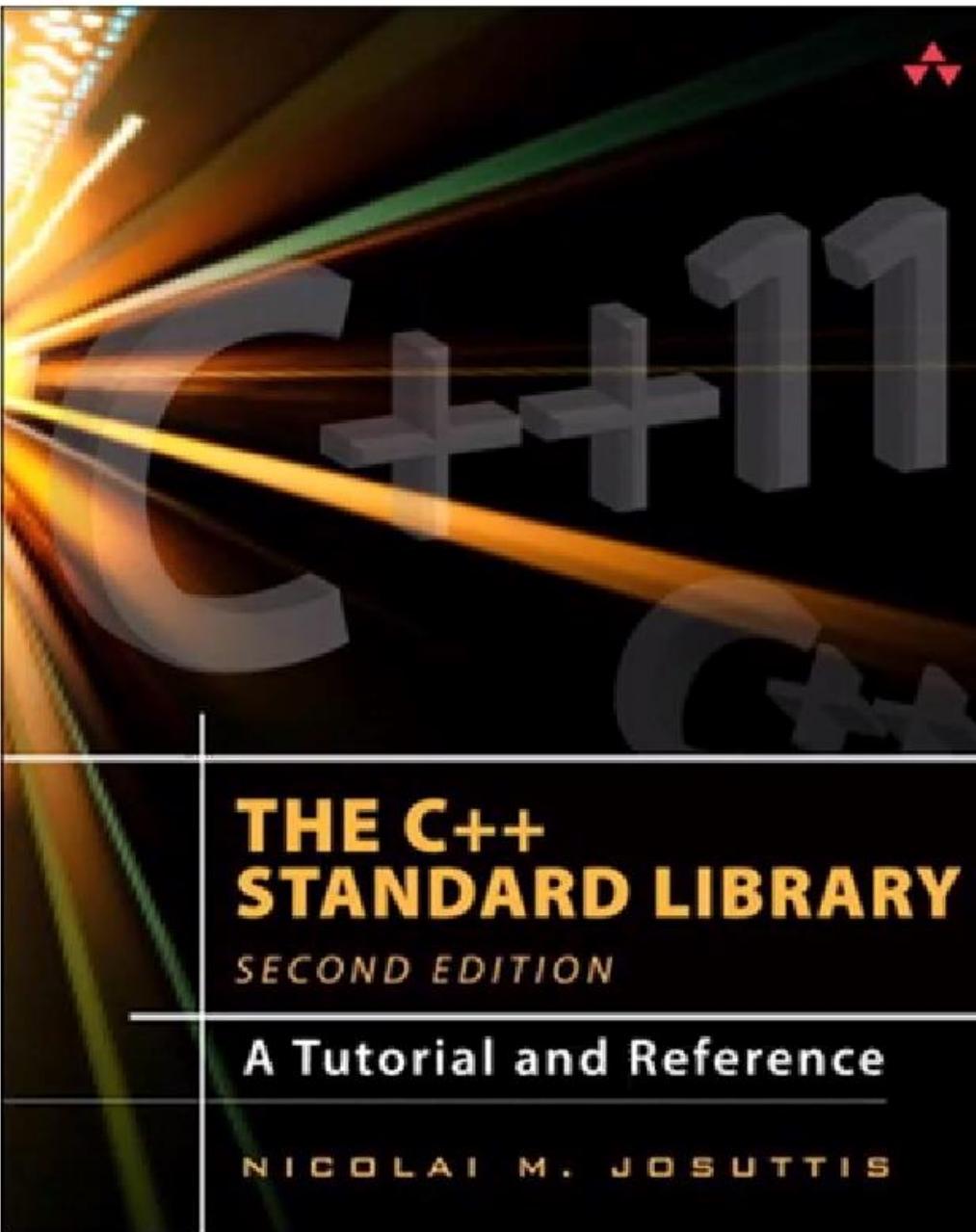


- banking and financial (funds transfer, financial modelling, teller machines)
- classical systems programming (compilers, operating systems, device drivers, network layers, editors, database systems)
- small business applications (inventory systems)
- desktop publishing (document viewers/editors, image editing)
- embedded systems (cameras, cell phones, airplanes, medical systems, appliances, space technologies)
- entertainment (games)
- graphics programming
- hardware design and verification
- scientific and numeric computation (physics, engineering, simulations, data analysis, geometry processing)
- servers (web servers, billing systems)
- telecommunication systems (phones, networking, monitoring, billing, operations systems)
- middleware



```
2
3 int main(void)
4 {
5     const int x = 10;
6     int* p = (int*) &x;
7
8     *p = 678; //ub
9
10
11
12 }
```

```
2
3 int main(void)
4 {
5     int x = 10;
6     int y = 230;
7
8     int* const p1 = &x;
9
10
11
12
13
14 // const pointer to int
15 // top level const
16 // right const
17
18 }
```

```
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
```

```
int main(void)
{
    int x = 10;
    int y = 230;

    const int* p| = &x;
```

x=6 diyebilirz

```
int x, y, z;  
  
int main()  
{  
    int* const p[] = { &x, &y, &z };
```

p ömrü boyunca x y z
'nin elemanlarını
gösterecek

```
+     x,  y,  z,
```

```
5
```

```
6 int main()
```

```
7 {
```

```
8     const int* p[] = { &x, &y, &z };
```

```
9
```

```
10    p[1] = 22;
```

logd /

```
11
```

```
12 }
```

```
13 }
```

```
14 }
```

```
}
```

```
int x, y, z;
```

```
int main()
{
    const int* p[3] = { &x, &y, &z };
```

```
*p[1] = |
```

A yellow circular icon containing a small black cursor arrow pointing upwards and to the right, indicating where the next character will be inserted.

```
2
3 // const ne'den önce gelirse const olan odur
4
5 int main()
6 {
7     int x = 24;
8     int y = 56;
9     int* p1 = &x;
