü (implicit)

```
double dval = 24.4;
Myclass mx;
mx = dval; // gecerli
```

--> önce standart conversion ile double -> int donusumu
daha sonra user define conversion ile int -> Myclass
turune sınıfın conversion ctoru ile donusum gerceklesir

user defined conversion
- conversion ctor
- type-cast operaor function

```
void func(Myclass);

Myclass foo()
{
	int ival{125};
	return ival;
}
int main()
{
	int ival{};
	double dval{};

	func(ival);
	func(dval);
}
```

```
int main()
{
	string str;
	str = "ali topu tut";
	// const dizi, array decay ile const char * türüne donusur
	sınıfın const char * parametreli conversion ctoru cagırıldı
}
```

-- explicit constructor --

```
class A{
public:
	explicit A(int); //explicit ctor
	//inline yazılırsa explicit anahtar sozcugu yazılmalı
	// sadece bildirimde kullanılır
}


A::A(int x); //explicit anahatar kelimesi kullanılamaz
```

---> explicit ctor: Sadece tur donusum operatoru acık olarak kullanılırsa tur dönusumu yapar. implicit donusum yapmaz

```
class A{
public:
	A()=default;
	explicit A(int x);
};
int main()
{
	A a;
	int ival{100};
	a = ival; // syntax error
	nec = static_cast<A>(ival); // gecerli
}
```

explicit: hatalı donusumlerin onune gecer

```
int x; // default initialization
int y(10); //direct initialization
int z{}; // value initialization
int t{50}; // direct list initialization
int m = 60; // copy initialization




class A {
public:
	explicit A(int);
```

```
        A(double);
};
int main()
{
        A x = 10; // gecerli
}
```

-- dynamic storage(dinamik ömürlü nesneler) --

- dinamik bellek yönetimiyle aynı şey degil

--> heap/free store: dinamik ömürlü nesnelerin yeri heap' de
ayrılır.

new expression (new operatorleri)
delete expression

new Myclass //type -> Myclass *

```
class Myclass {
public:
void foo();
void func();
};

int main()
{
  Myclass *p = new Myclass;
  // Myclass *p{new Myclass};
  // auto p = new Myclass;
  // auto p(new Myclass);

  p->foo();
  p->func();

  delete p;
}
```

```
void *operator new(size_t n);
-> operator new islevi basarısız olursa
        std::bad_alloc sınıfı turunden exception throw ediyor

// new Myclass
//static_cast<Myclass *>(operator new(sizeof(Myclass))) -> Myclass()



Myclass *p = Myclass;
        p hayata gelen dinamik ömürlü nesenin adresi



void operator delete(void *);
p->~Myclass;
operator delete(p);




class Myclass {
public:
Myclass()
{
  std::cout<< "myclass default ctor\n";
}
~Myclass()
{
  std::cout<< "destructor\n";
}
};

int main()
{
  std::cout<< "1\n";
  Myclass *p = new Myclass;
  std::cout<< "2\n";
  delete p;
  std::cout<< "3\n";
}
        // output
1
myclass default ctor
2
```

destructor
3

--> delete edilmezse destructor cagırılmaz.
	Kaynak sızıntısı olusur(resource leak)

```
#include <new>
using namespace std;

int main()
{
	size_t n;

	cout<<"kac tam sayı: ";
	cin>> n;

	int *p = static_cast<int *>(operator new(n * sizeof(int)));

	for(size_t i{}; i < n; ++i) {
	p[i] = i;
	}

	operator delete(p);
}
```

new ifadesi = operator new() => constructor
delete ifadesi = destructor => operator delete

overload operator new function

```
class Myclass {
public:
  Myclass()
  {
	std::cout<<"Myclass default ctor called this = " << this <<"\n";
  }
  ~Myclass()
  {
	std::cout<<"Myclass destructor called this = " << this <<"\n";
  }
```

```
private:
  char buffer[1024]{};
};



void *operator new(size_t n)
{
        std::cout << "operator new called n = " << n << "\n";
        void *vp = std::malloc(n);
        if(!vp) {
        throw std::bad_alloc{};
        }
        std::cout << "the adress of the allocated block = "<< vp <<"\n";

        return vp;
}

void operator delete(void *vp)
{
        if(!vp)
        return;
        std::cout << "operator delete called... vp " <<  vp << "\n";
        std::free(vp);
}

int main()
{
  std::cout<<"sizeof(Myclass) = "<<sizeof(Myclass) << "\n";

  Myclass *p = new Myclass;
  std::cout << "p = "<< p << "\n";
  delete p;
}
        //output
sizeof(Myclass) = 1024
operator new called n = 1024
the adress of the allocated block = 0x55cac1d5c2c0
Myclass default ctor called this = 0x55cac1d5c2c0
p = 0x55cac1d5c2c0
Myclass destructor called this = 0x55cac1d5c2c0
operator delete called... vp 0x55cac1d5c2c0

int main()
{
  std::string str{"bu yazıyı tutabilmek için heap de bellek edindi"};
}
        //output
```

```
operator new called n = 51
the adress of the allocated block = 0x562222ec02c0
operator delete called... vp 0x562222ec02c0
```

-> dinamik nesnelerin destructor'ı delete edilince cagırılır
-> otomatik omurlu nesnenin destructor'ı scope bitince cagırılır

```
class Myclass {
public:
  Myclass()
  {
        std::cout<<"Myclass default ctor called this = " << this <<"\n";
  }
  ~Myclass()
  {
        std::cout<<"Myclass destructor called this = " << this <<"\n";
  }
private:
  char buffer[16]{};
};
```

```
void *operator new(size_t n)
{
        std::cout << "operator new called n = " << n << "\n";
        void *vp = std::malloc(n);
        if(!vp) {
        throw std::bad_alloc{};
        }
        std::cout << "the adress of the allocated block = "<< vp <<"\n";

        return vp;
}

void operator delete(void *vp)
{
        if(!vp)
        return;
        std::cout << "operator delete called... vp " <<  vp << "\n";
        std::free(vp);
}
```

```
int main()
{
  auto p = new Myclass[4];
  std::cout << "p = " << p << "\n";
  delete[] p;
}
          //output
operator new called n = 72
the adress of the allocated block = 0x55a66a6892c0
Myclass default ctor called this = 0x55a66a6892c8
Myclass default ctor called this = 0x55a66a6892d8
Myclass default ctor called this = 0x55a66a6892e8
Myclass default ctor called this = 0x55a66a6892f8
p = 0x55a66a6892c8
Myclass destructor called this = 0x55a66a6892f8
Myclass destructor called this = 0x55a66a6892e8
Myclass destructor called this = 0x55a66a6892d8
Myclass destructor called this = 0x55a66a6892c8
operator delete called... vp 0x55a66a6892c0
```

*** static member - static member function ***

--> main fonks hayata gelmeden hayata gelir

```
.h
class Myclass{
public:
          static int x;
};

//.cpp
int Myclass::x;
```

- complete type
- incomplete type

```
class Myclass; //class declaration(forward declaration)
```

// derleyici sadece bildirimini görüyorsa incomplete

-> bir pointer degisken tanımlamak icin, pointer in gosterecegi
turun complete type olma zorunlugu yok.

-> fonksiyonların geri dönüs degeri ve parametlereleri incomplete
type olabilir

```
Myclass f1(Myclass);
Myclass *f2(Myclass);
Myclass &f3(Myclass&);

class A;
extern A ax;

class Myclass;

typedef Myclass* MPTR;
using MPTR = Myclass*;


class Myclass {
	Myclass y; // syntax error
	static Myclass x; //sınıfıların static veri elemanları incomplete type olabilir
}



class Myclass;
int main()
{
	Myclass mx; //syntax error
	Myclass *p = new Myclass; // syntax error
	//complete olmalı...
}


class Myclass;
Myclass *foo(); //gecerli
int main()
{
	Myclass *p = foo(); // gecerli
	auto x = *p; //syntax error
	Myclass::A; //syntax error
}
```

//pimpl idiom //pointer implementation

//sınıfın private bolumunu gizlemeye yonelik bir yontem

```
class A{
public:
	static int a;
	void func()
	{
	sa = 10;
	//Myclass::a = 11; // ayı
	}
}
```

-> static anatar kelimesi sadece bildirimde olucak. Tanımda syntax error

-> static member ctor ile initial edilemez

```
class Myclass {
	Myclass() : x{200} {} //syntax error
	static int x;
};
```

```
class Myclass {
	int x = 10;//gecerli
	static int y = 10; //syntax error
	const static int dx = 10; // gecerli
	const static int double a = 10.0; // syntax error
	static constexpr int l = 10; //gecerli
	static constexpr double k = 10;//gecerli
}
```

** inline variables **
ODR-> one rule defination

```
class A{
public:
	static int x = 10; //syntax error
	inline static int cx = 10; // gecerli
```

```
	static std::string str{"abc"}; // syntax error
	inline static std::string s{"abc"}; // gecerli c++17
}
```

- consteval - c++20
-yapılan cagıral her zaman sabit ifade garantisi

```
consteval int foo(int x) // compile time da deger ureten fonk.
{
	return x * x;
}

int main()
{
	foo(12); // gecerli

	int x = 20;
	foo(x) //syntax error
}
```

--> consteval ve constexpr -> implicit inline

** static member function **

- this pointer'ı yok
- global fonnksiyonlara alternatif
- .h icerisinde inline tanımlanabilir
- bildiriminde static anatar kelimesi olur .cpp de olmaz
- bir sınıf nesnesi icin cagırlamaz
- sınıfın non-static elemanları, static member funclarda kullanılamaz

```
class A {
	void foo(int); // non-static member func
	static void func(int);
}
int main()
{
	Myclass::func(12);
```

```
}


class Myclass {
public:
	static void foo();
	void func();
	static void f1()const; //syntax error
private:
	int a, b;
};

void Myclass::foo()
{
	a = 10; //syntax error
	func(); //syntax error

	Myclass x;
	x.a = 10;
	x.func(); // gecerli
}
...
void Myclass::fo(Myclass x)
{
	x.a = 10;
	x.func(); // gecerli
}




class A {
public:
  static void foo();
};
void A::foo()
{
  std::cout <<"hello\n";
}
int main()
{
  A m;
  A *p{&m};
  A::foo(); // gecerli
  m.foo(); // gecerli
  p->foo(); // gecerli
}
```

```
class Person {
public:
  Person()
  {
        ++count;
        std::cout<<count<<"\n";
  }
private:
  inline static size_t count{};
};

int main()
{
  Person p1;
  Person p2;
  Person p3;
}
        //output
//1
//2
//3



// .h kaynak dosyasında using bildirimleri yapilmamali









//Fighter.h
class Fighter {
public:
  Fighter(const std::string &name) : m_name{name}
  {
        m_svec.push_back(this);//adresi vektore sonda ekler
```

```
  }
  ~Fighter();
  //...
  void ask_help();
  std::string Name()const
  {
          return m_name;
  }
private:
  std::string m_name;
  inline static std::vector<Fighter *>m_svec;
};


//fighter.cpp
#include <algorithm>

Fighter::~Fighter()
{
  auto iter = std::find(m_svec.begin(), m_svec.end(), this);
  //auto -> std::vector<Fighter *>::iterator iter =

  if(iter != m_svec.end()) {
          m_svec.erase(iter);
  }
}

void Fighter::ask_help()
{
  std::cout << "ben savasci "<< m_name << "yardım edin\n";
  for(auto p : m_svec) { //p container da tutulan pointerlar
  if( p!= this)
          std::cout<< p->m_name << " ";
  }
}
//
int main()
{
  Fighter f1{"Ali"};
  Fighter f2{"Veli"};
  auto pf1 = new Fighter("Ayse");
  auto pf2 = new Fighter("Fatma");
  Fighter f3{"Metin"};

  delete pf1;

  pf2->ask_help();
  delete pf2;
```

```
}
        //output
ben savasci Fatma yardım edin Ali Veli Metin
```

** named constructor idiom **
//isimledirilmiş kurucu islev

```
class Myclass {
public:
  static Myclass *creatObect()
  { // bu fonksiyon olmasa syntax error
        // ctor private
        return new Myclass;
  }
private:
  Myclass();

};
```

//ayni sayida ve türde parametrelere sahip
farklı amaclı ctor syntax error

```
class Complex {
public:
  static Complex createPolar(double angle, double distance)
  {
        return Complex{angle, distance};
  }

  static Complex createCatesian(double r, double i)
  {
        return Complex{r, i,0};
  }
```

```
private:
Complex(double r, double i, int); //cartesian
Complex(double angle, double distance); //polar
//cartesian ctordaki dummy int parammetre olmasa syntax error

};
int main()
{
  auto c1 = Complex::createPolar(0.101, 0.74734);
  auto c2 = Complex::createCatesian(2.3, -1.2);
}
```

```
//singleton -> anti pattern
        -bir sınıf turunden tek bir nesne olusturmak
        database vs..
        -global access

design pattern

pattern -> dilden bagımsız
idiom -> dile bagımlı

gof patterns - gang of four
```

```
class Singleton {
public:

  static Singleton &get_instance()
  {
        if(!mp) { //ilk cagırıldıgında nullptr
        mp = new Singleton;
        }// sadece tek nesne olusturulabilir
        return *mp;
  }


  Singleton(const Singleton &) = delete; // nesne kopyalanamaz
  Singleton &operator = (const Singleton &) = delete;
```

```
  void func();
  void foo();
private:
  Singleton(); // nesne olusturulamaz
  inline static Singleton *mp{}; //nullptr

};
int main()
{
  Singleton::get_instance().foo();
  Singleton::get_instance().func();

  auto &rs = Singleton::get_instance();
  rs.func();
}
```

```
class Singleton { //Scott Meyers Singleton
public:

  static Singleton &get_instance()
  {
        static Singleton instance;

        return instance;
  }


  Singleton(const Singleton &) = delete; // nesne kopyalanamaz
  Singleton &operator = (const Singleton &) = delete;
  void func();
  void foo();
private:
  Singleton(); // nesne olusturulamaz


};
```

--> delete kullanıldıgında ilgili pointerın baglı oldugu nesnenin
once destructor cagırılır. sonra operator delete cagırlır.

```
class Myclass { //Scott Meyers
public:

static int foo()
{
  return 1;
}
static int x;

private:
};

int foo()
{
  return 2;
}

//name lookup
int Myclass::x = foo(); // ::foo() olsaydı output 2 olurdu

int main()
{
  std::cout << Myclass::x << "\n"; // output = 1
}
```

** friend bildirimleri **

- bir global fonksiyon (namespace) icin yapilabilir
- bir baska sinifin uye fonksiyonu için yapilabilir
- bir sınıf icin yapilabilir
    friend bildirimi yapilamaz

```
class Myclass {
public:
  friend int gfunc();

private:
  int mx;
  void foo();
};

int gfunc()
{
  Myclass m;
  m.mx = 10;
  m.foo();
}
```

- incomplete type olan sınıfların uye fonksiyonlarına

```
class Data {
public:
  int func(int);
};

class Myclass {
public:
  friend int Data::func(int);

private:
  int mx;
};

int Data::func(int x)
{
  Myclass m;
  m.mx = 10;
}
```

- friend bildiriminin public veya private bolumda yapilmasının
  farkı yok

```
class Data {
public:
  Data(); //def ctor
};

class Myclass {
public:
  friend Data::Data();
};
```

```
class Myclass {
public:
  friend class Data;
private:
  int mx;
};

class Data {
public:
  void func(Myclass m)
  {
          m.mx = 10;
  }
};
```

```
class A{
          friend class B;
};
class B {

};
```

-> B, A' nın elamanlarınaa erisenbilir. A, B' nin elamanlarına erisemez

***** operator overloading *****

-> bir sınıf nesnesi bir operatorun operandı yapildiğinda
derleyici ifadeyi fonksiyon cagrisina donusturur

-> static uye fonksiyon olamaz

-> compile-time etkiler

-> operandlardan en az biri sınıf turunden class veya enum type olmalı

-> bazı operatorler overload edilemiyor
:: scope resolation
. member selection, dot operator
?: ternay operator
sizeof
.* member pointer
typeid

-> global olmayan operator
operator[]
operator()
operator->
type-cast operator function
operator=

-> isimleriyle cagirilabilir
operator<(x, y)
x.operator+(y)

```
class A {

};
int main()
{
A ax, bx;
ax = bx; //ax.operator?(bx)
}
```

-> (biri haric) operator fonksiyonlari default argumant alamaz

- sadece fonksiyon cagrı operatoru alabilir

-> operator fonksiyonlarda operatorlerin (arity) degistirilemez
unary operatorler // tek operand !x
binary operatorler //a > b

	global
!X	operator!(x)
a + b	operator+(a, b)
a > b	operator>(a, b)

	member
!x	x.operator!()
a > b	a.operator>(b) //*this hehr zaman sol operand olur

```
class A {
public:
        A operator+(A); /toplama operatoru
        A operator+(); //gecerli //sign operator
}
```

a + b
+b sign operator (isaret opr)

a - b
-a

a * b
*ptr

x & y //bitwise and
&x

++mydate; //prefix
x++ //postfix

-> operatorlerin oncelik seviyesini (priority - precedence)
ve ocelik yonunu (associativity) degistirilemez!

---
atama operatoru disinda tum op. fon. global
x = !a * b +c > d;

x.operator=(operator>(operator+(operator*operator!(z), b), c), d)

---

member operator function ise;
x = !a * b +c > d;
x.operator=(a.operator!().operator*(b).operator+(c).operator>(d))

Myclass &operator
Myclass operator
a = b

-> bir fonksyion cagrısına karsılık geldigine
gore bu ifadeninL value olmasi icin & dondurmesi gerekir
++x
--x

-> +x ve -x gibi R value degerlerde ise geri donus degeri
ref & olmamali

A operator+(const A&, const A&)//yan etki yoksa const &

```
class A {
public:
        bool operator<(const A &)const;
        //sondaki const sol opr. degismeyecegini gosterir


        A &operator+=(const A&);
        // += l value oldugundan & donmeli
        // opr sag tarafı degismeyeceginden const & parametleri olmali

};
```

** const overload

```
class Vector {
public:
        int &operator[](size_t);
        const int &operator[](size_t)const; // const obj
};
```

simetrik binary op. global tercih edilir
- a > b
- b > a

-> eger .h dosyasında sadece bildirim yapilicaksa
#include <iosfwd> eklenebilir. Hafif kutuphanedir

```
int ival{12};
cout << ival;
cout.operator<<(ival); //aynı anlam
```

```
cout << "bilgisayar"; //const char * parametreli fonksiyon
operator<<(cout, "bilgisayar"); // aynı, global func

cout.operator<<("bilgisayar"); // bilgisayar yazmaz aynı degil,
// member function
// adresi yazdirir
cout.operator<<('A'); // output: 65
operator<<(cout,'A'); // output: A
```

```
ostream &operator<<(int); //int yazdiran
```

```
ostream &operator<<(double); //double yazdiran
ostream &operator<<(void *); //adres yazdiran
```

```
//inserter - formalı giriş
std::ostream& operator<<(std::ostream &os, const Mint &m)
{
	return os<<"(" << m.mval << ")";
}
```

extractor // formatlı cıkıs

```
// .h
class Mint {
public:
	explicit Mint(int x = 0) : mval{x} {}
	//inserter - formalı giriş
	friend std::ostream &operator<<(std::ostream&, const Mint&);

	//extractor  formatlı cıkıs
	friend std::istream &operator>>(std::istream&, Mint&);

private:
	int mval;
};
```

```
//.cpp
std::ostream& operator<<(std::ostream &os, const Mint &m)
{
	return os<<"(" << m.mval << ")";
}

std::istream &operator>>(std::istream& is, Mint& m)
{
	return is >> m.mval;
}
```

```
int main()
{
  Mint a, b, c;
  cout <<"uc tam sayi girin: ";
  cin >> a >> b >> c;
  cout << a << " " << b << " " << c <<endl;
}
//output
uc tam sayi girin: 1 5 7
(1) (5) (7)
```

equality
a == b

equivalance
!(a < b) && !(b < a)

```
// .h

class Mint
{
public:
        explicit Mint(int x = 0) : mval{x} {}
        //inserter - formalı giriş
        friend std::ostream &operator<<(std::ostream &, const Mint &);

        //extractor  formatlı cıkıs
        friend std::istream &operator>>(std::istream &, Mint &);

        friend bool operator<(const Mint &x, const Mint &y)
        {
        return x.mval < y.mval;
        }

        friend bool operator==(const Mint &x, const Mint &y)
        {
        return x.mval == y.mval;
        }
```

```
        Mint &operator+=(const Mint &r)
        {
        mval += r.mval;
        return *this;
        }

        Mint &operator-=(const Mint &r)
        {
        mval -= r.mval;
        return *this;
        }

        Mint &operator*=(const Mint &r)
        {
        mval *= r.mval;
        return *this;
        }

        Mint &operator/=(const Mint &r)
        {
        if (r.mval == 0)
        throw std::runtime_error("sifira bolunma hatasi...\n");

        mval /= r.mval;
        return *this;
        }

private:
        int mval;
};

// friend fonksiyonlarin sayisi kisitlanir
// sinifin priviate kısmında yapılan degisikliklerde sadece 2 funksiyon degistirilir
inline bool operator>=(const Mint &x, const Mint &y)
{
        return !(x < y);
        // return !operator<(x, y);
}

inline bool operator>(const Mint &x, const Mint &y)
{
        return (y < x);
        // return operator<(y, x);
}

inline bool operator<=(const Mint &x, const Mint &y)
{
```

```
        return !(y < x);
        // return !operator<(y, x);
}

inline bool operator!=(const Mint &x, const Mint &y)
{
        return !(y == x);
        // return !operator=(x, y);
}

inline Mint operator+(const Mint &left, const Mint &right)
{
        // Mint temp{left};
        // temp += right;
        // return temp;
        //VEYA
        return Mint{left} += right;
        //VEYA
        // inline Mint operator+(Mint left, const Mint &right) {
        // return left += right; }
}

inline Mint operator-(Mint left, const Mint &right)
{
        return left -= right;
}

inline Mint operator*(Mint left, const Mint &right)
{
        return left *= right;
}

inline Mint operator/(Mint left, const Mint &right)
{
        return left /= right;
}
```

-> + overload ediliyorsa += ' da overload edilmeli
-> += overload edilirken sınıfta, + ise simetrik oldugundan global yaparak += cagırılması

```
class A{
public:
	A &operator++(); // prefix
	A operator++(int); //posfix
}
// ++x L value,  y++ PR value
```

```
class Array {
public:
  explicit Array(size_t n) : msize{n}, mp{new int[msize]} {}
  ~Array()
  {
	delete[]mp;
  }

  size_t size()const {return msize;}
  int& operator[](size_t idx)
  {
	return mp[idx];
  }
  //copy ctor ve move yazilmali

  const int& operator[](size_t idx)const
  {
	return mp[idx];
  }

  friend std::ostream& operator<<(std::ostream& os, const Array& a)
  {
	os << "(";
	for(size_t i = 0; i< a.size() - 1; ++i)
	os << a[i] << ", ";

	return os << a[a.size() - 1] << ")";
  }

private:
  size_t msize;
  int *mp;
```

```
};


int main()
{
  Array a(10);
  std::cout<<a.size()<<"\n";
  for(int i = 0; i < a.size(); ++i)
          a[i] = 3*i;
  std::cout << a;
}
//10
//(0, 3, 6, 9, 12, 15, 18, 21, 24, 27)



SmartPtr;
p->x;
p.operator->()->x









----Smart Ptr-----

class ResourceUser {
public:
  ResourceUser()
  {
          std::cout <<"ResourceUser ctor kaynaklar edindi...\n";
  }

  ~ResourceUser()
  {
          std::cout <<"ResourceUser ctor kaynaklar geri verildi...\n";

  }

  void func()
  {
          std::cout <<"ResourceUser::func()\n";
```

```
  }

  void foo()
  {
        std::cout <<"ResourceUser::foo()\n";
  }

private:
};

class SmartPtr {
public:
  SmartPtr(ResourceUser *p) : mp{p} {}

  ~SmartPtr()
  {
        delete mp;
  }

  ResourceUser& operator*()
  {
        return *mp;
  }

  ResourceUser* operator->()
  {
        return mp;
  }
private:
  ResourceUser* mp;
};


void gf(ResourceUser &r)
{
}
int main()
{
  {
        SmartPtr p = new ResourceUser;
        p->foo();
        //p.operator->()->foo();
        p->func();

        gf(*p); //gf(p.operator*());
        (*p).foo();
  }
  std::cout<<"main devam ediyor...\n";
```

```
}
//output
ResourceUser ctor kaynaklar edindi...
ResourceUser::foo()
ResourceUser::func()
ResourceUser::foo()
ResourceUser ctor kaynaklar geri verildi...
main devam ediyor...




//unique ptr

class SmartPtr {
public:
  SmartPtr() : mp{nullptr} {}

  SmartPtr(const SmartPtr&) = delete; //copy ctor
  SmartPtr& operator=(const SmartPtr&) = delete; //copy assigment

  SmartPtr(SmartPtr&& other) : mp{other.mp}
  {
        other.mp = nullptr;
  }

  ~SmartPtr()
  {
        if(mp)
        delete mp;
  }

  ResourceUser& operator*()
  {
        return *mp;
  }

  ResourceUser* operator->()
  {
        return mp;
  }
private:
  ResourceUser* mp;
};
```

--- () overload ---

```
class Functor {
public:
  void operator()(int x = 0)
  {
          std::cout << "Functor::operator()() this = " << this <<"x = "<< x<< "\n";
  }
  void s()
  {
          std::cout << "Functor::s()() this = " << this << "\n";

  }
};

int main()
{
  Functor f;
  std::cout << "&f = " << &f <<"\n";
  f(12);
  f();
}
//output
&f = 0x7fff76679577
Functor::operator()() this = 0x7fff76679577x = 12
Functor::operator()() this = 0x7fff76679577x = 0
```

----- user define conversion ------

```
class Myclass {
public:
  operator int()const; //geri dönus turu int
};
```

```
int main()
{
  Myclass m;
  int ival = m; // m.operator int();
}
```

```
int square(int a)
{
  return a * a;
}

int main()
{
  Counter c{4};

  for(int i = 0; i < 10; ++i)
          ++c;

  cout<< c << "\n";
  int x = c; // gecerli
  //int x = c.operator int();
  //explicit olursa gecersiz. tur donusumu gerekli
  //int ival = static_cast <int> (c);

  cout<< square(c); // square(static_cast <int> (c))

}
```

conversion sequence
// derleyici yapar
std conv + user difened conv
user difened conv + std conversion

//derleyici donusum yapmaz
user dined conv + user difened conv

```
bool b = c; // Counter c;
```
counter' dan int e user difened, int -> bool std conversion

```
long double f = c;
```

*** operator bool ***

-> smartptr sınıfını lojik yorumlamaya almak icin

```
SmartPtr p = new ResourceUser;
if(p) {} //syntax error
```

```
explicit operator bool()const
{
return mval == 0;
}
```

```
SmartPtr p = new ResourceUser;
if(p) {} //gecerli
```

*** referance qualifier ***

```
class Myclass {
public:
	void func() &
	{
	std::cout << "Myclass::func() &\n";
	}

	void func() const&
	{
	std::cout << "Myclass::func() const&\n";
	}
```

```
        void func() &&
        {
        std::cout << "Myclass::func() &&\n";
        }

        void func() const&&
        {
        std::cout << "Myclass::func() const&&\n";
        }
};

int main()
{
        Myclass{}.func(); // Myclass::func() &&

        Myclass m;
        m.func(); // Myclass::func() &

        const Myclass z;
        z.func(); // Myclass::func() const&

        std::move(m).func(); // Myclass::func() &&

        std::move(z).func(); // Myclass::func() const&&
}
```

* enum class *

```
enum class Weekday {
  Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday,
};

Weekday& operator++(Weekday& wd)
{
  return wd = wd == Weekday::Saturday ? Weekday::Sunday :
static_cast<Weekday>(static_cast<int>(wd) + 1);

}

Weekday operator++(Weekday& wd, int)
```

```
{
  auto temp{wd};
  operator++(wd);
  return temp;
}

ostream& operator<<(ostream& os, const Weekday& wd)
{
  static const char* const pdays[]{
  "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday",
  };

  return os << pdays[static_cast<int>(wd)];
}

int main()
{
        Weekday wd{ Weekday::Monday };
        ++wd;
        cout << wd;
}
```

*** delegating ctor(c++11) ***

```
class Myclass {
public:
  Myclass(int a, int b, int c) : mx(a), my(b), mz(c) {}
  Myclass(int a) : Myclass(a, a, a) {}
  Myclass(int a, int b) : Myclass(a, b, 0) {}
  Myclass(const char *p) : Myclass(std::atoi(p), 0, 0) {}

private:
  int mx, my, mz;
};
```

*** if with initializer (c++17)***

-> if(T x = init_expr; x > 10)

```
if(int x = foo(); x > 10) { //auto x = foo();
        //
}
else {
        //
}
```

```
int main()
{
  string str;
  cout<<"yazi girin";
  getline(cin, str);

  auto idx = str.find('a'); // scopu gereksiz genisletir
  if(idx != string::npos) {
        //
        idx;
  }
}
-------
int main()
{
  string str;
  cout<<"yazi girin";
  getline(cin, str);


  if(auto idx = str.find('a'); idx != string::npos) {
        cout << "bulundu index = " << idx;
        str[idx] = '!';
        cout << str <<"\n";

  }
}
----
```

```
switch(int x = foo()) {
        //
}

while(int x = foo()) {
        //
}
--------
```

```
*** struct binding c++17 ***
struct  Data
{
  int a, b;
  double c;
};

//decomposition
int main()
{
  Data mydata = {10, 20, 1.3} ;

  //structured binding
  auto &[x, y, z] = mydata; // & olmazsa deger degismez
}
.....
auto [x, y, _] = mydata; // _ kullanılmayacak
...
```

```
Data func()
{
  return {1, 3, 4.5};
}

int main()
{
  Data mydata = {10, 20, 1.3} ;
```

```
  auto [x, y, z] = func();
}
-----




struct  Person
{
  string name;
  int age;
  double wage;
};

Person get_person()
{
//
  return{"ali", 32, 55.7};
}

int main()
{
  if(auto[name, age, salary] = get_person(); age > 40) {
          //
  }
}

---

int main()
{
  vector<Person> pvec(100);
  for(auto[name, age, wage] : pvec) {
          //
  }
}

---

int main()
{
  map<string, string> mymap;
  //

  for(const auto& [name, surname] : mymap) {
          //
  }
```

```
}
```

```
int main()
{
  map<string, string> mymap;

 if(auto [iter, inserted] = mymap.insert({"sali", "veli"}); !inserted) {
   std::cout << "ekleme yapilamadi\n";
   //
 }
}
```

```
class A {
public:
  A()
  {
        std::cout<<"1";
  }
  A(const A &)
  {
        std::cout<<"2";
  }
  ~A()
  {
        std::cout<<"3";
  }
};

class B {
public:
  B(A)
  {
        std::cout<<"4";
  }

  ~B()
  {
        std::cout<<"5";
  }
```

```
};

//most vexing parse -> hem degisken tanımlama, hem işlev bildirimi ise
// işlev bildiriminin onceligi var.
int main()
{
  //most vexing parse
  B b(A()); //A() -> function pointer par.
  // akrana bir sey yazmaz.

  //most vexing parse olmaz...
   B b( (A()) );
   B b( A{} );
   B b{ A() };
   B b{ A{} };
}
----
```

*** nested types/ type members/ member type ***

```
typedef int (*Fcmp)(const void*, const void*);
using FCMP = int (*)(const void*, const void*);
```

-> template haline getirilebilir(alias template)

-> bir sınıfın tanımı icinde yapilan tur bildirimleri
sınıfın nested type bildirimlerini olusturur.

```
class A {
private:
  void foo();
  static void sad();
  static int sx;

  class Nested {
  private:
          void func()
          {
```

```
        A ax;
        ax.foo();
        sad();
        sx = 5;
        }
  };
};
```

```
class A {

  class Nested {

  };

public:
  static Nested foo();
};

int main()
{
 auto x = A::foo(); // gecerli
 A::Nested y = A::foo(); //syntax error
}
```

```
// .h
class A {

  class Nested {
  public:
        void foo();
        void foo(Nested);
        Nested foo(Nested);
  };


};

int main()
{
```

```
 auto x = A::foo(); // gecerli
 A::Nested y = A::foo(); //syntax error
}

// .cpp
void A::Nested::foo()
{
  Nested nx; //gecerli
}

void A::Nested::foo(Nested x)
{
  //
}

A::Nested A::Nested::foo(Nested x) //Nested foo(Nested)
{

}
```

```
class A {
public:
  enum {size = 100};
};

int main()
{
 int x = A::size;
}
```

```
class Myclass {
  class Nested; //incomplete type

}
----
```

--> pimpl idiom(pointer implementation):
  Sınıfın private bolumunu gizlemeye yonelik

```
class Mint;
class Date;
class Counter;

class A {
private: // syntax error
  Mint mx;
  Date dx;
  Counter cx;
};

//non-static ilgili sınıfın complete type olması gerekir
// referans veya pointersa incomplete type gecerlidir

class Mint;
class Date;
class Counter;

class A {
private: // gecerli
  Mint *mx;
  Date &dx;
  static Counter cx;
};
```

```
// pimpl
class A {
public:
  A();
  ~A();
  void func();
  void foo();
private:
  struct Pimpl;
  Pimpl* mp;
};

//.cpp
struct A::Pimpl
```

```
{
  Mint mx;
  Date dx;
  Counter cx;
};

A::A() : mp{new Pimpl}
{
  //
}

A::~A()
{
  delete mp;
}

void A::func()
{
  ++mp->cx;
  mp->dx;
}
```

*** composition ***

dependancy
association
  aggregation -> omursel birliktelik yok
        composition -> nesne ile sahip olan nesne aynı anda haya gelir ve biter

```
class A {
public:
```

```
};

class Myclass {
// myclass hayata geldiğinde ctor ile A ax de hayata gelir.
// destr calıstıgında ax nesnesi de sonlanır
private:
  A ax;
};
```

-> composition interface edinme iliskisi degil.

```
class A {
public:
  void func();
  void foo();

};

class Myclass {

private:
  A ax;
};

int main()
{
  Myclass mc;
  mc.foo()//syntax error
}
```

```
class A {
public:
  void func();
  void foo();

};

class Myclass {
public:
  void foo()
  {
```

```
        ax.foo();
  }
private:
  A ax;
};

int main()
{
  Myclass mc;
  mc.foo(); // gecerli
}
---




---

class Member {
public:
  Member()
  {
        std::cout << "member default ctor\n";
  }
  ~Member()
  {
        std::cout << "member default destructor\n";
  }

};

class Owner {
public:
  Owner()
  {
        //scope'a girdiginde elemanlar hataya gelmiş olur
        std::cout << "Owner default ctor\n";
  }
  ~Owner()
  {
        std::cout << "Owner default destructor\n";
  }
private:
  Member mx;
};

int main()
{
```

```
  Owner ow;
}

//output
member default ctor
Owner default ctor
Owner default destructor
member default destructor
---


---
class Member {
public:
  Member()
  {
        std::cout << "member default ctor\n";
  }
        Member(const Member&)
  {
        std::cout << "Member copy ctor\n";

  }

};

class Owner {
public:
  Owner()
  {
        std::cout << "Owner default ctor\n";
  }


  Owner(const Owner &other)
  {
        std::cout << "Owner copy ctor\n";
  }

private:
  Member mx;
};

int main()
{
  Owner ow;
  Owner b = ow;
}
```

```
//output
member default ctor
Owner default ctor
member default ctor
Owner copy ctor
---



---
class Member {
public:
  Member()
  {
        std::cout << "member default ctor\n";
  }
        Member(const Member&)
  {
        std::cout << "Member copy ctor\n";

  }

};

class Owner {
public:
  Owner()
  {
        std::cout << "Owner default ctor\n";
  }


  Owner(const Owner &other) : mx(other.mx)
  {
        std::cout << "Owner copy ctor\n";
  }

private:
  Member mx;
};

int main()
{
  Owner ow;
  Owner b = ow;
}
//output
member default ctor
```

Owner default ctor
Member copy ctor
Owner copy ctor

```
class Member {
public:

        Member& operator=(const Member&)
 {
        std::cout << "Member copy assignment\n";
        return *this;
 }

};

class Owner { //copy assignment default
public:
private:
  Member mx;
};

int main()
{
  Owner a, b;
  a = b;
}
//output
Member copy assignment
```

-> owner' da copy assignment yaziliyorsa atanmasından da sorumludur.

```
class Member {
public:

        Member& operator=(const Member&)
 {
        std::cout << "Member copy assignment\n";
        return *this;
 }
```

```
};

class Owner {
public:

  Owner& operator=(const Owner &other)
  {
          std::cout << "Owner copy assignment\n";
          mx = other.mx;
          return *this;
  }

private:
  Member mx;
};

int main()
{
  Owner a, b;
  a = b;
}
//output
Owner copy assignment
Member copy assignment
---




---
class Owner {
public:
  Owner(Owner &&other) : a(move(other.a)), b(move(other.b))
  {

  }
private:
A a;
B b;
};
---



---
```

```
class Owner {
public:
  Owner& operator=(Owner &&other)
  {
        a = move(other.a);
        b = move(other.b);

        return *this;
  }
private:
A a;
B b;
};
---
```

*** namespace ***

```
--
//unnamed namespace
namespace {
}
--

---
int g = 1;

namespace A {
  int g = 2;
}

int main()
```

```
{
  int g = 10;
  g = 25;
  ::g = 5;
  A::g = 7;
}
---
```

*** nested namespace

```
---
namespace Ali {
  namespace Veli {
          namespace Elif {
          int x = 5;
          }
  }
}
int main()
{
  Ali::Veli::Elif::x = 20;
}
---
```

```
---
namespace ali {
  int x = 1;
}

namespace ali {
  int y = 2;
} // derleyici 2 namespace'i birlestirir
  -> bir cok baslık dosyasında std namespace oldugundan
  ayrı ayrı alınmaz derleyici aynı namespace isimleri birlestirir.
---
```

```
---
namespace A
{
  namespace B
```

```
 {
        class Myclass
        {
        };
        void func(Myclass)
        {
        std::cout << "AL::func()";
        }
 }

}

int main()
{
  A::B::Myclass x;
  func(x);
}
---
```

*** inline namespace
 -> nested namespace olsa da bu namespace icerisindeki
  isimler dogrudan onu kapsayan namespace'lerde visable olur.

```
---
namespace A
{
  inline namespace B
  {
        int x, y;
  }

}

int main()
{
  A::x = 10;

  //inline namespace olmasaydı
  //A::B::x = 20;
}
```

```
namespace A
{
  inline namespace B
  {
          int x, y;
          class Myclass
          {
          };
  }
          void func(Myclass x);
}

int main()
{
  A::Myclass m;
  func(m); //inline olmasa syntax error
}
```

```
// kod degistirmeden aynı sınıfın farklı yazılmış versiyonu kullanılabilir
#define VERSION_2

namespace Project
{
#ifdef VERSION_1
  inline
#endif
  namespace Version1
  {
          class Myclass{
          };
  }

#ifdef VERSION_2
  inline
#endif
  namespace Version2 {
          class Myclass {
```

```
        };
  }
}
int main()
{
  Project::Myclass m; //hangi classın kullanıldıgı makroya baglı

}
---
```

```
//unnamed namespace
-> linkage asamasında farklı varlıklara ait
---
namespace {
  //internal linkage
  int x;
}
int main()
{
  x = 5;
}
---
```

```
-----
//.h
const int x = 10;

---> internal linkage: Hangi kaynak dosya include ederse,
o kaynak dosyanın x'i ayrı.

-> kaynak dosyada adresi yazırılan &x ile, mainde
adresi yazdırıan &x farklı adreslerdir.

//.h
inline const int x = 10;
farklı kaynak dosyalarındaki &x aynıdır.
-----
```

```
//.h
int foo();
const int x = foo();
//bu tanımı iceren her kod dosyasında ayrı ayrı hayata gelir.
//inline olsaydı tek degisken olurdu 1 defa ilk deger verildi.

//.cpp
int foo()
{
  std::cout << "foo()\n";
  return 1;
}

int main()
{
}
//statik oldugu için program çalışmadan hayata gelir

//output
foo()
```

```
//namespace alias

namespace ali_veli {
  int x;
}
namespace ali = ali_veli;

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

```
namespace rgc = std::regex_constants;
//std::regex_constants::egrep;
rgc::egrep;
```

****** <string> ******
  //tur eş isim bildirimi olmasaydı
  std::basic_string<char, std::char_traits<char>, std::allocator<char>> str;
        std::string str;

-> dynamic array
-> heap
-> aynı zamanda STL container (Standart Template Library)
  - container: veri yapılarını implement eden, temsil eden sınıflara verilen isim
        - baglı liste, ikili arama agacı...

* STL *
  container
  //sequence containers(containers type)
        - vector
        - deque
        - list
        - forward_list
        - array
        - string

  //associative containers
  //unordered associative containers

-> algorithm: container ustunde calıstırılacak algoritmaları implement eden
  global fonk sablonlarına denir.

  std::cout << sizeof(std::string); //32
  std::cout << sizeof(char *) << "\n"; //8

-> string'in max boyutu belliyse baştan bellekte yer ayırılmalı

** str.size() **
- generic
- const
- geri donus degeri
  string::size_type //veya
  size_t len = str.size();

** str.length() **
- const
- size_t

** str.empty() **
- bool
- str boşsa true

** str.capacity() **
- size_t
- ayrılan toplam bellek boyutu

```
int main()
{
  std::string str{"ali veli"};
  std::cout << "uzunluk = " << str.size() <<"\n";
  std::cout << "kapasite = " << str.capacity() <<"\n";
}
//output
uzunluk = 10
kapasite = 15
// yazının uzunlugu 15 i gectiginde realloaction gerceklesir.
```

```
int main()
{
  std::string str{"ali veli sedaaa"};
  std::cout << "uzunluk = " << str.size() <<"\n"; //15
  std::cout << "kapasite = " << str.capacity() <<"\n"; //15

  str.push_back('A');
  std::cout << "uzunluk = " << str.size() <<"\n"; //16
  std::cout << "kapasite = " << str.capacity() <<"\n"; //30
}
// realloaction
// eski kapasiteyi tutan pointer invalid hale gelir
```

-- constructor

```
string str; //default ctor
string str {}; //default ctor
```

cstring --> null terminated byte stream
- yazının sonunda null karakter olması programcının sorumlulugunda

---not---
```
cout << "aa" << endl;
cout << "aa" << '\n';
```
//aynı degil
endl -> standart cıkıs akımının bufferını flash eder
  endl(cout);
  - ilave maliyet

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

-> static constexpr size_type npos: string in size_type'nın
  en buyuk degeridir.

-> yazıda arama yapıldıgında bulunursa string::size_type searc
  indexi dönderir. Bulamazsa npos döndürür

-> string::npos gecerli bir index olamaz

```
int main()
{
  string str;

  std::cout << "yazı girin: ";
  getline(cin, str); // '\n'  görene kadar

  if(auto idx = str.find('k'); idx != string::npos) {
          std::cout << "bulundu" << idx << "\n";
  }
  else
          std::cout << "bulunamadı";
}
```

```
func(string &str, size_type idx, size_type n);
// verilen indexten sonrasını yazdıran
// indexden sonra 10 karakter varsa

s.func(str, idx, 50); // ub degil
// idx den sonraki tüm karakterleri(10 karakter) alır

s.func(str, idx, string::npos);
// istenilen idx den sonraki tüm stringi alır
```

```
func(const std::string& str);//ise
  // str ne ise o yazı gönderilir
```

```
func(const std::string& str, size_t idx);
//idx başlayarak

func(const std::string& str, size_t idx, size_t n);//
//idx başlayarak n tane karakter oku


func(const char* str);//ise cstring
  // yazının sonunda null karakter olmalı

func(const std::string& str, size_type n);// data parametre
  // bu adresten başlayarak n kadar karakter
  // null karakterle ilgisi yok
  // idx + n tasması undifened behavior

func(size_type n, char c); // fill parametre


-> std::initializer_list<>
- #include <initializer_list>
func(std::initializer_list<char>);
  //initializer_list in char acılımı
  {23, 6, 7} gibi listedeki degerler int ise <int> acılımı
  auto x = {1, 5, 7, 9}; // std::initializer_list<int> x;
  auto y{12}; // int y



func(char *pe, char *pe) //range parametre
  [a,b) döngüde belli zamanda a = b olmalı
          yoksa gecerli bir range degildir

int a[5] = {1, 2, 3, 4, 5};
func(a, a+5) //tüm elemanlarını iceren aralık istenirse
  a + 5 dizinin bittigi yer
  [a, b) ilk eleman dahil son eleman dahil degil

ivec.begin() -> ilk ögenin konumunu dönderir
ivec.end() -> sondan sonraki (olmayan ögenin konumu) dönderir
```

```
void func(std::initializer_list<int> x)
{
  //
}
int main()
{
  std::initializer_list<int> a{5, 19, 20};
  func({1, 2, 4, 3});
  func(a);
}
---
```

```
---
class Myclass {
public:
  Myclass(std::initializer_list<int>)
  {
          std::cout << "initializer_list\n";
  }
  Myclass(int)
  {
          std::cout << "int\n";
  }
  Myclass(int, int)
  {
          std::cout << "int int\n";
  }
};

int main()
{
  Myclass m1{12};
  Myclass m2{12, 50};
  Myclass m3(10);
  Myclass m4(5, 7);
}
//output
initializer_list
initializer_list
int
int int
---
```

```
int main()
{
  std::string str(57, 'A');
  std::cout << str << "\n"; // AAAAAA...

  std::string s{57, 'A'}; // 9A
  std::cout << s << "\n";
}
```

```
void func(std::initializer_list<int> x)
{
  for(auto iter = x.begin(); iter != x.end(); ++iter) {
          std::cout << *iter <<" ";
  }
}

int main()
{
  func({2, 6 , 1,  7, 8});
}
//output
2 6 1 7 8
```

```
void func(std::initializer_list<int> x)
{
  for(auto val : x) //range based for loop
          std::cout << val <<"\n";
}
```

```
int main()
{
  func({2, 6 , 1,  7, 8});
}
//output
2
6
1
7
8
---
```

```
---
void func(int a)
{
  std::cout << "func cagırıldı a = " << a <<"\n";
}

int main()
{
  int x{12}, y{29}, z{43};

  for(auto val : {x, y, z, 33, 44, 56}) { //range based for loop
          func(val);
  }
}
//output
func cagırıldı a = 12
func cagırıldı a = 29
func cagırıldı a = 43
func cagırıldı a = 33
func cagırıldı a = 44
func cagırıldı a = 56
---
```

-> initializer_list her zaman const referans & sematiği kullanır

---
```
class Myclass {
public:
  Myclass() = default;

  Myclass(int)
  {
          std::cout << "int\n";
  }

  Myclass(const Myclass&)
  {
          std::cout << "copy ctor\n";
  }

  void foo() {}
};

int main()
{
  Myclass a[4];

  for(auto x : a)
  // Myclass yn = a[i] gibi
  x.foo();
// range based for loopdaher turda yerel degisken olusturur,
  ilk degerini a'daki ogelerden alır
}
//output
copy ctor
copy ctor
copy ctor
copy ctor


referans olsaydı copy ctor cagırılmayacaktı
  for(auto &x : a)
          x.foo();
```

```
int main()
{
```

```
  int a[] { 1, 5, 7, 9, 10, 5, 7, 9, 8, 7, 1, 2, 4 };

  ///ilk eleman adresi, son elemandan sonraki adres)
  std::sort(a, a + sizeof(a) / sizeof(*a));
  std::sort(std::begin(a), std::end(a)); // global fonk ile
}
---




---
string str('A'); //syntax error
string str{'A'}; // gecerli
string str{"A"}; // gecerli
string str{1, 'A'}; // gecerli/fill parametre
---



---
string s{"ali"};
cout << strlen(s.c_str()) ;
//string'den const char* dönüsüm
---



---
string s{"ali"};
const char* p = s.c_str();

//yazının adresi tutulursa ve yazı
  tasınırsa, pointer danglig hale gelir
---




string str{"ali"};
// adresini veren:
  str.c_str(); //const char*
  str.data(); //char*
  &str[0]
  &*str.begin();
```

```
string str{"cumhuriyet"};
// karakterlerine ulasmak:
  - str[i];
  - str[4] = 'x'; //const degilse
  - for(auto iter = str.begin(); iter != str.end(); ++iter)
          cout << *iter;

  - for (char& c : str)
          cout << c;

  - for (auto val : con) // kopyalama ile dolasma
          // sınıf türündendense her seferinde copy  ctor cagırılır

  - for (const char& c : str)
```

```
string str{"Ali"};
str.fropnt() = 'X;
str.back() = 'W';  //XiW
```

```
-> yazı bos degilse son karaktere erişme:
  - str[str.size() - 1]
  - str.at(str.size() - 1)
  - str.back()
  - *(str.end() - 1)
  - *str.rbegin()
```

```
-> yazının sonuna ekleme
str.push_back('a');
```

** insert **

```
string str{"ali"};
  // str.insert(2, "ayse"); //alaysei
  // str.insert(0, "ayse"); //ayseali
  // str.insert(0, 5, 'A'); //AAAAAali
string name{"veli"};
  // str.insert(0, name); //veliali
  // idx gecersiz ise throw
  // str.insert(0, name, 1, 2)// 1 idx' den baslayarak 2 idx ekle
          //elali
```

-> idx parametreli index fonksiyonlarının hepsi *this
  dönderir.

```
          str.insert(0, "ali").append;
```

	Full String	Part of String	C-string (*char*∗)	*char* Array	Single *char*	*num chars*	Iterator Range	Init list
constructors	Yes	Yes	Yes	Yes	—	Yes	Yes	Yes
`=`	Yes	—	Yes	—	Yes	—	—	Yes
`assign()`	Yes	Yes	Yes	Yes	—	Yes	Yes	Yes
`+=`	Yes	—	Yes	—	Yes	—	—	Yes
`append()`	Yes	Yes	Yes	Yes	—	Yes	Yes	Yes
`push_back()`	—	—	—	—	Yes	—	—	—
`insert()` for idx	Yes	Yes	Yes	Yes	—	Yes	—	—
`insert()` for iter.	—	—	—	—	Yes	Yes	Yes	Yes
`replace()` for idx	Yes	Yes	Yes	Yes	Yes	Yes	—	—
`replace()` for iter.	Yes	—	Yes	Yes	—	Yes	Yes	Yes
find functions	Yes	—	Yes	Yes	Yes	—	—	—
`+`	Yes	—	Yes	—	Yes	—	—	—
`==, !=, <, <=, >, >=`	Yes	—	Yes	—	—	—	—	—
`compare()`	Yes	Yes	Yes	Yes	—	—	—	—

---
std::string str{"yunus"};
std::string name{"ali"};
str.append(name, 0, 1); //yunusa

---
std::string str{"yunus"};
str.assign("veli"); // veli

---
std::string str{"yunus"};
const char *p = "0123456489"; //range parametre
str.assign(p + 2, p + 7);
//23456

---
std::string str{"yunus"};
str.erase(3, 1); //3 idx'den baslayarak 1 karakter sil
//yunus

---
std::string str{"yunus"};
str.erase(2); // 2 idx' den baslayarak siler
//yu

---
std::string str{"yunus"};
str.erase(0); // tüm yazıyı siler
// str.erase(); // ilk parametre 0, ikinci npos alır hepsini siler
---

```
std::string str{"yunus"};
str.erase(str.size() - 1); // son karakteri siler
// str.pop_back(); //son karakteri siler
// str.erase(str.end() - 1); // iter ile son karakteri siler
// str.erase(str.begin() + 1, str.end() - 1); // range //ys
```
---

string silmenin yolları
- std::string str{"yunus"};
- str.clear();
- str.erase(); //str.erase(0); // str.erase(0, string::npos);
- str.erase(str.begin(), str.end()); // range ile
- str = "";
- str = {};
- str.assign("");
- str = std::string{}; //default ctor ile gecici nesne olusturur

-> arama fonksiyonlarının hepsi idx döndürür
-> aramayı bir idx ile başlatabilirler

```
str.find('a'); // a karakterini yazının basından itibaren arar
str.find('a', 5); // 5 idx' den baslayarak ara

string s{"ali"};
string x{"deliali"};
x.find(s); //x' de s stingi arar //4
```

```
int main()
{
 std::string str;
 std::cout << "yazı girin: " ;
 getline(std::cin, str);
 std::string s{"ali"};
```

```
  if(auto idx = str.find(s); idx != std::string::npos);
          std::cout << "bulundu idx: " << str.find(s)<<"\n";

  // if(str.find(s))
  //   std::cout << "bulundu idx: " << str.find(s)<<"\n";
}
---
```

```
---
int main()
{
  std::string str;
  std::cout << "yazı girin: ";
  getline(std::cin, str);
  // argumen olarak gönderilenderden her hangi biri aranır
  str.find_first_of("aeoui"); //c'deki strpbrk
  str.find_first_of(s); // s stringinin karakterlerinden her hangi birini

  str.find_last_of(s); // s stringinin karakterlerinden en son bulundugu teri bulur

  str.find_first_not_of("0123456789"); // str de rakam olmayan ilk adresi bulur
  str.find_last_not_of("153"); // str de sondan baslayarak 153 rakamları olmayanı bulur

  str.rfind(s); // yazıyı sondan aramaya baslar

}
---
```

```
** substr **
-> string'in bir kısmını alır
-> kopyalama maliyeti
str.substr(6,3);
// 6. idx den baslayarak karakter alır

str.substr(3); // idx 3'ten geri kalan yazı
str.substr(); //stringin kopyasını elde eder
```

** replace **
str.replace(0, 2, "abcd");
// 0 idxden baslayarak 2 karakterlik kısmı abcd ile
  degistirir.

std::string str {"aliveli"};
str.replace(0, 2, "istanbul");
// output
istanbuliveli

String Function	Effect
stoi(*str*, *idxRet*=nullptr, *base*=10)	Converts *str* to an int
stol(*str*, *idxRet*=nullptr, *base*=10)	Converts *str* to a long
stoul(*str*, *idxRet*=nullptr, *base*=10)	Converts *str* to an unsigned long
stoll(*str*, *idxRet*=nullptr, *base*=10)	Converts *str* to a long long
stoull(*str*, *idxRet*=nullptr, *base*=10)	Converts *str* to an unsigned long long
stof(*str*, *idxRet*=nullptr)	Converts *str* to a float
stod(*str*, *idxRet*=nullptr)	Converts *str* to a double
stold(*str*, *idxRet*=nullptr)	Converts *str* to a long double
to_string(*val*)	Converts *val* to a string
to_wstring(*val*)	Converts *val* to a wstring

Table 13.6. Numeric Conversions for Strings

** stoi **
---
std::string str {"175aliveli"};
auto ival = stoi(str);
std::cout << ival << "\n"; // 175

```
std::string str {"175aliveli"};
size_t idx;
auto ival = stoi(str, &idx);
std::cout << ival << "\n"; // 175
std::cout << idx << "\n"; // 3
```

```
std::string str {"175aliveli"};
size_t idx;
auto ival = stoi(str, &idx, 16); //hex
std::cout << ival << "\n";  //5978
std::cout << idx << "\n"; // 4
//a da hex sistemde gecerli
```

```
int main()
{
  for (int i = 100; i < 1000; ++i)
  {
          auto square = i * i;
          auto s = std::to_string(i) + std::to_string(square);
          if (s.length() != 9)
          continue;
          sort(begin(s), end(s));
          if (s == "123456789")
          { // tüm rakamlar
          std::cout << i << " " << square << "\n";
          }
  }
}
// sayının kendisi ve karesi tüm rakamları 1 kez icerir
//output
567 321489
854 729316
```

----- inheritance -----

super class - subclass
parent class - child class
base - derived
super class -> complete type olmalı

```
class Base {

};

class Der : public Base { //der, public kalıtımı ile base sınıfından olusur
// public, private, protected yazılmazsa default : private Base olur
};

struct Der : Base { // default -> :public Base

};
```

-> her der nesnesi aynı zamanda Base türündendir

```
class Base {
public:
  void foo(int);

};
```

```
class Der : public Base {
  void foo(int ,int);
};


int main()
{
  Der myder;
  myder.foo(12); // syntax error
  // aranan isim bulundugunda isim arama biter
  //funtion overloading degil scope farklı

  myder.Base::foo(12); // gecerli

}
---

-> name lookup
  blok icinde -> Der class member icinde -> base class -> namespace

-> access control, name lookup sonrasında bakılır

---
class Base {
public:
  void foo(int);

};

class Der : public Base {
private:
  void foo();

};

int main()
{
  Der myder;
  myder.foo(250); // syntax error
  myder.foo(); // syntax error
  myder.Base::foo(12); // gecerli
}
---

-> türemiş sınıftan, taban sınıfa implicit type conversion var

-> taban sınıf türünden bir pointer türemiş sınıf türünden bir nesne adresi tutabilir
```

```
int main()
{
  Base *p = new Base;
  Base *p = new Der;
  //
  Der myder;
  Base *baseptr;
  baseptr = &myder;
  //referans icin de gecerli
  Base& baseref = myder;
}
```
---

upcasting -> türemiş sınıftan, taban sınıfa yapılan dönüşüm

```
Der myder;
Base mybase;
mybase = myder; // object slicing
//syntax hatası değil, fakat doğru işlem değil
```

```
class Base {
public:
  int x;
};

class Der : public Base {
};


int main()
{
  Der myder;
  Base mybase;
  std::cout<< &myder.x << "\n";
  std::cout<< &mybase.x << "\n";
}
//output
0x7fffcb36ce10
0x7fffcb36ce14
// taban sınıfın x nesnesinin adresi ile,
  türetilmiş sınıfın, taban sınıfa ait x nesnesinin
```

adresleri aynı değil, adresileri sıralı olmak zorunda değil

```
class Base {
public:
  int x, y;
};

class Der : public Base {
  int z;
};


int main()
{
  Der myder;
  Base mybase;
  std::cout << "sizeof(Base) = " << sizeof(Base) << "\n";
  std::cout << "sizeof(Der) = " << sizeof(Der) << "\n";
}

//output
sizeof(Base) = 8
sizeof(Der) = 12

// Der sınıfı içinde Base sınıfı nesnesi var
```

-> türemiş sınıfın default ctoru derleyici yazıyorsa,
taban sınıf nesnesi için default ctor cagıran kod üretir

```
class Base {
public:
  Base()
  {
          std::cout << "Base default ctor" << "\n";
  }
```

```
  ~Base()
  {
        std::cout << "Base destructor" << "\n";
  }
};

class Der : public Base {
};

int main()
{
  Der myder;
}
//output
Base default ctor
Base destructor

// ctor private veya delete edilseydi syntax error
---
```

```
---
class Base {
public:
  Base()
  {
        std::cout << "Base default ctor" << "\n";
  }

  ~Base()
  {
        std::cout << "Base destructor" << "\n";
  }
};

class Der : public Base {
public:
  Der()
  {
        std::cout << "Der default ctor" << "\n";
  // ctor initializer list ile cagrı yapılmazsa,
  // base in default ctorundaki cagrı alınır
  // initializer list ile deger verilmeyince default oluyordu
```

```
  }

  ~Der()
  {
        std::cout << "Der destructor" << "\n";
  }
};

int main()
{
  Der myder;
}
//output
Base default ctor
Der default ctor
Der destructor
Base destructor
---



---
class Base {
public:
  Base(int x)
  {
        std::cout << "Base int(x) ctor" << "\n";
  }

  ~Base()
  {
        std::cout << "Base destructor" << "\n";
  }
};

class Der : public Base {
public:
  Der() : Base(12) //Base{34}
  // Der() //syntax error , base in default ctoru yok
  // default degil int paramteri ctor cagırlır
  {
        std::cout << "Der default ctor" << "\n";
  }

  ~Der()
  {
        std::cout << "Der destructor" << "\n";
  }
```

```
};

int main()
{
  Der myder;
}
---




---
class MemberX {
public:
  MemberX()
  {
        std::cout << "MemberX default ctor" << "\n";
  }

  ~MemberX()
  {
        std::cout << "MemberX destructor" << "\n";
  }
};

class MemberY {
public:
  MemberY()
  {
        std::cout << "MemberY default ctor" << "\n";
  }

  ~MemberY()
  {
        std::cout << "MemberY destructor" << "\n";
  }
};

class Base {
public:
  Base()
  {
        std::cout << "Base default ctor" << "\n";
  }

  ~Base()
  {
```

```
        std::cout << "Base destructor" << "\n";
  }
};

class Der : public Base {
public:
  MemberX mx;
  MemberY my;
};

int main()
{
  Der myder;
}
//output
Base default ctor
MemberX default ctor
MemberY default ctor
MemberY destructor
MemberX destructor
Base destructor
---
```

copy ctor

```
class Der{
public:
  Der(const Der& other) : Base(other), ax,(ather.ax)...
  {

  }
};
```

```
class Base {
public:
  Base()
  {
        std::cout << "Base default ctor" << "\n";
```

```
  }

  Base(const Base&)
  {
        std::cout << "Base copy ctor" << "\n";

  }
};

class Der : public Base {
public:
};

int main()
{
  Der myder;
  Der y(myder);
}
//output
Base default ctor
Base copy ctor
---




---
class Base {
public:
  Base()
  {
        std::cout << "Base default ctor" << "\n";
  }

  Base(const Base&)
  {
        std::cout << "Base copy ctor" << "\n";

  }
};

class Der : public Base {
public:
  Der() = default;
  Der(const Der& other)
  {
        // Base copy degil default ctoruna cagrı yapar
  }
```

```
};

int main()
{
  Der myder;
  Der y(myder);
}
//output
Base default ctor
Base default ctor
---



---
class Base {
public:
  Base()
  {
        std::cout << "Base default ctor" << "\n";
  }

  Base(const Base&)
  {
        std::cout << "Base copy ctor" << "\n";

  }
};

class Der : public Base {
public:
  Der() = default;
  Der(const Der& other) : Base(other)
  {

  }
};

int main()
{
  Der myder;
  Der y(myder);
}
//output
Base default ctor
Base copy ctor
---
```

copy assignment

```
class Base {
public:
  Base& operator=(const Base&)
  {
        std::cout << "Base copy assignment" << "\n";
        return *this;
  }
};

class Der : public Base {
public:

};

int main()
{
  Der x, y;
  x = y;
}
//output
Base copy assignment
```

```
class Base {
public:
  Base& operator=(const Base&)
  {
        std::cout << "Base copy assignment" << "\n";
        return *this;
  }
};
```

```
class Der : public Base {
public:
  Der& operator=(const Der& other)
  {
        //operator=(other); //recurisive
        //*this = other; //recurisive

        Base::operator=(other);
        static_cast<Base &>(*this) = other; // anlamsal aynı
        return *this;
  }
};

int main()
{
  Der x, y;
  x = y;
}
//output
Base copy assignment
---
```

- sınıf ici using kullanımı

```
---
class Base {
public:
  void func(int)
  {
        std::cout << "Base::func(int)\n";
  }

};

class Der : public Base {
public:
  // void func(int x)
  // {
  //   Base::func(x);
  // }
  // bunun yerine ovaerloading icin
  using Base::func; // ile bildirim scope da visable hale gelir
  void func(double)
  {
```

```
        std::cout << "Der::func(double)\n";
  }

};

int main()
{
  Der myder;
  myder.func(12);
  myder.func(1.2);
}
//output
Base::func(int)
Der::func(double)
---
```

-> taban sınıfın protected memberları
  türemiş sınıfta using bildirimi ile tanımlaranarak
  mainde türemiş classtan erişilebilir

  *** runtime polymorphism ***

airplane

1) hem bir arayüz (interface) hem de bir kod (implementation) veriyor

2) hem bir arayüz (interface) hem de default bir kod (default implementation) veriyor
  override etmek
  -en az bir fonksiyonu varsa böyle sınıflara polimorphic class denir

3) bir arayüz veren ama implementasyon vermeyen
  override etmek

- en az bir fonksiyonu varsa abstract (soyut) class

-> c++' da bir sınıf ya abstract, ya da concrete(somut)

```
class Airplane {
public:
  void takoff();
  virtual void land(); //sanal üye fonk
  // eger sanal fonksiyonu olmasaydı airplane classı non-polimorphic

  virtual void fly() = 0; // pure virtual function(saf sanal)
  // class'ın en 1 pure virtual fonk sahip olması abstract class yapar

};

int main()
{
  Airplane ax; // soyut sınıftan nesne olusturulamaz
  // syntax  error
}
```

-> eger hangi fonksiyonun cagırırılacagı derleme zamanında
belli oluyorsa early binding (static binding)

-> eger hangi fonksiyonun cagırırılacagı çalışma zamanında
belli oluyorsa late binding (dynamic binding)

```
void game(Car &car)
{
  car.start();
  car.run();
  car.stop();
}
// hangi araba icin calıstıgı programın calısma zamanında belli olur
```

```
class Car
{
```

```
public:
  void start()
  {
        std::cout << "Car has started!\n";
  }
};

class Audi : public Car
{
public:
  void start()
  {
        std::cout << "Audi has started!\n";
  }
};

void car_game(Car& car)
{
  car.start();
}

int main()
{
  Audi ax;
  car_game(ax);
}
//output
Car has started!

// early binding
---



---
class Car
{
public:
  virtual void start()
  {
        std::cout << "Car has started!\n";
  }
};
//output
Audi has started

// late binding (dynamic bindig)
---
```

-> bu mekanizmaya virtual dispatch denir

```
class Car
{
public:
  virtual void start()
  {
        std::cout << "Car has started!\n";
  }
};

class Audi : public Car
{
public:
  void start()
  {
        std::cout << "Audi has started!\n";
  }
};

class Mercedes : public Car
{
public:
  void start()
  {
        std::cout << "Mercedes has started!\n";
  }
};

class Fiat : public Car
{
public:
  void start()
  {
        std::cout << "Fiat has started!\n";
  }
};

void car_game(Car& car)
{
  car.start();
}
```

```
int main()
{
  Audi audi;
  Mercedes mercedes;
  Fiat fiat;
  car_game(audi);
  car_game(mercedes);
  car_game(fiat);
}
//output
Audi has started!
Mercedes has started!
Fiat has started!

//runtime da belli olur
```

-> override etmek için gem geri donusu hem de imzası aynı olmalı

```
class Base {
public:
 virtual void func(int, int);
};

class Der : public Base {
public:
  // void func(int, int); //gecerli ama kullanılmamalı
  void func(int, int)override;
};
```

-> eger taban sınıfın bir sanal fonksiyonu
  bir taban sınıf poiteri ya da bir taban sınıf referansı ile
  çagırılırsa, çalışma zamanında çagırılan fonksiyon
  hangi türden bir sınıf nesnesini gösteriyorsa/bagalanıyorsa
  o sınıfın ute fonk olur.

```
class Base {
public:
 virtual void func()
 {
          std::cout << "Base::func()\n";
  }
};

class Der : public Base {
public:
  void func()override
  {
          std::cout << "der::func()\n";
  }
};
int main()
{
  Der myder;
  Base * baseptr = &myder;
  baseptr->func(); //virtual dispatch
}
// int main()
// {
//   Der myder;
//   Base& baseptr = myder;
//   baseptr.func(); //virtual dispatch
// }

//output
der::func()
---


---
int main()
{
  Der myder;
  Base base = myder; //object slicing
  // virtual dispatch artık olmayacak
  // artık compile time' da belli olucak
  base.func();
}
//outpu
Base::func()
---
```

```
class Base
{
public:
  virtual void func()
  {
          std::cout << "Base::func()\n";
  }
  void foo() // this ile gerceklestiğinden hangi türden nesneyi gösteriyorsa o cagırlır
  {
          func();
  }
};

class Der : public Base
{
public:
  void func() override
  {
          std::cout << "der::func()\n";
  }
};
int main()
{
  Der myder;
  myder.foo(); //dispatch gerceklesir
}
//output
der::func()
```

```
class Base
{
public:
  virtual void func()
```

```
  {
        std::cout << "Base::func()\n";
  }
};

class Der : public Base
{
public:
  void func() override
  {
        std::cout << "der::func()\n";
  }
};

class Myclass : public Der
{
public:
  void func() override //virtual void func() override
  {
        std::cout << "Myclass::func()\n";
  }
};

void gfoo(Base& baseref)
{
  baseref.func();
}
int main()
{
  Myclass ax;
  gfoo(ax);
}
//output
Myclass::func()
---




contextual keyword (baglamsal anahtar sözcük)
  - override
  - final

int for; //syntax error
int override = 5; //gecerli


void func()override // bu context'te kullanıldıgında key niteliğinde
```

virtual dispatch' in olmadığı durumlar
- fonk cagrısı taban sınıf türünden bir değişken ile yapılırsa
- fonk cagrısında nitelenmiş isim kullanılırsa
        carptr->Car::run()
- ctor icide yapılan sanal fonk cagrıları

-> taban sınıf ctor içinde sınıfın sanal fonk cagrı yapılırsa
  virtual dispatch devreye girmez

-> taban sınıf destructor içinde sınıfın sanal fonk cagrı yapılırsa
  virtual dispatch devreye girmez

```
//bmw nin start fonk private olarak override edilmiştir
Bmw* p = new Bmw;
Car* cp = p;
cp->start(); // gecerli
p->start(); //syntax error

delete p;
```

-> çükü access control compile time ile ilişkin,
-> virtual dispatch olsaydı runtime oldugundan hata olmazdı
-> isim arama static type a ilişkin
-> virtual dispatch dynamic type

```
class Base
{
public:
  virtual void func(int x = 77)
  {
          std::cout << "Base x = " << x << "\n";
  }
};

class Der : public Base
{
public:
  void func(int x = 33)override
  {
          std::cout << "Der x = " << x << "\n";
  }

};

void foo(Base *p)
{
  p->func();
}

int main()
{
  foo(new Der);
}
//output
Der x = 77

// sanallık devreye girer,
  varsayılan arguman static type gore yapılır
```

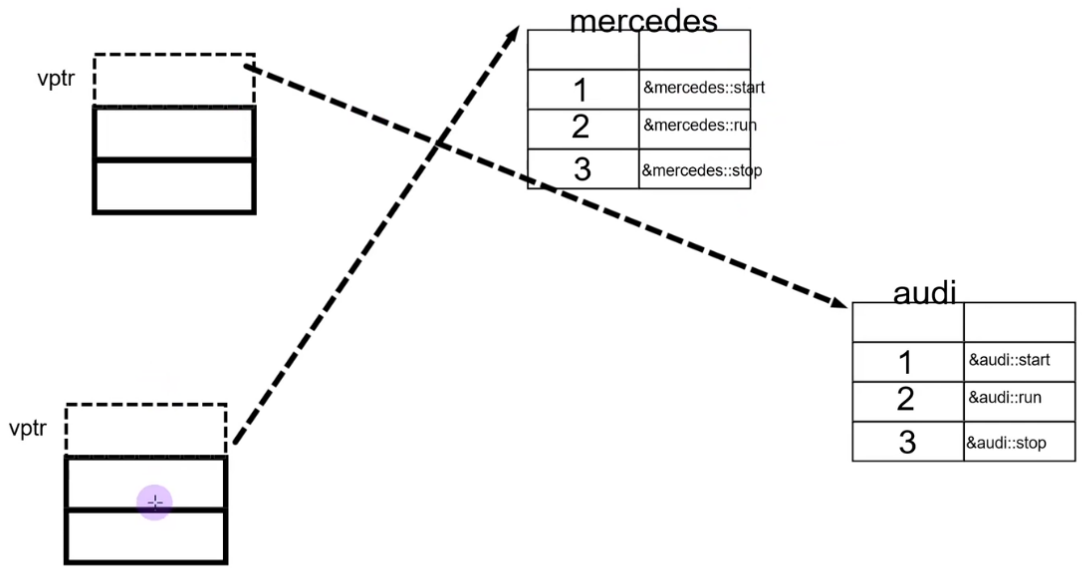
-> NVI

```
class Base
{
public:
  void bar() // NVI -> non virtual inerface
  {
          //code
          foo();
          //code
  }
private:
  virtual void foo()
  {
          std::cout << "base::foo()\n";
  }

};

class Der : public Base
{
public:
  virtual void foo() override
  {
          std::cout << "der::foo()\n";
  }
};
```

-> bir sınıfın (int x,y) 8byte
  en az 1 virtual oldugunda 4 byte eklenebilir
  virtual fonk sayısı arttıkca byte artmaz

-> bu değişim sınıfın polimorphic olup olmamasıyla ilgili

vptr
-> her bir sınıf için bir sanal tablosuna ihtiyac
var.



-> non virtual dispatch oldugunda
  ptr->foo() // derleyici hangi fonk cagırıldıgını compile time da bilir

-> polimorphic cagırılarda bu maliyete ek
  +2 derefencing maliyeti eklenir
  - sınıf nesnesinin icindeki vptr degerini get
        edicek ki sanal fonksiyon tablosuna erişssin
  - sonra fonksiyonun indexli ile adresi elde edilir

devirtualization: virtual dispatch olan fonksiyonun
  koduna bakarak derleyicinin compile time'da belirlemesi
   - bu optimizasyon yapılmazsa 2 derefencing maliyeti var

-- virtual constructor
  - ctor icin virtual olması soz konusu değil

-- virtual constructor idiom (clone)

```
class Car
{
public:
  virtual Car *clone() = 0;
};

class BMW : public Car
{
public:
  Car *clone() override
  {
          return new BMW(*this);
  }
};
```

--> clone eger abstract class degilse copy ctor ve copy assignment delete edilmeli.
  -> object slicing

```
void car_game(Car* p)
{
          Car* pnewcar = p->clone();
          p->start();
          pnewcar->start(); // aynı nesne

}

int main()
{
    srand(time(nullptr));
          for(;;) {
          auto cp = createRandomCar();
          car_game(cp);
          delete cp;
```

```
        }
}
---
```

```
---
class Car
{
public:
  virtual Car *clone() {return new Car;} //abstract degilse
  Car(const Car& other) = delete;
  Car& operatorler=(const Car& other) = delete;
  // yanlıslıkla object slicing senaryosunda copy ctor ve assignment delete edildiğinden
        compile time hatası olur
};

void car_game(Car* p)
{
        Car mycar= *p; //object slicing
        // artık virtual dispatch yok
        mycar.run();
}
```

```
--virtual destructor

class Base {
public:
  ~Base()
  {
        std::cout << "~Base()\n";
  }
};

class Der : public Base {
public:
  ~Der()
  {
        std::cout << "~Der()\n";
  }
};
```

```
int main()
{
   Base* p = new Der; //ub olabilir
         //
         delete p;
}
//output
~Base()
```

-> p Base* türünden oldugundan derleyici ~base cagırıldı
-> p hangi classı gösteriyorsa o nesnenin destructor cagırılmalı (~Der)

-> taban sınıfın destructor'ı virtual olmalı!

```
class Base {
public:
  virtual ~Base()
  {
         std::cout << "~Base()\n";
  }
};

class Der : public Base {
public:
  ~Der()
  {
         std::cout << "~Der()\n";
  }
};

int main()
{
   Base* p = new Der;
         //
         delete p;
}
//output
~Der()
~Base()
```

-> türemiş sınıf nesnelerin, taban sınıf pointeri ile kullanılması istenmiyorsa:
  taban sınıf destructor'ı virtual yapmayıp, protected yapılabilir.
  - taban sınıf pointerı delete edildiğinde hata olusur syntax error
        cünkü delete dolaylı olarak destructor cagırır, access control yapıldıgında
        protected oldugundan erisemez(compile time)

```
class Base {
public:
  ~Base()
  {
        std::cout << "~Base()\n";
  }
};

class Der : public Base {
public:
  ~Der()
  {
        std::cout << "~Der()\n";
  }
};

int main()
{
    Base* p = new Der; //ub olabilir
        //
        delete p; //syntax error

        // Der* p = new Der;
        // delete p; // gecerli
        -> taban sınıfın protected ogelerine türemiş sınıfın ogeleri erişebilir
}
```

- global fonks idiomatik yapılar dışında virtual olamaz

```
class Car {
public:
    virtual void start() = 0;
    virtual void run() = 0;
```

```
    virtual void stop() = 0;
    virtual ~Car() {}
    virtual Car* clone() = 0;
    virtual void print(std::ostream&)const = 0;
};

void Bmw::print(std::ostream& os)const
{
    os << "I am a Bmw\n";
}

int main()
{
    srand(time(nullptr));
        Car* p = createRandomCar();
        p->start();
        std::cout << *p << "";
        p->run();
}
//output
Bmw has just started
I am a Bmw
Bmw is running
---


---
int main()
{
    std::vector<Car*> cvec;
        srand(time(nullptr));

        for(int i = 0; i < 20; ++i) {
        cvec.push_back(createRandomCar());
        }

        for(auto p : cvec)
        {
        std::cout << *p <<"\n";
        p->start();
        p->run();
        p->stop();
        }

        for(auto p : cvec)
        delete p;
}
---
```

-- variant return type (covariance)

```
class Base {
public:
	virtual int foo(int);
};

class Der : public Base {
public:
	float foo(int)override;// syntax error
}
```

```
class A {

};

class DerA : public A {

};

class Base {
public:
	virtual A * foo(int);
};

class Der : public Base {
public:
	DerA *foo(int)override;
}
// pointer veya referans sematiği ile olmalı
```

-> eğer taban sınıfın sanal fonk geri dönüş değeri sınıf türünden
  bir pointersa türemiş sınıf bunu override ederken türemiş

sınıfları türünden bir adres döndürebilir

```
class Car {
public:
   virtual Car* clone() = 0;
};


class Bmw  : public Car {
public:
   Bmw* clone() override; // Car* clone() override
};

int main()
{
  Bmw* pb = new Bmw;
  Bmw* px = pb->clone(); // covariance
  // Car* clone()override olsaydı syntax error,
  //  static_cast<Bmw*>(pb->clone()) gerekli olurdu
}
```

-- inherited constructor

```
class Base {
public:
        Base() = default;
        Base(int);
        Base(int, int);
        Base(double);
        void foo();
};

class Der : public Base {
public:
        void g();
};
```

```
int main()
{
	Der myder(12, 10); //syntax error Der classının 2 parametreli ctoru yok
}
---



---
class Base {
public:
	Base() = default;
	Base(int);
	Base(int, int);
	Base(double);
	void foo();
};

class Der : public Base {
public:
	Der(int x) : Base{x} {}
	Der(int a, int b) : Base{a, b} {}
	Der(double a) : Base{a} {}
	void g();
};


int main()
{
	Der myder(12, 10); // geçerli
}
---



---
class Base {
public:
	Base() = default;
	Base(int);
	Base(int, int);
	Base(double);
	void foo();
};

class Der : public Base {
```

```
public:
        using Base::Base; // inherited ctor icin
        void g();
};

int main()
{
        Der myder(12, 10); // geçerli
}
```

-> Der sınıfı için default ctor cagırıldıgında, derleyicinin Der için
  yazdıgı cagırlır

-> copy veya move ctor taban sınıftan alımıyor.

-> yukardaki Der sınıfı örneğinde Der(int) türünden ctor olsa ve çağırılsa
  Base(int) değil, Der(int) calısır

```
class Base {
protected:
        void foo(int );
};

class Der : public Base {
public:
        using Base::foo;
};

int main()
{
        Der myder;
        myder.foo(12); // gecerli
}
```

```
class Base {
protected:
        Base(int);
};
```

```
class Der : public Base {
public:
	using Base::Base;
};

int main()
{
	Der myder(12); //syntax error
}
---
```

```
---
class Base {
public:
	Base() : mx{0}{}
	Base(int x) : mx{x}{}
	void print()const
	{
	std::cout << "mx = " << mx <<"\n";
	}
private:
	int mx;
};

class Der : public Base {
public:
	using Base::Base;
	void print()const
	{
	Base::print();
	std::cout << "mval = " << mval <<"\n";
	}
private:
	int mval;
};


int main()
{
	Der myder(40);
	myder.print();
}
//output
mx = 40
```

mval = 32765 // çöp deger, default initial edildi

-- multiple inheritance

```
class A {

};

class B {

};

class C : public A, public B {

};
```
—
```
int main()
{
	C cx;
	A& ra = cx;
	B* pb = &cx;
	// upcasting
}
```

```
class A {
public:
	A()
	{
```

```
        std::cout << "A ctor\n";
        }
        ~A()
        {
        std::cout << "A destructor\n";
        }
};

class B {
public:
        B()
        {
        std::cout << "B ctor\n";
        }
        ~B()
        {
        std::cout << "B destructor\n";
        }
};

class C : public A, public B { // yazım sırasına göre cagırlır

};


int main()
{
        C cx;
}
//output
A ctor
B ctor
B destructor
A destructor
---
```

--> birden fazla taban sınıf bulunması durumunda isim arama
  her taban sınıf içinde yapılır.