10    int* p2 = &y;
11
12    int const* * const ptr = &p1;
13
14    ptr = &p2;
15    *ptr = &x;
16    **ptr = 876; ~
```

```
3
4     typedef int* iptr;
5
6     int main()
7     {
8         int x = 10;
9         int y = 345;
10
11        const iptr p = &x;
12
13        *p = 46;           I
14
15
16
17
18
19
20
21
22 }
```

```
3
4     typedef int* iptr;
5
6 int main()
7 {
8     int x = 10;
9     const iptr p = &x; I
10
11 //int* const ptr = &x; + -
12 //const int* ptr = &x; -
13
14
15
16
17
18
19 }
```

```
1
2 #define PRIVATE static
3 #define PUBLIC
4
5 PUBLIC int g = 10;
6 PRIVATE double gdval = 4.657;
```

A screenshot of a C++ development environment. The top bar shows tabs for 'notlar.txt' and 'main.cpp', and a dropdown for 'Debug'. The status bar indicates 'Local Windows Debugger' and 'x64'. The code editor displays the following C++ code:

```
#include <iostream>

int main()
{
    bool flag = false;
    int x = 0;

    flag = x;

    std::cout << std::boolalpha << flag;
}


```

The closing brace of the main function is highlighted with a yellow circle containing a white 'I', indicating it is the current cursor position or a point of interest.

```
6 void allocate_memory(int** pd, size_t n)
7 {
8     *pd = (int*)malloc(n * sizeof(int));
9     if (!*pd) {
10         fprintf(stderr, "cannot allocate memory!\n");
11         exit(EXIT_FAILURE);
12     }
13 }
14
15 int main()
16 {
17     int n = 10;
18     int* p = &n;
19
20     allocate_memory(&p, n);
21 }
```

```
class A {  
};
```

most-vexing parse

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

```
int main()  
{  
    B bx(A());  
}
```



```
5  
6  
7 using FUNC = int(int);  
8  
9 int foo(int);  
10  
11 int main()  
12 {  
13     //FUNC* fp = &foo;  
14     int (*fp)(int) = &foo;  
15 }
```

```
void func(int(int));  
void func(int(*)(int));
```

```
int main()
{
    int x = 10;
    int a[20] = { 0 };
    int* ptr = &x;
    I
    // a
    // *ptr
    // ptr[0]
    // x.a
    // ptr->
```