```
class A {
public:
	void func(int);
};

class B {
public:
	void func(int, int);

};

class C : public A, public B {

};


int main()
{
	C cx;
	cx.func(1); // syntax error
	//ambiguitiy
}
---



---
class A {
public:
	void func(int);
};

class B {
public:
	void func(int, int);

};

class C : public A, public B {
public:
  void func(int);
};


int main()
{
	C cx;
	cx.func(1); // gecerli ilk C' de aranır
```

```
        cx.func(15, 5) //syntax error C' de (int, int) yok
        cx.A::func(5); // gecerli
        cx.B::func(15, 20); // gecerli

}
---



---
class A {
public:
        void func(int);
};

class B {
public:
        void func(int, int);

};

class C : public A, public B { // yazım sırasına göre
public:
        using A::func;
        using B::func;
};


int main()
{
        C cx;
        cx.func(5); // gecerli
        cx.func(5, 25); //gecerli
}
---
```

--> bütün saf sanal fonk override ederse concrete,
  etmezse abstract class olur

```
---
class Shape {
public:
```

```
        virtual double get_area()const = 0;
        virtual double get_perimeter()const = 0;
        virtual ~Shape() = default;
};

class Drawable {
        virtual void draw() = 0;
};

class Paintable {
public:
        virtual void paint() = 0;
};

class Circle : public Shape, public Drawable, public Paintable {
public:
        double get_area()const override;
        double get_perimeter()const override;
};
---
```

```
-- diamond formation
  - DDD -> dreaded diamond of derivation

        Base
  Der1          Der2
        Mder

---
class Base {

};

class Der1 : public Base {

};

class Der2 : public Base {
public:
};
class Mder : public Der1, public Der2
```

```
{
};
---
```

-> Mder içinde 2 Base nesnesi olur //ambiguitiy olusur
  Der1::Base, Der2::Base
  - Der1 de Base nesnesi değişirse Der2 de değişmez

```
---
class Base {
public:
	void foo();
};

class Der1 : public Base {

};

class Der2 : public Base {
public:
};

class Mder : public Der1, public Der2
{
};

int main()
{
	Mder md;
	md.foo(); //syntax error ambiguitiy
	// 1 foo fonk var, 2 tane Base nesnesi var

	md.Der1::foo(); // gecerli
	md.Der2::foo(); // gecerli
}
---
class Mder : public Der1, public Der2
{
public:
	void bar()
	{
	Der1::foo();
	Der2::foo();
	}
};
---
```

```
int main()
{
	Mder md;
	// Base* bptr = &x; // syntax error
	Base& bptr = x; // synatx error
	Base* bptr = static_cast<Der1*>(&x); // gecerli

}
```

```
class Device
{
public:
	void turnon()
	{
	std::cout << "cihaz acıldı...\n";
	on_flag = true;
	}
	void turnoff()
	{
	std::cout << "cihaz kapatıldı...\n";
	on_flag = false;
	}
	bool is_on() const
	{
	return on_flag;
	}
	virtual ~Device() = default;

private:
	bool on_flag{false};
};

class Printer : public Device
{
public:
	virtual void print()
	{
```

```
        if (!is_on())
        std::cout << "cihaz kapali, print islemi yapilamıyor\n";
        else
        std::cout << "print islemi yapildi\n";
        }
};

class Scanner : public Device{
public:
        virtual void scan()
        {
        if(!is_on())
        std::cout << "cihaz kapalı, tarama yapılamıyor\n";
        else
        std::cout << "tarama yapıldı\n";
        }
};

class Combo : public Printer, public Scanner
{
public:
};

int main()
{
        Combo cx;

        cx.Printer::turnon();
        cx.print();
        cx.scan();
        cx.Printer::turnoff();
        cx.print();
}
//output
cihaz acıldı...
print islemi yapildi
cihaz kapalı, tarama yapılamıyor
cihaz kapatıldı...
cihaz kapali, print islemi yapilamıyor
// Combo sınıdında, Device değiştiğinde sadece birinde
  degisir, 2 Device var, çözümü virtual inheritance
---




--> Mder içinde 1 Base nesnesi olması  (DDD olmaması)
  sağlayan araç virtual inheritance
```

virtual inheritance
-> birden fazla taban sınıf nesnesinden birden fazla değil de
  1 tane olması icin için virtual inheritance

```
//virtual base
class Device
{
public:
        void turnon()
        {
        std::cout << "cihaz acıldı...\n";
        on_flag = true;
        }
        void turnoff()
        {
        std::cout << "cihaz kapatıldı...\n";
        on_flag = false;
        }
        bool is_on() const
        {
        return on_flag;
        }
        virtual ~Device() = default;

private:
        bool on_flag{false};
};

class Printer : virtual public Device
{
public:
        virtual void print()
        {
        if (!is_on())
        std::cout << "cihaz kapali, print islemi yapilamıyor\n";
        else
        std::cout << "print islemi yapildi\n";
        }
};

class Scanner : virtual public Device{
public:
```

```
        virtual void scan()
        {
        if(!is_on())
        std::cout << "cihaz kapalı, tarama yapılamıyor\n";
        else
        std::cout << "tarama yapıldı\n";
        }
};

class Combo : public Printer, public Scanner
{
public:
};

int main()
{
        Combo cx;

        cx.turnon();
        cx.print();
        cx.scan();
        cx.turnoff();
        cx.print();

        // cx.Printer::turnon();
        // cx.print();
        // cx.scan();
        // cx.Printer::turnoff(); // scanner' da kapanır
        // cx.print();

        -> combo' nun icinde 1 Device var
}
//output
cihaz acıldı...
print islemi yapildi
tarama yapıldı
cihaz kapatıldı...
cihaz kapali, print islemi yapilamıyor
---



-> virtual inheritance varsa,
  her zaman ilk virtual Base hayata gelir

-> Combo class içerisinde Device ctoruna çağr yapılmalı
```

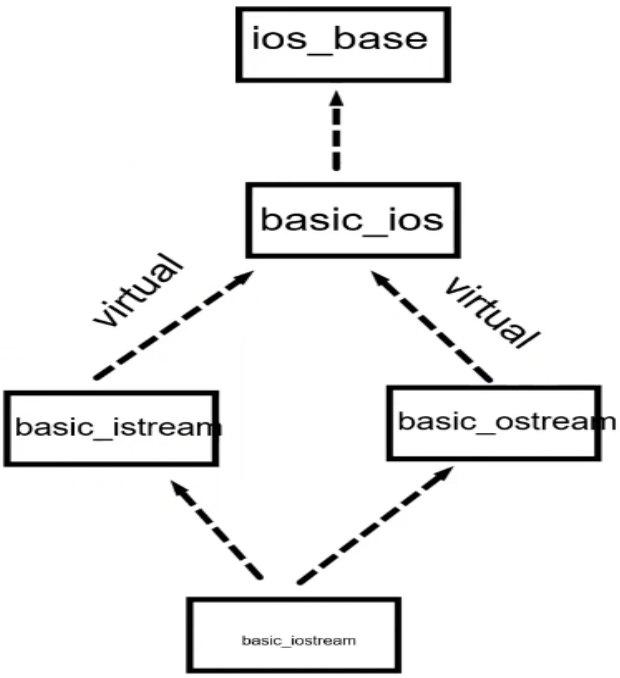
```
//virtual base
class Device
{
public:
	Device()
	{
	std::cout << "Device() ctor\n";
	}
};

class Printer : virtual public Device
{
public:
	Printer()
	{
	std::cout << "Printer() ctor\n";
	}
};

class Scanner : virtual public Device{
public:
	Scanner()
	{
	std::cout << "Scanner() ctor\n";
	}
};

class Combo : public Printer, public Scanner
{
public:
};

int main()
{
	Combo cx;
}
//output
Device() ctor
Printer() ctor
Scanner() ctor
```

-> Device ctorunu çağıran Combo ctor

```
//virtual base
class Device
{
public:
	Device()
	{
	std::cout << "Device() ctor\n";
	}
};

class Printer : virtual public Device
{
public:
	Printer()
	{
	std::cout << "Printer() ctor\n";
	}
};

class Scanner : virtual public Device{
public:
	Scanner()
	{
	std::cout << "Scanner() ctor\n";
	}
};

class Combo : public Printer, public Scanner
{
public:
	Combo()
	{
	std::cout << "Combo() ctor\n";
	}
};

class MultiCombo : public Combo {
public:
	MultiCombo()
	{
	std::cout << "MultiCombo() ctor\n";
	}
};

int main()
```

```
{
	MultiCombo cx;
}
//output
Device() ctor // MultiCombo ctoru Device ctor' u cagırdı
Printer() ctor
Scanner() ctor
Combo() ctor
MultiCombo() ctor
---




---
//virtual base
class Device
{
public:
	Device(int x)
	{
	std::cout << "Device(int) x = "<< x <<"\n";
	}
};

class Printer : virtual public Device
{
public:
	Printer(int x) : Device(x)
	{
	std::cout << "Printer() ctor\n";
	}
};

class Scanner : virtual public Device{
public:
	Scanner(int x) : Device(x)
	{
	std::cout << "Scanner() ctor\n";
	}
};

class Combo : public Printer, public Scanner
{
public:
	Combo() : Device(3), Printer(4), Scanner(5)
	{ // virtual base class olmasa Combno : Device(3) syntax error olurdu
```

```
        std::cout << "Combo() ctor\n";
        }
};

class MultiCombo : public Combo {
public:
        MultiCombo() : Device(4)
        {
        std::cout << "MultiCombo() ctor\n";
        }
};

int main()
{
        MultiCombo cx;

        // Combo cx; çağırılırsa Device(int) x = 3 olur

        // Printer px(340); cağırılırsa Device(int) x = 340
}
//output
Device(int) x = 4
Printer() ctor
Scanner() ctor
Combo() ctor
MultiCombo() ctor
---
```



```
//virtual base
class Device
{
public:
	Device(int x)
	{
	std::cout << "Device(int) x = "<< x <<"\n";
	}
};

class Printer : virtual public Device
{
public:
	Printer(int x) : Device(x)
	{
	std::cout << "Printer() ctor\n";
	}
};

class Scanner : virtual public Device{
public:
	Scanner(int x) : Device(x)
	{
	std::cout << "Scanner() ctor\n";
	}
};

class Combo : public Printer, public Scanner
{
public:
	Combo() : Device(3), Printer(4), Scanner(5)
	{ // virtual base class olmasa Combno : Device(3) syntax error olurdu
	std::cout << "Combo() ctor\n";
	}
};

class MultiCombo : public Combo {
public:
	MultiCombo() : Device(4)
	{
	std::cout << "MultiCombo() ctor\n";
	}
};

int main()
{
```

```
	MultiCombo cx;

	// Combo cx; çağırılırsa Device(int) x = 3 olur

	// Printer px(340); cağırılırsa Device(int) x = 340
}
//output
Device(int) x = 4
Printer() ctor
Scanner() ctor
Combo() ctor
MultiCombo() ctor
---
```

- final class -> kalıtımda taban sınıfı olarak kullanılamaz
- final override

```
---
class Base {
};

class Der final : public Base {
};

class A : public Der { // final class type can not be used as a base class
//syntax error
};
---
```

final override

```
---
class Base {
public:
	virtual void func(int);

};

class Der : public Base {
public:
	void func(int) override final;
};

class Myclass : public Der {
```

```
public:
	void func(int) override; // syntax error
	// Cannot override final function
};
```
---

public inheritance -> IS A type

private ve protected inheritance kullanımındaki temel amac composition

-- private inheritance

```
class Base {
};

class Der : private Base {

};


int main()
{
	Der myder;
	Base* bptr = &myder; // syntax error
	// Base& bt = myder; // syntax error
	// public kalıtımı olsaydı upcasting olurdu
	// her Der bir Base anlamına gelmez
	// upcasting client kodlara legal degil, sınıfın kendi üye fonk.da legal
}
```

```
class Der : private Base {
	void foo()
	{
	Der myder;
	Base* bptr = &myder; // gecerli
	// türemiş sınıflara friend fonksiyonlarında da upcasting gecerli
	}
```

```
};
---




---
class Base {
public:
	void foo();
};

class Der : private Base {

};

int main()
{
	Der myder;
	myder.foo(); // syntax error
	// inaccessible
	// Base sınıfının ögeleri,
	// Der sınıfının private ogeleri gibi ele alınır
}
---




----
//	benzerlikleri
// Owner içinde bir Member nesnesi var
// Owner Member interface'ini kendi arayüzüne katar
// Owner isterse Member fonksiyonlarını kendi arayüzüne ekler

//	farklılıkları
// Her Owner bib member olarak kullanılamaz
// Owner member'in sanal işlevlerini override edemez
// Owner Member'in protected bolümüne erişemez
class Member { // Containment
public:
	void foo();
};
```

```
class Owner {
public:
	void foo()
	{
	mx.foo();
	}
private:
	Member mx;
};

//////////////////////////////////////////////////////

//	benzerlikleri
// Der içinde bir Base nesnesi var
// Der Base interface'ini kendi arayüzüne katar
// Der isterse Base fonksiyonlarını kendi arayüzüne ekler

//	farklılıkları
// her Der Base olarak kullanılamıyo, bir farkla üye fonk ve friend fonk kullanılabilir
// Der Base'in sanal fonksiyonlarını override edebilir
// Der Base'in protected bölümüne erişebilir
class Base { // private inheheritance
public:
	void foo();
};

class Der : private Base {
public:
	// using Base::foo;
	void foo()
	{
	Base::foo();
	}
};
----
```

--> Neden private kalıtım?
  - containment yoluyla composition'a alternatif

--> Neden composition değil de private inheritance?
  - taban sınıfın protected bölümüne erişmek
  - taban sınıfın sanal fonksiyonunu override etmek
  - kasıtlı olarak upcasting'den faydalanmak

EBO -> Empty Base Optimization
 - bazı durumlarda empty class türlerinden memberlar alındıgında
 bu durumuda empt class için 1 byte yer ayrılıyor,
 1 byte'da çogunulukla hizalama (alignment) nedeniyle,
 o elemana sahip olan sınıfın sizeof'unu işlemcinin kelime uzunlugu
 kadar arttırabilir. Buna alternatif olarak private kalıtım kullanıldıgında
 derleyiciye optimizasyon yapma imkanı verilir.

```
class Empty { // sizeof(empty) = 1
public:
	void func();
};

class Myclass { // sizeof(Myclass) = 8 //alignment
private:
	int mx; // mx olmasa sizeof = 1
	Empty e;
};
```

```
class Empty { // sizeof(empty) = 1
public:
	void func();
};

class Myclass : private Empty { // sizeof(Myclass) = 4
private:
	int mx;
};


int main()
{
	std::cout << "sizeof(Myclass) = " << sizeof(Myclass)<< "\n";
}
```
---

restricted polymorphism // kısıtlanmıs

```
class Base{
public:
	virtual void vfunc();
};

class Der : private Base{
public:
	void vfunc()override;
	friend void foo1();
};

void gfunc(Base& r)
{
	r.vfunc();
	//...
}

void foo1()
{
	Der der;
	gfunc(der); // gecerli
	//...
}

void foo2()
{
	Der der;
	gfunc(der); // syntax error
	//...
}
```

-- protected inheritance

```
class Base{
public:
	void func();
```

```
};

class Der : private Base{
public:
};

class Myclass : public Der {
	void bar()
	{
	func(); //syntax error
	}
};

---

class Base{
public:
	void func();
};

class Der : protected Base{
public:
};

class Myclass : public Der {
	void bar()
	{
	func(); // gecerli
	}
};
---
```

----- exception handling -----

- olağan dışı durumların işlenmesi

- run time hataları ikiye ayrılır
  - programming errors

assertion (doğrulama)
  - static assertion (compile time assertion)
        static_assert(sizeof(int) >= 4, "sizeof int must be greater or equal 4");
  - dynamic assertion (runtime assertion) -> assert(makro)

-> non forcing - sınamak zorunlu değil

try block -> nerde hata bekleniyorsa
catch block -> hataya mudale edicek kod
throw statement (exception throw)

a) terminative -> hata yakalanır, kontrollü sonlandırılır
b) resumptive -> hata yakalanır, program kayıp olmadan hizmet vermeye devam eder

std::terminate => abort cagırır
std::set_terminate -> terminate davranısını özelleştirir

using terminate_handler = void(*)()
typdef void(*terminate_handler)()

terminate_handler set_terminate(terminate_handler);
// terminate_handler function pointer type

```
void  f1()
{
        std::cout << "f1 cagırıldı\n";
        throw 1;
}

void myfunc()
{
        std::cout <<"myfunc cagirildi myfunc abort'u cagıracak\n";
        abort();
}
```

```
int main()
{
	std::set_terminate(myfunc);
	std::cout <<"main basladı\n";
	f1();
	std::cout <<"main sona erdi\n";


}
//output
main basladı
f1 cagırıldı
myfunc cagirildi myfunc abort'u cagıracak
Aborted (core dumped)
---
```

-> bir try blok olusturmak ile,
  bu blok içinde çalışan kodlardan gönderilecek
  hata nesnesini yakalamaya aday olur

```
throw expr;

auto exception_object = expr; // derleyicinin yazdiği kod



void f()
{
	int ival = 10;
	throw ival; // gönderilen ival' in kendisi değil
	// gönderilen nesne bu ifadenin türünden derleyicinin olusturdugu nesne
}



---
class Myclass {
};

void f()
{
	throw Myclass{}; // copy alligion
}
---
```

olusturulan nesneyle iletilen 2 ayrı bilgi var
  - nesnenin türü (hata neyle ilgili)
  -

```
class exception {
public:
  virtual const char* what()const;
};

try {

}
catch (exception& ex) {
  ex.what();
}
```

```
try {

}
catch (exception &ex) { // & yaparak is a ilişkisi

}
```

```
void f1()
{
	std::cout<<"f1()\n";
	std::string str{"veli"};
	auto c = str.at(20);
}

int main()
{
//amac sadece std kütüphanenin exception' ını yakalamk ise;
	try {
```

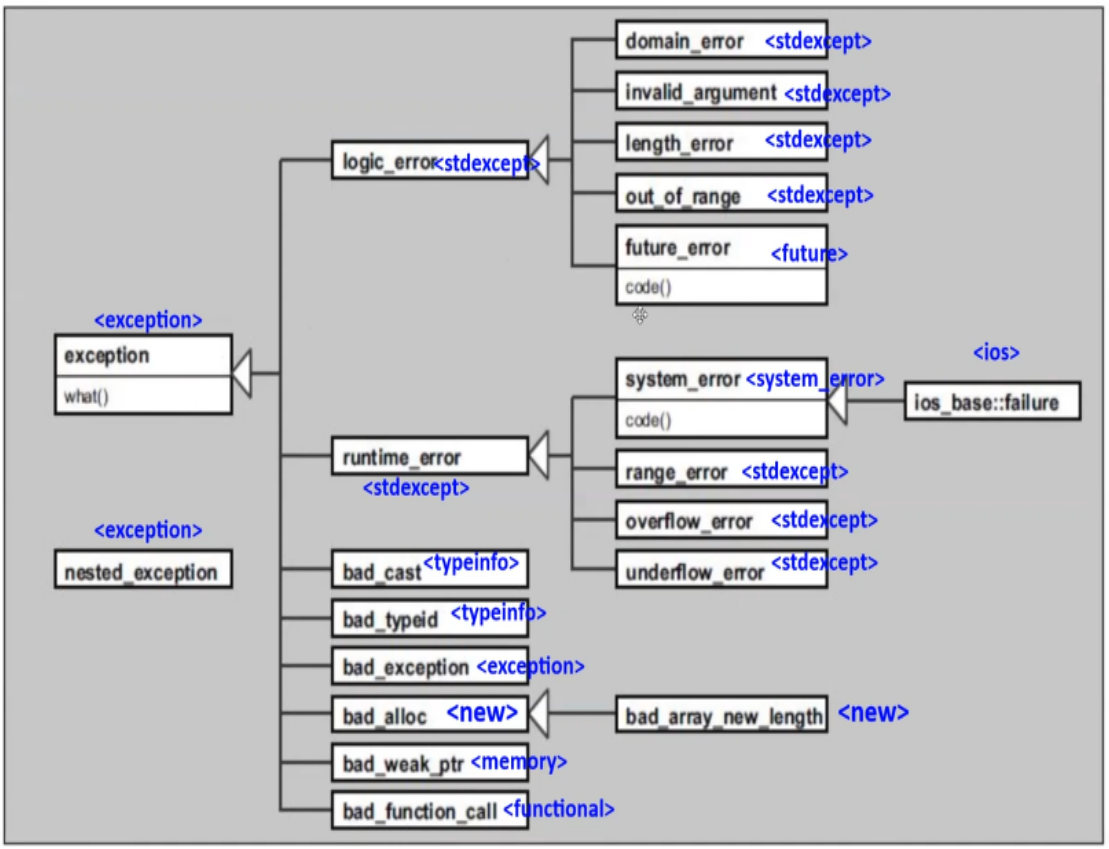
```
        f1();
        }
        catch(std::exception& ex) { //& olmazsa object slicing
        std::cout << "hata yakalandi: " << ex.what() << "\n";

        }
}
//output
f1()
hata yakalandi: basic_string::at: __n (which is 20) >= this->size() (which is 4)
---
```

-> catc(T ex) ise // & değilse
  1) copy ctor kullanılır
  2) polimorphic ise object slicing olustugundan,
        dinamik tür bilgisi kaybolur
  3) virtual dispatch devreye girmez

```
class Myexception {
public:
        Myexception()
        {
        std::cout << "default ctor\n";
        }

        ~Myexception()
        {
        std::cout << "destructor\n";
        }

        Myexception(const Myexception &)
        {
        std::cout << "copy ctor\n";
        }
};

void func()
{
        throw Myexception{}; // copy alligion
}
```

```
int main()
{
  try {
        func();
        }
        catch(Myexception &) {
        std::cout << "hata yakalandi\n";
        // hata nesnesinin hayati devam ediyor
        }
}
//output
default ctor
hata yakalandi
destructor
//
-> copy alligion, copy ctora cagrı yapılmadı
---




---
class Myexception {
public:
        Myexception()
        {
        std::cout << "default ctor\n";
        }

        ~Myexception()
        {
        std::cout << "destructor\n";
        }

        Myexception(const Myexception &)
        {
        std::cout << "copy ctor\n";
        }
};

void func()
{
        Myexception mx;
        throw mx;
}

int main()
```

```
{
  try {
        func();
        }
        catch(Myexception &) {
        // burdaki referans derleyicinin olusturdugu nesnenin yerine gecti/copy mx
        std::cout << "hata yakalandi\n";
        // hata nesnesinin hayati devam ediyor
        }
}
//output
default ctor
copy ctor
destructor
hata yakalandi
destructor
---
```




```
---
class bad_date : public std::logic_error {
public:
};
```

```
int main()
{
	throw bad_date{}; // syntax error / deleted func
}
---



---
class bad_date : public std::logic_error {
public:
	bad_date() : std::logic_error{"gecersiz tarih\n"}{}
};

int main()
{
	throw bad_date{};
}
//output
what():  gecersiz tarih
Aborted (core dumped)

---



---
class bad_date : public std::logic_error {
public:
	bad_date(const char *p) : std::logic_error{p}{}
};


int main()
{
	throw bad_date{"gecersiz"};
}
---



---
class bad_date : public std::logic_error {
public:
	using std::logic_error::logic_error;
};
```

```
int main()
{
	throw bad_date{"hata"}; // cagırılan taban sınıf ctoru
}
```
---

*
exception safety
  - uncode exception olmamalı
  - kaynak sızıntısı olmamalı
  - hayatı devam eden nesneler geçersiz hale gelmeyecek (invalid state)

-> basic guarantee: kabul edilebilir min garanti
	- programın durumu değişebilir
	- program devam edebilecek durumda kalacak
	- kaynak sızıntısı olmayacak
	- fonksiyon hiçbir nesneyi gecersiz durumda bırakmayacak

-> strong gurantee: commit or rollback (ya iş yapılmış olacak ya da fonk cagırılmadan önceki nesnenin state'i değişmez)
	- program durumu değişmeyecek
	- commit or rollback, ya işini gör ya da hiçbir şey yapılmamış durumda bırak
	- program devam edebilecek durumda kalacak

-> nothrow gurantee: En güçlü garanit, çalışma sırasında exception gönderilmez
	- fonksiyon işini yapma garantisi veriyor
	- hata gönderilirse kendi yakalayıp işini görücek

emit: exception'ının dısarı sızması
propagate: ancak onu cagıran fonkdan yakalanması
*

re-throw statment: yakalanan hata nesnesi neyse onu döndürür
```
  throw;
```

```
catch (...) { // catch all
}
```

```
catch(std::out_of_range &){
  //buraya girerse out_of_range türünden
}
```

```
catch(std::exception &){
  // exception türünden ama out_of_range değil
}

catch(...){
  // std kütüphanenin hata sınıfı türünden değil
  throw; // aynı hata nesnesini daha yukardaki kodalara gönderir
}
```

```
int main()
{
	try { //all code in main
	// tüm hataları yakalamaz
	// global değişkenler olabilir
	// global değişkenler veya sınıfın static elemanları
	//main cagırılmadan önce calısır
	// dolayısıyla bu kod bu try içerisinde değil

	}
	catch(...) {

	}

}
```

```
void func()
{
	try {
	throw std::out_of_range{"range hatasi"};
	}
	catch(const std::exception& ex) {
```

```
        std::cout<<"hata func içinde yakalandi : " << ex.what() << "\n";
        //iki farklı throw statment
        throw ex; // türü out_of_range değil, tepedeki sınıf türünden
        //throw;
        }
}


int main()
{
        try {
        func();
        }

        catch(const std::out_of_range&) {
        std::cout << "hata yakalandi std::out_of_range\n";
        }

        catch(const std::exception&) {
        std::cout << "hata yakalandi std::exception\n";
        }
}
//output
hata func içinde yakalandi : range hatasi
hata yakalandi std::exception
---



---
void func()
{
        try {
        throw std::out_of_range{"range hatasi"};
        }
        catch(const std::exception& ex) {
        std::cout<<"hata func içinde yakalandi : " << ex.what() << "\n";
        //iki farklı throw statment
        //throw ex;
        throw; // yeni nesne yaratılmaz, dinamik tür değişmez
        }
}


int main()
{
        try {
        func();
```

```
        }

        catch(const std::out_of_range&) {
        std::cout << "hata yakalandi std::out_of_range\n";
        }

        catch(const std::exception&) {
        std::cout << "hata yakalandi std::exception\n";
        }
}
//output
hata func içinde yakalandi : range hatasi
hata yakalandi std::out_of_range
---
```

-> terminate fonksiyonu, uncaught exception ve
eğer yakalanmış bir hata nesnesi yokken rethrow statment
yürütülürse terminate fonksiyonu çağırılır.

-> RAII (Resource Acquisition Is Initialization)

-> stack unwinding (yığının geri sarımı): This process of destroying local objects and calling destructors is called stack unwinding.

```
class Myclass {
public:
        Myclass()
        {
        std::cout << this << " adresinde nesne olustu\n";
        }
        ~Myclass()
        {
        std::cout << this << " adresindeki nesne için destructor çağirildi\n";
        }
};

void f4()
{
        Myclass x,y;
        throw 1;
}
```

```
void f3()
{
	Myclass x;
	f4();
}

void f2()
{
	Myclass x;
	f3();
}

void f1()
{
	Myclass x;
	f2();
}

int main()
{
	try {
	f1();
	}

	catch(double x) {
	std::cout << "hata yakalandi x = " << x << "\n";
	}
}
//output
0x7fffffffdc07 adresinde nesne olustu
0x7fffffffdbd7 adresinde nesne olustu
0x7fffffffdba7 adresinde nesne olustu
0x7fffffffdb76 adresinde nesne olustu
0x7fffffffdb77 adresinde nesne olustu
terminate called after throwing an instance of 'int'
---
// farklı main()
int main()
{
	try {
	f1();
	}

	catch(int x) {
	std::cout << "hata yakalandi x = " << x << "\n";
	}
}
```

```
//output
0x7fffffffdc07 adresinde nesne olustu
0x7fffffffdbd7 adresinde nesne olustu
0x7fffffffdba7 adresinde nesne olustu
0x7fffffffdb76 adresinde nesne olustu
0x7fffffffdb77 adresinde nesne olustu
0x7fffffffdb77 adresindeki nesne için destructor çağirildi
0x7fffffffdb76 adresindeki nesne için destructor çağirildi
0x7fffffffdba7 adresindeki nesne için destructor çağirildi
0x7fffffffdbd7 adresindeki nesne için destructor çağirildi
0x7fffffffdc07 adresindeki nesne için destructor çağirildi
hata yakalandi x = 1
---


-> exception yakalanırsa,
  otomatik ömürlü nesneler exception yakalandıgında
  stack unwinding sürecinde destroy edilir. // catch bloğuna girmeden




---
class Myclass {
public:
};

void foo(); // throw XXX

void func()
{
        Myclass* p = new Myclass;
        foo();
        delete p;
        // exception handling acısından büyük bir risk
        // sadece dinamik ömürlü nesneleri için değil, her tür kaynak için geçerli
        // smart pointer kullanılabilir
}
---




---

class Resource {
```

```
public:
	Resource(int x)
	{
	std::cout << this << " adresinde x = "<< x <<"nesne olustu\n";
	}
	~Resource()
	{
	std::cout << this << " adresindeki nesne için destructor çağirildi\n";
	}
};

void f4()
{
	std::cout << "f4 cagirildi\n";
	auto pr = new Resource(4);
	throw 1;
	delete pr;
}

void f3()
{
	std::cout << "f3 cagirildi\n";
	auto pr = new Resource(3);
	f4();
	delete pr;
}

void f2()
{
	std::cout << "f2 cagirildi\n";
	auto pr = new Resource(2);
	f3();
	delete pr;
}

void f1()
{
	std::cout << "f1 cagirildi\n";
	auto pr = new Resource(3);
	f2();
	delete pr;
}

int main()
{
	try {
	f1();
	}
```

```
        catch(int x) {
        std::cout << "hata yakalandi x = " << x << "\n";
        }
}
//output
f1 cagirildi
0x55555556e2c0 adresinde x = 3nesne olustu
f2 cagirildi
0x55555556e2e0 adresinde x = 2nesne olustu
f3 cagirildi
0x55555556e300 adresinde x = 3nesne olustu
f4 cagirildi
0x55555556e320 adresinde x = 4nesne olustu
hata yakalandi x = 1
// delete edilmedi
---
```

-- noexcept specifier ve noexcept operator
  -> iki ayrı işlev

void func()noexcept; // exception göndermeme garantisi
  void func()noexcept(true); // noexcept garantisi verir

void func()noexcept(false); // noexcept garantisi verilmez
  void func(); // aynı anlam

-> exception garantisi verip vermemeyi koşula bağlamış olur(conditional noexcept)

- operator noexcept

```
int main()
{
        int x = 10;
        constexpr auto b = noexcept(x++);
        std::cout << "x = " << x << "\n";
}
//output
```

10

-> destructor noexcept garantisi verir

```
class Person
{
public:
        Person (const char * pname) : name_{pname} {}

        std::string get_name()const
        {
        return name_;
        }

        Person (const Person& p) : name_{p.name_}
        {
        std::cout << "copy " << name_ << "\n";
        }

        Person(Person&& p) : name_{std::move(p.name_)}
        {
        std::cout << "move " << name_ << "\n";

        }

private:
        std::string name_;

};


int main()
{
  std::vector<Person> pvec{"cahit sitki taranci", "ahmet muhip diranas", "fazil hüsnü
daglarca"};
  std::cout << "\n";
  pvec.push_back("ahmet hamdi tanpınar");


}
//output
copy cahit sitki taranci
```

```
copy ahmet muhip diranas
copy fazil hüsnü daglarca

move ahmet hamdi tanpınar
copy cahit sitki taranci
copy ahmet muhip diranas
copy fazil hüsnü daglarca
---



---
class Person
{
public:
        Person (const char * pname) : name_{pname} {}

        std::string get_name()const
        {
        return name_;
        }

        Person (const Person& p) : name_{p.name_}
        {
        std::cout << "copy " << name_ << "\n";
        }

        Person(Person&& p) noexcept : name_{std::move(p.name_)}
        {
        std::cout << "move " << name_ << "\n";

        }

private:
        std::string name_;

};


int main()
{
  std::vector<Person> pvec{"cahit sitki taranci", "ahmet muhip diranas", "fazil hüsnü
daglarca"};
  std::cout << "\n";
  pvec.push_back("ahmet hamdi tanpınar");


}
//output
```

copy cahit sitki taranci
copy ahmet muhip diranas
copy fazil hüsnü daglarca

move ahmet hamdi tanpınar
move cahit sitki taranci
move ahmet muhip diranas
move fazil hüsnü daglarca

```
struct A{
	A()
	{
	std::cout << "A ctor kaynak edindi\n";
	}

	~A()
	{
	std::cout << "A destructor \n";

	}
};

class Myclass {
public:
	Myclass() : mp(new A)
	{
	std::cout << "Mylcass ctor\n";
	throw 1;  // bu durumda myclass hayata gelmemiş oluyor, fakat elemanları hayatta
	}
	~Myclass()
	{
	std::cout << "Myclass destructor\n";
	if(mp)
	delete mp;
	}
private:
	A* mp;
};
```

```
int main()
{
	try{
	Myclass m;
	}
	catch(int){
	std::cout << "hata yakalandı\n";
	}
}
//output
A ctor kaynak edindi
Mylcass ctor
hata yakalandı
// destructor cagırılmadı
---
-> çözüm için kaynak akıllı pointera bağlanabilir




---
class Myclass {
public:
	Myclass()
	{
	std::cout << "Mylcass ctor\n";
	throw 1;  // dinamik nesne yok, stack unwinding ile destructor cagırılır
	}
	~Myclass()
	{
	std::cout << "Myclass destructor\n";

	}
private:
	A mp;
};

int main()
{
	try{
	Myclass m;
	}
	catch(int){
	std::cout << "hata yakalandı\n";
	}
}
//output
A ctor kaynak edindi
```

```
Mylcass ctor
A destructor
hata yakalandı
```

--> eger ctor exception throw ederse fakat nesne dinamik oluşmuşsa,
destructor cagırılmasa da operator new tarafından elde edilen
sizeof(class) türü kadar bellek bloğu delete operatorun kodu çalışmasa da
derleyicinin ürettiği kodla operator delete çağırılacak.

-> eğer noexcept garantisi veren kod exception throw ederse
- bu derleme zamanı kontrolüne tabi değil
- std::terminate

```
void foo()
{
	throw 1;
}

void func()noexcept
{
	std::cout << "func cagırıldı\n";
	foo();
}

int main()
{
	std::set_terminate(&my_terminate);
	try {
	func();
	}
	catch(int){ // programın akışı catch bloguna girmeyecek
	std::cout <<"HATA\n";
	}
}
//output
func cagırıldı
```

```
std::terminate cagirildi...
myterminate cagrildi...
std::abort() cagrildi...
---


---
void foo()
{
	throw 1;
}

struct A {
	~A()
	{
	std::cout <<"A dtor\n";
	foo();
	}
};

void func()
{
	A ax;
}

int main()
{
	try{
	func();
	} // terminate cagırılır, catch bloguna girmez
	catch(int) {
	std::cout << "hata yakalandi\n";
	}
}
//output
A dtor
terminate called after throwing an instance of 'int'
---
```

-> destructor yazılsa da yazılmasa da noexcept

-> hata yakalaan otomatik ömürlü nesneler için destructor çağırılmakta,
fakat cagırılan destructor da exception throw ederse std::terminate

-> ya destructor'dan exception throw edilmeyecek veya destructor içinde yakalanıcak

** function try block

-> member initial list ile gönderilmiş nesne try catch ile yakalanmaz!

-> function try block tüm fonksiyonlar için kullanılabilir.
  Fakat kulanılma sebebi elemanların ctorlarından gönderilen exception'ı
  o elemana sahip sınıfın ctoru içinde yakalamak

```
void func()
try {
        //
}
catch(int) {

}
```

```
class A {
public:
        A(int) {
        throw 1;
        }
};

class Myclass {
public:
        Myclass() try : ax(10)
        {
        std::cout <<"Myclass ctor\n";
        }
        catch(int x) {
        std::cout << "hata yakalandı x = " << x << "\n";
        // derleyicinin yazdıgı kodla bu kod rethrow olur
        //throw;
        }

private:
        A ax;
};
```

```
int main()
{
	Myclass bx;
}
//output
hata yakalandı x = 1
terminate called after throwing an instance of 'int'
---
```

RTTI -> runtime type information

```
void car_game(Car* ptr)
{
	// buraya gelen araba acaba bir Mercedes mi?
	// eğer *ptr dinamik türü Auidi ise cam_ac()
}
```

operator
- dynamic_cast
- typeid

yardımcı sınıf
- std::type_info

--> Car sınıfından, Mercedes sınıfına dönüşüm down-casting denir
-> türemiş sınıftan taban sınıfa değil, taban sınıftan türemiş sınıfa doğru yapılan dönüşüm

```
---
Car* carptr;
//
```

```
Mercedes *p = dynamic_cast<Mercedes *>(carptr);
// Başarı ise hedef türden adres üretir
//  Mercedes değilse nullptr
---

if(Mercedes *p = dynamic_cast<Mercedes *>(carptr)) {
        }
---




---
class Base {
};

class Der : public Base {
};

void func (Base* baseptr)
{
        if(Der* derp = dynamic_cast<Der *>(baseptr)) { // compile time error
        // dynamic_cast must have a polymorphic class type
        }
}
---




---
void car_game(Car &carref)
{
        Fiat &fr = dynamic_cast<Fiat &>(carref);
        auto &fr = dynamic_cast<Fiat &>(carref);
        // fiat nesnesi değilse nullref olmadığından exception throw edilir
        // bad_cast
}
---
```

--> typeid

- operandı bir ifade (expression) olacak
- operandı birr tür olacak
- typeid operatoru type_info türünden nesneye eriştirir
- operandı polimorphic olmak zorunda değil

```
class Myclass {};
int main()
{
	int x = 10;
	double y = 45;
	std::cout << typeid(x).name() << "\n";
	std::cout << typeid(y).name() << "\n";
	std::cout << typeid(Myclass).name() << "\n";
}
//output //gcc
i
d
7Myclass
```

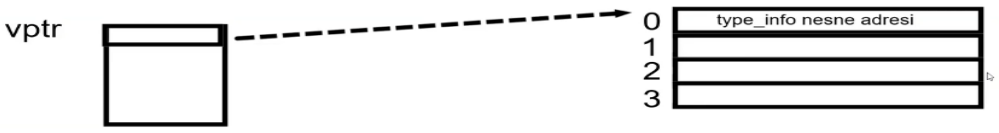
polymorphic değilse

```
int x = 10;
std::cout << typeid(x++).name() << "\n";
std::cout << x << "\n"; // x = 10
```

```
class Base {
public:
	virtual ~Base() {}
};

class Der : public Base {

};

void foo(Base& baseref)
{
```

```
        std::cout << typeid(baseref).name() << "\n";
        std::cout << (typeid(baseref) == typeid(Der)) << "\n";
}

int main()
{
        Der myder;
        foo(myder);
}
//output
class Der
1 //true
---
```

```
---
void car_game(Car* ptr)
{
        ptr->start();

        if(typeid(*ptr) == typeid(Fiat)) {
        // her fiat124 fiat olmasına rağmen if'in doğru kısmına girmez
        // sadece Fiat
        static_cast<Fiat*>(ptr)->activate_aebs();
        }
        //...
}
---
```



--> ilgili type_info nesnesine erişildiğinde,
  türemiş sınıflarına bakma mecburiyeti yok
  - Fiat

--> dynamic_cast, ilgili sınıftan türetilen türlere de
  bakmak zorunda
  - Fiat -> Fiat124 ...

----- generic programming -----

```
void gswap(void* vp1, void* vp2, size_t n)
{
        char* p1 = (char*)vp1;
        char* p2 = (char*)vp2;

        while (n--) {
        char temp = *p1;
        *p1++ = *p2;
        *p2++ = temp;
        }
}

void* greverse(void* vpa, size_t size, size_t sz)
{
        char* p = (char*)vpa;
        for(int i = 0; i < size/2; ++i) {
        gswap(p + i * sz, p + (size -1 -i) * sz, sz);
        }
}
```

-> derleyiciye kod yazdıran şablonlar
  --> function templates
  --> class templates
  --> alias template (eş isim) c++11
  --> variable template (değişken şablonlar) c++11
  --> concept c++20

template <>
// type parameter
  - template <typename T>
// non-type parameter
  - template <int x>
-> template<typename T, int size>
// template template

-> tür parametresi için 2 anahtar sözcük var
// class
// typename

// c++17 ile,

--> CTAD = Class template argument deduction

// Derleyici türü nasıl anlar?
  1- deduction

  2- explicit template argumant

```
template<typename T>
void func(T x)
{

}
int main()
{
```

```
        func<int>(10);
        }
```

3- default template argumant

```
        template<typename T, typename U = int>
```

```
// template argument deduction
  bir istisna haricinde auto type deduction ile ayni
```

```
template <typename T>
void func(T x);

int main()
{
   func(10);
   auto y = 10; // kurallar aynı
}
```

```
template <typename T>
class TypeTeller;

template <typename T>
void func(T)
{
        TypeTeller<T> x;
}

int main()
{
// örnekler auto ile aynı
   int i{3};
   const int& r = i; // const ve & düşer
   func(r); // T = int

        int a[] {5,4,8};
```

```
    func(a); // T = int*

    const int b[] {5,4,8};
    func(b); // T = const int*

    func("ali"); // T = const char*

            func(foo); // T = int (*)(int)

}
---



---
template <typename T>
void func(T &)
{
            TypeTeller<T> x;
}

int foo(int);

int main()
{
    const int x = 10;
    func(x); // T = const int

    int a[5]{};
    func(a); // T = int [5]
    // void func(int (&r)[5]) // derleyicinin yazdığı fonks

    const int b[5]{};
    func(b); // T = const int [5]

    func("ali"); // T = const char [4]
    // array decay olmaz

            func(foo); // T = int(int) // function type
            // geri dönüş türü ve parametresi int olan fonk türü
            // fonk parametre değişken türü ise int(&)(int)
}
---




---
template <typename T>
```

```
void func(T &&) // forwarding reference - universal reference
// sağ taraf parametresi değil
{
	TypeTeller<T> x;
}

void foo(int &&); // parametre pr value
---




// reference to reference oluşursa,
// olusturulan türün ne olduğunu belirleyen kurallara,
// reference collapsing
	sonuç:
  T&   &  T&
  T&   && T&
  T&&  &  T&
  T&&  && T&&


---
int main()
{
	using MRef = Myclass&;
	Myclass mx;
	MRef &x = mx; // x type = Myclass &
	MRef &&x = mx; // x type = Myclass &

	using MR = Myclass&&;
	Myclass my;
	MR &&y = my;// an rvalue reference cannot be bound to an lvalue

	using Type = Myclass&;
	using A = Type; // A = Myclass&
}
---




---
template<typename T>
void func(T &&)
{
```

```
}

int main()
{
	func(10); // R value expr
	// T = int
	// fonk parametre türü int &&

	int x = 10;
	func(x); // T = int &
	//fonk parametresi sol taraf ref oldugundan,
	fonk parametresi int & türünden //ref collapsing
}
---



-- başarısız olma durumları :
  1- derleyici tür çıkarımı yapamaz
---
template<typename T>
void func(T )
{
}

int main()
{
	foo();
}
---


  2- ambiguitiy ()
---
template<typename T>
void func(T x, T y);

int main()
{   // geçerli
	func(1, 2);
	func("ali", "veli");
	func(3.4, 2.4);

	func(2, 3.4); // syntax error

}
---
```

```
---
template<typename T>
void func(T &x, T &y); // &

int main()
{   // geçerli
        func("ali", "can"); // T& oldugundan array decay olmaz,
        // T = const char[4]


        func("ali", "veli"); // syntax error
        // const char[4] const char[5]

        int a[5]{}, b[5]{}, c[4]{};

        func(a, b); // gecerli
        func(a, c) // syntax error
}
---




---
template<typename T>
void func(T &&x, T &y);

int main()
{
        int i = 10;
        func(i, i); // syntax error
        // ilk T = int & olur
        // ikinci T int
}
---




---

template<typename T>
void func(T &&x, T y);

int main()
{
        func(12, 34); // T = int // gecerli
```

```
}
```

--> PARAMETRE TÜRÜNÜN DEĞİL T' NİN TÜRÜNÜN AYNI OLMASI GEREKİR.

```
template<typename T, int size>
void func(T(&)[size]);

int main()
{
	int a[10];
	func(a); // T = int

	//derleyicinin yazdıgı fonk parametresi:
	void func(int(&r)[10])
}
```

```
template<typename T>
void func(T** p);

int main()
{
	int x{};
	int* ptr{&x};
	int**p{&ptr};

	// void func(T** p) ise;
	func(p); // T = int

	// void func(T* p) ise ;
	func(p); // T = int*

	// void func(T p) ise ;
	func(p); // T = int**
}
```

```
template<typename T, int n>
constexpr int Size(const T(&)[n])
{
	return n;
}

int main()
{
	int a[10];
	float b[]{1.f, 2.f};

	constexpr auto size_a = Size(a); // 10
	constexpr auto size_b = Size(b); //2
}
```

```
template<typename T, typename U>
void func(T(*)(U))
{

}

int foo(double);

int main()
{
	func(foo);
	// T = int
	// U = double
}
```

--- mülakat sorusu
```
class Person{};
int foo();
Person func();

int main()
```

```
{
	int foo() = 10; //syntax error

	Person per;
	func() = per; // gecerli
	//func().operator=(per);
}
---
```

```
--- auto ve template arasındaki tek fark
template <typename T>
void f(T);

int main()
{
	auto x = {1,2,3,4};
	f({1,2,3,4}); // syntax error
}
---
```

```
--- ms
void foo(int&)
{
	std::cout<< "int &\n";
}

void foo(int&&)
{
	std::cout<< "int &&\n";
}

void func(int&& r)
{
	foo(r);
}

int main()
{
	func(12); // int &
	// r L value
```

```
        int&& x = 10;
        int& r = x; // geçerli
        // x ifadesi l value
}
---
```

```
---
template<typename T>
void Swap(T& x, T& y)
{
        T temp = std::move(x);
        x = std::move(y);
        y = std::move(temp);


        //-----------------------------------
        // string' de gereksiz kopyalama oluşur
        T temp = x; // copy
        x = y; // copy assign
        y = temp; // copy assign
        //-----------------------------------
}
---
```

```
// fonksiyon şabonları ile,
// gerçek fonksiyonlar birbirini overload edebilir
template<typename T>
void func(T x);
void func(int x);
```

```
---
template<typename T>
void func(T x) = delete;
void func(int x);
// parametreye sadece tam sayı girilebilir
```

```
int main()
{
	func('a'); // syntax error
	func(2.3); // syntax error
	func(6);
}
---
```

```
---
// partial ordering rules of templates
// daha niteleyici olan seçilir
template<typename T>
void func(T x)
{
	std::cout << "1";
}

template<typename T>
void func(T* x)
{
	std::cout << "2";
}

int main()
{
	int x;
	func(&x); // 2
}
---
```

```
--- // 1. yol
template<typename Result, typename T, typename U>
Result sum(T x, U y)
{
	return x + y;
}

int main()
```

```
{
	auto y = sum<double>(12, 4.5);
}
---
```

--> trailing return type

```
auto foo()->int // geri dönüs türü
{
	//
	return 1;
}
```

```
--- // 2. yol
template<typename T, typename U>
auto sum(T x, U y)  -> decltype(x + y)
{
	return x + y;
}

int main()
{
	auto y = sum(12, 4.5);
}
---
```

--> auto return type

```
auto foo(int x, double y)
{
	return x * y + 1;
}
```

-> decltype' in operandı olan ifade
  a) PR value expression ise elde edilen tür T türü
  b) L value expression ise elde edilen tür T& türü

c) X value expression ise elde edilen tür T&& türü

```
int&& foo();
int main()
{
	int x = 10;
	decltype(foo()) y = 10; // y türü sağ taraf ref
	//foo X value expr
}
```

```
template<typename T>
decltype(auto) foo(T x)
{
	return (x); // decltype' nın operandı gibi davranır
}

int main()
{
	foo(12); // elde edilen tür int &
}
```

```
int foo()
{
	const int x = 10;
	return x;
}
int main()
{
	foo(); // fonk çağı ifadesinin türe int
}
```

```
decltype(auto) foo()
{
```

```
	const int x = 10;
	return x;
}
int main()
{
	foo(); // fonk çağı ifadesinin türe const int
}
---
```

```
template <typename T>
typename T::value_type func(T x, int y);
// geri dönüş değeri T' nin nested type
// eğer template tür parametresina bağlı bir türse,
// çözünürlüklük oprt ile niteleniyorsa typname anahtar sözcüğü
// kullanmak zorunlu
```

```
// SFINAE
// substitution failure is not an error

// substitution aşamasında bir türün geçerli olmaması sebebiyle
// syntax hatası olursa,
// derleyici syntax hatası vermez, overload resolation' dan çıkar
template <typename T>
typename T::Myclass func(T x);

void func(double); // bu fonk çağırılır

int main()
{
	func(12);
}
```

STL

containers (class templates)
iterators (class templates)
algorithms (function templates)

intantiate -> bir şablondan derleyicden gercek kodu yazmasına,
derleyicinin o şablonu instantiate etmesi denir.

specialization -> derleyicinin yazdıgı fonksiyona
template specialization denir

--- class template ---

```
template <typename T>
class Myclass {
};
// Dikkat Myclass bir sınıf değil
// Myclass<int> bir sınıf.
// Myclass<double> ayrı bir sınıf.
// her birine bu sınıfın specialization'ı denir
```

```
template <typename T>
class Myclass {
public:
        T f(T x);
        int foo(const T&);
```

```
private:
	T mx;
};
```

```
template <typename T>
class Myclass {
public:
	T foo(T x);
	T func(T x, T y);
private:
	T mx;
};

template<typename T>
T Myclass<T>::foo(T x)
{
	//
}

template<typename T>
T Myclass<T>::func(T x, T y)
{
	//
}
```

```
template <typename T>
class Myclass {
public:
	Myclass foo(T x);
private:
	T mx;
};

template<typename T>
Myclass<T> Myclass<T>::foo(T x)
{
```

```
	//
}
---
```

```
---
template <int n>
class Myclass {
};

int main()
{
	Myclass<3> mx;
	Myclass<5> my;
	my = mx; // syntax error
	// farklı sınıf türleri
}
---
```

```
---
template <typename T>
class Myclass {
};

template <typename T>
bool operator==(const Myclass<T>& left, const Myclass<T>& right);

int main()
{
	Myclass<double> x,y;
	x == y;
}
---
```

```
---
template <typename T>
class Counter {
```

```
public:
	Counter() = default;
	Counter(T val) : mval{val} {}

	T get()const
	{
	return mval;
	}
private:
	T mval{};
};

template <typename T>
std::ostream& operator<<(std::ostream& os, const Counter<T>& c)
{
	return os << "(" << c.get() << ")";
}

int main()
{
	Counter<long> cnt{15};
	Counter<std::string> tnt{"ali"};
	std::cout << cnt << "\n";
	std::cout << tnt << "\n";
}
---
```

```
---
template <typename T>
struct Myclass
{
	Myclass()
	{
	std::cout << typeid(*this).name() << "\n";
	}
};

int main()
{
	Myclass<int> x; // struct Myclass<int>
	Myclass<Myclass<int>> y; // struct Myclass<struct Myclass<int>>
}
---
```

```
template <typename T, size_t n>
struct Array
{
	T a[n];
};

int main()
{
	Array<int, 20> ar; // sınıf nesneni olarak kullanılır
	// maliyeti yok

	// int x[20];
}
```

--> pair template

--- c' de birden fazla dönüş değeri
```
struct Data
{
	int x;
	float y;
};

Data f(); // int ve float döner
```

--- c++' da
```
template <typename T, typename U>
struct Pair
{
	T first;
	U second;
};

std::pair<int, long> foo(); // bir int bir long döner
std::pair<double, int> func();
```

```
int main()
{
	std::pair<int, double> px;
	px.first = 10;
	px.second = 1.3;
}
```

```
// default template argument
template<typename T = int>
class Myclass {

};

int main()
{
	Myclass<double> x;
	Myclass<> y; // int
}
```

```
template<typename T, typename A = std::allocator<T>>
class Vector {
//
};

int main()
{
	Vector<int> vx;
}
```

```
class Myclass{};

template<typename T, typename U = Myclass>
void func(T x, U = U{}); // U = default ctor
```

--- member templates ---

```
template <typename T>
class Myclass {
public:
	void func(T x)
	{
	std::cout << x;
	}
};

int main()
{
	Myclass<int> mx; // T = int
	mx.func(1.3); // 1
}
```

```
template <typename T>
class Myclass {
public:
	void func(T&& x); // R value referans
	// Myclass<int> x;
	// forwarding & olması için type deduction olmalı
	// fonksiyona çağrı yapıldığında tür belli
};


void foo(std::vector<int>&&); // forwarding ref değil
// R value ref

void f1(std::vector<T>&&) // R value

void f2(const T &&); // R value
```

- int açılımı türünten bir sınıfın double türünden sınıf parametresi
  nasıl yapılır ?

--- soru

```
template <typename T>
class Myclass {
public:
	void func(Myclass);
};

int main()
{
	Myclass<int> mx;
	Myclass<double> my;
	mx.func(my); // syntax error
}
---


--- çözüm
template <typename T>
class Myclass {
public:
	template<typename U>
	void func(Myclass<U> x)
	{
	std::cout << typeid(*this).name() << "\n";
	std::cout << typeid(x).name() << "\n";
	}
};

int main()
{
	Myclass<int> mx;
	Myclass<double> my;
	mx.func(my); // geçerli
}
---


--- Pair class---


---
template <typename T, typename U>
struct Pair {
	Pair()=default;
	Pair(const T& t, const U& u) : first(t), second(u){}

	template<typename K, typename M>
	Pair(const Pair<K, M> &other) : first(other.first), second(other.second) {}
```

```
        T first{};
        U second{};// pointer nullptr ile sınıflar default ctor ile başlar
};


// karsılatırma yapıldıgında first'ü kücük olan kücüktür
// firstler eşit ise second kucuk olan kucuktur
template <typename T, typename U>
bool operator<(const Pair<T, U>& left, const Pair<T, U>& right)
{
        return left.first < right.first || !(right.first < left.first) && left.second < right.second;
        // (a<b) || !(a<b) --> = ise
}
---



---
template <typename T, typename U>
Pair<T, U> MakePair(const T& t, const U& u)
{
        return Pair<T, U>(t, u);
}

int main()
{
        auto p = MakePair(12, 4.5); // <int, double>
}
---




---
int main()
{
        auto x = std::make_pair(12, 3.5);
        auto y = std::make_pair(x, 3L); //  pair<pair<int, double>, long>
}
---



---
template <typename T, typename U>
std::ostream& operator<<(std::ostream& os, const std::pair<T, U>& p)
{
        return os << "[" << p.first <<", " << p.second <<"]";
}
```

```
int main()
{
	std::cout << std::make_pair(std::make_pair(12, 4.5), std::bitset<32>{25u}) <<
std::endl;
}
// output
[[12, 4.5], 00000000000000000000000000011001]
---
```

--> template explicit specialization

```
// template explicit specialization
// template full specialization
// class template ve function template için
// function template overloading oldugundan yanıltabilir
// template <>
```

```
template <typename>
class Myclass{

};

template <>
class Myclass<int> // yukardaki template'e explicit specialization olustu
{

};
```

```
---
template <typename> // primary template
class Myclass{
public:
```

```
        Myclass()
        {
        std::cout << typeid(*this).name() <<"\n";
        }
};

template <>
class Myclass<int> // yukardaki template'e explicit specialization olustu
{
public:
        Myclass()
        {
        std::cout << "explici specialization for int\n";
        }
};

int main()
{
        Myclass<char> cx;
        Myclass<double> dx;
        Myclass<int> ix;
}
//output
7MyclassIcE
7MyclassIdE
explici specialization for int
---




--- mülakat sorusu
// döngü kullanmadan 0-100 arasındaki sayıları yazdırma
// çüzüm 1

struct A
{
        A()
        {
        static int x = 0;
        std::cout << x++ << " ";
        }
};

int main()
{
        A a[100];
```

```
}
---


---
// döngü kullanmadan 0-100 arasındaki sayıları yazdırma
// çüzüm 2

template<int n>
struct A : A<n - 1>
{
	A()
	{
	std::cout << n << " ";
	}
};

template<>
struct A<0>
{
};


int main()
{
	A<100> ax;
}
---




---
// factorial
// compile time'da hesaplar

template<int n>
struct Factorial
{
	const static int value = n * Factorial<n - 1>::value;
};

template<>
struct Factorial<0>
{
	const static int value = 1;
};
```

```
int main()
{
	int a[Factorial<6>::value]; // int a[720]
	// compile time da hesaplar
}
```

```
// function specialization.

template<typename T>
void func(T x)
{
	//
}

// Dikkat!
explicit specialization function overoad set' e dahil etmez

template<>
void func(int x)
{
	//
}
```

```
template<typename T>
void func(T x)
{
	std::cout << "1";
}

// explicit specialization function overoad set' e dahil etmez
template<>
void func(int *x)
```

```
{
	std::cout << "2";
}


template<typename T>
void func(T *x)
{
	std::cout << "3";
}

int main()
{
	int x{};
	func(&x);
}
// output
3
---

---
template<typename T>
void func(T x)
{
	std::cout << "1";
}


template<typename T>
void func(T *x)
{
	std::cout << "3";
} // bu fonk seçildiğinden specialization'i çağırır


template<>
void func(int *x)
{
	std::cout << "2";
}

int main()
{
	int x{};
	func(&x);
}
// output
2
```

--> partial specialization of templates

```
template<typename T>
class Myclass {
public:
	Myclass()
	{
	std::cout << typeid(*this).name() <<"\n";
	}
};

template<typename T>
class Myclass<T *> // T * (pointer) türleri için bu template kullanılacak
{
public:
	Myclass()
	{
	std::cout << "partial specialization for T *\n";
	}
};


int main()
{
	Myclass<int *> m1;
	Myclass<char> m2;
	Myclass<long> m3;
	Myclass<char *> m4;
	Myclass<int *> m5;
}
// output
partial specialization for T *
7MyclassIcE
7MyclassIlE
partial specialization for T *
```

partial specialization for T *

```
template<typename T>
class Myclass {
public:
	Myclass()
	{
	std::cout << typeid(*this).name() <<"\n";
	}
};

template<typename T, typename U>
class Myclass<std::pair<T, U> >
{
public:
	Myclass()
	{
	std::cout << "partial specialization for std::pair<T, U>\n";
	}
};


int main()
{
	Myclass<int *> m1;
	Myclass<char> m2;
	Myclass<std::pair<int,double>> m3;
}
// output
7MyclassIPiE
7MyclassIcE
partial specialization for std::pair<T, U>
```

--- perfect forwarding ---

--> bir fonksiyonun aldığı argümanlarının const ve value
  kategorisini değiştirmeden başka bir fonksiyona geçmesine
  perfec forwarding denir

```
class Myclass{};

void foo(Myclass &)
{
	std::cout << "Myclass &\n";
}

void foo(const Myclass &)
{
	std::cout << "const Myclass &\n";
}

void foo(Myclass &&)
{
	std::cout << "Myclass &&\n";
}

template <typename T>
void func(T x) // perfec forwarding yok
{
	foo(x);
}

int main()
{
	Myclass m1;
	const Myclass m2;

	foo(m1);
	foo(m2);
	foo(Myclass{});
	std::cout << "\n";

	func(m1);
	func(m2);
	func(Myclass{});
```

```
}
// output
Myclass &
const Myclass &
Myclass &&

Myclass &
Myclass &
Myclass &
---




---
class Myclass{};

void foo(Myclass &)
{
	std::cout << "Myclass &\n";
}

void foo(const Myclass &)
{
	std::cout << "const Myclass &\n";
}

void foo(Myclass &&)
{
	std::cout << "Myclass &&\n";
}

template <typename T>
void func(T&& x)
{
	foo(std::forward<T>(x)); // argümanın const ve value category korur
}

int main()
{
	Myclass m1;
	const Myclass m2;

	func(m1);
	func(m2);
	func(Myclass{});
}
```

```
// output
Myclass &
const Myclass &
Myclass &&
---
```

```
---
class Myclass{};

void foo(Myclass&, int &&, const double &)
{
}

template <typename T, typename U, typename K>
void func(T&& t, U&& u, K&& k)
{
        foo(std::forward<T>(t), std::forward<U>(u), std::forward<K>(k));
}
---
```

```
--- alias template ---
```

```
--
template <typename T>
using iptr = T*;

int main()
{
  iptr<int> ptr = nullptr;
}
--
```

```
---
```

```
template <typename T, size_t n>
using Array = T[n];

int main()
{
	Array<int, 20> ar;
	int a[20];
}
---




---
template <typename T>
using eqpair = std::pari<T, T>;

int main()
{
	eqpair<int> x;
	// std::pair<int, int> x;
}
---




---
template <typename T>
struct Myclass {
	typedef int type;
};

template <typename T>
using mytype = typename Myclass<T>::type;


int main()
{
	mytype<double> x;
}
---



---
template <typename T>
using gmap = std::map<T, std::greater<T>>;
```

```
int main()
{
	gmap<double> x;
}
--
```

--> variables template

```
template <typename T>
struct A {
	//
	constexpr static bool value = true;
};

template <typename T>
constexpr bool A_v = A<T>::value;


int main()
{
	//A<int>::value;
	A_v<int> ;
}
```

--> meta function: compile time' da değer üreten yapılardır
- derleme zamaınında tür bilgisi elde eden meta fonksiyonlar
- derleme zamaınında sabit elde eden meta fonksiyonlar

```
---
template <typename T>
struct Remove_Reference { // referans almaz
	using type = T;
};
```

```
template <typename T>
struct Remove_Reference<T&> { // partial spec.
	using type = T;
};


template <typename T>
struct Remove_Reference<T&&> { // partial spec.
	using type = T;
};

template <typename T> // template alias
using Remove_Reference_t = typename Remove_Reference<T>::type;

int main()
{

	Remove_Reference<int&>::type x; // x = int
	Remove_Reference_t<int&&> y; // y = int
}
---
```

```
---
// değişken türünün void olup olmadığını compile time'da anlama

template <bool x>
struct bool_constant {
	static constexpr bool value = x;
};

template <typename T>
struct is_void : bool_constant <false> {};

template <>
struct is_void<void> : bool_constant <true> {};

template <>
struct is_void<const void> : bool_constant <true> {};

template <>
struct is_void<volatile void> : bool_constant <true> {};
```

```
template <>
struct is_void<const volatile void> : bool_constant <true> {};

template <typename T>
constexpr bool is_void_v = is_void<T>::value;

int main()
{
	is_void<int>::value; // false
	is_void<int *>::value; // false
	is_void<long >::value; // false
	is_void<void>::value; // true

	is_void_v<const void>; // true
	is_void_v<const int>; // false
	is_void_v<char>; // false
}
---
```

```
--- #include <type_traits> ---

namespace std {
  // helper class:
  template <class T, T v> struct integral_constant;

  template <bool B>
  using bool_constant = integral_constant<bool, B>;
  using true_type = bool_constant<true>;
  using false_type = bool_constant<false>;

  // primary type categories:
  template <class T> struct is_void;
  template <class T> struct is_null_pointer;
  template <class T> struct is_integral;
  template <class T> struct is_floating_point;
  template <class T> struct is_array;
  template <class T> struct is_pointer;
  template <class T> struct is_lvalue_reference;
  template <class T> struct is_rvalue_reference;
  template <class T> struct is_member_object_pointer;
  template <class T> struct is_member_function_pointer;
  template <class T> struct is_enum;
  template <class T> struct is_union;
```

```
  template <class T> struct is_class;
  template <class T> struct is_function;

  // composite type categories:
  template <class T> struct is_reference;
  template <class T> struct is_arithmetic;
  template <class T> struct is_fundamental;
  template <class T> struct is_object;
  template <class T> struct is_scalar;
  template <class T> struct is_compound;
  template <class T> struct is_member_pointer;

  // type properties:
  template <class T> struct is_const;
  template <class T> struct is_volatile;
  template <class T> struct is_trivial;
  template <class T> struct is_trivially_copyable;
  template <class T> struct is_standard_layout;
  template <class T> struct is_empty;
  template <class T> struct is_polymorphic;
  template <class T> struct is_abstract;
  template <class T> struct is_final;
  template <class T> struct is_aggregate;

  template <class T> struct is_signed;
  template <class T> struct is_unsigned;
  template <class T> struct is_bounded_array;
  template <class T> struct is_unbounded_array;
  template <class T> struct is_scoped_enum;
  // ...
```

--> std::integral_constant

```
template<class T, T v>
struct integral_constant {
        static constexpr T value = v;
        using value_type = T;
        using type = integral_constant; // using injected-class-name
        constexpr operator value_type() const noexcept { return value; }
        constexpr value_type operator()() const noexcept { return value; } // since c++14
};
```

```
using true_type = integral_constant<bool, true>;
using false_type = integral_constant<bool, false>;

template <typename T>
struct is_void : false_type {};

template<> struct is_void<void> : true_type{};
template<> struct is_void<const void> : true_type{};

// variable template
template <typename T>
constexpr bool is_void_v = is_void<T>::value;
```

```
// is_same tür ilişkisi trait' inin gerçekleştirimi

template<class T, class U>
struct IsSame : std::false_type{};

template <class T>
struct IsSame<T, T> : std::true_type{};

template<typename T, typename U>
inline constexpr bool IsSame_v = IsSame<T, U>::value;

int main()
{
	std::cout << IsSame_v<int, int> << "\n"; // true
	std::cout << IsSame_v<int, const int> << "\n"; // false
	std::cout << IsSame_v<unsigned, int> << "\n"; // false
}
```

```
// tag dispatch
// tam sayi için ayrı kod
```

```
template <typename T>
void func_impl(T x, std::true_type)
{
	std::cout << "tam sayi türleri icin olan kod\n";
}


template <typename T>
void func_impl(T x, std::false_type)
{
	std::cout << "tam sayi sayi türleri olmayanlar için kod\n";
}

template<typename T>
void func(T x)
{
	func_impl(x, std::is_integral<T>::type());
}

int main()
{
	func(12); // tam sayi ..
}

---




---
// pointer mı?

template <typename T>
struct ispointer : std::false_type{};

template <typename T> // partial spec.
struct ispointer<T *> : std::true_type{};

template <typename T> // partial spec.
struct ispointer<const T *> : std::true_type{};

template <typename T> // partial spec.
struct ispointer<volatile T *> : std::true_type{};

template <typename T> // partial spec.
struct ispointer<const volatile T *> : std::true_type{};
```

```
int main()
{
	ispointer<const int *>::value; // ture
}
```

```
// if constexpr

template <typename T>
void func(T x)
{
	if constexpr (std::is_integral_v<T>) {
	//
	}
	else if constexpr (std::is_floating_point_v<T>) {

	}
	else {

	}
}
```

```
template <typename T>
auto func(T x)
{
	if constexpr (std::is_pointer<T>::value)
	return *x;
	else
	return x;
}

int main()
{
	int x = 5;
	int *ptr = & x;
	std::cout << func(1) << "\n"; // 1
```

```
	std::cout << func(x)<< "\n"; // 5
	std::cout << func(*ptr)<< "\n"; // 5
	std::cout << func(ptr)<< "\n"; // 5
}
---




—
// static_assert

template <typename T>
auto func(T x)
{
	static_assert(std::is_integral<T>::value,"template argument must be of an integral
type\n");
	//
}

int main()
{
	func(10); // gecerli
	func('A'); // gecerli
	func(5.3); // template argument must be of an integral type
}
—







---
--> variadic template

// template parameter pack
template<typename ...Args>
void func(Args ...args) { // function paramater pack

}

// derleyicinin yazdığı
void func(int p1, double p2, long p3);

int main()
{
	func(1, 3.5, 4L);
```

```
}
---



---
template<typename ...Args>
void func(Args &...args); // parametreleri referans
void func(Args &&...args); // parametreleri perfect fowarding
---


---
template<typename ...Args>
void func(Args ...args) {
	std::cout << sizeof...(Args) << "\n"; // 3
	std::cout << sizeof...(args) << "\n"; // 3
}

int main()
{
	func(1, 3.5, 4L);
}
---



---

template<typename ...Args>
void func(Args ...args) {
  if constexpr (sizeof...(args) == 1){
	std::cout << "1 arguman\n";
  }
}

int main()
{
	func(1); // 1 arguman
}
---



---

template<typename T, typename ...Args>
void func(T x, Args &&...args) {
```

```
        f(std::forward<Args>(args)...);
        // f(std::forward<int>(p1), std::forward<double>(p2), std::forward<long>(p3));
}
---
```

```
---
class Myclass {
public:
        Myclass(int&, int, double &&);
};

template<typename ...Args>
void func(Args &&...args) {
        // aldığı argümanların value kategorsini ve constlugunu değiştirmez
        Myclass x{std::forward<Args>(args)...};
}

int main()
{
        int ival{10};
        func(ival, 20, 3,4);
}
---
```

```
args...             f(p1, p2, p3)   //f(args..)
f(args)...   f(p1), f(p2), f(p3)
f1(f2(args))...   f1(f2(p1)), f1(f2(p2))
```

```
---
class Myclass {
public:
        Myclass(int a, int b)
        {
        std::cout << a << "\n" << b << "\n";
        }

};
```

```
int main()
{
	auto prt = std::make_unique<Myclass>(10, 7); // perfect fowarding ile new Myclass{..}
	// make_unique variadic
	//std::unique_ptr<Myclass> uptr(new Myclass{1, 5});
}
---
```

```
--- make_unique
class Myclass {
public:
	Myclass(int a, int b)
	{
	std::cout << a << "\n" << b << "\n";
	}

};

template <typename T, typename ...Args>
std::unique_ptr<T> MakeUnique(Args&& ...args)// perfect f.
{
	return std::unique_ptr<T>(new T(std::forward<Args>(args)...));
}


int main()
{
	auto uptr = MakeUnique<Myclass>(1, 25); // myclass(int, int) ctor cagırılır
}
---
```

-> variadic template için 3 farklı teknik var
- compile time'da recursive (eksiltme)
- küme parantezi içinde bir liste
- if constexpr

```
// 1. teknik
template<typename T>
void print(const T& t) // döngüyü durduran şablon
{
	std::cout << t << "\n";
}

template<typename T, typename ...Args>
void print(const T& t, const Args&... args) // recursive şablon
{
	std::cout << t << "\n";
	print(args...);
}

int main()
{
	std::string name{"veli"};
	int ival{15};
	print(10, name, ival);
}
// output
10
veli
15
```

```
template<typename T>
T summer(T v) // döngüyü durduran şablon
{
	return v;
}

template<typename T, typename ...Args>
T summer(const T& first, const Args&... args) // recursive şablon
{
	return first + summer(args...);
}
```

```
int main()
{
	std::cout << summer(10, 20, 3, 50); //83
	std::string s1{"ali"};
	std::string s2{"veli"};
	std::cout << summer(s1, s2,"kerem"); //aliverlikerem
}
```

--> int a[] = {(12,5), (1,7)}; // a[1]=5, a[2]=7

```
template<typename ...Types>
void print(const Types& ...params)
{
	//(std::cout << params, 0)...
	(std::cout << p1, 0), (std::cout << p2, 0)
}
```

```
// 2. teknik
// recursive olmayan print
template<typename ...Types>
void print(const Types& ...params)
{
	int a[] = { (std::cout << params << "\n", 0)... };
}

int main()
{
	print(23, 5.2, "veli");
}
// output
23
5.2
veli
```

```
// daha profesyonel
template<typename ...Types>
void print(const Types& ...params)
{
	(void)(std::initializer_list<int>) { (std::cout << params << "\n", 0)... };
}

int main()
{
	print(23, 5.2, "veli");
}
---
```

```
---
// 3. teknik
template<typename T, typename ...Args>
std::ostream& print(std::ostream& os, const T& t, const Args& ...rest)
{
	os << t;

	if constexpr (sizeof...(rest) > 0)
	os << ", ";
	else
	os << "\n";

	if constexpr (sizeof...(rest) != 0)
	print(os, rest...);

	return os;
}

int main()
{
	print(std::cout, 23, 5.2, "veli");
}
// output
23, 5.2, veli
---
```

----- STL -----

- Standard template library
- OOP değil
- kalıtımı tamamen statik olarak kullanıyor (compile time)
- run time polymorphism kullanılmıyor
- containers // veri yapılarını implement eden sınıf şablonu
- iterators // pointer-like
    - dinamik ömürlü nesnelerin ömürlerini kontrol etmek
    - kaplarda tutulan öğelerin konumlarını tutan nesneler
- algorithms // fonosiyon şablonları
- function objects

--> containers
  -- sequence containers // ekleme belirli bir konumla
    - std::vector
    - std::list
    - std::deque
    - std::forward_list
    - std::array
    - std::string
  -- associative containers // ikili arama ağaçları
    - std::set
    - std::multi_set
    - std::map
    - std::multi_map

  -- unordered associative containers // hash

- std::unordered_set
- std::unordered_multi_set
- std::unordered_map
- std::unordered_multi_map

```
// range [a, b)
template<typename Iter>
void display_elems(Iter beg, Iter end)
{
	while (beg != end) {
	std::cout << *beg << " ";
	++beg;
	}
	std::cout << "\n";
}

int main()
{
	int a[10]{1,2,3,4,5,6,7,8,9,10};
	display_elems(a, a + 10);

	std::vector<std::string> str {"ali", "veli", "cem"};
	display_elems(str.begin(), str.end());
}
// output
1 2 3 4 5 6 7 8 9 10
ali veli cem
```

```
// iterator sınıfı, vector sınıfının bir nested type'ı

namespace std {
template<typename T>
class Vector {
public:
	class iterator {
	public:
```

```
        operator*
        operator++
        operator--
        };
        iterator begin();
        iterator end();
};
---




---
int main()
{
        std::vector<int> ivec{2, 5, 10};
        auto iter = ivec.begin();
        //std::vector<int>::iterator iter = ivec.begin();
}
---









--- find()
template <typename Iter, typename T>
Iter Find(Iter beg, Iter end, const T& val)
{
        while (beg != end) {
        if(*beg == val)
        return beg;
        ++beg;
        }
        return end;
}

int main()
{
        std::vector<std::string> svec{"ali", "can", "veli", "seda"};
        std::string name = "veli";

        if(auto iter = Find(svec.begin(), svec.end(), name); iter != svec.end()) {
        std::cout << "bulundu: " << *iter << "\n";
```

```
        }
        else
        std::cout << "bulunamadı\n";
}
// output
bulundu: veli
----




---
int main()
{
        std::vector<std::string> svec{"ali", "can", "veli", "seda"};
        sort(begin(svec), end(svec)); // global fonksiyon
        // ADL -> Argument-dependent lookup,
        // fonksiyona gönderilen argümanın türe ilişkin olması
        // std namespace içinde
}
—




---
int main()
{
   std::vector<std::string> svec;
   auto iter_beg = svec.begin();
   // svec boş ise geçerli fakat dereference tanımsız davranıştır ub
   // exception throw etmez
   // end konumu dereference etmek her koşulda ub

   // eğer vector boşsa
   if(svec.begin() == svec.end()) // true
}
---
```

--- iterator category ---

-> bir iteratorun hangi işlemlerde kullanılabileceğini
  belirleyen iterator kategorisidir

- input iterator
- output iterator
- forward iterator
- bidirectional iterator
- random access iterator

-> bir iterator varsa yukardakilerden birine ait olmak zorunda

-> iterator kategorsinin ne oldugunu anlamanın yolu
  iterator_category nested type'nın ne olduguna bakmak

```
using type = std::istream_iterator<int>::iterator_category;
int main()
{
    std::cout << typeid(type).name() << "\n";
}
// output
input_iterator_tag
```
---

iterator kategorileri	operasyonlar	
output iterator	copy constructible ++it it++ *it it-> (sol taraf değeri)	ostream_iterator ostreambuf_iterator
input iterator	copy constructible ++it it++ *it it-> (sağ taraf değeri) it1 == it2 it1 != it2	istream_iterator istreambuf_iterator
forward iterator	copy constructible default constructible ++it it++ *it it-> (sağ taraf değeri) (sol taraf değeri) it1 == it2 it1 != it2	forward_list unordered_set unordered_multiset unordered_map unordered_multimap
bidirectional iterator	copy constructible default constructible ++it it++ --it it-- *it it-> (sağ taraf değeri) (sol taraf değeri) it1 == it2 it1 != it2	list set multiset map multimap
random access iterator	copy constructible default constructible ++it it++ --it it-- *it it-> (sağ taraf değeri) (sol taraf değeri) it1 == it2 it1 != it2 it + n n + it it - n it += n it -= n it1 - it2 it[n] it1 < it2 it1 <= it2 it1 > it2 it1 >= it2	vector deque array string C arrays

-> pointerla yapılan her şey random access iterator ile yapılabilir

---
```
// template parametresini minimal beklenti olan iterator kategorisini,
// çağdıştıracak isimler seçilmeli
// desbeg L value, * ve ++ kullanıldıgından output iterator olması yeterli

template <typename InIter, typename OutIter> //
OutIter Copy(InIter beg, InIter end, OutIter destbeg)
{
   while(beg != end) {
      *destbeg++ = *beg++;
   }
   return destbeg; // en son yazdıgı yerden sonraki konum
}
```

--- find()
```
// linear search
// STL' de arama algoritmaları InIterator döndürür
// aranan değer bulunamazsa end() döner
template <typename InIter, typename T>
InIter Find(InIter beg, InIter end, const T& val)
{
   while (beg != end) {
      if(*beg == val)
         return beg;
      ++beg;
   }
   return end;
}
```

-> predicate = geri dönüş değeri bool olan callable' lar

find_if -> predicate isteyen algoritmalar

--- find_if()

```
// Pred f function pointer
// Pred f function object (functer) olabilir
template <typename InIter, typename Pred>
InIter Find(InIter beg, InIter end, Pred f)
{
    while (beg != end) {
        if(f(*beg))
            return beg;
        ++beg;
    }
    return beg;
}
```

```
bool is_len_4(const std::string& str)
{
    return str.length() == 4;
}

int main()
{
    std::vector<std::string> svec {"istanbul", "ankara", "bursa", "bolu", "adana"};
    if(auto iter = find_if(svec.begin(), svec.end(), is_len_4); iter != svec.end())
    {
        std::cout << *iter << "\n";
        std::cout << iter - svec.begin()<< "\n"; //index

    }
}
// bolu
// 3
```

```
class LenPred { // functor
```

```
public:
   LenPred(size_t len) : mlen(len) {}
   bool operator()(const std::string& s) {
      return s.size() == mlen;
   }
private:
   size_t mlen;
};

int main()
{
   size_t len;
   std::cout << "uzunuk giriniz: ";
   std::cin >> len;
   std::vector<std::string> svec {"istanbul", "ankara", "bursa", "bolu", "adana"};
   if(auto iter = find_if(svec.begin(), svec.end(), LenPred{len}); iter != svec.end())
   {
     // sınıf nesnesi, fonk çağrı operatorunun operandı olur
     // geçici nesne parametresine uzunluk verildi
       std::cout << *iter << "\n";
       std::cout << iter - svec.begin()<< "\n";
   }
}
---
```

--- lambda expression ---

-> derleyici bir sınıfın kodunu yazar
-> ifadeyi geçici nesne oluşturma ifadesi oluşturur
-> lambda ifadesi geçen her yerde geçici nesne türünden sınıf var
-> derleyicinin oluşturduğu sınıfa closer type
-> derleyicinin oluşturduğu geçici nesneye closer object denir
-> []{}
-> [//lambda capture](int x // parametreler){return x+5 // fonk ana blok}

-> auto f = [](int x) {return x + 5; };
   // f derleyicinin olusturdugu sınıf türünden değişken

-> fonksiyon parametresina argüman olarak gönderme

```
template <typename T>
void func(T f)
{
}
int main()
{
    func([](int x, int y) {
        return x * x + y * y;
    });
}
```

-> parametre değişkeni ve betimleyici yoksa

```
auto f = [](){return 1;};
auto f = []{return 1;}; // aynı anlam
```

-> []{}; // void

-> []() {std::cout << "merhaba dunya\n";}();

-> fonk çağrı operatorunun operandı

```
class A {
public:
    void operator()() {}
};

int main()
{
    A{}();
    // aynı anlamda
    []() {}(); // fonk çağrı operatorunun operandı oldu
}
```

->

```
int x = 10;
[](int y){ return x;}; // syntax error
```

->

```
static int y = 10;
[](int y){ return y;}; // geçerli
```

->

```
int x = 10;
[x](int y){ return x;}; // bu sınıfın veri elemanı olan x' i kullanır
// kopyalama yoluyla x'i alır
```

-> lambda' ların üye fonksiyonları default olarak const

->

```
int x = 10;
[x](int y){ return x = y;}; // x default const oldugundan syntax error
```

-> int x = 10;
   [x](int y)mutable { return x = y;}; // artık const member değil, geçerli

-> PR value

-> ifade aynı olsa bile derleyici yeni sınıf oluşturur

-> int x = 10;
   // fonksiyonun çağrılması için
   auto f = [x](int a)mutable {x *= a;}; // capture by copy, x değişmez
   f(10); // veya
   [x](int a)mutable {x *= a;}(10);
   // x' in değeri değişmez

-> int x = 10;
   auto f = [&x](int a) {x *= a;}; // capture by reference
   f(10); // x = 100int x = 10;
   auto f = [&x](int a) {x *= a;}; // capture by reference
   f(10); // x = 100

-> int x = 10;
   int y = 10;
   auto f1 = [=](int a) {return a * (x + y);}; // capture by copy
   auto f2 = [&](int a) {++x; ++y;}; // capture by ref

-> []<typename T> () ->int, mutable, noexcept, constexpr {}

-> // generalize lambda expression
   auto f = [](auto x) {return x * x;};

-> int *ptr;
   [x = *ptr](){}; // lambda init capture
   // birinci varlık nedeni move-only type için

-> IIFE
   const int ival = [x, y](int a) {
      //...
      return x * a - y;
   }(); // () fonksiyon çağrı oprt

```
int main()
{
    auto sum = [](int x, int y) {
        return x + y;
    };
    std::cout << sum(5, 10);
}
// output
15
```

```
class xyz_984 { // derleyicinin yazdıgı sınıf
public:
  auto operator()(int x) {
    return x + 5;
  }
};

int main()
{
  [](int x){return x + 5};
}
```

```
int main()
{
    size_t len;
    std::cout << "uzunuk giriniz: ";
    std::cin >> len;
    std::vector<std::string> svec {"istanbul", "ankara", "bursa", "bolu", "adana"};
    if(auto iter = find_if(svec.begin(), svec.end(), [len](const string& s) {return s.size == len});
    {
      // sınıf nesnesi, fonk çağrı operatorunun operandı olur
      // geçici nesne parametresine uzunluk verildi
        std::cout << *iter << "\n";
        std::cout << iter - svec.begin()<< "\n";
    }
}
```