I value reference

	C	C++
<code>++x</code>	R	L
<code>--x</code>	R	L
<code>a = b</code>	R	L
<code>x, y</code>	R	L
<code>x > y ? a : b</code>	R	L



I

```
int main()
{
    int x = 34;
    int* const ptr = &x;
```

l r value expression

```
|  
| int main()  
{  
|     int x = 9;           I  
|     int* p{ &x };  
  
|     int* & r = p; //r ^'ye referans r demek p demek  
  
|     ++* r;             ref  
|     //++* p;  
| }  
|
```

```
int main()    refbasic
{
    int a[5]{ 1, 2, 3, 4, 5 };
    int(&r)[5] = a;
```



```
2
3
4     typedef int inta5[5];
5
6     int main()
7     {
8         int a[5]{ 1, 2, 3, 4, 5 }; //int[5] ==> inta5
9         inta5& r = a;
10        inta5* p = &a;
11
12
13
```

```
typedef int inta5[5];

int main()
{
    int a[5]{ 1, 2, 3, 4, 5 }; //int[5] ==> inta5
    //inta5& r = a;
    int(&r)[5] = a;           I
    //inta5* p = &a;
    //int (*p)[5] = &a;
```

```
using inta5 = int[5];
```

```
int g = 50;

int& foo()
{
    return g;
}

int main()
{
    std::cout << "g = " << g << "\n";
    ++*foo();
    std::cout << "g = " << g << "\n";
    *foo() = 999;
    std::cout << "g = " << g << "\n";

    int* p = foo();
}
```

```
3 int g = 50;  
4  
5 int& foo()  
6 {  
7     return g;  
8 }
```

fonk lvalue olduu için
legaldir

```
9  
10 int main()  
11 {  
12     std::cout << "g = " << g << "\n";  
13     foo() = 888;  
14 }  
15 }
```

```
int* foo()
{
    int x;
    scanf("%d", &x);
    return &x;
}
```

otomatik bir deeri döndürüyoruz
dive hata

```
1
2
3
4 int* foo()
5 {
6     static int g = 6;
7     // 
8     return &g; I
9 }
```

legal

```
3
4 const char* foo(void)
5 {
6
7     //code
8
9     return "necco";
10}
```

I

```
int& foo(int &r) {
    r *= 2;
    return r;
}
```

string statitctr

```
int main()
{
    int x = 10;

    int& r = foo(x);
    ++r;

    std::cout << "x = " << x << "\n";
}
```

```
1 #include <iostream>
2
3 using T = int;
4
5 void f1(T* );
6 void f1_(T &);
7
8 void f2(const T* );
9 void f2_(const T& );
10
11 T* f3(void);
12 T& f3_(void);
13
14 const T* f4();
15 const T& f4_();
16
17
18 int main()
19 {
20     ...
21 }
```

```
3
4 int main()
5 {
6     int x = 10;
7     const int* ptr = &x;
8
9 }
```

constructual const

```
auto x = expr;  
=====  
auto &y = expr; I  
=====  
auto &&z = expr; //forwarding reference  
=====
```

Dikkat!

Bir değişken isminin oluşturduğu ifadelerin değer kategorisi
L value

```
int a[5]{};
```

```
//auto &b = a; //int[5]
```

```
auto& r1 = "fatih"; //const char[6]
```

```
using carr = const char[6];

int main()
{
    auto& r1 = "fatih"; //const char[6]

    const char(&r2)[6] = "fatih";
    carr &r3 = |
```

 int foo(int);

using FPTR = int (*)(int);

```
int main()
{
    auto fp1 = foo; //int (*)(int)
    FPTR fp2 = foo;
```

```
2
3 int foo(int);
4
5 using FUNCTION = int(int);
6
7 int main()
8 {
9     FUNCTION *fp = foo;
10 }
11 }
```

legal



```
int foo(int);
```

```
using FUNCTION = int(int);  
using FPTR = int(*)(int);
```