```
int main()
```

```
{
    // training return type
    auto f = [](int x)->double {return x * 5;};
    auto x = f(5); // x-> double
}
---


---
 // training return type olmasa syntax error
    auto f = [](int x)->double { // çıkarım yapmaz açıkça belirtilir
        if(x <5)
            return x; // int
        else
        return 1.4 * x; // double
        };
---




---
auto func(int x)
{
    return [&x](int a) {return a * x;};
}

int main()
{
    auto f = func(12);
    auto x = f(5); // ub , danglig reference
    std::cout << "x = " << x << "\n"; //-45643
}
---




---
int main()
{
    std::vector<std::string> svec;
    rfill(svec, 100, rname);
    print(svec);

    char c;

    std::cout << "karakter giriniz: ";
```

```
    std::cin>> c;
    if (auto iter = std::find_if(begin(svec), end(svec),
    [c](const auto& s) {return s.std::find(c) != std::string::npos;}); iter != end(svec))
    {
        std::cout << "buludnu " << iter - svec.begin() << "indis:" << *iter;
    }
}
---
```

```
---
int main()
{
    vector<string> svec;
    rfill(svec, 100, rname);
    print(svec);

    sort(svec.begin(), svec.end()); // s1 < s2
    sort(svec.begin(), svec.end(),
    [](const string& s1, const string& s2){return s1 > s2;}); // s1 > s2
}
---
```

//

-> iterator

```
---
int main()
{
    vector<int> ivec{2, 5, 10};
    const vector<int>::iterator iter = ivec.begin(); // low level const değil
    *iter = 45; // geçerli
    ++iter; //syntax error

}
---
```

```
int main()
{
   vector<int> ivec{2, 5, 10};
   vector<int>::const_iterator iter = ivec.begin(); // low level const
   *iter = 40; // syntax error

}
```
---

cbegin(//)
cend(//) -> const
iter = cbegin(a)
auto *iter = 45; // syntax error

```
template<typename InIter, typename OutIter>
OutIter Copy(InIter beg, InIter end, OutIter destbeg)
{
   while(beg != end)
      *destbeg++ = *beg++;
   return destbeg;
}

using namespace std;

int main()
{
   vector<int> x{3, 5, 7, 20};
   vector<int> y{0, 1, 8};

   Copy(x.begin(), x.end(), y.begin()); // ub
   // olmayan konuma yazma girişimi

}
```

```
template<typename InIter, typename OutIter>
```

```
OutIter Copy(InIter beg, InIter end, OutIter destbeg)
{
   while(beg != end)
      *destbeg++ = *beg++;
   return destbeg;
}



template <typename C> // Container tutucak
class BackInsertIterator {
public:
   BackInsertIterator(C &c) : mc(c) {}
   BackInsertIterator& operator*() { return *this; }
   BackInsertIterator& operator++() { return *this; }
   BackInsertIterator& operator++(int) { return *this; }
   BackInsertIterator& operator=(const typename C::value_type& val) // containerda tutula
öğenin türündense
   {
      mc.push_back(val);
      return *this;
   }
private:
C& mc;
};

int main()
{
   vector<int> x{3, 5, 7, 20};
   vector<int> y;
   copy(x.begin(), x.end(), BackInsertIterator<vector<int>>{y});
   for(int i : y)
      std::cout << i << " ";


}
// output
3 5 7 20
---




---
template <typename Iter>
```

```
void print(Iter t)
{
    for(auto iter = t.begin(); iter != t.end(); iter++)
        std::cout << *iter << " " ;
    std::cout << "\n";
}
---

---
template<typename Iter>
void Print(Iter beg, Iter end)
{
    while(beg != end)
        cout << *beg++ << " ";
    cout << "\n";
}
---


--- orjinali
int main()
{
    vector<int> x{3, 5, 7, 20};
    vector<int> y {1};

    copy(x.begin(), x.end(), back_inserter(y));
    print(y);

}
// output
1 3 5 7 20
---




--- copy_if
template <typename InIter, typename OutIter, typename Pred>
OutIter CopyIf(InIter beg, InIter end, OutIter destbeg, Pred f)
{
    while (beg != end) {
        if (f(*beg))
            *destbeg++ = *beg;
        ++*beg;
    }
    return destbeg;
}
```

```
int main()
{
    vector<string> townvec;
    rfill(townvec, 100, rtown);
    print(towlower);
    size_t len;

    std::cout << "uzunlugu kac olanlar kopyalansin";
    cin >> len;

    list<string> slist;

    copy_if(townvec.begin(), townvec.end(), back_inserter(slist),
    [len](const string& s){return s.size() == len;});

}
```

--- reverse_iterator ---

```
// reverse_iterator iterator adaptörü
// reverse_iterator nested type

template<typename Iter>
void Print(Iter beg, Iter end)
{
    while(beg != end)
        cout << *beg++ << " ";
    cout << "\n";
}
```

```
int main()
{
    vector<int> ivec {1, 4, 7, 6, 3};
    Print(ivec.begin(), ivec.end());
    Print(ivec.rbegin(), ivec.rend());
}
// output
1 4 7 6 3
3 6 7 4 1
---
```

```
---
int main()
{
    vector<int> ivec {1, 4, 7, 6, 3};
    auto iter = ivec.rbegin();
    cout << *iter << "\n";
    ++iter; // reverse iterator oldugundan --iter;
    cout << *iter << "\n";

}
// output
3
6
---
```

```
---
int main()
{
   vector<int> ivec{10, 20, 30, 40};

   auto riter = ivec.rbegin();
   std::cout << *riter << "\n"; // 40

   auto iter = riter.base(); // end tutar *iter syntax error
}
```

```
int main()
{
   vector<int> ivec{10, 20, 30, 40};

   auto riter = ivec.rbegin();
   ++riter;
   std::cout << *riter << "\n"; // 30

   auto iter = riter.base();
   std::cout << *iter << "\n"; // 40

}
```

```
int main()
{
   vector<int> ivec{10, 20, 30, 40};

   auto rbeg = ivec.rbegin();
   auto rend = ivec.rend();

    print(rbeg, rend);
    print(rend.base(), rbeg.base());

}
// output
40 30 20 10
10 20 30 40
```

--- mülakat sorusu

```
int main()
{
    // soru sondaki 30' u diziden sil
    vector<int> ivec{10, 30, 20, 30, 40};

    int ival = 30;
    print(ivec);

    auto iter = find(ivec.rbegin(), ivec.rend(), ival);
    if(iter != ivec.rend()) { // bulunursa
        ivec.erase(iter.base() - 1); // erase iter parametreli elemanı siler
        // ivec.erase(iter); // syntax error iter_reverse türünden
        // *iter 30 fakat iter 30'u değil ondan sonraki elemanı gösterir
        // iter.base() 40'ı gösterir, 30'u göstermesi için iter.base() - 1
    }
    print(ivec);

}
// output
10 30 20 30 40
10 30 20 40
```

-> 4 begin()

```
c.begin();   container::iterator
c.cbegin();  container::const_iterator
c.rbegin();  container::reverse_iterator
c.crbegin(); container::const_reverse_iterator
```

--- iterator fonksiyon şablonları ---

-> advance()
// iterator kategorisine bağlı olarak döngü veya döngüsüz iteratorü arttırır

```
vector<int> ivec{10, 30, 20, 30, 40};
auto iter = ivec.begin();
std::cout << *iter << "\n"; // 10
advance(iter, 3);
std::cout << *iter << "\n"; // 30
```

--- advance() // tag dispatch ile

```
namespace details { // compile time' da iterator kategorisine göre
   template <typename Rainter, typename Distance> // random access iterator tür ise
   void advance(Rainter&  it, Distance n, std::random_access_iterator_tag)
   {
      it += n;
   }

   template <typename Rainter, typename Distance> // bidirectional iterator türse
   void advance(Rainter&  it, Distance n, std::random_access_iterator_tag)
   {
      if( n > 0)
      {
         while (n--)
            ++it;
      }
      else
      {
         while(n++)
            --it;
      }
   }


   template <typename Rainter, typename Distance> // input iterator ise
   void advance(Rainter&  it, Distance n, std::random_access_iterator_tag)
   {
      while(n--)
         ++it;
   }
}
```

```
template <typename Iter, typename Distance>
void advance(Iter&  it, Distance n)
{
   advance(it, n, typename std::iterator_traits<Iter>::iterator_category{}); // pointer türleri de
kullanılabilir

   // 2) advance(it, n, typename Iter::iterator_category{}); // recursive değil
   // iter türü pointer olursa 2. advance kullanılamaz
}
---
```

```
--- advance() if constexpr ile
template <typename Iter, typename Dist>
void advance(Iter&  pos, Dist n)
{
   using cat = typename std::iterator_traits<Iter>::iterator_category;
   if constexpr (std::is_same_v<cat, sd::random_acces_iterator_tag>)
   { // std::is_same<cat, std::random_acces_iterator_tag>::value
      while(n--)
         ++pos;
   }
   else {
      while(n++)
         --pos;
   }
   else { //input iterator tag
      while(n--)
         ++pos;
   }
}
---
```

```
---
int main()
{
```

```
    list<int> vec{1, 2, 3, 4};
    auto iter_x = vec.begin();
    auto iter_y = vec.end();
    // distance() = end() - begin()
    // int n = iter_y - itere_x; syntax error,  bidirectional tag

    auto n = distance(iter_x, iter_y);
    std::cout << "n = " << n << "\n"; // 4
}
---
```

advance
distance
next
prev
iter_swap

advance(iter, 5) -> iter değişir call by ref
next(iter, 5); -> call by value
next(iter) // default (iter, 1)
prev(iter, 3);

```
---
int main()
{
    list<int> vec{1, 2, 3, 4};
    *prev(vec.end(), 2) = -1;
    print(vec);
}
// output
1 2 -1 4
---
```

iter_swap -> iter_swap doesnt swap iterator
  - 2 iterator konumundaki nesneyi swap eder

```
int main()
{
    list<int> vec{1, 2, 3, 4};
    iter_swap(vec.begin(), prev(vec.end()));
    print(vec);
}
// output
4 2 3 1
```

--- reverse algorithm
```
int main()
{
    vector<string> svec {"istanbul", "ankara", "bursa", "adana"};
    print(svec);

    for(auto &s : svec) {
        reverse(begin(s), end(s));
    }
    print(svec);

    reverse(begin(svec), end(svec));
    print(svec);
}
// output
```

```
istanbul ankara bursa adana
lubnatsi arakna asrub anada
anada asrub arakna lubnatsi
---
```

```
---
int main()
{
    list<string> x;
    rfill(x, 100, rname);

    reverse(x.begin(), x.end()); // maliyeti daha fazla, nesneleri takas eder

    x.reverse(); // ilk öncelik her zaman üye fonksiyon olmalı
    // node' ları değiştirir

}
---
```

```
--- for_each()
template <typename Iter, typename Ufunc>
Ufunc foreach(Iter beg, Iter end, Ufunc f)
{
    while(beg != end) {
        f(*beg);
    }
    return f;
}
---
```

```
---
int main()
{
    vector<string> svec {"ali", "cem", "ece", "veli"};
    print(svec);
    for_each(svec.begin(), svec.end(), [](auto& s){s += "can";});
```

```
    print(svec);

}
// output
ali cem ece veli
alican cemcan ececan velican
---
```

```
--- transform
template <typename InIter, typename OutIter, typename Func>
OutIter Transform(InIter beg, InIter end, OutIter dest, Func f)
{
    while(beg != end) {
        *dest++ = f(*beg++);
    }
    return dest;
}
---
```

```
---
int main()
{
    vector<int> ivec{1, 3, 7, 4};
    list<int> ilist(ivec.size());
    int n;

    std::cout << "kac kat";
    cin >> n; //5

    print(ivec);
    transform(ivec.begin(), ivec.end(), ilist.begin(),
    [n](int x){return x * n;});
    print(ilist);
}
// output
1 3 7 4
5 15 35 20
```

--- remove_copy

```
template<typename InIter, typename OutIter, typename T>
OutIter RemoveCopy(InIter beg, InIter end, OutIter destbeg, const T& val)
{
   while(beg != end)
   {
      if(*beg != val)
         *destbeg++ = *beg;
      ++beg;
   }
   return destbeg;
}
```

```
int main()
{
   vector<int> ivec{1, 3, 7, 4};
   vector<int> dvec;

   int ival;
   std::cout << "silinecek degeri girin: ";
   cin >> ival;
   remove_copy(ivec.begin(), ivec.end(), back_inserter(dvec), ival);
   print(ivec);
   print(dvec);
}
// output
1 3 7 4
1 7 4
```

--- remove_copy_if

```
template<typename InIter, typename OutIter, typename UnPred>
OutIter RemoveCopyIf(InIter beg, InIter end, OutIter destbeg, UnPred f)
{
   while(beg != end)
   {
      if(!f(*beg))
         *destbeg++ = *beg;
      ++beg;
   }
   return destbeg;
}
```

----- containers -----

-- sequence containers
- vector
- deque
- list
- forward_list
- array
- string
- C arrays

-> C dizileri hariç ortak bütün container'da ortak olan
üye fonk var, sequence container'a özel fonksiyonları var

// CTAD

```
vector<int> ivec;
vector ivec {1, 5, 9}; // vector<int>  ,c++17
// derleyici ctora gönderilen argümanlardan çıkarım yapar
```

--- vector ---

```
int main()
{
   vector<int> ivec(10);

   ivec.push_back(2);
   ivec.push_back(3);
   ivec.push_back(5);

   std::cout << "size = " << ivec.size() << "\n"; // 13
   std::cout << "cap = " << ivec.capacity() << "\n"; // 20
   // reallocation olmadan 7 eleman eklenebilir
}
```

```
--> ivec.reserve(10000); // capacity bu degerden başlar
--> ivec.shrink_to_fit(); // capacity = size olur
```

-> realloaction oldugunda eski pointerlar danglig hale gelir!

```
int main()
{
```

```
    vector<int> ivec; // default ctor
    //vector<int> ivec{};

    if(ivec.empty())
    // if(std::empty(x)) global func
    {
        // ...
    }

    if(ivec.size() < 5)
    //if(size(ivec) < 5)  global func
    {
        //...
    }
}
---
```

```
int main()
{
    vector<int> ivec;
    vector<int> x = ivec; // copy ctor
    vector<int> y {ivec}; // copy ctor
}
```

```
int main()
{
    int n;
    cin >> n;
    vector<int> ivec(n); // size = n
    // tüm öğeleri default initial eder
    // aritmatik tür ise 0, pointer ise nullptr
}
```

```
int main()
```

```
{
    vector<int> x(10); // 10 elamanlı
    std::cout << x.size() << "\n"; // 10
    // tüm değerler 0


    vector<int> y{10}; // initializer_list<int> açılıımı türünden ctor
    std::cout << y.size() << "\n"; // 1
    // değeri 10
}
```

-> initializer_list parametreli fonksiyonlar dışında
normalde {} veya () ile ctor kullanılabilir

```
class Myclass {
public:
    Myclass(int)
    {
        std::cout << "Myclass(int)\n";
    }
    Myclass(int, int)
    {
        std::cout << "Myclass(int, int)\n";
    }
    Myclass(std::initializer_list<int>)
    {
        std::cout << "Myclass(initializer_list)\n";
    }
};

int main()
{
    Myclass a(12);
    Myclass b(12, 1);
    Myclass c{12};
    Myclass d{12, 1};
    // initializer_list parametreli ctor olmasaydı,
    // (int) veya (int, int) çıktı olurdu
}
// output
Myclass(int)
Myclass(int, int)
```

Myclass(initializer_list)
Myclass(initializer_list)

```
int main()
{
    //vector(size_type n, const T& val);
    // n değer ile başlatır, her birinin değeri val

    vector<int> ivec(10, 5);
    print(ivec);
}
// output
5 5 5 5 5 5 5 5 5 5
```

-> range ctor

```
vector<int> myvec{1,2,3,4,5};
vector<double> dvec(myvec.begin(), myvec.end());
print(dvec); // 1 2 3 4 5
```

```
int main()
{
    vector<const char *> vec{"ali", "can", "oya"};
    list<string> mylist{vec.begin(), vec.end()};
    print(mylist); // ali can oya
}
```

```
int a[]{1, 5, 10, 7, 8};
vector ivec{begin(a), end(a)};
```

--- getter
- size
- capacity
- max_size
- empty

```
int main()
{
    vector<int> vec{1, 2, 3, 4, 5};
    for(size_t i = 0; i < vec.size(); ++i)
    {
        std::cout << vec[i] << "\n";
        // std::cout << vec.operator[](i) << "\n";
    }
}
```

```
int main()
{
    vector<int> vec{1, 2, 3, 4, 5};
    try {
        auto val = vec[100]; // ub, hata yakalanmaz
    }
    catch(const std::exception& ex) {
        std::cout << "exception caught : " << ex.what() << "\n";
    }
}
```

```
int main()
{
    vector<int> vec{1, 2, 3, 4, 5};
    try {
        auto val = vec.at(100); // hata yakalanır
    }
    catch(const std::exception& ex) {
        std::cout << "exception caught : " << ex.what() << "\n";
    }
}
// output
exception caught : vector::_M_range_check: __n (which is 100) >= this->size() (which is 5)
```

--- front(), back()

```
int main()
{
    vector<string> svec{"ali", "ayse", "cem"};

    svec.front() += "can"; // svec[0]
    svec.back() += "tok"; // svec[svec.size() - 1]
    print(svec);
}
// output
alican ayse cemtok
```

--- resize()

```
int main()
{
    vector<string> svec{"ali", "ayse", "cem"};

    std::cout << svec.size() << "\n"; // 3
    svec.resize(40);
    std::cout << svec.size() << "\n"; // 40
    // diğer öğeler default init edilir str "", int{0} ...
}
```

```
int main()
{
    vector<string> svec{"ali", "ayse", "cem"};

    std::cout << svec.size() << "\n";
    print(svec);
    svec.resize(1); // silmek için
    std::cout << svec.size() << "\n";
    print(svec);


}
// output
3
ali ayse cem
1
ali
```

```
int main()
{
    vector<string> svec{"ali", "ayse", "cem"};

    std::cout << svec.size() << "\n";
    print(svec);
    svec.resize(10, "kerem");
```

```
    std::cout << svec.size() << "\n";
    print(svec);


}
// output
3
ali ayse cem
10
ali ayse cem kerem kerem kerem kerem kerem kerem kerem
---
```

```
-- sequence container insert/erase fonksiyonları --

erase(iter);
erase(iter1, iter2);
// silinen öğeden sonraki öğenin konumunu döndürür
```

```
int main()
{
    vector<string> svec{"ali", "ayse", "cem", "kerem"};
    print(svec);

    svec.erase(svec.begin() + 1); // 2. öğeyi siler
    // random access iterator olmasaydı geçersizdi

    svec.erase(next(svec.begin())); // list sınıfında da geçerli, ortak arayüz


    svec.erase(prev(svec.end())); // son öğeyi siler
    svec.pop_back(); // son öğeyi siler, geri dönüş değeri yok

    svec.clear(); öğeleri siler
}
```

Operation	Effect
`vector<Elem> c`	Default constructor; creates an empty vector without any elements
`vector<Elem> c(c2)`	Copy constructor; creates a new vector as a copy of *c2* (all elements are copied)
`vector<Elem> c = c2`	Copy constructor; creates a new vector as a copy of *c2* (all elements are copied)
`vector<Elem> c(rv)`	Move constructor; creates a new vector, taking the contents of the rvalue *rv* (since C++11)
`vector<Elem> c = rv`	Move constructor; creates a new vector, taking the contents of the rvalue *rv* (since C++11)
`vector<Elem> c(n)`	Creates a vector with *n* elements created by the default constructor
`vector<Elem> c(n,elem)`	Creates a vector initialized with *n* copies of element *elem*
`vector<Elem> c(beg,end)`	Creates a vector initialized with the elements of the range [*beg*,*end*)
`vector<Elem> c(initlist)`	Creates a vector initialized with the elements of initializer list *initlist* (since C++11)
`vector<Elem> c = initlist`	Creates a vector initialized with the elements of initializer list *initlist* (since C++11)
`c.~vector()`	Destroys all elements and frees the memory

Table 7.9. Constructors and Destructor of Vectors

Operation	Effect
`c[idx]`	Returns the element with index *idx* (*no* range checking)
`c.at(idx)`	Returns the element with index *idx* (throws range-error exception if *idx* is out of range)
`c.front()`	Returns the first element (*no* check whether a first element exists)
`c.back()`	Returns the last element (*no* check whether a last element exists)

Table 7.12. Direct Element Access of Vectors

Operation	Effect
`c.push_back(`*elem*`)`	Appends a copy of *elem* at the end
`c.pop_back()`	Removes the last element (does not return it)
`c.insert(`*pos*`,`*elem*`)`	Inserts a copy of *elem* before iterator position *pos* and returns the position of the new element
`c.insert(`*pos*`,`*n*`,`*elem*`)`	Inserts *n* copies of *elem* before iterator position *pos* and returns the position of the first new element (or *pos* if there is no new element)
`c.insert(`*pos*`,`*beg*`,`*end*`)`	Inserts a copy of all elements of the range [*beg*,*end*) before iterator position *pos* and returns the position of the first new element (or `pos` if there is no new element)
`c.insert(`*pos*`,`*initlist*`)`	Inserts a copy of all elements of the initializer list *initlist* before iterator position *pos* and returns the position of the first new element (or *pos* if there is no new element; since C++11)
`c.emplace(`*pos*`,`*args...*`)`	Inserts a copy of an element initialized with *args* before iterator position *pos* and returns the position of the new element (since C++11)
`c.emplace_back(`*args...*`)`	Appends a copy of an element initialized with *args* at the end (returns nothing; since C++11)
`c.erase(`*pos*`)`	Removes the element at iterator position *pos* and returns the position of the next element
`c.erase(`*beg*`,`*end*`)`	Removes all elements of the range [*beg*,*end*) and returns the position of the next element
`c.resize(`*num*`)`	Changes the number of elements to *num* (if `size()` grows new elements are created by their default constructor)
`c.resize(`*num*`,`*elem*`)`	Changes the number of elements to *num* (if `size()` grows new elements are copies of *elem*)
`c.clear()`	Removes all elements (empties the container)

Table 7.14. Insert and Remove Operations of Vectors

Operation	Effect
`c = `*c2*	Assigns all elements of *c2* to `c`
`c = `*rv*	Move assigns all elements of the rvalue *rv* to `c` (since C++11)
`c = `*initlist*	Assigns all elements of the initializer list *initlist* to `c` (since C++11)
`c.assign(`*n*`,`*elem*`)`	Assigns *n* copies of element *elem*
`c.assign(`*beg*`,`*end*`)`	Assigns the elements of the range [*beg*,*end*)
`c.assign(`*initlist*`)`	Assigns all the elements of the initializer list *initlist*
`c1.swap(c2)`	Swaps the data of `c1` and `c2`
`swap(c1,c2)`	Swaps the data of `c1` and `c2`

Table 7.11. Assignment Operations of Vectors

--- assign(iter1, iter2)

```
int main()
{
    vector<string> svec{"ali", "ayse", "cem", "kerem"};
    list<string> mylist{"arda", "veli"};

    svec.assign(mylist.begin(), mylist.end());
    print(svec); // arda veli
}
```

```
int main()
{
    vector<int> ivec;
    ivec.assign(10, 3);
    print(ivec);
}
// output
3 3 3 3 3 3 3 3 3 3
```

```
int main()
{
    vector<string> svec{"ali", "ayse", "cem", "kerem"};

    auto iter = svec.insert(svec.begin() + 1, "berk");
    std::cout << *iter << "\n"; // berk
    print(svec); // ali berk ayse cem kerem
}
```

```
int main()
{
    vector<string> svec{"ali", "ayse", "cem", "kerem"};
```

```
    vector<string> d;

    for(const auto& s : svec)     {
        d.insert(d.begin(), s); // her öğeyi dizinin başına ekler
        print(d);
        getchar();
    }
}
// output
kerem cem ayse ali
---
```

```
---
int main()
{
    int a[] {1,2,3,4,5};
    vector<int> ivec{10, 20, 30, 40,50};
    print(ivec);
    ivec.insert(ivec.begin() + 2, begin(a), end(a));
    print(ivec);
}
// output
10 20 30 40 50
10 20 1 2 3 4 5 30 40 50
---
```

```
---
int main()
{
    vector<int> ivec{10, 20, 30, 40,50};
    print(ivec);
    ivec.insert(ivec.begin() + 2, 5, 100); // fill insert
    print(ivec);
}
// output
10 20 30 40 50
10 20 100 100 100 100 100 30 40 50
---
```

--> emplace
// doğrudan nesneyi taşımak veya kopyalamak yerine,
   nesneyi vector'un adresinde oluşturur

```
int main()
{

    vector<Date> dvec;
    Date mydate(10, 4, 2000);

    dvec.push_back(Date::random());
    dvec.push_back(mydate);
    dvec.emplace(4, 4, 1999); // doğrudan container'da hayata gelir

    dvec.emplace(dvec.begin(), 1, 1, 2005); // başa ekler
}
```

```
int main()
{

    // n. indeks sonrasındaki tüm elmanları silme

    vector<int> ivec(100);
    size_t idx = 15;

    ivec.resize(idx);
    //veya
    ivec.erase(ivec.begin() + idx, ivec.end());
}
```

--- swap

```
int main()
{

    vector<int> odds{10, 20, 30, 40,50};
    vector<int> evens{2, 4, 5, 9, 11};

    // vector'un pointer elemanlarını swap eder
    // heap'deki öğeleri takas etmez
    swap(odds,evens);
    // veya
    odds.swap(evens);

}
```

--- data

```
void display_array(const int* p, size_t n)
{
    while(n--)
        std::cout << *p++ << " ";
    std::cout << "\n";

}

int main()
{
    vector<int> ivec{10, 20, 30, 40,50};
    display_array(ivec.data(), ivec.size()); // //10 20 30 40 50
    // veya
    display_array(&ivec[0], ivec.size());
    // veya
    display_array(&*ivec.begin(), ivec.size());

    display_array(ivec.begin(), ivec.size()); // syntax error
    // iterator türünden
```

```
}
---
```

-- silme algoritmaları --

// container' ın üye fonksiyonu erase,
  global algorithm ismi remove

-> remove çağırılmadan önceki size ile sonraki size anyı
-> geri dönüş değeri logic_end -> silinmemiş ilk öğeden sonraki konum
-> remove doesnt remove

```
---
int main()
{
    vector<int> ivec{1, 5, 1, 2, 4, 7, 1, 3, 1 };
    cout << ivec.size() << "\n";

    auto iter = remove(begin(ivec), end(ivec), 1);
    print(ivec.begin(), iter); // silinmemiş öğeler

    cout << "silinmemis ogeler : " << distance(ivec.begin(), iter) << "\n";
    cout << "silinmis ogeler : " << distance(iter, ivec.end()) << "\n";
    cout << ivec.size() << "\n";
}
// output
9
5 2 4 7 3
silinmemis ogeler : 5
silinmis ogeler : 4
9
```

--- remove-erase idiom

```
int main()
{
    // remove-erase idiom
    vector<int> ivec{1, 5, 1, 2, 4, 7, 1, 3, 1 };

    cout << "size = " << ivec.size() << "\n"; // 9
    ivec.erase(remove(ivec.begin(), ivec.end(), 1), ivec.end());
    cout << "size = " << ivec.size() << "\n"; // 5
    print(ivec); // 5 2 4 7 3

}
```

```
int main()
{
    vector<string> svec{"ali","ayşe", "ali","erdem", "su" };

    erase(svec,"ali"); // kendi içinde remove-erase kullanıyo
    // geri dönüş değeri silinmiş öğe sayısı
    // c++20
}
```

--- remove_if

```
int main()
{
    // remove_if
    vector<string> svec;
    rfill(svec, 100, rname);

    size_t len;
    std::cout << "uzunlugu kac olanlar silinsin: ";
```

```
    cin >> len;
    svec.erase(remove_if(svec.begin(), svec.end(),
    [len](const string& s){return s.size() == len;}), svec.end());

}
---
```

```
--- unique(iter1, iter2)
int main()
{
    vector<int> ivec{1, 2, 2, 3, 4, 5, 5, 5, 6, 6, 7};
    print(ivec);
    ivec.erase(unique(ivec.begin(), ivec.end()), ivec.end());
    print(ivec);
}
// output
1 2 2 3 4 5 5 5 6 6 7
1 2 3 4 5 6 7
---
```

```
--- ms
int main()
{
    // her değerden bir tane olması için önce sort sonrar unique

    vector<int> ivec{1, 2, 3, 1, 7, 2, 3, 4, 5, 5, 5, 6, 6, 7};
    print(ivec);
    sort(ivec.begin(), ivec.end());
    ivec.erase(unique(ivec.begin(), ivec.end()), ivec.end());
    print(ivec);
}
// output
1 2 3 1 7 2 3 4 5 5 5 6 6 7
1 2 3 4 5 6 7
---
```

```
--- unique(iter1, iter2, f)
int main()
{
    // unique(iter1, iter2, f)

    vector<int> ivec{1, 2, 3, 1, 7, 2, 3, 4, 5, 5, 5, 6, 6, 7};
    print(ivec);
    ivec.erase(unique(ivec.begin(), ivec.end(),
    [](int x, int y) {return x % 2 == y % 2;}), ivec.end());
    print(ivec);
    // tek ve çift ardışık olmaz
}
// output
1 2 3 1 7 2 3 4 5 5 5 6 6 7
1 2 3 2 3 4 5 6 7
---
```

```
--- unique_copy(iter1, iter2, iter_dest)
int main()
{
    vector<int> ivec{1, 2, 3, 1, 7, 2, 3, 4, 5, 5, 5, 6, 6, 7};
    list<int> ilist;

    unique_copy(ivec.begin(), ivec.end(), back_inserter(ilist));
    print(ilist);
}
// output
1 2 3 1 7 2 3 4 5 6 7
```

```
--- ms
int main()
{
    // aranan ilk değeri silme

    vector<string> svec{"ali","ayse", "ali","erdem", "su" };
```

```
    print(svec);

    cout << "silinecek isim: ";
    string name;
    cin >> name;

    if(auto iter = find(svec.begin(), svec.end(), name); iter != svec.end())
    {
        svec.erase(iter);
        print(svec);
    }
    else
        cout << "isim bulunamadi\n";

    // sondan başlayarak aranan ilk öğeyi silme
    // if(auto iter = find(svec.rbegin(), svec.rend(), name); iter != svec.rend())
    // {
    //     svec.erase(iter.base() - 1);
    //     print(svec);
    // }
}
```

Expression	Effect
`negate<type>()`	*- param*
`plus<type>()`	*param1 + param2*
`minus<type>()`	*param1 - param2*
`multiplies<type>()`	*param1 * param2*
`divides<type>()`	*param1 / param2*
`modulus<type>()`	*param1 % param2*
`equal_to<type>()`	*param1 == param2*
`not_equal_to<type>()`	*param1 != param2*
`less<type>()`	*param1 < param2*
`greater<type>()`	*param1 > param2*
`less_equal<type>()`	*param1 <= param2*
`greater_equal<type>()`	*param1 >= param2*
`logical_not<type>()`	*! param*
`logical_and<type>()`	*param1 && param2*
`logical_or<type>()`	*param1 || param2*
`bit_and<type>()`	*param1 & param2*
`bit_or<type>()`	*param1 | param2*
`bit_xor<type>()`	*param1 ^ param2*

Table 10.1. Predefined Function Objects

-- standart function object --

```
#include <functional>



template<typename T>
struct Equal_to {
    T operator()(const T& x, const T& y)
    {
        return x == y;
    }
};
```

```
int main()
{
    vector<int> ivec;
    rfill(ivec, 100, Irand(0, 1000));

    sort(ivec.begin(), ivec.end()); // default x < y

    sort(ivec.begin(), ivec.end(), greater{}); //  x > y
    sort(ivec.begin(), ivec.end(), greater<int>{}); //  aynı
    sort(ivec.begin(), ivec.end(), greater<>{}); //  aynı

    sort(ivec.begin(), ivec.end(), [](int a, int b){return a > b;}); //  x > y
}
```

--- ostream iterator

```
template <typename InIter, typename OutIter> //
```

```
OutIter Copy(InIter beg, InIter end, OutIter destbeg)
{
   while(beg != end) {
      *destbeg++ = *beg++;
   }
   return destbeg;
}

// çıkış akımına vektorü kopyalar/yazdırır
template <typename T>
class OstreamIterator {
public:
   OstreamIterator(std::ostream& os, const char* p = "") : mos{os}, mp{p}{}
   OstreamIterator& operator++() {return *this;}
   OstreamIterator& operator++(int) {return *this;}
   OstreamIterator& operator*() {return *this;}
   OstreamIterator& operator=(const T& val)
   {
      mos << val << mp;
      return *this;
   }
private:
   std::ostream& mos;
   const char* mp;
};



// ostream iterator
int main()
{
   vector<int> ivec{1, 2, 3, 4, 5, 7, 10};

   Copy(ivec.begin(), ivec.end(), OstreamIterator<int>{cout, " "});
   // dosyaya da yazabilir
}
// output
1 2 3 4 5 7 10
---




--- generate(iter1, iter2, f)
int main()
{
   vector<int> ivec{1, 2, 3, 4, 5, 7, 10};
```

```
    generate(ivec.begin(), ivec.end(), [](){return 1;});
    copy(ivec.begin(), ivec.end(), OstreamIterator<int>{cout, " "});
}
// output
1 1 1 1 1 1 1
---



--- generate_n(iter, int size, f)
int main()
{
    vector<int> ivec{1, 2, 3, 4, 5, 7, 10};

    generate_n(ivec.begin(), 3, [](){return 1;});
    copy(ivec.begin(), ivec.end(), OstreamIterator<int>{cout, " "});
}
// output
1 1 1 4 5 7 10
---




---ms
// birden fazla boşluk varsa siler
int main()
{
    string str;
    cout << "yazi girin: ";
    getline(cin, str);
    cout << '|'<< str << "|\n";

    str.erase(unique(str.begin(), str.end(),
    [](char c1, char c2){return isspace(c1) && isspace(c2);}), str.end());
    cout << '|'<< str << "|\n";

}
// output
|c++  java sharp    asp py   th|
|c++ java sharp asp py th|
---
```

--- sıralama algoritmaları ---

--- partial_sort(iter1, itern, iter2, f)

```
int main()
{
    // ilk 10 değer sıralı
    partial_sort(ivec.begin(), ivec.begin() + 10, ivec.end());

    partial_sort(ivec.begin(), ivec.begin() + 10, ivec.end(), greater{});
    // büyükten küçüğe
}
```

stable_sort -> sıralamadan önceki izafi konumlarını koruyarak sıralanır

sort stable olmak zorunda değil

```
using namespace std;

using na_pair = std::pair<std::string, int>;

int main()
{
    vector<na_pair> vec;

    generate_n(back_inserter(vec), 10'000, [] {return na_pair{ rname(), Irand{10, 50}() }; });

    std::ofstream ofs{ "out.txt" };
    if (!ofs) {
        std::cerr << "out.txt dosyasi olusturulamadi\n";
        exit(EXIT_FAILURE);
    }


    sort(vec.begin(), vec.end(), [](const auto& p1, const auto& p2) {
        return p1.first < p2.first; });


    stable_sort(vec.begin(), vec.end(), [](const auto& p1, const auto& p2) { //stable olmak zorunda değil
        return p1.second < p2.second; });
                ▲ 1 of 4 ▼ void stable_sort<_BidIt, _Pr>(const _BidIt _First, const _BidIt _Last, _Pr _Pred)

    ofs << left;

    for (const auto& [name, age] : vec) {
        ofs << setw(20) << name << " " << age << "\n";
    }
```

nth_element -> n' inci öğeye göre sıralar, n den öncekiler küçük
  sonraikiler büyük olur

partition(Iter1, iter2, f); // koşula göre sıralar
  - geri dönüş değeri partial point, koşulu sağlamayanların ilk konumu
  - hepsi sağlıyorsa geri dönüş x.end()

partition_copy -> koşulu sağlayanları bir yere, koşulu sağlamayanları başka bir yere

---
```
int main()
{
    vector<Date> dvec;
    rfill(dvec, 100, Date::random);
    vector<Date> ok_vec;
    vector<Date> not_ok_vec;

    auto iter_pari = partition_copy (dvec.begin(), dvec.end(), back_inserter(ok_vec),
back_inserter(not_ok_vec),
    [](const Date& x) {return x.week_day() == 6;});

    cout << "kosulu sağlayanlar";
    print(ok_vec);

    cout << "kosulu sağlayamayanlar";
    print(not_ok_vec);
}
```

--- is_sorted(iter1, iter2)
```
int main()
```

```
{
    vector<int> ivec{2, 5, 7, 9, 15};
    // sıralı mı?
    cout << is_sorted(ivec.begin(), ivec.end()); // true
}
---



--- is_sorted(iter1, iter2, f)
int main()
{
    vector<int> ivec{2, 5, 7, 9, 15};
    cout << is_sorted(ivec.begin(), ivec.end(), greater{}); // false
}
---



--- is_sorted_until(iter1, iter2)
int main()
{
    vector<int> ivec{2, 4, 10, 9, 15};
    // sıranın bozuldugu adresi dönderir
    auto iter = is_sorted_until(ivec.begin(), ivec.end());
    cout << *iter; // 9
}
---









--- ms
// vector' de uzunluğu 3 olan öğeler silinecek, 4 olanlar tekrar eklenecek
int main()
{
    vector<string> svec{"ali", "seda", "cem", "can", "veli", "arda", "berkecan"};
    auto iter = svec.begin();
    print(svec);

    // iterator invalidation olmaması için...
    while (iter != svec.end()) {
        if(iter->length() == 3)
```

```
            iter = svec.erase(iter); // geri dönüsü sildiği öğeden sonraki konum
        else if(iter->length() == 4) {
            iter = svec.insert(iter, *iter);
            advance(iter, 2); // iter +=2;
        }
        else
            iter++;
    }
    print(svec);
}
// output
ali seda cem can veli arda berkecan
seda seda veli veli arda arda berkecan
---
```

```
--- ms
// oN karmasıklıgı (erase) olmadan o1 karmasıklıgında silme
int main()
{
    vector<int> ivec{1,23,4,5,6,7};

    auto iter = ivec.begin() + 4;
    iter_swap(iter, ivec.end() - 1); // son öğe yapılır
    //swap(*iter, ivec.back())
    ivec.pop_back();
}
---
```

```
--- deque veri yapısı (double ended queue) ---
```

- indeks ile sabit zamanda erişmek
- baştan ve sonran eklemek sabit zaman
- amortised constant time
#include <deque>
- random acces iter
- realloaction yok
- baş ve sondan ekleme yapılmıyosa her şey invalid olur
- ekleme baş veya sondan yapılıyorsa iteratorler invalid olur,
  referanslar invalid olmaz
- deque<bool> fiilen bool tutar, vector bit seviyesinde tutar

--- list ---

- O(n)
- konumu bilinen yere ekleme silme sabit zaman, O(1)
- daha fazla cache miss (önbelleğe çekilmemiş), az performanslı
- swap fonksiyonları sınıf üye fonksiyonları,
  nesneleri, iteratorleri değil düğümleri takas eder
#include<list> çift yönlü bağlı liste
#include<forward_list> tek yönlü bağlı liste // forward iterator category
- iteratorleri bidirectional iterator
- iteratorleri toplama, çıkarma,[] yapılamaz
- iterator invalidation geçerli değil

---> eğer elemanı doğrudan allocated edilmiş adreste oluşturmak için,
  argümanlara sahipse, emplace kullanılmalı, kopyalam veya taşıma
  maliyeti yok!!!
    - nesnesin sadece ctoru çağırılır

```
int main()
{
  list<int> mylist;
  rfill(mylist, 10, rname);
  mylist.reverse();
  mylist.sort();
  mylist.sort(greater{});
```

```
}
---




---
int main()
{
    list<int> mylist;
    rfill(mylist, 10, rname);
    int ival;
    cout << "hangi değer silinecek";
    cin >> ival;
    auto n = mylist.remove(ival);
    //auto n = erase(mylist, ival); //c++20

    cout << "toplam " << n << " öge silindi\n";

}
---




---
int main()
{
    list<int> mylist;
    rfill(mylist, 10, rname);
    int ival;
    cout << "kaca tam bolunenler silinecek";
    cin >> ival;
    auto n = mylist.remove_if([ival](int x) {return x % ival == 0;});
    cout << "toplam " << n << " öge silindi\n";

}
---




---
int main()
{
    list<int> mylist;
    rfill(mylist, 10, rname);
```

```
    auto n = mylist.unique();
    cout << n << "eleman silindi";

}
---
```

```
---
int main()
{
    list<int> mylist;
    rfill(mylist, 10, rname);

    auto n = mylist.unique([](int x, int y){return x % 2 == y % 2;});;
    cout << n << "eleman silindi";