I

```
- int main()  
{  
    FUNCTION *fp1 = foo;  
    FPTR fp2 = foo;  
    auto fp3 = foo;  
}
```

```
#include <iostream>
#include <ctime>

void process_date(int mday = -1, int month = -1, int year = -1)
{
    if (year == -1) {
        std::time_t timer{};
        std::time(&timer);
        auto tp = localtime(&timer);
        year = tp->tm_year + 1900;
        if (month == -1) {
            month = tp->tm_mon + 1
        }
    }
}

int main()
{
    process_date(12, 5, 1993);
    process_date(12, 5);
    process_date(12);
    process_date();
```

```
void process_date(int mday = -1, int month = -1, int year = -1)
{
    if (year == -1) {
        std::time_t timer{};
        std::time(&timer);
        auto tp = localtime(&timer);
        year = tp->tm_year + 1900;
        if (month == -1) {
            month = tp->tm_mon + 1;
            if (mday == -1) {
                mday = tp->tm_mday;
            }
        }
    }
}
```

```
int main()
{
    process_date(12, 5, 1993);
    process_date(12, 5);
    process_date(12);
```

```
3
4
5 int main()
6 {
7     int x = 45235;
8     const int* cptr = &x;
9     //
10
11     int* p = (int *)cptr;
12
13     constcast
14
15
16 }
17
18
```



```
char* Strchr(const char* p, int c)
{
    while (*p) {
        if (*p == c) { I
            return (char*) p;
        }
        ++p;
    }

    if (c == '\0')
        return (char*) p;

    return NULL;
}
```

cstylecast

```
5 struct Myclass,
6
7     Myclass f1(Myclass);
8     Myclass* f2(Myclass* );
9     Myclass& f3(Myclass&);  I
10    Myclass* f4();
11
12    typedef Myclass * MyclassPtr;
13    using MyclassPtr = Myclass*;
```

```
14
15    /
```

incomplite

```
16
17 int main()
18 {
19     Myclass *ptr = nullptr;
20
21     ptr = f4();
22 }
```

```
1 enum Color : unsigned int {Blue, Black, Purple, Brown};  
2  
3
```

I

```
1 // #include "screen.h"  
2  
3 enum class ScreenColor { Blue, Black, Purple, Brown, Red };  
4  
5 void foo()  
6 {  
7     using enum ScreenColor;  
8  
9     auto x = Blue; I  
10 }  
11  
12
```

```
constexpr bool isprime(int x)
{
    if (x < 2)
        return false;
    if (x % 2 == 0)
        return x == 2;

    if (x % 3 == 0)
        return x == 3;

    if (x % 5 == 0)
        return x == 5;

    for (int i = 7; i * i <= x; i += 2)
        if (x % i == 0)
            return false;

    return true;
}
```

I

```
void foo(int(int));  
void foo(int(*)(int));
```

redeclaration

I

```
4
5     int a[10]{};
6     using inta10 = int[10];
7     using inta10ptr = int(*)[10];
8
9
10    int(* foo())[10]
11 {
12     return &a;
13 }
14
15    inta10* bar()
16 {
17     return &a;
18 }
19
20    inta10ptr baz()
21 {
22     return &a;
23 }
```

scoped enum ==> int —

unscoped enum ==> int +

double ==> char +

int * ==> void * +

int * ==> bool +

nullptr_t ==> int *

int

bool => int * —

```
4 class MyClass {  
5     public:  
6         MyClass();  
7         MyClass(int);  
8     };  
9 }
```

user-defined conversion

```
0  
1  
2 int main()  
3 {  
4     MyClass x;  
5  
6     x = 5;  
7 }
```

```
class Myclass {  
public:  
    operator int()const;  
};
```