}
---
```

```
---
// list::merge // sıralı birleştirme
int main()
{
    list<int> x, y;
    rfill(x, 10, rname);
    rfill(y, 10, rname);

    x.sort();
    y.sort();

    x.merge(y); // x kendine y' yi katar

    cout << y.size(); // 0
    cout << x.size(); // 20

}
---
```

```
int main()
{
    list<int> x, y;
    rfill(x, 10, rname);
    rfill(y, 10, rname);

    auto fpred = [](const string& s1, const string& s2) {
        return s1.size() < s2.size() || s1.size() == s2.size() && s1 < s2;
    }

    x.sort(fpred);
    y.sort(fpred);

    x.merge(y, fpred);
}
```

```
int main()
{
    list<string> x, y;
    rfill(x, 5, rname);
    rfill(y, 5, rname);

    // x'in basında y' den gelen öğeler olur
    x.splice(x.begin(), y) ;
    cout << y.size(); // 0
    cout << x.size(); // 10
}
```

--- forward_list ---

- tekli bağlı liste
- ekeleme verilen konumdan sonrakine yapar

```
int main()
{
    forward_list<string> mylis{"ali", "ece", "seda", "cem"};
    print(mylis);
    mylis.insert_after(mylis.begin(), "sezen");
    print(mylis);
}
// output
ali ece seda cem
ali sezen ece seda cem
```

```
int main()
{
    forward_list<string> mylis{"ali", "ece", "seda", "cem"};
    print(mylis);
    mylis.push_front("sezen");
    print(mylis);
}
// output
ali ece seda cem
sezen ali ece seda cem
```

--- ms

```
int main()
{ // döngüsüz listenin başına 3 kerem ekle
    forward_list<string> mylis{"ali", "ece", "seda", "cem"};
    print(mylis);
    mylis.insert_after(mylis.before_begin(), 3, "kerem");
    // before_begi() pointerı dereference edilmemeli
    print(mylis);
}
// output
ali ece seda cem
kerem kerem kerem ali ece seda cem
```

```
int main()
{
    forward_list<string> mylist{"ali", "ece", "seda", "cem"};
    print(mylist);
    mylist.erase_after(mylist.begin()); // baştaki öğeden sonrakini siler

    print(mylist);
}
// output
ali ece seda cem
ali seda cem
```

```
int main()
{
    forward_list<string> mylist{"ali", "ece", "seda", "cem"};
    print(mylist);
    mylist.pop_front(); // ilk öğeyi siler
    mylist.erase_after(mylist.before_begin()); // ilk öğeyi siler

    print(mylist);
}
```

```
int main()
{ // döngüsüz listenin başından 3 öğe sil
    forward_list<string> mylist{"ali", "ece", "seda", "cem"};
    print(mylist);

    mylist.erase_after(mylist.before_begin(), next(mylist.begin(), 3));

    print(mylist);
}
// output
ali ece seda cem
cem
```

-> size fonksiyonu yok distance(iter1, iter2) ile öğrebilebilir

--> associative containers
- set
- map
- multiset
- multimap

--- set ve multiset containers ---

- değerle arama Ologn karmasıklığı
- algoritma hali binary searc

equality x == y

equivalance !(x < y) && !(y < x)

-> strict weak ordering (sağlaması gereken kriterler)
- a operation b true, b op a false (op -> <, >...)
- a op a false
- a op b true, b op c true, a op c must be true
- !(a op b) && !(b op a)   !(b op c) && !(c op b)
   !(a op c) && !(c op a)

- set'de bir anahtardan bir tane var, multiset' de birden fazla olabilir

- bidirectional iterator

- iterator konumundaki nesne const

```
template<typename T, typename comp = std::less<T>, typename A = std::allocator<T>>
```

```
set<int> myset; // set<int, less<int>, allocater<int>> myset;
```

```
int main()
{
    set<int> myset;

    for(int i = 0; i < 100; ++i)
        myset.insert(Irand(0, 120)());

    cout << myset.size(); // 80
    // aynı değerleri set etmez
}
```

```
int main()
{
    // initializer_list ctor
    set<int> myset{1, 4, 3, 1, 2, 5, 9, 7};
    print(myset);
}
```

```
// output
1 2 3 4 5 7 9
---




---
template <typename T>
using gset = std::set<T, std::greater<T>>;

int main()
{
    gset<string> myset {"mehmet", "ali", "veli", "zeynep", "hayrettin"};
    print(myset);
}
zeynep veli mehmet hayrettin ali
---




---
int main()
{
    srand(time(nullptr));

    multiset<int> myset;

    for(int i = 0; i < 100; ++i)
        myset.insert(rand() % 100);

    cout << "size = " << myset.size() << "\n"; // 100
    print(myset); // unique değil
}
---




---
int main()
{
    set<int> myset{1, 4, 3, 1, 2, 5, 9, 7};
    auto iter = myset.begin();
    *iter = 10; // syntax error
}
```

--> set'in insert işlevinin geri dönüş değeri,
    pair<iterator, bool> bool ekleme işlevinin yapılıp yapılmadıgı
    - ekleme yapıldıysa ekleme yapılan değere iterator,
      ekleme yapılmadıysa (aynı öğeden var demek) var olana iterator

---

```
int main()
{
    set<string> myset {"mehmet", "ali", "veli", "zeynep", "hayrettin"};

    string name;
    cout << "ekleme yapilacak isim: ";
    cin >> name;

    // structured binding ve if with initializer
    if(auto [iter, flag] = myset.insert(name); flag) {
        cout << *iter << " eklendi\n";
    }
    else
        cout << "set'te var\n";
}
```

```
int main()
{
    set<string> myset {"mehmet", "ali", "veli", "zeynep", "hayrettin"};

    string name;
    cout << "isim giriniz: ";
    cin >> name;

    // arama alg. 1. yol
    // count ile nesneye erişilemez
    if(myset.count(name)) {
```

```
        cout <<  "bulundu\n";
    }
    else
        cout << "yok\n";
}
---
```

```
---
int main()
{
    set<string> myset {"mehmet", "ali", "veli", "zeynep", "hayrettin"};

    string name;
    cout << "isim giriniz: ";
    cin >> name;

    // arama alg. 2. yol
    if(auto iter = myset.find(name); iter != myset.end()) {
        cout <<  "bulundu : "<< *iter << "\n";
    }
    else
        cout << "yok\n";
}
// output  isim giriniz: ali
bulundu : ali
---
```

```
---
int main()
{
    set<string> myset {"mehmet", "ali", "veli", "zeynep", "hayrettin"};

    string name;
    cout << "isim giriniz: ";
    cin >> name;

    // arama alg. 3. yol
    if(myset.contains(name))) { // c++20
        cout <<  "bulundu : \n";
```

```
    }
    else
        cout << "yok\n";
}
---
```

```
---
int main()
{
    set<string> myset {"mehmet", "ali", "veli", "zeynep", "hayrettin"};

    string name;
    print(myset);

    myset.erase(myset.begin()); // ilk öğeyi siler
    myset.erase(prev(myset.end())); // son öğeyi siler
    myset.erase(next(myset.begin()), prev(myset.end())); // ilk ve son öğe hariç siler

    myset.erase("veli") ; // varsa geri dönüs değeri 1 yoksa 0
    // multiset olursa kaç tene varsa onu döner
}
---
```

```
--- ms
// bir anahatrı değitirme ?
// c++17 öncesi


int main()
{
    set<string> myset {"mehmet", "ali", "veli", "zeynep", "hayrettin"};

    string old_key, new_key;
    cout << "eski ve yeni isimleri girin: ";
```

```
    cin >> old_key >> new_key;

    auto iter = myset.find(old_key);
    if(iter != myset.end()) {
        myset.erase(iter);
        myset.insert(new_key);
    }
    else
        cout << "bulunamadi\n";

    print(myset);
}
// output
eski ve yeni isimleri girin: ali
cem
cem hayrettin mehmet veli zeynep
---




---
// bir anahatrı değitirme ?
// c++17 sonrası


int main()
{
    set<string> myset {"mehmet", "ali", "veli", "zeynep", "hayrettin"};

    string old_key, new_key;
    cout << "eski ve yeni isimleri girin: ";
    cin >> old_key >> new_key;

    if(auto iter = myset.find(old_key); iter != myset.end()) {
        auto handle = myset.extract(iter); // düğümü çıkartır
        handle.value() = new_key; // destructor çağırılmaz
        myset.insert(move(handle)); // taşıma ile
    }
    else
        cout << "bulunamadi\n";

    print(myset);
}

// VEYA

int main()
```

```
{
    set<string> myset {"mehmet", "ali", "veli", "zeynep", "hayrettin"};

    string old_key, new_key;
    cout << "eski ve yeni isimleri girin: ";
    cin >> old_key >> new_key;

    if(auto handle = myset.extract(old_key); handle) {
        handle.value() = new_key; // destructor çağırılmaz
        myset.insert(move(handle)); // taşıma ile
    }
    else
        cout << "bulunamadi\n";

    print(myset);
}
// output
eski ve yeni isimleri girin: veli
kerem
ali hayrettin kerem mehmet zeynep
---
```

```
---

int main()
{
    set<string> myset {"mehmet", "ali", "veli", "zeynep", "hayrettin"};

    string name;

    cout << "eklenecek isim";
    cin >> name;

    myset.insert(myset.begin(), name); // hint
    // garantisi yok, eklenebilirse ekler
    // maliyet avantajı
}
---
```

```
int main()
{
   set<string> myset {"mehmet", "ali", "veli", "zeynep", "hayrettin"};

   myset.emplace(10, 'a');
   myset.emplace_hint(myset.begin(), 10, 'a'); // konum veriliyorsa
   print(myset);
}
```

--- sıralanmış öğelerde kullanılan algoritmalar
- lower_bound
- upper_bound
- equal_range

-> set map, multimap, multiset containerlarında üye fonksiyonlar

--> lower_bound: verilen anahtara eşit veya büyük ilk konum
- sırayı bozmadan eklenebilecek ilk konum

```
int main()
{
   srand(time(nullptr));
   multiset<int> myset;

   int n = 20;
   while(n--)
```

```
    {
        myset.insert(rand() % 10);
    }

    int x;
    cout << "tam sayi girin: ";
    cin >> x;

    auto iter = myset.lower_bound(x);
    cout << "lower_bound icin idx : " << distance(myset.begin(), iter) << "\n";


    print(myset);
}
// output
tam sayi girin: 7
lower_bound icin idx : 16
0 1 1 3 3 3 3 3 3 4 5 5 5 6 6 6 7 8 9 9
---
```

--> upper_bound > anahtarı olan ilk konum
  - key'i sırayı bozmadan eklenebilecek son konum

```
int main()
{
    srand(time(nullptr));
    multiset<int> myset;

    int n = 20;
    while(n--)
    {
        myset.insert(rand() % 10);
    }

    int x;
    cout << "tam sayi girin: ";
    cin >> x;
```

```
    auto iter = myset.upper_bound(x);
    cout << "lower_bound icin idx : " << distance(myset.begin(), iter) << "\n";


    print(myset);
}
// output
tam sayi girin: 1
lower_bound icin idx : 5
0 0 0 0 1 2 2 2 3 3 3 4 5 5 6 6 6 6 7 9
---
```

[lover_bound, upper_bound) => equal_range

```
---
int main()
{
    srand(time(nullptr));
    multiset<int> myset;

    int n = 20;
    while(n--)
    {
        myset.insert(rand() % 10);
    }

    int x;
    cout << "tam sayi girin: ";
    cin >> x;

    auto iter_u = myset.upper_bound(x);
    auto iter_l = myset.lower_bound(x);
    cout << "distance for equal range : " << distance(iter_l, iter_u) << "\n";

    print(myset);
}
// output
tam sayi girin: 3
distance for equal range : 4
0 0 1 2 2 2 3 3 3 3 4 4 5 5 5 6 6 7 8 9
---
```

```
-> 3'ü de  aynı anlam
// pair<multiset<int>::iterator, multiset<int>::iterator> ip = myset.equal_range(x);
// auto ip = myset.equal_range(x);
// auto [iter_lower, iter_upper] = myset.equal_range(x);
```

```
// algoritm
    vector<int> ivec;
    // ...
    auto iter = lower_bound(ivec.begin(), ivec.end(), "ali");
    print(ivec);
```

```
// algoritm
    auto [iter_lower, iter_upper] = equal_range(ivec.begin()), ivec.end(), 9);
```

```
// en küçük öğenin konumu döner
    auto iter_min = min_element(svec.begin(), svec.end());
```

```
// en büyük öğenin konumu döner
    auto iter_max = max_element(svec.begin(), svec.end());
```

```
// en büyük ve en küçük öğeyi bulur pari döner
    auto [itermin, itermax] = minmax_element(svec.begin(), svec.end());
    cout << "min = " << *itermin << " max = " << *itermax;
```

--- ms
// vector sürekli sıralı olacak şekilde öğe ekleme

```
int main()
{
    vector<string> svec;

    for(int i = 0; i < 10; ++i) {
        auto s = rname();
        cout << s << "\n";
        auto iter = lower_bound(svec.begin(), svec.end(), s);
        svec.insert(iter, s);
    }
}
```

--- map ---
- include <map>
- logn
- ikili arama ağacı
- sadece key tutmak yerine value' da tutar
- map ve multi_map öğeler pairler, set, multiset sadece key
- belirli bir anahtara karşılık değere erişmek için kullanılır
  isimlerden telefon numarasına erişmek vs...
- map, multimap ve set multiset arasındaki fark aynı

```
template<typename Key, typename Value, typename Comp = less<Key>, typename A =
allocater<pair<Key, Value>>>
class map {};


map<int, double> x;
//map<int, double, less<int>, allocater<pair<int, double>> x;




int main()
{
    map<int, string> ismap;
    ismap.insert(pair<int, string> {125, "ali"});
    ismap.insert(make_pair(99, "veli"));
    ismap.insert({50, "zeliha"}); //c++17

    ismap.emplace(55, "ece");

    for(auto iter = ismap.cbegin(); iter != ismap.cend(); ++iter) {
        cout << iter->first << " " << iter->second << "\n";
    }

    // VEYA

    for( const auto &p : ismap) {
        cout << p.first << " " << p.second << "\n";
    }

    // VEYA

    for( const auto& [age, name] : ismap) {
        cout <<age << " " << name << "\n";
    }
}




// initializer_list ctor
map<string, int> mymap { {"ali", 25}, {"seda", 50}, {"cem", 10} };



//range ctor
vector<pair<string, int>> myvec { {"ali", 25}, {"seda", 50}, {"cem", 10} };
```

```
map<string, int> mymap{myvec.begin(), myvec.end()};
```

-> print pair

```
template<typename T, typename U>
std::ostream& operator<<(std::ostream& os, const std::pair<T, U>& p)
{
    return os << "[" << p.first << ", " << p.second << "]";
}


template<typename C>
void print(const C& c, const char* p = " ", std::ostream& os = std::cout)
{
    for(const auto& elem : c)
        os << elem << p;
}
```

```
template<typename C, typename F>
void rfill(C& con, size_t n, F f)
{
    while(n--)
        con.insert(con.end(), f());
}
```

```
int main()
{
    map<string, int> mymap;

    rfill(mymap, 10, []{return make_pair(rname(), rand());});

    string name;
    cout << "bir isim girin: ";
    cin >> name ;

    mymap[name] = 9999; // map' de tutulan öğenin second' ına referans
    // değeri 9999 yapar
    // eğer aranan değer yoksa o türden pair nesesi ekler
    // mymap.insert({key, int{}}); // yoksa
    // multimap' in [] yok
}
```

```
--- ms
// aynı isimden(değerden) kaç tane oldugunu sayma
int main()
{
    vector<string> svec;

    rfill(svec, 1000, rname);

    map<string, int> cmap; // int değerden kaç adet oldugunu tutucak

    for(const auto& s : svec) {
        ++cmap[s];
    }

    for(const auto& [name, count] : cmap)
        cout << name << " "  << count;

}
```

--- ms
```
int main()
{
    // tutulan anahtarları değerlerine göre sıralama
    vector<string> svec;

    rfill(svec, 1000, rname);

    map<string, int> cmap;

    vector<pair<string, int>> myvec {cmap.begin(), cmap.end()};

    auto pred = [](const auto& p1, const auto& p2) {
        return p2.second < p1.second;
    };

    sort(myvec.begin(), myvec.end(), pred);

    for(const auto& [name, count] : myvec)
        cout << name << " "  << count;

}
```

```
// anahtar (key) değeri değiştirme
// c++17 öncesi
int main()
{
    map<string, int> mymap;

    rfill(mymap, 10, []{return make_pair(rname(), rand()%1000);});

    cout << "eski ve yeni anatarı girin: ";
    string oldkey, newkey;
    cin >> oldkey >> newkey;

    if(auto iter = mymap.find(oldkey); iter != mymap.end()) {
        auto val = iter->second;
        mymap.erase(iter);
```

```
        mymap.insert({newkey, val});
        cout << "anahtar degistirildi\n";
        print(mymap, "\n");
    }
    else
        cout << "bulunamadi\n";
}
---




---
int main()
{
    // c++17

    map<string, int> mymap;

    rfill(mymap, 10, []{return make_pair(rname(), rand()%1000);});

    cout << "eski ve yeni anatarı girin: ";
    string oldkey, newkey;
    cin >> oldkey >> newkey;

    if(auto iter = mymap.find(oldkey); iter != mymap.end()) {
        auto handle = mymap.extract(iter);
        handle.key() = newkey;
        mymap.insert(std::move(handle));// nesne destroy edilme<
        print(mymap, "\n");
    }
    else
        cout << "bulunamadi\n";
}
---




--- insert_or_assign
int main()
{
    map<string, int> mymap;
    // ...
    mymap.insert_or_assign("ali", 750);
    // maliyeti azaltmaya yönelik bir fonksiyon
    // pairi oluşturmadan map' de o değerden var mı kontrol eder
    // varsa nesne oluşturma ve yok etme maliyetinden kaçınır
```

```
    // yoksa pair'i oluştutur ve ekler // insert()
}
---




mymap.at(name) = 150; // c++17, yoksa exception throw eder
// mymap[name] = 150;
```

---> unordered associative containers
- equality
--> h table veri yapısı
  - unordered_set
  - unordered_multiset
  - unordered_map

- unordered_multimap

- include<unordered_set>
-> anahtarla erişimin O(1)

```
template <typename T, typename H = std::hash<T>,
    typename E = std::equal_to<T>, typename A = std::allocator<T>>
```

```
template<typename T>
struct Hash {
    size_t operator()(const T&)const;
};
```

```
int main()
{   // hasher function

    hash<string> hasher_str;
    hash<int> hasher_int;

    cout << hasher_int(196546) << "\n";
    cout << hasher_int(196546) << "\n";
    cout << hasher_int(196547) << "\n";

    cout << hasher_str("ali") << "\n";
    cout << hasher_str("ali") << "\n";
    cout << hasher_str("veli") << "\n";

    cout << hash<string>{}("yunus") << "\n";
    // temporary object
}
// output
196546
196546
196547
11306657118794567581
11306657118794567581
6855463604528545353
---
```

```
namespace std{
   template<>
   struct hash<Date> {
      size_t operator()(const Date& d)const
      {
         std::hash<int> hasher;
         return hasher(d.get_year()) + hasher(d.get_month()) + hasher(d.get_month_day());
      }
   };
}


int main()
{
   unordered_set<Date> myset;
   for(int i = 0; i < 100; ++i) {
      myset.insert(Date{}.random_date());
   }

}
```

-> Date nesnesinin operator==() olmalı

```
struct DateHasher {
   size_t operator()(const Date& d)const
      {
         std::hash<int> hasher;
         return hasher(d.get_year()) + hasher(d.get_month()) + hasher(d.get_month_day());
      }
};
```

```
struct DateEqual {
        bool operator()(const Date& dx, const Date& dy)const
        {
            return dx.get_year() == dy.get_year() &&
            dx.get_month() == dy.get_month() &&
            dx.get_month_day() == dy.get_month_day();
        }
};

int main()
{
    unordered_set<Date, DateHasher, DateEqual> myset;
    for(int i = 0; i < 100; ++i) {
        myset.insert(Date{}.random_date());
    }

}
---
```

```
---
int main()
{
    vector<string> svec;
    rfill(svec,1000, rname);

    unordered_map<string, int> cmap;

    for(const auto& name : svec)   // her bir isimdan kaç tane var
        ++cmap[name];

    vector<pair<string, int>> vec{cmap.begin(), cmap.end()};
    sort(vec.begin(), vec.end(),
    [](const auto& px, const auto& py) {
        return px.second > py.second || (px.second == py.second && px.first < py.first);
    });

    for(const auto& [name, count] : vec)
        cout << name << " " << count;
}
---
```

```
int main()
{
    unordered_set<int> us(128); // bucket sayısı ctor

     for(int i = 0;i < 100; ++i)
         us.insert(get_random_number());

    cout << "bucket count: " << us.bucket_count()<< "\n"; // 137
    cout << "size:         " << us.size()<< "\n";  // 65

    // load factor
    cout <<  (double)us.size() / us.bucket_count()<< "\n";  // 0.47445
    cout << "load factor: " << us.load_factor() << "\n"; // 0.47445

    cout << "max load factor: " << us.max_load_factor() << "\n"; // 1
}
// output
bucket count: 137
size:         65
0.474453
load factor: 0.474453
max load factor: 1
```

--> max load factor'e eriştiğinde rehash yapılacak, bucket sayısı arttırılacak,
    tüm öğeler yeniden konumlandırılacak

- max load factor değiştirilebilir

```
    us.max_load_factor(0.75f);
```

```
int main()
{
   std::unordered_set<std::string> us(128);
   for(int i = 0; i<100; ++i)
      us.insert(rname());

   std::ofstream ofs{"out.txt"};
   if(!ofs) {
      std::cerr<< "out.txt dosya olusturulamadi\n";
      exit(EXIT_FAILURE);
   }

   ofs << "bucket count: " << us.bucket_count()<< "\n"; // 137
   ofs << "size:         " << us.size()<< "\n";  // 65
   ofs << "load factor: " << us.load_factor() << "\n"; // 0.47445
   ofs << "max load factor: " << us.max_load_factor() << "\n"; // 1

   for(int i = 0; i < us.bucket_count(); ++i) {
      ofs << "|" << std::setw(2) << i << "| [" << us.bucket_size(i) << "]\n";
   }

}
// out.txt
bucket count: 137
size:         84
load factor: 0.613139
max load factor: 1
| 0| [1]   // ilk değer bucket index, ikincisi kaç tane elemanı oldugu
| 1| [2]
| 2| [1]
| 3| [1]
| 4| [0]
| 5| [1]
| 6| [0]
| 7| [0]
| 8| [1]
| 9| [0]
|10| [0]
|11| [0]
//...
---




---
```

```
int main()
{
    std::unordered_set<std::string> us(128);
    for(int i = 0; i<100; ++i)
        us.insert(rname());

    std::ofstream ofs{"out.txt"};
    if(!ofs) {
        std::cerr<< "out.txt dosya olusturulamadi\n";
        exit(EXIT_FAILURE);
    }

    ofs << "bucket count: " << us.bucket_count()<< "\n"; // 137
    ofs << "size:         " << us.size()<< "\n";  // 65
    ofs << "load factor: " << us.load_factor() << "\n"; // 0.47445
    ofs << "max load factor: " << us.max_load_factor() << "\n"; // 1

    for(int i = 0; i < us.bucket_count(); ++i) {
        ofs << "|" << std::setw(2) << i << "| [" << us.bucket_size(i) << "] ";
        for(auto iter = us.begin(i); iter != us.end(i); ++iter)
            ofs << *iter <<" ";
        ofs << "\n";
    }

}
// out.txt
bucket count: 137
size:         84
load factor: 0.613139
max load factor: 1
| 0| [1] cemil
| 1| [2] saadet ahmet // ahmet için bir sorgulama daha yapılır
| 2| [1] atalay
| 3| [1] egemen
| 4| [0]
| 5| [1] tansel
---
```

```
us.rehash(400); // bucket sayısı 400
us.reserve(400) // max count tane bucket düzenlenebilir(öğe sayısı)
std::cout << us.bucket_size(1) << "\n"; // bucket'ta kaç öğe var onu döner
```

--- container adaptor ---

--> abstract data type

- stack (last in first out)
- queue (first in first out)
- priority_queue (öncelik yüksek olan ilk çıkar)
  heap

-> container değiller, yardımcı sınıflar

<algorithm>
- make_heap
- push_heap
- pop_heap
- sort_heap

-- stack --

```
#include <stack>
```

- bir sınıf şablonu
std::stack<int> x; //std::stack<int, std::deque<int>> x;

```
// template alias
template<typename T>
using vstack = std::stack<T, std::vector<T>>;
int main()
{
   vstack<int> vx;
}
```

```
int main()
{
   stack<int> s{1,4,5,7,90}; // syntax error
   // init ctor yok
}
```

```
template<typename T, typename C = std::deque<T>>
class Stack {
protected:
   C c;
};
```

```
int main()
{
   std::stack<int> s;

   for(int i = 0; i < 10; ++i)
      s.push(i);

   std::cout << "size = " << s.size()<< "\n"; // 10
   s.top() = 40; // en üstteki öğeye referance ile eriştirir
   s.pop() // void geri dönüşlü, stack' den çıkartır
}
```

```
int main()
{
   std::stack<std::string> name_stack;

   std::string s;

   for(int i = 0; i< 10; ++i) {
      s = rname();
      std::cout << s << "yiğina push ediliyor\n";
      name_stack.push(move(s)); // taşıma ile
      // moved from state (destructor çağırılmaz )
         // kaynağı çalışnımş nesne geçerlidir, kullanılabilir,
         // Değeri hakkında garanti yok, ub yok
         // yeniden atama yapılabilir
   }
}

// moved from state olmasaydı
int main()
{
   std::stack<std::string> name_stack;



   for(int i = 0; i< 10; ++i) {
      std::string s = rname(); // döngün her turunda copy elision olucaktı
      // döngünün her turunda s destroy olucaktı
      std::cout << s << "yiğina push ediliyor\n";
      name_stack.push(move(s)); // taşıma ile

   }
}
```

```
int main()
{
    std::stack<std::string> name_stack;



    for(int i = 0; i< 5; ++i) {
        std::string s = rname();
        std::cout << s << " yigina push ediliyor\n";
        name_stack.push(move(s));
    }
    std::cout << "size = " <<  name_stack.size() << "\n";

    while(!name_stack.empty()) {
        std::cout << name_stack.top() << " yigindan cikartiliyor\n";
        name_stack.pop();
    }
}
// output
berk yigina push ediliyor
hilal yigina push ediliyor
ahmet yigina push ediliyor
metin yigina push ediliyor
melih yigina push ediliyor
size = 5
melih yigindan cikartiliyor
metin yigindan cikartiliyor
ahmet yigindan cikartiliyor
hilal yigindan cikartiliyor
berk yigindan cikartiliyor
---
```

```
---
class Mystack : public std::stack<int> {
public:
    //
    int& back()
    {
        return c.front();
```

```
        // c, container veri elemanı
        // isminin c olması garanti

    }
}
---




template<typename T, typename C = std::deque<T>>
class Stack {
public:
    void push(const T& tval)
    {
        c.push_front(tval);
    }
    T& top()
    {
        return c.front();
    }

    //...
protected:
    C c;
};









---
int main()
{
    std::deque dx = {1,4,5,7,8,4,2,10};
    // stack' in int açılımının bir ctoru deque' nin bir açılımı
    std::stack mystack(dx);

    while(!mystack.empty()) {
        std::cout << mystack.top() << " ";
```

```
        mystack.pop();
    }
}
// output
10 2 4 8 7 5 4 1
---



---
int main()
{
    std::vector dx = {1,4,5,7,8,4,2,10};

    std::stack mystack(dx); // stack<int, vector<int>>

    while(!mystack.empty()) {
        std::cout << mystack.top() << " ";
        mystack.pop();
    }
}
---
```

-- queue --

#include<queue>
  - priority_queue
  - queue

- template class

- default template argümanı deque
- kalıtımla c isimli container nesnesi kullanılabilir

```
int main()
{
    queue<string> x;
    for(int i = 0; i< 5; ++i) {
        auto s = rname();
        cout << s << " kuyruga giriyor\n";
        x.push(move(s));
    }

    cout << "kuyrukta : " << x.size() << " kisi var\n";
    cout << "kuyruk basinda : " << x.front() << " var\n";
    cout << "kuyruk sonunda : " << x.back() << " var\n";

    while(!x.empty()) {
        cout << x.front() << " kuyruktan cikiyor\n";
        x.pop();
    }
}

// output
berk kuyruga giriyor
hilal kuyruga giriyor
ahmet kuyruga giriyor
metin kuyruga giriyor
melih kuyruga giriyor
kuyrukta : 5 kisi var
kuyruk basinda : berk var
kuyruk sonunda : melih var
berk kuyruktan cikiyor
hilal kuyruktan cikiyor
ahmet kuyruktan cikiyor
metin kuyruktan cikiyor
melih kuyruktan cikiyor
```

```
int main()
{
    queue<string, vector<string>> x;
    // syntax error
    // vector' un pop_font fonksiyonu yok
}
```

-- priority_queue --
- heap data structer
- heap haliene getirmek O(n)
- heapi sıralama onlogn

```
int main()
{
    vector<int> ivec;
    rfill(ivec, 16, get_random_number);
    print(ivec);
    make_heap(ivec.begin(), ivec.end()); // default less
    print(ivec);
}

// output
41 467 334 500 169 724 478 358 962 464 705 145 281 827 961 491
962 705 961 500 464 724 827 491 467 41 169 145 281 334 478 358
```

```
int main()
{
    vector<int> ivec;
    rfill(ivec, 16, get_random_number);
    print(ivec);
    make_heap(ivec.begin(), ivec.end());
    print(ivec);

    cout << "max = " << ivec.front() << "\n";
    pop_heap(ivec.begin(), ivec.end()); // silinecek öğeyi vektorün en sonuna getirdi
    ivec.pop_back(); // son elemanı sildi, heap yapısı korundu
    print(ivec);
}

// output
41 467 334 500 169 724 478 358 962 464 705 145 281 827 961 491
962 705 961 500 464 724 827 491 467 41 169 145 281 334 478 358
max = 962
961 705 827 500 464 724 478 491 467 41 169 145 281 334 358
```

```
int main()
{
    vector<int> ivec;
    rfill(ivec, 16, get_random_number);
    print(ivec);
    make_heap(ivec.begin(), ivec.end());
    print(ivec);

    ivec.push_back(500);
    push_heap(ivec.begin(), ivec.end());
    print(ivec);
}

// output
41 467 334 500 169 724 478 358 962 464 705 145 281 827 961 491
```

```
962 705 961 500 464 724 827 491 467 41 169 145 281 334 478 358
962 705 961 500 464 724 827 500 467 41 169 145 281 334 478 358 491
---
```

```
---
int main()
{
    vector<int> ivec;
    rfill(ivec, 16, get_random_number);
    print(ivec);
    make_heap(ivec.begin(), ivec.end());
    print(ivec);

    ivec.push_back(500);
    push_heap(ivec.begin(), ivec.end());
    print(ivec);
    // heap'i sıralamak icin..
    sort_heap(ivec.begin(), ivec.end()); // O(nlongn)
    print(ivec);
}

// output
41 467 334 500 169 724 478 358 962 464 705 145 281 827 961 491
962 705 961 500 464 724 827 491 467 41 169 145 281 334 478 358
962 705 961 500 464 724 827 500 467 41 169 145 281 334 478 358 491
41 145 169 281 334 358 464 467 478 491 500 500 705 724 827 961 962
---
```

```
---
int main()
{
    vector<int> ivec;
    rfill(ivec, 10, get_random_number);
    print(ivec);

    make_heap(ivec.begin(), ivec.end()); // O(n)
```

```
    while(!ivec.empty()) { // onlogn algoritmayla öcelikli olan çıkartılır
        pop_heap(ivec.begin(), ivec.end());
        cout << "onceligi yuksek olan " << ivec.back() <<" kuyruktan cikiyor\n";
        ivec.pop_back();
        getchar();
    }

}

// output
41 467 334 500 169 724 478 358 962 464
onceligi yuksek olan 962 kuyruktan cikiyor
onceligi yuksek olan 724 kuyruktan cikiyor
onceligi yuksek olan 500 kuyruktan cikiyor
---
```

---> random acces iterator veran her hangi bir container' ı
make_heap(), sort_heap(), pop_heap()... fonksiyonlarıyla manipüle edilebilir

```
priority_queue<int> x;
// priority_queue<int, vector<int>, less<int>> x;
```

```
template<typename T, typename C = std::vector<T>, typename Comp = std::less<T>>
class PriorityQueue {
public:
    void push(const T& tval)
    {
        c.push_back(tval);
        push_heap(c.begin(), c.end());
    }

    T& top()
    {
        return c.front();
    }

    void pop()
```

```
    {
        pop_heap(c.begin(), c.end());
        c.pop_back();
    }

protected:
    C c;
};
```

```
int main()
{
    priority_queue<Date, vector<Date>, greater<Date>> x;

    for(int i = 0; i < 10; ++i)
        x.push(Date().random_date());

    while(!x.empty()) {
        cout << x.top() << "\n";
        x.pop();
        getchar();
    }
} // output
7 June 1901
24 August 1907
16 June 1922
13 December 1941
15 April 1949
---
////////////////////////////////////////////////////////////
```

--- reference wrapper ---

- referanslar containerlarda tutulamaz, rebandleble değiller
- referance wrapper bir pointer'ı sarmalar, rebandleble yapısı oluşturur
- functional başlık dosyasında referance_wraper sınıfı var
- sağ taraf değerine bağlanamaz

***

```
template< typename T>
class ReferenceWrapper {
public:
   ReferenceWrapper(T &r) : mptr{&r} {}
   ReferenceWrapper& operator=(T& r) // yeni öğeyi tutar
   {
      mptr = &r;
      return *this;
   }

   T& get()
   {
      return *mptr;
   }

   operator T&() // tür dönüştürme
   {
      return *mptr;
   }
private:
   T* mptr;
};

int main()
{
   int x = 10;
   int y = 20;
   ReferenceWrapper<int> r = x;
```

```
    r = y;
    int ival = 10 + r;
    cout << ival << "\n"; // 30
}

// fonksiyonlu hali

template< typename T>
ReferenceWrapper<T> Ref(T& t)
{
    return ReferenceWrapper<T>{t};
}

int main()
{
    int x = 10;
    int y = 20;
    Ref(x);
}
***
```

```
--- reference_wrapper
int main()
{
    int x = 10;
    int y = 20;

    reference_wrapper<int> r = x;

    ++r;
    cout << "x = " << x << "\n";

    r.get() = 100;
    cout << "x = " << x << "\n";

    r = y;
    cout << "r = " << r << "\n";

    ++r;
    cout << "y = " << y << "\n";
}
```

```
// output
x = 11
x = 100
r = 20
y = 21
---
```

```
---
template<typename T>
void func(T x)
   {
      ++x;
   }
int main()
{
   int ival = 1;
   func(ival);
   cout << ival << "\n"; // 1

   func(ref(ival)); // std::reference_wrapper<int>
   cout << ival << "\n"; // 2

   func<int &>(ival);
   cout << ival << "\n"; // 3
}
---
```

```
---
int main()
{
   vector<int> ivec(100);
   // ...
   mt19937 eng;
   generate(ivec.begin(), ivec.end(), eng); // kopyalama maliyeti
   generate(ivec.begin(), ivec.end(), ref(eng)); // ref sematiği ile
}
```

```
int main()
{
    int x = 10, y = 20;

    auto p1 = make_pair(x, y); // int, int
    p1.first = 1;
    p1.second = 2; // x ve y değişmez, call by value
    cout << "x, y = " << x <<", " <<  y<< "\n";


    pair<int&, int&> p2(x, y);
    p2.first = 1;
    p2.second = 2;
    cout << "x, y = " << x <<", " <<  y<< "\n";

    auto p3 = make_pair(ref(x), ref(y));
    p3.first = 7;
    p3.second = 5;
    cout << "x, y = " << x <<", " <<  y<< "\n";

}
// output
x, y = 10, 20
x, y = 1, 2
x, y = 7, 5
```

```
int main()
{
    list<int> mylist{10,40,50,70,80};
    vector<reference_wrapper<int>> myvec{mylist.begin(), mylist.end()};
```

```
    for(auto &x : myvec)
        ++x;

    for(auto &x : mylist)
        cout << x << " " ;
}
// output
11 41 51 71 81
---




---
int main()
{
    int x = 10;
    reference_wrapper r = x; // CTAD c++17
}
---



---
int foo(int x)
{
    //..
    return 10 * x;
}
int main()
{

    // reference_wrapper<int(int)> r = foo;
    reference_wrapper r = foo; // CTAD
    cout << r(100) << "\n"; // 1000
}
---



-> cref(x); // const
```

--- std::function ---

#include <functional>

- belirli bir parametrik yapıdaki tüm callable' ları
  sarmalayan sınıf şablonu

- callable'ları container' da tutmak

```
int foo(int x)
{
    cout << "foo cagirildi\n";
    return 3 * x;
}

int main()
{
    std::function<int(int)> f(foo);
    //
    cout << f(50);
}
// output
foo cagirildi
150
```

```
int main()
{
    std::function<int(int)> f([](int val){return val * val;});
    //
    cout << f(5);
}
```

```
// output
25
---
```

```
---
class Functor {
public:
    int operator()(int x)
    {
        return x * x * x - 1;
    }
};

int main()
{
    Functor fx;
    function<int(int)> f{fx};
    cout << fx(10) ; // 999

    f = [](int a){return a + 5;};
    cout << f(10); //15
}
---
```

```
---
int main()
{
    function<int(int)> f;
    try {
        f(10);
    }
    catch(const std::exception& ex) {
        cout << "exception caught : " << ex.what() << "\n";
    }
}
// output
exception caught : bad_function_call
---
```

```
void func(int(*fptr)(int)) // sadece fonksiyon adresi
{
   ///
   auto val = fptr(12);
}
```

```
void func(std::function<int(int)> f) // herhangi bir callable
{
   ///
   auto val = f(12);
}

int foo(int);

int main()
{
   func(foo);
   func(Functer{});
   func([](int x){return x + 1;});
}
```

```
// callable tutar
vector<function<double(double)>> myvec;
```

```
double f1(double);
double f2(double);
double f3(double);

int main()
{
```

```
    vector<function<double(double)>> myvec;
    myvec.emplace_back(f1);
    myvec.emplace_back(f2);
    myvec.emplace_back([](double d){return d*d;});
    myvec.push_back(function<double(double)>{f3});

    for (auto f : myvec) {
        auto n = f(1.1);
        //
    }
}
---
```

```
--- std::tuple ---

#include <tuple>

---
int main()
{
    tuple<int, double, long> t{12, 5.2, 5L};
    cout << get<0>(t) << "\n";
    cout << get<1>(t) << "\n";
    cout << get<2>(t) << "\n";

}
// output
12
5.2
5
---
```

```
tuple t{12, 5.2, 5L}; // CTAD
```

```
tuple t{12, 5.2, "ali"}; // std::tuple<int, double, const char *> t

tuple t{12, 5.2, "ali"s}; //std::tuple<int, double, std::string> t
// user defined literal
```

-> get fonksiyonun geri dönüş değeri referans

```
tuple t{12, 5.2, "ali"};
get<0>(t) = 50;
```

```
tuple t{12, 5.2, "ali"s};

get<int>(t) = 20;
get<double>(t) = 2.5;
get<string>(t) = "veli";
// birden fazla int olursa syntax error
```

```
using Age = int;
using Name = string;
using Wage = double;
using BirthDate = Date;

using PersonInfo = tuple<Age, Name, Wage, BirthDate>;

int main()
{
    PersonInfo info;
    get<Age>(info);
    get<Name>(info);
    get<Wage>(info);
}
```

```
std::tuple<int, double, string> foo(int x, double dval, const char *p)
{
    // ...
    return {x, dval, p};
}
```

```
int main()
{
    int x = 1;
    double d = 1.5;
    Date mydate{1,1,2000};

    auto t = make_tuple(x, d, mydate);

    cout << get<Date>(t); // 1 January 2000
}
```

```
std::tuple<int, double, string> func()
{
    return {50, 48.4, "veli"};
}

int main()
{
    auto t = func();
    int ival = get<0>(t);
    double d = get<1>(t);
    string s = get<2>(t);
    // yerine ...

    auto [id, wage, name] = func();
}
```

```
int main()
{
    int age;
    double wage;
    string name;

    // tie(...) fonksiyonu referance ile alır değerleri değiştirebilir
    // global variadic fonsksiyon şablonu
    tie(age, wage, name) = func();

    cout << age << "\n";
    cout << wage << "\n";
    cout << name << "\n";
}
// output
```

```
50
48.4
veli
---
```

```
// ignore global fonksiyon, öğeyi göz ardı eder
tie(age, ignore, name) = func();
```

-> structure binding ile ignore yapılamaz

```
int main()
{
    tuple t1{12, 5.2, "ali"s};
    tuple t2{12, 5.2, "veli"s};
    cout << (t1 < t2) << "\n"; // 1
}
```

```
---
using Person = std::tuple<int, std::string, double>;

int main()
{
    std::vector<Person> vec;
    vec.reserve(100);
```

```
    for(int i = 0; i < 100; ++i) {
        vec.push_back(Person{rand(), rname(), Drand{1.5, .7}()}); // geçici nesne ile
       //veya
        vec.push_back({rand(), rname(),  Drand{1.5, .7}()});
        // veya
        vec.emplace_back(rand(), rname(),  Drand{1.5, .7}());
    }

    for(const auto& p : vec) {
        std::cout << get<0>(p) <<" "<< get<1>(p) << " " << get<2>(p);
    }
}
---
```

```
---
class Myclass {
public:
    friend bool operator<(const Myclass& left, const Myclass& right)
    {
        return std::tuple{left.mx, left.str, left.dval} < std::tuple{left.mx, left.str, left.dval};
    }

private:
    int mx;
    std::string str;
    double dval;
};
---
```

```
---
int main()
{
    // fonksiyon şablonu
    constexpr auto n1 = tuple_size<tuple<int, double>>::value;
    // veya
    constexpr auto n2 = tuple_size_v<tuple<int, double>>;
```

```
    tuple x = {12, 4.5, 'a'};
    constexpr auto n3 = tuple_size_v<decltype(x)>;
}
---
```

```
---
int main()
{
    tuple x = {12, 4.5, 'a'};
    tuple_element<0, decltype(x)>::type ival{}; // ival -> int
    tuple_element<2, decltype(x)>::type c{}; // c -> char
    // veya
    tuple_element_t<1, decltype(x)> dval{}; // dval -> double
}
---
```

--- std::array ---

- bir sınıf şablonu
- varlık nedeni C dizilerini sarmalamak
- STL' e daha iyi suyum sağladığından,
    interface' e sahip ,

```
    bazı yerlerde exception throw edebiliyor,
    array decay söz konusu değil,
    bir fonksiyonun geri dönü değeri olabilir,
    kopyalama sematiği kazandırıldığından tercih edilir
#include <array>
- ekstra maliyet yok
- random acces iterator
- get<>() interface' ini destekliyor


---
template<typename T, size_t n>
struct Array {
    T ar[n];
    //
};
int main()
{
    // structerlara c' de oldugu gibi ilk değer verme
    Array<int, 10> ax{1,5,7,4,5};
}
---




---
int main()
{
    // default initil. değerler çöp değer
    array<int, 5> ar; // int a[5]

    // value initial
    array<int, 5> ar{}; // değerler 0 ile başlar
}
---




---
int main()
{

    array<int, 5> ar{};
    ar.at(2); // arr[2]
    // ar.at(6) exception throw eder [6] etmez
```

```
}
---




---
int main()
{

    array<int, 5> ar;
    ar.fill(10); // tüm elemanları 10 yapar
}
---




---
int main()
{
    array<int, 7> ax{1,2,3,4,5,6,7};
    array<int, 7> ay;

    copy(ax.begin(), ax.end(), ay.begin());
    print(ay); // 1 2 3 4 5 6 7
}
---




---
int main()
{
    array<int, 7> ax{1,2,3,4,5,6,7};
    array<int, 7> ay;

    auto p = ax.data(); // başlangıç adresi döner
    cout << p; // 0x557cbff770
}
---




---
int main()
```

```
{
    array<int, 5> arx{1,2,7,9,5};
    array<int, 5> ary{-1,9,1,2,8};

    // swap linnear complex
    swap(arx, ary); // global fonk
    // veya
    arx.swap(ary); // üye fonks
}
---
```

```
---
int main()
{
    array<int, 5> arx{1,2,7,9,5};
    array<int, 5> ary{-1,9,1,2,8};

    // bütün containerlarda geçerli
    cout << (arx < ary); // karşılıklı öğelerden ilk küçük olan öğe
}
---
```

```
---
// array sınıf sablonu için bir template inserter yazimi
template<typename T, size_t n>
std::ostream& operator<<(std::ostream &os, const std::array<T, n>& ar)
{
    os << "[" << ar.size() << "] |";
    for(size_t i{}; i < ar.size() - 1; ++i) {
        os << ar[i] << ", ";
    }
    return os << ar.back() << "|";
}
int main()
{
    array<int, 5> arx{1,2,7,9,5};
    cout << arx; // [5] |1, 2, 7, 9, 5|
```

```
    array x = {1.1, 6.6, 5.5, 1.5}; // CTAD
    cout << x; //[4] |1.1, 6.6, 5.5, 1.5|
}
---
```

```
---
array<int, 3> foo( int x, int y, int z)
{
    // ...
    return {x, y, z};
}

int main()
{
    auto [a, b, c] = foo(10, 20, 30);
    cout << a << "\n";
    cout << b << "\n";
    cout << c << "\n";

} // output
10
20
30
---
```

-> structer bindig' de eksik öğe olusra synatx error

```
---
template<typename T, size_t n>
std::ostream& operator<<(std::ostream &os, const std::array<T, n>& ar)
{
    os << "[" << ar.size() << "] |";
    for(size_t i{}; i < ar.size() - 1; ++i) {
```

```
        os << ar[i] << ", ";
    }
    return os << ar.back() << "|";
}

int main()
{
    array<array<int, 3>, 4> ar{{{1,1,1}, {2,2,2}, {3,3,3}, {4,4,4}}};
    ar[2][1] = 10;

    // şablondan 2 farklı fonksyion yazar
    cout << ar; // [4] |[3] |1, 1, 1|, [3] |2, 2, 2|, [3] |3, 10, 3|, [3] |4, 4, 4||

}
---
```

```
---
int main()
{
    array<int> ax= {1,2,3,4,5}; // syntax error
    array<>ax = {1,2,3,4,5}; // syntax error
    array ax = {1,2,3,4,5}; // CTAD geçerli
}
---
```

```
--> move işlevinin vector, deque gibi sınıflarda
  karmaşıklığı O(1) pointerları atanır,
  array heap'de bellek alanı tutmadıgından
  karşılıklı elemanlar taşınır
```

```
---
// array' de copy
class Myclass {
```

```
public:
   Myclass() = default;

   Myclass(const Myclass&)
   {
      cout << "copy ctor\n";
   }
   Myclass(const Myclass&&)
   {
      cout << "move ctor\n";
   }
};
int main()
{
   array<Myclass, 5> arx;
   array<Myclass, 5> ary = arx;
}
copy ctor
copy ctor
copy ctor
copy ctor
copy ctor
---


---
// array' de move
class Myclass {
public:
   Myclass() = default;

   Myclass(const Myclass&)
   {
      cout << "copy ctor\n";
   }
   Myclass(const Myclass&&)
   {
      cout << "move ctor\n";
   }
};
int main()
{
   array<Myclass, 5> arx;
   array<Myclass, 5> ary = move(arx);
}
move ctor
move ctor
move ctor
```

```
move ctor
move ctor
---
```

-> array' de copy ve move arasında maliyet farkı yok

```
---
// vector' de copy
class Myclass {
public:
    Myclass() = default;

    Myclass(const Myclass&)
    {
        cout << "copy ctor\n";
    }
    Myclass(const Myclass&&)
    {
        cout << "move ctor\n";
    }
};
int main()
{
    vector<Myclass> arx(5);
    vector<Myclass> ary = arx;
}
copy ctor
copy ctor
copy ctor
copy ctor
copy ctor
---
```

```
---
// vector' de move
class Myclass {
public:
    Myclass() = default;

    Myclass(const Myclass&)
    {
        cout << "copy ctor\n";
    }
    Myclass(const Myclass&&)
```

```
    {
        cout << "move ctor\n";
    }
};
int main()
{
    vector<Myclass> arx(5);
    vector<Myclass> ary = move(arx); // output boş
    // pointerları aldı (tipik olarak vektörlerin 3 pointerı var)
}
---
```

```
---
int main()
{
    array<int, 5> ar{};
    int a[5]{};

    int *p = a; // array decay
    int *ptr = ar; // geçersiz, bunun yerine,

    int *ptr1 = &ar[0];
    int *ptr2 = ar.data();
    int *ptr3 = &*ar.begin(); // ile ilk adrese erişilebilir
}
---
```

```
---
void set_array_random(int *p, size_t size);

int main()
{
    array <int, 10> ax{1,2,3,4,5,6,7,8,9,10};
    set_array_random(ax.data(), ax.size());
}
---
```

```
int main()
{
    constexpr array <int, 10> ax{1,2,3,4,5,6,7,8,9,10};
    constexpr auto val = ax[2];// compile time'da belli 3
}
```

--- bitset ---

- bitsel işlemler için oluşturulmuş sınıf şablonu
- container değil

```
#include<bitset>
```

- default ctoru tüm bitleri 0 ile başlatır
- ilave maliyet yok

--
```
int main()
{
    bitset<16> bs{};
    cout << bs; // 0000000000000000
}
```

```
int main()
{
    cout << "bir tam sayi girin: ";
    int x;
    cin >> x; // 16
    // sınıfın ctor' unun parametresi unsigned long long oldugundan
    // bitset<32>{x} daraltıcı dönüşüm olur
    cout << bitset<32>(x) << "\n";
}
// output
00000000000000000000000000010000
```

```
int main()
{
    bitset<32> x;
    cout << typeid(x[5]).name() << "\n"; // bitset<32>::reference, nested type
    // [] ile bool türünden değer elde edilmez
}
```

```
int main()
{
    bitset<32> x{550u};
    x[5]; // bitset',n reference isimli nested type' ı


    bool b = x[5]; // ile
    bool bx = x[5].operator bool(); // aynı anlam

    if (x[5]) // veya x[5].operator bool()
    {
        /* code */
    }
}
```

```
int main()
{
    bitset<32> x{550u};

    x[3] = true;
    x[7] = false;
    x[16] = 1;
}
```

```
int main()
{
    bitset<16> x{550u};

    cout << x << "\n";
    cout << x[5] << "\n"; // 1
    x[5].flip();
    cout << x[5] << "\n"; // 0

    // ~ overload
    cout << x[2] << "\n"; // 1
    x[2] = ~x[2]; // x[2]' nin değili,
    cout << x[2] << "\n"; // 0
}
```

```
int main()
{
    bitset<16> x{550u};
    cout << x << "\n";

    // set edilmiş bit sayısını döndürür
    cout << x.count() << "\n";
}
```

0000001000100110
4

---
int main()
{
    bitset<16> x{550u};

    cout << x.any(); // en az 1 bit set edilmişse true döner
    cout << x.all(); // bütün bitler 1 ise true
    cout << x.none(); // bütün bitler 0 ise true
}
1 0 0

---
enum Color{
    white, Gray, Red, Blue, Brown, Black, No_of_Colors
};

int main()
{
    bitset<No_of_Colors> colors;

    colors[Brown] = true;
    colors[Red] = true;
    colors[Black] = false ;
}

---
int main()
{
    bitset<No_of_Colors> colors;

    colors.set(); // bütün bitleri set eder
    colors.set(2); // 2. biti set eder

```
    colors[2] = true;
    cout << colors << "\n";
    colors.set(2, false);
    cout << colors << "\n";
}
// output
111111
111011
---
```

```
---
int main()
{
    bitset<32> bs{2342u};

    bs.reset(); // tüm bitleri sıfırlar
    bs.reset(10); // 10. biti sıfırlar

    bs.flip(); // tüm bitler flip edildi
    bs.flip(10); // 10. bit flip edildi
}
---
```

```
---
int main()
{
    bitset<16> bs{101011001} ;
    cout << bs.to_ullong() << "\n";
    cout << bs.to_ulong() << "\n";
}
---
```

```
---
int main()
{
    bitset<16> bx{101011001} ;
```

```
    bitset<16> by{111100000} ;
    cout << bx << "\n";
    cout << by << "\n";

    // kaydırma işlemleri

    cout << (bx & by) << "\n";
    // 0100111000111001 bx
    // 0100000001100000 by
    // 0100000000100000 bx & by
}
---




---
int main()
{
    bitset<16> bx{101011001} ;
    bitset<16> by{111100000} ;
    cout << bx << "\n";
    cout << by << "\n";

    // kaydırma işlemleri

    cout << (bx ^ by) << "\n";
    // 0100111000111001 bx
    // 0100000001100000 by
    // 0000111001011001 bx ^ by
}
---




---
int main()
{
    bitset<16> bx{101011001} ;
    bitset<16> by{111100000} ;
    cout << bx << "\n";
    cout << by << "\n";

    // kaydırma işlemleri
    cout << (bx << 3) << "\n";
    cout << (bx >> 5) << "\n";
}
---
```

```
int main()
{
    bitset<16> bx{101011001} ;
    bitset<16> by{111100000} ;
    cout << bx << "\n";
    cout << by << "\n";

    // kaydırma işlemleri
    cout << (bx | by) << "\n";
    bx |= by;
    cout << bx ;
}
```

--- std:: bind() ---
#include <functional>
- tüm callable' larda çalışır
- function object
- maliyetli

```
using namespace std;

void foo(int x, int y, int z)
{
    cout << "x = " << x << " y = " << y << " z = " << z << "\n";
}
```

```
int main()
{
    using namespace placeholders; // nested namespace
    auto f = std::bind(foo, _1, 20, 30);
    f(100);
}
// output
x = 100 y = 20 z = 30
---
```

```
---
int main()
{
    using namespace placeholders; // std nested namespace
    auto f = std::bind(foo, _1, 20, 30); // _1 f' e verilen argüman
    // auto f = std::bind(foo, 10, 20, _1); // 10,20, 100 olucak
    f(100);
}
---
```

```
---
int main()
{
    using namespace placeholders;
    auto f = std::bind(foo, _1, _1, _1);
    f(100);
}
// output
x = 100 y = 100 z = 100
---
```

```
---
int main()
{
    using namespace placeholders;
    auto f = std::bind(foo, _1, 500, _2);
    f(100, 200);
}
// output
x = 100 y = 500 z = 200
---
```

```
void foo(int x, int y, int z)
{
    cout << "x = " << x << " y = " << y << " z = " << z << "\n";
}

int main()
{
    using namespace placeholders;
    auto f = std::bind(foo, _3, _1, _2);
    f(100, 200, 50);
}
// output
x = 50 y = 100 z = 200
```

```
class Functer {
public:
    int operator()(int x, int y)
    {
        cout << "Functor::operator()\n";
        cout << "x = " << x << "\n";
        cout << "y = " << y << "\n";
        return x * y;
    }
};
int main()
{
    using namespace placeholders;
    auto f = std::bind(Functer{}, _1, _2);
    f(200, 50);
}
// output
Functor::operator()
x = 200
y = 50
```

```
int main()
{
    using namespace placeholders;
    auto f1 = [](int a, int b, int c){return a * b * c;};
    auto f2 = std::bind(f1, _1, 5, _2);
    // veya
    auto f3 = std::bind(f1, _1, 5, _2)(50, 100);
    cout << f2(5, 10) << "\n";
}
// output
250
```

```
void foo(int &r1, int &r2)
{
    r1 +=100;
    r2 += 50;
}
int main()
{
    using namespace placeholders;

    int x = 1;
    int y = 5;

    auto f = std::bind(foo, x, y); // x ve y kopyalama sematiği ile alınır
    f();
    cout << x << ", " << y << "\n"; // 1, 5

    // x ve y' yi referans sematiği ile  değiştirme
    auto f1 = std::bind(foo, ref(x), ref(y));
    f1();
    cout << x << ", " << y << "\n"; // 101, 55
```

```
}
---
```

```
---
void myprint(std::ostream& os, int x, int y)
{
   os << x << " " << y << "\n";
}
int main()
{
   using namespace std::placeholders;

   myprint(cout, 2, 5);

   // ostream sınıfıkopyalamaya karşı kapatılmış nesne oldugundan syntax error
   auto f = std::bind(myprint, std::cout, _1, _2);

   // geçerli, reference wraper
   auto f = std::bind(myprint, ref(std::cout), _1, _2);
}
---
```

```
---
class Myclass {
public:
   void set(int a, int b)
   {
      cout << "myclass::set(int, int)\n";
      cout << "a = " << a << "\n";
      cout << "b = " << b << "\n";
   }
};
```

```
int main()
{
    using namespace std::placeholders;

    // non_static class oldugundan bind için nesne gerekli
    Myclass m;

    auto f = std::bind(&Myclass::set, m, 5, _1);
    f(10);

}
// output
myclass::set(int, int)
a = 5
b = 10
---
```

```
---
int main()
{
    using namespace placeholders;

    vector<int> ivec(100);
    randomize();

    // generate için vector'de değerler olmalı
    generate(ivec.begin(), ivec.end(), []{return rand() % 1000;});

    int ival;
    cout << "kactan buyuk degerler yazilsin: ";
    cin >> ival;

    // lambda expression
    // copy_if(ivec.begin(), ivec.end(), ostream_iterator<int>{cout, " "},
    //[ival](int x){return x > ival;});

    copy_if(ivec.begin(), ivec.end(), ostream_iterator<int>{cout, " "},
```

```
    std::bind(std::greater<int>{}, _1, ival));
}
// output
kactan buyuk degerler yazilsin: 960
976 983 983
---
```

```
--- mem_fn() ---
```

```
---
class Myclass {
public:
    Myclass() = default;
    void print()const
    {
        cout << "(" << mx << ")";
    }

    void set(int val)
    {
        mx = val;
    }
private:
    int mx{};
};

int main()
```

```
{
    Myclass mx;
    mx.print();

    auto f = mem_fn(&Myclass::set);
    f(mx, 20);
    mx.print();

    auto f2 = mem_fn(&Myclass::print);
    f2(mx);
}
// output
(0)(20)(20)
---




---
int foo(const Date& d)
{
    return d.get_month_day();
}

int main()
{
    vector<Date> myvec;
    rfill(myvec, 100, Date::random_date);

    // 1. yol
    transform(myvec.begin(), myvec.end(), ostream_iterator<int>{cout, " "}, &foo);

    // 2. yol
    transform(myvec.begin(), myvec.end(), ostream_iterator<int>{cout, " "},
    [](const Date& d){return d.get_year_day();});

    // 3. yol
    transform(myvec.begin(), myvec.end(), ostream_iterator<int>{cout, " "},
        mem_fn(&Date::get_year_day));

}
---
```

--- not_fn ---

```
int main()
{
    cout << "bir tam sayi girin: ";
    int x;
    cin>> x;

    auto f = not_fn(&isprime);
    cout << isprime(x) ; // asalsa true
    cout << f(x) ; // isprime(x)' in tersini döner asalsa false
}
```


```
int main()
{
    vector<int> myvec;
    rfill(myvec, 100, Irand{0, 1000});
    print(myvec);

    // vektördeki asal olmayan sayıları yazar
    copy_if(myvec.begin(), myvec.end(), ostream_iterator<int>{cout, " "}, not_fn(isprime));
}
```

--- member function pointers ---

```
class Myclass {
public:
   static void func(int x) // static member function
   {
      cout << "Myclass::func(int)\n";
   }
};

int main()
{
   // aynı anlam
   void (*fp)(int) = Myclass::func;
   void (*fp1)(int) = &Myclass::func;
   auto fp2= &Myclass::func;

   fp(1);
   fp1(1);
   fp2(1);
}
```

```
class Myclass {
public:
   void func(int x)
   {
      cout << "Myclass::func(int)\n";
   }
};

int main()
{
   void (Myclass:: *fp)(int); // non_static member function pointer
   auto fp2= &Myclass::func;
}
```

---
class Myclass {
public:
   void func(int x)
   {
      cout << "Myclass::func(int)\n";
   }
   int foo(int, int);
};

int main()
{
   void (Myclass:: *fp)(int) = &Myclass::func;
   int (Myclass:: *fp1)(int, int) = &Myclass::foo;
}

---
using Mfptr = void (Myclass:: *)(int);
int main()
{
   void (Myclass:: *fp)(int) = &Myclass::func;
   Mfptr fpx = &Myclass::func;
}

---
class Myclass {
public:
   int f1(int x);
   int f2(int x);
   int f3(int x);
   int f4(int x);
   int f5(int x);
};

```
using Mfptr = int(Myclass::*)(int);
//typedef int(Myclass::*Mfptr)(int);

int main()
{
    Mfptr fa[] ={ // int(Myclass::*fa[])(int) = ...
        &Myclass::f1,
        &Myclass::f2,
        &Myclass::f3,
        &Myclass::f4,
        &Myclass::f5,
    };
}
---
```

```
---
class Myclass {
public:
    int mx;
};

int main()
{
    Myclass m;
    auto p = &m.mx; // p -> int *

    auto p1 = &Myclass::mx; // p1 -> int Myclass::*p1
}
---
```

---   .* ve ->* operatorleri ---

- C dilinde yok
- ptr->* sınıf adresi
- ptr.*   sınıf nesnesi

```
class Myclass {
public:
   void func(int x)
   {
      cout << "Myclass::func(int x)\n";
      cout << "x = " << x << "\n";
      cout << "this = " << this << "\n";

   }
};

int main()
{
   Myclass m;
   cout << "&m = " << &m << "\n";

   void (Myclass::*fp)(int) = &Myclass::func;

   fp(10); // syntax error
   m.fp(10); // name lookup hatası, syntax error
   m.*fp(10); // operator önceliği syntax error
   (m.*fp)(10); // geçerli
}
```

```
class Myclass {
public:
   void func(int x)
   {
      cout << "Myclass::func(int x)\n";
```

```
        cout << "x = " << x << "\n";
        cout << "this = " << this << "\n";
        cout << "-------------------------------------\n";

    }
    void foo(int x)
    {
        cout << "Myclass::foo(int x)\n";
        cout << "x = " << x << "\n";
        cout << "this = " << this << "\n";
        cout << "-------------------------------------\n";
    }
};

int main()
{
    Myclass m;
    cout << "&m = " << &m << "\n";

    void (Myclass::*fp)(int) = &Myclass::func;
    (m.*fp)(10);

    fp = &Myclass::foo;
    (m.*fp)(20);

}
// output
&m = 0xba159ff68f
Myclass::func(int x)
x = 10
this = 0xba159ff68f
-------------------------------------
Myclass::foo(int x)
x = 20
this = 0xba159ff68f
-------------------------------------
---
```

```
---
int main()
{
    void(Myclass::*fp)(int) = &Myclass::foo;
    Myclass *mp = new Myclass;
```

```
    cout << mp << "\n";
    (mp->*fp)(50);

}
// output
0x1e3029d13e0
Myclass::foo(int x)
x = 50
this = 0x1e3029d13e0
---




---
int main()
{
    void(Myclass::*fpa[])(int) = {
    &Myclass::foo,
    &Myclass::func
    };

    Myclass m;

    for(auto fptr : fpa) {
        (m.*fptr)(2);
    }

    for(int i = 0; i< std::size(fpa); ++i) {
        (m.*fpa[i])(i);
    }
}
---




// fonksiyon parametresi
void g(Myclass &r, void (Myclass::*fp)(int));
```

```
struct Myclass {
    int x = 1;
    int y = 2;
};

int main()
{
   Myclass mx;

   int Myclass::*ptr = &Myclass::x;

   cout << "mx.x = " << mx.x << "\n";
   cout << "mx.y = " << mx.y << "\n";

   mx.*ptr = 10;
   cout << "mx.x = " << mx.x << "\n";

   ptr = &Myclass::y;
   mx.*ptr = 50;
   cout << "mx.y = " << mx.y << "\n";
}
// output
mx.x = 1
mx.y = 2
mx.x = 10
mx.y = 50
```

--- std::invoke c++17 ---

- fonksiyon şablonu
- #include<functional>

```
void gfunc(int x, int y)
{
    cout << "gfunc cagirildi\n";
    cout << "x = " << x << "\n";
    cout << "y = " << y << "\n";
}

int main()
{
   std::invoke(gfunc, 10, 20);
}
// output
gfunc cagirildi
x = 10
y = 20
```

```
int main()
{
    auto f = [](int x){return x * 5 + 5;};
    std::cout << std::invoke(f, 10); // 55
}
```

```
class Functor {
public:
    Functor(int x) : mx{x}{}
```

```
    int operator()(int x, int y)const
    {
        return  mx * (x + y);
    }
private:
    int mx;
};

int main()
{
    std::cout << std::invoke(Functor{10}, 10, 20) << "\n"; // 300
}
---
```

```
---
class Myclass {
public:
    int func(int x, int y)
    {
        std::cout << "Myclass int int\n";
        return x * y;
    }
};

int main()
{
    Myclass mx;
    std::cout << std::invoke(&Myclass::func, mx, 5, 2) << "\n"; // 10
    // int(Myclass::*fptr)(int , int);

}
---
```

```
---
struct Myclass {
    int x;
};
```

```
int main()
{
    int Myclass::*ptr = &Myclass::x;

    Myclass mc;
    mc.*ptr = 10;
    std::invoke(ptr, mc) = 15;
}
```
—

---- dynamic array ----

- runtime sırasında oluşturup silenebilen nesneler

- operator new ve operator delete overload edilebilir

```
void *operator new(size_t n)
```

```
{
    std::cout << "operator new called n = " << n << "\n";
    void *vp = std::malloc(n);
    if(!vp) {
        throw std::bad_alloc{};
    }
    std::cout << "the adress of the allocated block = "<< vp <<"\n";

    return vp;
}

void operator delete(void *vp)
{
    if(!vp)
        return;
    std::cout << "operator delete called... \nvp" <<  vp << "\n";
    std::free(vp);
}

class Myclass {
public:
    Myclass()
    {
        std::cout << "default ctor\n";
        std::cout << "this: " << this << "\n";
    }
    unsigned char buffer[1024];
};

int main()
{
    Myclass *p = new Myclass;
    std::cout << "-------------------------\n";
    delete p;

}
// output
operator new called n = 1024
the adress of the allocated block = 0x1e8dab51750
default ctor
this: 0x1e8dab51750
-------------------------
operator delete called...
vp0x1e8dab51750
---
```

```
using new_handler = void(*)(void);
typedef void(*new_handler)(void);

new_handler gp = nullptr;

new_handler set_new_handler(new_handler p)
{
   auto fp = gp;
   gp = p;
   return fp;
}
new_handler get_new_handler()
{
   return gp;
}

void *operator new(size_t n)
{
   while(true) {
      void* vp = std::malloc(n); // malloc çağırmak derleyiciye bağlı
      if (vp)
         return vp;
      else {
         auto fptr = std::get_new_handler();
         if(!fptr)
            throw std::bad_alloc{};
         fptr();
      }
   }
}

// programcı set_new_handler işlevini çağırır
// set_new_handler(my_new_handler);

void my_new_handler()
```

```
{
    //        aşağıdaki seçeneklerden birini yapmak zorunda
    // operator new' in başarılı olmasını sağlayacak koşulları oluştur
    // doğrudan bad_alloc throw et
    // set_new_handler nullptr ile çağır
    // std::terminate çağır
    // set_new_handler başka bir new handler işlevi set et
}
---
```

--> operator new başarısız oldugunda exception throw eder,
doğrudan exception throw etmesi istenmediğinde istenmiyorsa
std::set_new_handler çağırılır, kendi fonksiyonun parametresi
geçilir.

```
class Myclass {
public:

    unsigned char buffer[1024 * 1024];
};

Myclass *p;

int main()
{
    try{
        while(true) {
            p = new Myclass;
            std::cout << '.';
        }
    }
    catch(const std::bad_alloc& ex)
    {
        std::cout << "\nexception cauhgt: " << ex.what() << "\n";
    }
}
// output
```

.....*n
exception cauhgt: std::bad_alloc

---
class Myclass {
public:

   unsigned char buffer[1024 * 50000];
};

Myclass *p;

void another_handler()
{
   std::cout << "anoter_handler called...\n";
   getchar();
}

void myhandler()
{
   static int count = 0;
   std::cout << "myhandler called...\n";
   if(++count == 3) {
      // throw std::bad_alloc{}; // 1. yol
      // std::terminate(); // 2
      // std::set_new_handler(nullptr); // 3
      // std::set_new_handler(another_handler); // 4, başka bir hadler fonk
   }
   getchar();
}

int main()
{
   std::set_new_handler(&myhandler);

   try{
      while(true) {
         p = new Myclass;
         std::cout << '.';

```
        }
    }
    catch(const std::bad_alloc& ex)
    {
        std::cout << "\nexception cauhgt: " << ex.what() << "\n";
    }
}
// output
.......*n
myhandler called...
myhandler called...
myhandler called...
---
```

```
---
class Myclass {
public:

    void foo();
};


int main()
{
    auto ptr = new const Myclass;

    ptr->foo(); // syntax error
}
---
```

new T

new T []

placement new

- operator new'i overload edilemez
- var olan bir bellekte nesneyi hayata başlatmak

nothrow new
- exception throw etmek yerine nullptr döndürür

-> placement new operator fonks
-> overload edilemez, syntax error

```
void *operator new(size_t size, void* vp )
{
   return vp;
}
```

```
int main()
{
   char buffer[sizeof(Date)];
   std::cout << static_cast<void *>(buffer) << "\n";
   // buffer adresinde Date nesnesini hayata getirmek için ,
   // kullanılan new operatorlerine placement new operatorleri denir

   Date *p = new(buffer)Date{1, 1, 2020}; // allocation yok

   delete p; // ub, tanımsız davranış,
   // dinamik olarak allocate edilmemiş bellek blogunu free etmek ub
   // delete kullanılamadığından destructor'da çağırılmaz

   // destructor' ın ismiyle çağırıldığı tek senaryo, placement new
   p->~Date();
}
```

// nothrow new
// exception throw etmek yerine nullptr döndürür

```
class Data {
    char buffer[1024 * 1024];
};

int main()
{
    std::vector<Data*> vec;

    for(;;) {
        auto pd = new(std::nothrow)Data;
        if (!pd) {
            break;
        }
        vec.push_back(pd);
    }
    std::cout << "vec.size: " << vec.size(); // 22032
}
```

--- attributes ---

[[nodiscard]] -> fonksiyon değeri kullanılması zorunlu hale gelir

---
```
[[nodiscard]]
bool isprime(int val);
```

```
int main()
{
    int x{50};
    isprime(x);
}
// output
//warning: ignoring return value of 'bool isprime(int)',
    //declared with attribute 'nodiscard' [-Wunused-result]
---
```

```
---
class Myclass {
public:
    Myclass()
    {
        std::cout << "Myclass() this: " << this << "\n";
    }
    ~Myclass()
    {
        std::cout << "~Myclass() this: " << this << "\n";
    }


// operator new overload memeber function
// static void* operator new(size_t n); // geçerli
    void* operator new(size_t n) // static anahtarı olmasa da static
    {
        auto p = std::malloc(n);
        std::cout << "Myclass::operator new called! n = " << n << "\n";
        if(!p)
            throw std::bad_alloc{};
        std::cout << "address of allocated block = " << p << "\n";
        return p;
    }
```

```
    void operator delete(void *vp) // static anahtarı olmasa da static
    {
        std::cout << "Myclass::operator delete called! n = " << vp << "\n";
        free(vp);
    }
private:
    char buffer[1024]{};
};

int main()
{
    // auto p = new std::string; // sınıfın operator new' i çağırılmaz

    auto p = new Myclass;
    delete p;

    // eğer
    // ::delete p; olsaydı
    // Myclass::operator delete çağırılmazdı
}
// output
Myclass::operator new called! n = 1024
address of allocated block = 0x134137a1750
Myclass() this: 0x134137a1750
~Myclass() this: 0x134137a1750
Myclass::operator delete called! n = 0x134137a1750
---
```

```
---
// .h
class Myclass {
public:
    constexpr static size_t buffer_size = 1024;
    constexpr static size_t max_no_of_dynamic_objects = 100;
    void* operator new(size_t n);
    void operator delete(void*);
private:
    unsigned char m_buf[buffer_size]{};
```

```
    // max_no_of_dynamic kadar nesne tutabilecek bellek alanı
    static unsigned char s_buffer[];

    // hangi indx elemanının kullanılıp kullanılmadıgını tutar
    static bool s_flags[];
};


//.cpp
unsigned char Myclass::s_buffer[max_no_of_dynamic_objects * sizeof(Myclass)];

bool Myclass::s_flags[max_no_of_dynamic_objects]{};

void* Myclass::operator new(size_t n)
{
    auto iter = find(begin(s_flags), end(s_flags), false);
    if (iter == end(s_flags)) // yoksa bellek dolu
    {
        throw bad_alloc{};
    }
    auto idx = iter - begin(s_flags);
    s_flags[idx] = true; // indeksin dolu oldugunu diziye yazar(true)
    return s_buffer + idx * sizeof(Myclass);
}

void Myclass::operator delete(void* vp)
{
    if (!vp)
        return;

    auto idx = (static_cast<unsigned char*>(vp) - s_buffer) / sizeof(Myclass);
    s_flags[idx] = false;
}



int main()
{
    std::vector<Myclass*> myvec;

            for (size_t i = 0; i < Myclass::max_no_of_dynamic_objects; i++)
            {
                    myvec.push_back(new Myclass);
            }

            std::cout << "myvec.size(): " << myvec.size() << "\n";

            delete myvec[0];
```

```
	try {
		auto p = new Myclass;
	}
	catch (const std::exception& ex) {
		std::cout << "exception caught: " << ex.what() << "\n";
	}
}
---
```

--- smart pointers ---

- move only type, kopyalamaya kapatılmış sınıf

unique_ptr -> exclusive ownership
- aynı kaynagı gösteren 1 pointer olabilir

- shared_ptr
  - weak_ptr
  - aynı kaynagı birden fazla pointer gösterebilir

```
#include <memory>
```

```
template<typename T, typename Deleter = std::default_delete<T>>
class unique_ptr {
public:
   ~unique_ptr()
   {
      if(mp)
         Deleter{}(mp);
   }
private:
   T *mp;
};

template <typename T>
struct default_delete {
   void operator()(T *p)
   {
      delete p;
   }
};
```

```
//boş mu?

int main()
{
   std::unique_ptr<Myclass> uptr;
   if(uptr) {}// if(uptr.operator bool())

   std::cout << "uptr" << (uptr ? "dolu" : "bos");

   if (uptr == nullptr) {}

   if (uptr.get()) {}
}
```

```
int main()
{
   Triple x;
   Triple *p = &x;
   std::unique_ptr<Triple> uptr{&x}; // ub
   // dinamik ömürlü nesne adresi olmalı
}
```

```
int main()
{
   std::cout <<"main basliyor\n";
   {
      // T* parametreli ctor
      std::unique_ptr<Triple> uptr{new Triple{1,2,3}};
   }
   std::cout << "main devam ediyor\n";

}
// output
main basliyor
(1, 2, 3) degerinde 0x1ddd1cb13e0 adresinde Triple nesnesi hayata geldi
(1, 2, 3) degerindeki 0x1ddd1cb13e0 adresindeki Triple nesnesinin hayati sona erdi
main devam ediyor
```

```
int main()
{
   auto ptr = new Triple;
```

```
    // biri silindiğinde danglig pointer olucağından tanımsız davranış
    std::unique_ptr<Triple> upx{ptr};
    std::unique_ptr<Triple> upy{ptr}; // ub
}
---




---
int main()
{
    // fonksiyon şablonu
    // ctor'a gönderilecek argümanlar yazılır,
    // perfect fowarding ile gönderilir
    // new make_uniq içerisinde kullanılır
    auto up = std::make_unique<Triple>(1, 2, 5);
}
// output
(1, 2, 5) degerinde 0x205f61113e0 adresinde Triple nesnesi hayata geldi
(1, 2, 5) degerindeki 0x205f61113e0 adresindeki Triple nesnesinin hayati sona erdi
---




---
// make_uniq fonksiyon şablonu
template <typename T, typename ...Args>
std::unique_ptr<T> MakeUnique(Args&& ...args)
{
    // geçici nesne ile PR value oluşturuldu
    return std::unique_ptr<T>{new T{std::forward<Args>(args)...}};
}

int main()
{

    auto up = MakeUnique<Triple>(1, 2, 5);
}
// output
(1, 2, 5) degerinde 0x1ed411213e0 adresinde Triple nesnesi hayata geldi
(1, 2, 5) degerindeki 0x1ed411213e0 adresindeki Triple nesnesinin hayati sona erdi
```

```
int main()
{
    auto up = std::make_unique<std::string>("ali emre");
    //std::unique_ptr<std::string> up = std::make_unique<std::string>("ali emre");
    std::cout << *up; // uptr.operator*()
}
// output
ali emre
```

```
int main()
{
    auto up = std::make_unique<std::string>("ali emre");

    std::cout << (*up).length() << "\n"; // 8
    std::cout << up->length() << "\n"; // 8
    std::cout << up.operator->()->length() << "\n"; // 8
}
```

---> bir unique_ptr nesnesini ok veya içerik operatorunun
operandı yapılacaksa dolu oldugundan emin olunmalı,
exception throw etmez, ub

```
int main()
{
    std::unique_ptr<Triple> upx;
    auto upy = upx; // syntax error, copy deleted function
}
```

```
void func(std::unique_ptr<Triple>);

int main()
{
    auto uptr = std::make_unique<Triple>(1, 2, 3)    ;
    func(uptr); // syntax error, copy ctor çağırılmalı
    // sınıfın copy ctor' u delete edilmiş

    func(std::move(uptr)); // geçerli
}
```

```
int main()
{
    auto uptr = std::make_unique<Triple>(1, 2, 3)    ;
    std::unique_ptr<Triple> upy;
    upy = uptr; // syntax erorr, copy assignment
    upy = std::move(uptr); // geçerli
}
```

```
int main()
{
    auto uptr = std::make_unique<Triple>(1, 2, 3)    ;
    std::unique_ptr<Triple> upy;

    std::cout << "uptr :" << (uptr ? "dolu" : "bos") <<"\n";
    std::cout << "upy :" << (upy ? "dolu" : "bos") <<"\n\n";

    upy = std::move(uptr);
    std::cout << "uptr :" << (uptr ? "dolu" : "bos") <<"\n";
    std::cout << "upy :" << (upy ? "dolu" : "bos") <<"\n";
}
// output
(1, 2, 3) degerinde 0x1bfe2f013e0 adresinde Triple nesnesi hayata geldi
```

```
uptr :dolu
upy :bos

uptr :bos
upy :dolu
(1, 2, 3) degerindeki 0x1bfe2f013e0 adresindeki Triple nesnesinin hayati sona erdi
---
```

-> her ikisi de değere sahipse , move edilen ilk önce
kendi değerini bırakıp move edilenin değerini çalar

```
---
int main()
{
    auto uptr = std::make_unique<Triple>(1, 2, 3);
    auto upy = std::make_unique<Triple>(1, 1, 1);


    std::cout << "uptr :" << (uptr ? "dolu" : "bos") <<"\n";
    std::cout << "upy :" << (upy ? "dolu" : "bos") <<"\n\n";

    uptr = std::move(upy);
    std::cout << "uptr :" << (uptr ? "dolu" : "bos") <<"\n";
    std::cout << "upy :" << (upy ? "dolu" : "bos") <<"\n";
}
// output
(1, 2, 3) degerinde 0x17f8fea13e0 adresinde Triple nesnesi hayata geldi
(1, 1, 1) degerinde 0x17f8fea1400 adresinde Triple nesnesi hayata geldi
uptr :dolu
upy :dolu

(1, 2, 3) degerindeki 0x17f8fea13e0 adresindeki Triple nesnesinin hayati sona erdi
uptr :dolu
upy :bos
(1, 1, 1) degerindeki 0x17f8fea1400 adresindeki Triple nesnesinin hayati sona erdi
---
```

```
int main()
{
    auto uptr = std::make_unique<Triple>(1, 2, 3);

    std::cout << "uptr :" << (uptr ? "dolu" : "bos") <<"\n";

    //      uptr.reset(T*) nesneyi delete eder, uptr boş
    uptr.reset();
    uptr.reset(nullptr);
    uptr = nullptr; // yukardaki kullanımlar ile aynı anlam

    std::cout << "uptr :" << (uptr ? "dolu" : "bos") <<"\n";
}
// output
(1, 2, 3) degerinde 0x213993f13e0 adresinde Triple nesnesi hayata geldi
uptr :dolu
(1, 2, 3) degerindeki 0x213993f13e0 adresindeki Triple nesnesinin hayati sona erdi
uptr :bos
```

```
int main()
{
    auto uptr = std::make_unique<Triple>(1, 2, 3);

    std::cout << "get(): " << uptr.get() << "\n";
    std::cout << uptr << "\n";
    std::cout << *uptr << "\n";

}
// output
(1, 2, 3) degerinde 000001EF51DA1830 adresinde Triple nesnesi hayata geldi
get(): 000001EF51DA1830
000001EF51DA1830
(1, 2, 3)
(1, 2, 3) degerindeki 000001EF51DA1830 adresindeki Triple nesnesinin hayati sona erdi
```

```
int main()
{
    // release -> unique_ptr boşa çıkıyor fakat nesneyi silmiyor
    auto uptr = std::make_unique<Triple>(1, 2, 3);
    std::cout << "uptr :" << (uptr ? "dolu" : "bos") <<"\n";

    Triple* p = uptr.release();
    std::cout << "uptr :" << (uptr ? "dolu" : "bos") <<"\n";

    delete p; // delete edilmezse destructor çağırılmaz
}
// output
(1, 2, 3) degerinde 0x28e2c9e13e0 adresinde Triple nesnesi hayata geldi
uptr :dolu
uptr :bos
(1, 2, 3) degerindeki 0x28e2c9e13e0 adresindeki Triple nesnesinin hayati sona erdi
```

```
int main()
{

    auto uptr = std::make_unique<Triple>(1, 2, 3);
    auto upy = std::make_unique<Triple>(10,20,30);
    std::cout << "uptr :" << (uptr ? "dolu" : "bos") <<"\n";
    std::cout << "upy :" << (upy ? "dolu" : "bos") <<"\n";

    upy.reset(uptr.release()); // move assignment aynı anlamda
    // upy = std::move(uptr);

    std::cout << "main devam ediyor\n";

}
// output
(1, 2, 3) degerinde 0x180bcc213e0 adresinde Triple nesnesi hayata geldi
(10, 20, 30) degerinde 0x180bcc21400 adresinde Triple nesnesi hayata geldi
uptr :dolu
upy :dolu
(10, 20, 30) degerindeki 0x180bcc21400 adresindeki Triple nesnesinin hayati sona erdi
main devam ediyor
```

(1, 2, 3) degerindeki 0x180bcc213e0 adresindeki Triple nesnesinin hayati sona erdi
---

-> 2 unique_ptr birbiri ile karşılaştırılabilir (==)

-> otamatik ömürlü nesneler return edildiğinde,
   taşıma sematiği devreye girer

-> otomatik ömürlü nesne edileriken move kullanılmamalı

```
class Myclass {
   // Move only class
};

Myclass func()
{ // otomatik ömürlü bir nesne ile return edildiğinde
   // L value olsa bile R value dönüşür
   Myclass m;
   return m;
}

int main()
{
   Myclass m = func();
   Myclass m2;
   m2 = func();
   // geçerli
}
```

--- kaplarda unique_ptr tutmak ---

- kopyalama yapıldığından initializer_list ctor' u kullanılamaz

---

```
using usptr = std::unique_ptr<std::string>;

int main()
{
   using namespace std;

   vector<usptr> vec(100);

   for(auto up : vec) // syntax error, copy deleted

   for(auto &up : vec) // geçerli
   for(auto &&up : vec) // geçerli,fowarding-universal reference
}
```

```
using usptr = std::unique_ptr<std::string>;

int main()
{
    using namespace std;

    // syntax error
    vector<usptr> vec {usptr{new string{"ali"}}, usptr{new string{"veli"}}};
}
```

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


    vector<usptr> vec;
    auto up = make_unique<string>("ali");

    vec.push_back(move(up));
    vec.push_back(usptr{ new string ("ali")});
    vec.push_back(make_unique<string>("ali"));

    // perfect fowardig ile ctor'a gönderilecek argüman yazılır
    vec.emplace_back(new string{"veli"});

    // vector nesnesinin hayatı bittiğinde,
        // bellekte tutulan nesnelerin destructor'u çağırılır
}
```

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


    vector<usptr> vec;
    auto up = make_unique<string>("ali");
```

```
    vec.push_back(move(up));
    vec.push_back(usptr{ new string ("ali")});
    vec.push_back(make_unique<string>("ali"));

    list<usptr> mylist;
    for (auto& up : vec)
        mylist.push_back(std::move(up));

    // vectordeki öğeler boş, listedikiler dolu hale gelir
    // vector size' ı değişmez
    std::cout << "vec.size() " << vec.size() << "\n"; // 3

    // syntax error, kopyalama var
    list<usptr> mylist{vec.begin(), vec.end()};


    list<usptr> myl(100);
    // syntax error, kopyalama var
    copy(vec.begin(), vec.end(), myl.begin());
}
---
```

```
---
// move iterator dereference edildiğinde, değeri move'a argüman gönderir
template <typename InIter, typename OutIter> //
OutIter Copy(InIter beg, InIter end, OutIter destbeg)
{
    while(beg != end) {
        *destbeg++ = std::move(*beg++);
    }
    return destbeg;
}

int main()
{
    using namespace std;


    vector<usptr> vec;
    auto up = make_unique<string>("ali");

    vec.push_back(move(up));
    vec.push_back(usptr{ new string ("ali")});
```

```
    vec.push_back(make_unique<string>("ali"));

    list<usptr> myl(100);
    // geçerli
    Copy(vec.begin(), vec.end(), myl.begin());
}
---
```

```
---
// move iterator dereference edildiğinde, değeri move'a argüman gönderir
template <typename InIter, typename OutIter> //
OutIter Copy(InIter beg, InIter end, OutIter destbeg)
{
    while(beg != end) {
        *destbeg++ = std::move(*beg++);
    }
    return destbeg;
}

int main()
{
    using namespace std;


    vector<usptr> vec;
    auto up = make_unique<string>("ali");

    vec.push_back(move(up));
    vec.push_back(usptr{ new string ("ali")});
    vec.push_back(make_unique<string>("ali"));

    list<usptr> myl(100);
    // geçerli
    Copy(vec.begin(), vec.end(), myl.begin());
}
---
```

```
---
int main()
{
```

```
    using namespace std;
    vector<usptr> vec;

    auto uptr = *vec.begin(); //syntax error, L value
    auto uptr = move(*vec.begin()); //geçerli
}
---




---

int main()
{
    using namespace std;
    vector<usptr> vec;
    auto up = make_unique<string>("ali");

    vec.push_back(move(up));
    vec.push_back(usptr{ new string ("ali")});
    vec.push_back(make_unique<string>("ali"));

    // syntax error, kopyalama var
    list<usptr> mylist{vec.begin(), vec.end()};

    // geçerli
    list<usptr> mylist{make_move_iterator(vec.begin()), make_move_iterator(vec.end())};

}
---




---
int main()
{
    using namespace std;
    vector<usptr> vec;
    auto up = make_unique<string>("ali");

    vec.push_back(move(up));
    vec.push_back(usptr{ new string ("ali")});
    vec.push_back(make_unique<string>("ali"));

   // auto uptr = *vec.begin(); // syntax error
    auto uptr = move(vec.begin()); // geçerli
```

```
    unique_ptr<string> x = *make_move_iterator(vec.begin());
}
---



---
int main()
{
    using namespace std;
    vector<usptr> vec;
    auto up = make_unique<string>("ali");

    vec.push_back(move(up));
    vec.push_back(usptr{ new string ("ali")});
    vec.push_back(make_unique<string>("ali"));

    list<usptr> mylist{make_move_iterator(vec.begin()), make_move_iterator(vec.end())};

    for (auto& up : mylist)
        cout << (up ? "dolu" : "bos") << "\n";


    for (auto& up : vec)
        cout << (up ? "dolu" : "bos") << "\n";
}
// output
dolu
dolu
dolu
bos
bos
bos
---









---
int main()
{
    using namespace std;
    vector<usptr> vec;
    auto up = make_unique<string>("ali");
```

```
    vec.push_back(move(up));
    vec.push_back(usptr{ new string ("ali")});
    vec.push_back(make_unique<string>("ali"));

    list<usptr> mylist(4);

    // geçerli
    copy(make_move_iterator(vec.begin()), make_move_iterator(vec.end()), mylist.begin());

    // tür dönüşümü olan move ile aynı değil
    // copy' nin yaptığını yapar
    move(vec.begin(), vec.end(), mylist.begin());
}
---
```

```
---
struct TripleDeleter {
    void operator()(Triple *p)
    {
        std::cout << p << " adresindeki nesne delete ediliyor\n";
        delete p;
    }
};


int main()
{
    using namespace std;

    {
        unique_ptr<Triple, TripleDeleter> uptr(new Triple{1,2,3});
    }

    cout << "main devam ediyor\n";
}
// output
(1, 2, 3) degerinde 0x221a98b13e0 adresinde Triple nesnesi hayata geldi
0x221a98b13e0 adresindeki nesne delete ediliyor
(1, 2, 3) degerindeki 0x221a98b13e0 adresindeki Triple nesnesinin hayati sona erdi
main devam ediyor
```

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

    auto f = [](Triple *p){
        std::cout << p << " adresindeki nesne delete ediliyor\n";
        delete p;
    };


    { // c++20
    { // c++20 öncesi lambda default ctor delete edildiğinden synax error
        unique_ptr<Triple, decltype(f)> uptr(new Triple{1,2,3});
    }

    // geçerli
    { // lambda ifadesi argüman olarak gönderilir copy ctor ile hayata gelir
        unique_ptr<Triple, decltype(f)> uptr(new Triple{1,2,3}, f);
    }



    cout << "main devam ediyor\n";
}
```

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

    auto f = [](int x){
        return x * x;
    };

    decltype(f) g; // c++20 öncesi syntax error, default ctor yok
}
```

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

    auto f = [](int x, int y){
        return x % 10 < y %  10;
    };

    // default init oldugundan c++20 öncesi syntax error
    set<int,decltype(f)> myset;

    //c++20 öncesinde kullanmak için callable ctor kullanılır
    set<int,decltype(f)> myset(f);
}
```

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

    // geçerli
    unique_ptr<Triple[]> uptr(new Triple[5]);

    // ub
    unique_ptr<Triple> uptr(new Triple[5]);
}
```

--- ms

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

    auto p = new Triple{1,2,3};
    unique_ptr<Triple> px{p};

    auto ptr = px.release(); // px mülkiyeti bıraktı, delete etmedi
```

```
    cout << *ptr << "\n";
    ptr->set(1,7,9); // delete edilmedi!!
}
// output
(1, 2, 3) degerinde 0x25804c713e0 adresinde Triple nesnesi hayata geldi
(1, 2, 3)
// destructor çağırılmadı
---




--- ms
int main()
{
    using namespace std;

    auto p = new Triple{1,2,3};
    unique_ptr<Triple> px{p};

    auto ptr = px.get();

    cout << *ptr;
    ptr->set(1,1,1);
    delete ptr;

    cout << * px ; // danglig pointer, ub
}
---




---
int main()
{
    using namespace std;

    auto uptr = make_unique<Triple>(1,2,3);
    unique_ptr<Triple> upx;

    // no operator "=" matches these operands
    upx = uptr.release(); // synax error

    upx.reset(uptr.release()); // veya
    upx = move(uptr);
}
```

```
int main()
{
    auto fdel = [](FILE* f){fclose(f);};

    unique_ptr<FILE, decltype(fdel)> uptr{fopen("out.txt", "w"), fdel};
    fprintf(uptr.get(), "ali");
    // FILE *f = fopen("out.txt", "w");
    //fclose(f);
}
```

--- lambda perfec fowardig
```
int main()
{
    auto f = [](auto&& ...args){
        foo(std::forward<decltype(args)>(args)...);
    };
}
```

```
class Myclass {
public:
    Myclass(int val) : mx {val}{}
    void set(int val)
    {
        mx = val;
    }
    void func()
    {
        auto f = [this](int val) {return mx + val;}; // geçerli

        // burdaki this capture edilen this
        auto f = [this](int val) {return this->mx + val;}; // geçerli
```

```
        auto f = [&](int val) {return this->mx + val;}; // geçerli
        auto f = [=](int val) {return this->mx + val;}; // deprecated

        // *this nesnesini kopyalama yoluyla capture eder
        // lambda init capture
        // c++17 öncesi
        auto f = [strathis = *this](){return strathis;}; // geçerli

        // *this nesnesini kopyalama yoluyla capture eder
        // c++17
        auto f =[*this]()mutable{++mx;};

        // auto f = [](int val) {return rhis->mx + val;}; //syntax error
        // auto f = [mx](int val) {return mx + val;}; // synax error
    }
private:
    int mx;
};
---
```

```
---
int main()
{
    auto uptr = make_unique<string>("ali");
    // lambda init
    auto f = [uptr = std::move(uptr)]() {cout << *uptr << "\n";};
    f();
    if(uptr)
        cout << "dolu\n";
    else
        cout << "bos\n";
}
// output
ali
bos
---
```

```
---
// lambda in unevaluated context
int main()
```

```
{
    auto fcomp = [](int a, int b){return a < b;};
    set<int, decltype(fcomp)> myset;
    // yerine
    // isimlendirme zorunlulugu ortadan kalkıyor
    set<int, decltype([](int a, int b){return a < b;})> myset;
}
---
```

```
---
// templated lambda
int main()
{
    // 1. yol
    // derleyici closer type için yazdığı operator fonksiyonu template olarak yazar
    auto f = [](auto x){};

    // 2. yol
    []<typename T>(T x, T y){};

    // farkları
    auto f = [](auto x, auto y){}; // parametrenin türleri farklı olabilir
    f(12, 5.5); // geçerli

    auto f =[]<typename T>(T x, T y){}; // parametrelerin türleri aynı
    // ve örneğin sadece vector sınıfı türünden olsun kısıtlamaları yapılabilir
    f(12, 5.5); // syntax error
    auto f =[]<typename T>(std::vector<T> x){}; // parametrelerin türleri aynı
}
---
```

```
int main()
{
    auto f = [](auto ...args) {
        foo(std::forward<decltype(args)>(args)...);
    };

    auto g = []<class ...Args>(Args ...args) {
        foo(std::forward<Args>(args)...);

    };
}
```

--- shared_ptr ---

- paylaşımlı sahiplik
- reference counting uygulanması maliyeti arttır
- deleter template tür parametresi değil
- unique_ptr gibi default init edilebilir
- kopyalamaya kağatılmış bir sınıf değil

```
int main()
```

```
{
    // deleter tür olmadıgındani deleter olucaksa argüman olarak geçilicek
    shared_ptr<int> spx {new int{42}, [](int *p){
        cout << p << " adresindeki nesne delete ediliyor...\n";
        delete p;
     }};

    cout << "main devam ediyor\n";
}
// output
0x179c7fe13e0 adresindeki nesne delete ediliyor...
---
```

```
---
int main()
{
    // deleter tür olmadıgındani deleter olucaksa argüman olarak geçilicek
    shared_ptr<Triple> spx {new Triple(1,1,1)};

    // referans sayacının değerini get eder
    cout << spx.use_count() << "\n" ;
    auto x = spx;
    auto y = spx;
    cout << spx.use_count() << "\n" ; // x.use_count()

    cout << "main devam ediyor\n";
}
// output
(1, 1, 1) degerinde 0x21323c213e0 adresinde Triple nesnesi hayata geldi
1
3
main devam ediyor
(1, 1, 1) degerindeki 0x21323c213e0 adresindeki Triple nesnesinin hayati sona erdi
---
```

```
---
void foo(shared_ptr<Triple> p)
{
```

```
    cout << p.use_count() << "\n" ;
    p->set(1,2,3);
    cout << *p << "\n";

}

int main()
{
    shared_ptr<Triple> spx {new Triple(1,1,1)};
    foo(spx);
    cout << "main devam\n";
}
// output
(1, 1, 1) degerinde 0x2e64e0d13e0 adresinde Triple nesnesi hayata geldi
2
(1, 2, 3)
main devam
(1, 2, 3) degerindeki 0x2e64e0d13e0 adresindeki Triple nesnesinin hayati sona erdi
---
```

```
---
int main()
{
    cout << sizeof(unique_ptr<Triple>) << "\n";
    cout << sizeof(shared_ptr<Triple>) << "\n";
    // shared_ptr kontrol bloğuna pointer tutar
}
// output
4
8
---
```

```
---
int main()
{
    auto sp = make_shared<Triple>(10,20,30);
```

```
}
---
```

--> make_shared kontrol bloğu ile nesne adresini birleştirerek,
optimizasyon yapar.

```
---

int main()
{
    auto sp = make_shared<string>("ali");
    cout << *sp->begin();
    sp.reset();
}
---
```

```
---
int main()
{

    unique_ptr<int[]> uptr{new int[10]};
    uptr[2]; // geçerli
    *uptr; // syntax error

    unique_ptr<int> uptr1{new int(10)};
    uptr1[2]; // syntax error
    *uptr1; // geçerli
}
---
```

-> make_shared ve make_unique fonksiyonlarının [] specializationı yok

--- weak_ptr ---

- ayrı bir sınıf
- bir shared_ptr' den bir weak_ptr oluşturulabilir
  - shared_ptr'nin kaynağının oluşturulan weak_ptr yoluyla kontrol edilebilir
    -a) lock() // geri dönüş değeri shared_ptr
      auto sptr = wp.lock(); // veya kapsam azaltmak için,
      if(shared_ptr<string> sptr = wp.lock()){}

    -b) weak_ptr' den shared_ptr olusturulabilir, weak_ptr'yi
      yaratan shared_ptr kaynagı sonlanmışsa exception throw eder

- weak ptr shared_ptr'nin referans sayacını arttırmıyor
- weak_ptr de içerik ve ok operatorü yok
- kaynak hayatta mı sorgulaması için kullanılır

```
int main()
{
    shared_ptr<string> sp{new string{"ali"}};
    cout << sp.use_count() << "\n";

    weak_ptr<string> wp{sp};
    cout << sp.use_count() << "\n";
}
// output
1
1
```

```
int main()
{
    shared_ptr<string> sp{new string{"ali"}};
    cout << sp.use_count() << "\n";

    weak_ptr<string> wp{sp};
    cout << sp.use_count() << "\n";

    sp.reset();
    auto sptr = wp.lock(); // yeni bir shared ptr oluştur
    cout << sp.use_count() << "\n";

    if(sptr)
    {
        cout << "kaynak henuz sonlanmadi\n";
    }
    else {
        cout << "kaynak sonlanmis\n";
    }

}
// output
1
1
0
kaynak sonlanmis
```

```
int main()
{
    shared_ptr<string> sp{new string{"ali"}};
    cout << sp.use_count() << "\n";

    weak_ptr<string> wp{sp};
    cout << sp.use_count() << "\n";
```

```
    sp.reset();
    cout << sp.use_count() << "\n";

    if(!wp.expired())
    {
        cout << "kaynak henuz sonlanmadi\n";
    }
    else {
        cout << "kaynak sonlanmis\n";
    }

}
// output
1
1
0
kaynak sonlanmis
---
```

--> bir nesnenin delete edilmesi için 1 shared_ptr göstermesi gerekir

```
---
int main()
{
    vector<shared_ptr<string>> vec;

    for(int i = 0; i < 10; ++i)
        vec.emplace_back(new string{rname()});

    for(const auto& sp : vec)
        cout << *sp << " ";
```

```
    list<shared_ptr<string>> ls{vec.begin(), vec.end()};

    for(const auto& sp : ls)
        *sp += "can";
    cout << "\n";

    for(const auto& sp : vec)
        cout << *sp << " ";
}
// output
berk hilal ahmet metin melih cetin hulya burhan yurdanur hulusi
berkcan hilalcan ahmetcan metincan melihcan cetincan hulyacan burhancan yurdanurcan
hulusican
---
```

```
---
using svec = std::vector<std::string>;

class Booklist {
public:
    Booklist(std::initializer_list<std::string> blist) : mpvec{ new svec {blist}} {}
    void add_book(std::string name)
    {
        mpvec->push_back(std::move(name));
    }

    void delete_book(const std::string& name)
    {
        if (auto iter = find(mpvec->begin(), mpvec->end(), name); iter != mpvec->end())
            mpvec->erase(iter);
    }
    void print() const
    {
        std::cout << "listedeki kitaplar\n";
        for (const auto& s : *mpvec)
            std::cout << s << "\n";
    }
```

```
private:
    std::shared_ptr<svec> mpvec;
};

int main()
{
    Booklist x{"savas ve baris", "kara kitap", "genclik"};

    x.add_book("mai ve siyah");

    auto y = x;
    y. add_book("masum");

    auto z = y;
    z.add_book("cinali");

    x.delete_book("genclik");
    x.print();

    Booklist a{"c", "python", "java"};
    auto b = a;
    auto c = a;

    // eğer static veri elemanı ile oluşturulsaydı başka Booklist oluşturulamazdı,
        // tek liste olurdu, static member'dan farkı bu
}
---
```

```
---
// Eğer bir sınıfın üye fonksiyonu içinde shared_ptr ile hayatı kontrol edilen * this nesnesini
gösteren
// shared_ptr'nin kopyasını çıkartmak isterseniz sınıfınızı CRTP örüntüsü ile kalıtım yoluyla
std::enable_shared_from_this
// sınıfından elde etmelisiniz


class Myclass : public std::enable_shared_from_this<Myclass> {  //CRTP
public:
        Myclass()
        {
```

```
		std::cout << "Myclass ctor this : " << this << "\n";
	}

	void func()
	{
		std::cout << "Myclass::func() islevi : " << this << "\n";
		//ben func islevinin bir shared_ptr ile kontrol edilen dinamik Myclass nesnesi
icin cagrildigina eminim
		auto sptr = shared_from_this();
		std::cout << "sptr.use_count() = " << sptr.use_count() << "\n";
	}

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


int main()
{
	auto sp = make_shared<Myclass>();
	sp->func();
	//Myclass *p = new Myclass;

	//try {
	//	p->func();
	//}
	//catch (const std::exception &ex) {
	//	std::cout << "hata yakalandi : " << ex.what() << "\n";
	//}

}
---









---
/*
   shared_ptr aliasing constructor
```

```
	shared_ptr ile hayatı kontrol edilen bir sınıf nesnesinin veri elemanlarından birini
	başka bir shared_ptr nesnesinin göstermesini istiyoruz.
	Eğer bir önlem alınmaz ise sahip olan nesneyi gösteren shared_ptr'nin hayatı
bitince elemanı gösteren shared_ptr dangling hale gelirdi.
	Buradaki problemi çözmek için shared_ptr sınıfınıun "aliasing ctor" denilen ctor'u ile
elemana shared_ptr oluşturuyoruz:
	shared_ptr<Member> spm (spowner, spowner->mx);
*/



class Member {
public:
	Member()
	{
		std::cout << "Member constructor\n";
	}

	~Member()
	{
		std::cout << "Member destructor\n";
	}
};

class Owner {
public:
	Owner()
	{
		std::cout << "Owner constructor\n";
	}

	~Owner()
	{
		std::cout << "Owner destructor\n";
	}
	Member mx;
private:

};


using namespace std;

int main()
{
	auto sp = make_shared<Owner>();
   // ana nesnenin ayatı bitse dahi, onun elemanını gösteren shared_ptr oldugu sürece
delete edilmez
```

```
        auto spm = shared_ptr<Member>(sp, &sp->mx);

        cout << "spm.use_count() = " << spm.use_count() << "\n";
        cout << "sp.use_count()  = " << sp.use_count() << "\n";
        (void)getchar();

        sp.reset();
        cout << "after sp.reset() call\n";
        cout << "spm.use_count() = " << spm.use_count() << "\n";
        cout << "sp.use_count()  = " << sp.use_count() << "\n";
        (void)getchar();
}
// output
Member constructor
Owner constructor
spm.use_count() = 2
sp.use_count()  = 2

after sp.reset() call
spm.use_count() = 1
sp.use_count()  = 0

Owner destructor
Member destructor
---
```

```
--- make_uniq' den shared_ptr
int main()
{
        auto up = make_unique<Triple>(1,2,3);
    shared_ptr<Triple> sptr(std::move(up));

    std::cout << (up ? "dolu" : "bos") << "\n";

    if (sptr)
        std::cout << *sptr << "\n";
}
// output
(1, 2, 3) degerinde 0x1fb4e5813e0 adresinde Triple nesnesi hayata geldi
```

```
bos
(1, 2, 3)
(1, 2, 3) degerindeki 0x1fb4e5813e0 adresindeki Triple nesnesinin hayati sona erdi
---




---
// make_uniq' den shared_ptr

std::unique_ptr<Triple> make_triple(int a, int b, int c)
{
    return std::make_unique<Triple>(a,b,c);
}

int main()
{
    std::shared_ptr<Triple> sptr{make_triple(1,2,3)};
    auto p1 = sptr;
    auto p2 = sptr;
    std::cout << p2.use_count() << "\n";
}
// output
(1, 2, 3) degerinde 0x12458ce13e0 adresinde Triple nesnesi hayata geldi
3
(1, 2, 3) degerindeki 0x12458ce13e0 adresindeki Triple nesnesinin hayati sona erdi
---
```

```
    --- input/output operations ---

using ostream = std::basic_ostream<char>;
using wostream = std::basic_ostream<wchar_t>;



int main()
{
        cout << "ali" << 123 << " " << 2.5 << bitset<16>(45) << "\n";
        operator<<(operator<<(operator<<(cout, "ali").operator<<(123), " ").operator<<(2.5),
bitset<16>(45));
}
// output
ali123 2.50000000000101101
ali123 2.50000000000101101







// ostream manipulator
ostream& operator<<(ostream&(*fp)(ostream &))
{
        return fp(*this);
}

ostream& myos(ostream& os)
{
        std::cout << "myos cagirildi\n";
        return os;
}







// endl manipulator
ostream& Endl(ostream& os)
{
        os.put('\n');
```

```
        os.flush();
        return os;
}
```

```
int main()
{
        cout << (10 < 5) << "\n";

        cout.setf(ios_base::boolalpha);
        cout << (10 < 5) << "\n";

        cout.unsetf(ios_base::boolalpha);
        cout << (10 < 5) << "\n";
}
// output
0
false
0
```

```
int main()
{
        cout << hex ; // manipilator
        cout << 65487 << "\n";

        cout.setf(ios::uppercase);
        cout << 65487 << "\n";

        cout.unsetf(ios::uppercase);
        cout << 65487 << "\n";
}
// output
ffcf
```

FFCF
ffcf

```
int main()
{
	cout.setf(ios::boolalpha);
	cout.flags(cout.flags() | ios::boolalpha); // aynı değer
}
```

```
if (os.flags() & ios::boolalpha) {
		true false yazar
	}
	else
		0 1 yazar
```

```
int main()
{
	cout.setf(ios::boolalpha);
	cout.flags(cout.flags() | ios::boolalpha); // aynı anlam

	cout.unsetf(ios::boolalpha);
	cout.flags(cout.flags() & ~ios::boolalpha); // aynı anlam
}
```

// boolalpha ve noboolalpha manipilatorleri

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

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

int main()
{
	bool b1{}, b2{}, b3{};
	cout << Boolalpha << b1 << NoBoolalpha << b2 << b3;
}
```

// output
false00

```
int main()
{
	int x = 10;
	cout << hex << uppercase << showbase << x <<"\n";
}
```

// output
0XA

```
int main()
{
	double dval = 35;
	cout << dval << "\n";

	cout.setf(ios::showpoint);
	cout << showpoint;

	cout << dval;
}
// output
35
35.0000
```

-> bir manipilator unsetf edilne kadar geçerli olur

```
int main()
{
	double dval = 35;
	cout << dval << "\n";

	cout.setf(ios::showpos);
	cout << showpos;

	cout << dval << "\n";
	cout << noshowpos << dval << "\n";

}
// output
35
+35
35
```

// setw(//)

```
int main()
{
	int x{10};
	cout << setw(16) << x << "\n"; // default sağa dayalı yazar
	// parametreli manipilatorler için #include <iomanip>
}
// output
	  10
```

```
int main()
{
	int x{10};
	cout << left; // cout.setf(ios::left, ios::adjustfield);
	cout << setw(16) << x << "\n";
	// parametreli manipilatorler için #include <iomanip>
}
// output
10
```

```
int main()
{
	// bir dosyayı yazdırmanın en kısa yolu
	ifstream ifs{"out.txt"};
	cout << ifs.rdbuf();
}
```

```
int main()
{
	// cout move only type class
	ostream mycout{cout}; // syntax error
}
```

```
int main()
{
  // farklı formatlamalar için önemli bir özellik
	ostream mycout {cout.rdbuf()};
	cout << "aliveli\n";
	mycout << "aliveli\n";
}
// output
aliveli
aliveli
```

```
int main()
{
	int x = 156756;
	// ortaya hizalı yazar
	cout << format("|{:^20|", x) << "\n"; // c++20
}
```

```
class sp {
public:
	sp(int n = 0, char c = ' ') : mx{n}, mc{c}{}
	friend std::ostream& operator<<(std::ostream& os, const sp& x)
	{
		int n = x.mx;
		while(n--)
			os.put(x.mc);
		return os;
	}
private:
	int mx;
	char mc;
};

int main()
{
	cout << "ali" << sp(10) << "veli" << sp(5, '*') << sp(15) << 123;
}
// output
ali         veli*****               123
```

--- ms

```
int main()
{
	// cümleyi ayırma
	string sline;
	cout << "bir cumle girin: ";
	getline(cin, sline);
	istringstream iss(sline);
```

```
        string word;

        while(iss >> word)
                cout << word << "\n";
}
// output
bir cumle girin: ali veli ekrem
ali
veli
ekrem
---
```

--- dosya okuma yazma ---

- ofstream
- ifstream //okuma
- fstream // hem okuma hem yazma interface'i var

-> ortak özellikleri default init edilebilirler,
kopyalanamaz ama taşınabilirler

```
int main()
{
    // dosya varsa sıfırlanır, yoksa oluşturulur
    ofstream ofs{"out.txt"}; // defalut text modu ve yazmaya müsait
```

```
    ofstream ofs{"out.txt", ios::out};  // yazma amaçlı açılacağını
    ofstream ofs{"out.txt", ios::in};  // okuma amaçlı açılacağını
    ofstream ofs{"out.txt", ios::app};  // sona ekleyerek yazma amaçlı açılacağını
    ofstream ofs{"out.txt", ios::trunc};  // dosyanın yazma amaçlı sıfırlanarak açılır
    ofstream ofs{"out.txt", ios::ate};  // dosya göstericisini sonra başlatır
}
```

```
int main()
{
    // okuma
    ifstream ofs{"out.txt"}; // default
    ifstream ofs{"out.txt", ios::in}; // default değer ile aynı anlam
}
```

```
int main()
{

    fstream ofs{"out.txt"}; // default
    fstream ofs{"out.txt", ios::in | ios::out}; // default değer ile aynı anlam
}
```

```
int main()
{
    ofstream ofs{"out.txt"};

    // if(ofs.fail())

    if(!ofs) {
        cerr << "dosya olusturulamadi\n";
        return 1;
    }
}
```

```
int main()
{
    ofstream ofs;

    if(ofs.is_open())
        cout << "acik dosya var\n";
    else
        cout << "acik dosya yok\n";

    ofs.open("out.txt");

    if(ofs.is_open())
        cout << "acik dosya var";
    else
        cout << "acik dosya yok";

}
// output
acik dosya yok
acik dosya var
```

```
int main()
{
    ofstream ofs;

    if(ofs.is_open())
```

```
        cout << "acik dosya var\n";
    else
        cout << "acik dosya yok\n";

    if(ofs)
        cout << "true\n";

    ofs.open("out.txt");

    if(ofs.is_open())
        cout << "acik dosya var";
    else
        cout << "acik dosya yok";

}
// output
acik dosya yok
true
acik dosya var
---
```

--> dosyanın açılması ctor veya open() ile ,
dosyanın kapatılması destructor veya close() ile

-> close() çağırılmazsa nesnenin hayatı bitene kadar,
dosya açık kalır

```
---
std::ofstream open_write_file(const std::string fname)
{
    std::ofstream ofs{fname};
    if(!ofs)
        throw std::runtime_error{"file: " + fname + " cannot created!"};
    return ofs;
}

int main()
{
    auto fs = open_write_file("out.txt") ;

    fs << left << setfill('.');
    for (int i = 0; i < 100; i++)
    {
        fs << setw(20) << rname() << setw(20) << rtown() << "\n";
    }
}
// output
berk.................hatay.................
ahmet................kahramanmaras.......
melih................mugla.................
hulya................kirsehir.............
---
```

```
---
std::ofstream open_write_file(const std::string fname)
{
    std::ofstream ofs{fname};
    if(!ofs)
        throw std::runtime_error{"file: " + fname + " cannot created!"};
    return ofs;
}

int main()
{
    auto ofs = open_write_file("out.txt") ;
```

```
    vector<string> svec;

    rfill(svec, 5000, []{return rname() + ' ' + rtown();});
    copy(svec.begin(), svec.end(), ostream_iterator<string>{ofs, "\n"});
}
// output
hilal mardin
metin artvin
cetin igdir
---
```

```
---
std::ifstream open_read_file(const std::string &fname)
{
    std::ifstream ifs{fname};
    if(!ifs)
        throw std::runtime_error{"file: " + fname + " cannot opened!"};
    return ifs;
}

int main()
{
    auto ifs = open_read_file("out.txt") ;

    // dosyadakileri diziye alır
    vector<string> svec{istream_iterator<string>{ifs}, {}};
    std::cout << "svec.size(): " << svec.size() << "\n"; // 10000

    // en büyük 10 isim listenin başına gelir
    partial_sort(svec.begin(), svec.begin() + 10, svec.end(), greater{});

    // yazdırır
    copy(svec.begin(), svec.begin() + 10, ostream_iterator<string>{cout, "\n"});
}
svec.size(): 10000
```

zubeyde_yalova
zubeyde_trabzon
zubeyde_sirnak

```
int main()
{
    auto ifs = open_read_file("out.txt") ;

    // dosyayı yazdırmanın en kolay yolu,
    // buffer pointerını çıkış akımına inserter olarak vermek
    cout << ifs.rdbuf();

    // dosyaya yazma
    ofstream ofs{"outt.txt"};
    ofs << ifs.rdbuf();
}
```

```
int main()
{
    auto ifs = open_read_file("out.txt") ;

    string x{};

    // dosyadan okur
    while(ifs >> x)
    {
        cout << x;
        getchar();
    }
}
```

```
int main()
{
	auto ifs = open_read_file("out.txt") ;

	int ival;
	double dval;
	string name;

	using person = tuple<string, int, double>;
	vector<person> pvec;

	while(ifs >> ival >> dval >> name) {
	pvec.emplace_back(name, ival, dval);
	}

	sort(pvec.begin(), pvec.end());

	auto ofs{open_write_file("person.txt")};
	ofs << left << setprecision(4) << fixed;

	for(const auto& [name, grade, wage] : pvec) {
	ofs << setw(16) << name << setw(8) << grade << wage <<"\n";
	}
}
```

```
// byte byte okuma
int main()
{
	auto ifs = open_read_file("out.txt") ;

	char c;
```

```
	// byte byte okuma
	while(ifs.get(c)) { // istream& döner, operator bool cagrılır
	cout.put(c);
	}

	// VEYA

	auto ifs = open_read_file("out.txt") ;
	int k;
	while((k = ifs.get()) != EOF){
	cout.put((char)c);

	}
}
---
```

```
---
int main()
{
	auto ifs = open_read_file("out.txt") ;

	string sline;

	// satır satır okuma
	while(getline(ifs, sline)) { // istream& döner
	cout  << "#"<< sline << ")\n";
	}
}
---
```

```
int main()
{
	auto ifs = open_read_file("out.txt") ;

	string sline;
	vector<string> svec;

	// satır satır okuma
	while(getline(ifs, sline)) { // istream& döner
	svec.push_back(std::move(sline)); // taşınır
	}
	cout << "svec.size(): " << svec.size() << "\n";

	for(const auto& s : svec) {
	cout << s << "\n";
	}
}
// output
svec.size(): 20
hilal_mardin
metin_artvin
cetin_igdir
```

```
int main()
{
	auto ifs = open_read_file("out.txt") ;

	string name;
	int ival;
	char c;

	vector<pair<string, int>> myvec;

	while(ifs >> name >> c >> ival) // c nin değeri '='
	myvec.emplace_back(std::move(name), ival);
```

```
        cout << "myvec.size(): " << myvec.size() << "\n";
        for(const auto& s : myvec) {
        cout << s << "\n";
        }
}
// output
myvec.size(): 1000
[haluk, 886]
[hilmi, 915]
[kasim, 835]
---
```

```
---
// formatsız dosya işlemleri
int main()
{
        ofstream ofs{"primes.dat", ios::binary};
        if(!ofs) {
        cerr << "dosya olusturulamadi\n";
        return 1;
        }

        int prime_count{};
        int x{};
        while(prime_count < 1'000'000) {
        if(isprime(x)) {
        ofs.write(reinterpret_cast<char *>(&x), sizeof(int));
        ++prime_count;
        }
        ++x;
        }
}
---
```

```
// formatsız dosya işlemleri
int main()
{
	ifstream ifs{"primes.dat", ios::binary};
	if(!ifs) {
	cerr << "dosya acilamadi\n";
	return 1;
	}

	int x;

	while(ifs.read(reinterpret_cast<char *>(&x), sizeof(int))) // dönüş değeri istream&,
bool operator
	{
	cout << x << " ";
	}
}
// output
... 15485837 15485843 15485849 15485857 15485863
```

```
int main()
{
	ifstream ifs{"primes.dat", ios::binary};
	if(!ifs) {
	cerr << "dosya acilamadi\n";
	return 1;
	}

	constexpr int size{10};
	array<int, size> ar;
```

```
        while(ifs.read(reinterpret_cast<char *>(ar.data()), sizeof(int) * size)) // dönüş değeri
istream&, bool operator
        {
        print(begin(ar), end(ar));
        getchar();
        }

}
// output
2 3 5 7 11 13 17 19 23 29
31 37 41 43 47 53 59 61 67 71
73 79 83 89 97 101 103 107 109 113 ...
---









---
int main()
{
        ifstream ifs{"primes.dat", ios::binary};
        if(!ifs) {
        cerr << "dosya acilamadi\n";
        return 1;
        }

        constexpr int size{10};
        vector<int> ivec(100);

        while(ifs.read(reinterpret_cast<char *>(ivec.data()), sizeof(int) * ivec.size())) // dönüş
değeri istream&, bool operator
        {
        print(begin(ivec), end(ivec));
        getchar();
        }

}
// output
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101
103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197
199 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293 307 311
313 317 331 337 347 349 353 359 367 373 379 383 389 397 401 409 419 421 431
433 439 443 449 457 461 463 467 479 487 491 499 503 509 521 523 541
```

547 557 563 569 571 577 587 593 599 601 607 613 617 619 631 641 643 647 653 659
661 673 677 683 691 701 709 719 727 733 739 743 751 757 761 769 773 787 797 809
811 821 823 827 829 839 853 857 859 863 877 881 883 887 907 911 919 929 937 941
947 953 967 971 977 983 991 997 1009 1013 1019 1021 1031 1033 1039 1049 1051 1061
1063 1069 1087 1091 1093 1097 1103 1109 1117 1123 1129 1151 1153 1163 1171 1181
1187
1193 1201 1213 1217 1223 ...

// dosya kopyalama

```
// filecopy ali.exe veli.exe
int main(int argc, char** argv)
{
	using namespace std;

	if(argc != 3) {
	cerr << "usage: <filecopy> <source file name> <dest file name>\n";
	return 1;
	}

	constexpr int size{1024};
	unsigned char buffer[size];

	ifstream ifs{argv[1], ios::binary};
	if(!ifs) {
	cerr << "file " << argv[1] << "cannot be opened\n";
	}

	ofstream ofs{argv[2], ios::binary};
	if(!ofs) {
	cerr << "file " << argv[2] << "cannot be created\n";
```

```
        }

        streamsize read_bytes{};
        while(ifs.read(reinterpret_cast<char*>(buffer), size)) {
        auto n = ifs.gcount();
        ofs.write(reinterpret_cast<char*>(buffer), n);
        read_bytes += n;
        }
        cout << "file " << argv[1] << " of " << read_bytes << " copied as file "
        << argv[2] << "\n";
}
// /run out.txt primes.txt
---
```

```
---
// RAII
class fmguard {
public:
        fmguard(std::ostream& os) : os_{os}, flags_{os.flags()} {}
        ~fmguard()
        {
        os_.flags(flags_);
        }
private:
        std::ostream& os_;
        std::ios::fmtflags flags_;
};
---
```

```
//////////////////////////////////////////////////////////////////////////
// virtaul dispatch / CRTP karşılaştırması
---
class Animal {
public:
	virtual void cry() = 0;
	virtual ~Animal() {}
};

class Dog : public Animal{
public:
	void cry()override
	{
	std::cout << "kopek havliyor\n";
	}
};


class Cat : public Animal{
public:
	void cry()override
	{
	std::cout << "kedi miyavliyor\n";
	}
};

void game(Animal& r)
{
	r.cry();
}

int main()
{
	Dog mydog;
	Cat mycat;
	// virtual dispatch
	game(mydog);
	game(mycat);
}
// output
kopek havliyor
kedi miyavliyor
---

// virtual dispatch maliyeti:
  - sanal fonk tablosu olusturulur,
  - taban sınıf nesnesi içinde virtual pointer var
  - her sınıfın virtual table' ı ayrı
```

- bir bellek maliyeti ve ilave runtime maliyeti var

--> CRTP ile runtime maliyeti yok!

```
// CRTP
// non-virtual
template <typename Der>
class Animal {
public:
	void make_sound()
	{
	Der& der = static_cast<Der&>(*this);
	der.cry();
	}
};

class Dog : public Animal<Dog>{
public:
	void make_sound()
	{
	std::cout << "kopek havliyor\n";
	}
};


class Cat : public Animal<Dog>{
public:
	void make_sound()
	{
	std::cout << "kedi miyavliyor\n";
	}
};


template <typename T>
void game(Animal<T>& r)
{
	r.make_sound();
}

int main()
{
	Dog mydog;
	Cat mycat;

	mydog.make_sound();
```

```
        mycat.make_sound();
        //veya
        game(mydog);
        game(mycat);
}
// output
kopek havliyor
kedi miyavliyor
kopek havliyor
kopek havliyor
---
////////////////////////////////////////////////////////////////////////////////
```