user-defined conversion

```
int main()  
{  
    Myclass x;  
    int ival{};  
  
    ival = x;  
}
```

```
9 void func(const Myclass&)
10 {
11     std::cout << "const LVALUE REF\n";
12 }
13
14 void func(Myclass&)
15 {
16     std::cout << "LVALUE REF\n";
17 }
18
19 void func(Myclass&&)
20 {
21     std::cout << "RVALUE REF\n";
22 }
23
24
25
26
27 int main()
28 {
29     Myclass m;
30     const Myclass cm;
31
32     func(m);
33     //func(cm);
34     //func(Myclass{});
35 }
```

```
3  ifndef __cplusplus
4  extern "C" {
5  #endif
6  extern c
7      int fnec(int a, int b);
8      int f1(int a, int b);
9      int f2(int a, int b);
10     int f3(int a, int b);
11
12  ifdef __cplusplus
13  }
14  endif
15
```

I

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

```
//void func(Myclass *p, int);
```



this kullanımı

```
3
4
5 class Neco {
6     public:
7         void func();
8     private:
9         int x;
10    };
11
12 void f1(Neco* );
13 void f2(Neco & );
14 void f3(Neco );
15
16 void Neco::func()
17 {
18     f1(this);
19     f2(*this);
20     f3(*this);
21 }
```

I

```
5
6 class Myclass {
7
8 public:
9     Myclass& foo()
10    {
11        //return *this;
12    }
13
14};
```

The code shows a C++ class definition named Myclass. It contains a public member function foo() which returns a reference to the object itself. The code is annotated with several vertical lines and boxes:

- A thick green vertical line is positioned to the left of the first brace of the class definition.
- A thin grey vertical line extends from the top of the class definition down to the final closing brace at the bottom.
- A thin orange vertical line is positioned to the left of the opening brace of the foo() function.
- A small square box is placed above the opening brace of foo().
- A dashed vertical line connects the top of the class definition to the top of the foo() function body.
- A rectangular box highlights the line "Myclass& foo()".
- A small square box is placed above the opening brace of the foo() function body.
- A dashed vertical line connects the top of the foo() function body to the brace of the function body.
- A rectangular box highlights the line "return *this;".
- A dashed vertical line connects the end of the foo() function body to the brace of the class definition.
- A small square box is placed above the final closing brace at the bottom.
- A dashed vertical line connects the end of the class definition to the bottom of the page.

A black rectangular box labeled "legal" is overlaid on the text between the two braces at the bottom of the code.

```
public:  
    Myclass(); //default ctor  
    ~Myclass(); //destructor  
  
    Myclass(const Myclass&); //copy ctor  
    Myclass(Myclass&&); //move ctor  
  
    Myclass& operator=(const Myclass&); //copy assignment  
    Myclass& operator=(Myclass&&); //move assignment  
  
    Myclass(int);  
};
```

```
01 class Myclass {  
02     public:  
03         delete eder  
04     private:  
05         const int mx;  
06     };
```

```
01  
02  
03  
04  
05 derleyicinin yazdığı default ctor  
06 sınıfın bütün veri elemanlarını  
07 default init eder:  
08
```

```
default(ed)
```

```
=====
```

I

```
A, B ve C
```

copyconst

```
class MyClass {
```

```
public:
```

```
    MyClass(const MyClass& other) :
```

```
        m_a(other.m_a),
```

```
        m_b(other.m_b),
```

```
        m_c(other.m_c))
```

```
{
```

```
}
```

```
private:
```

```
    A m_a;
```

```
    B m_b;
```

```
    C m_c;
```

```
};
```

```
class String {
public:
    String(const char* p) :
        mlen{ std::strlen(p) },
        mp{ static_cast<char*>(std::malloc(mlen + 1)) }
    {
        std::strcpy(mp, p);
    }

    String(const String& other) : mlen(other.mlen), mp{ static_cast<char*>(std::malloc(mlen + 1)) }
    {
        std::strcpy(mp, other.mp);
    }

    ~String()
    {
        std::free(mp);
    }

    void print()const
    {
        std::cout << mp << '\n';
    }
}
```

copyctorbutpointerli



```
6 void func(Myclass&&)
7 {
8     std::cout << "func(Myclass &&)\n";
9 }
10
11 void func(const Myclass&)
12 {
13     std::cout << "func(const Myclass &)\n";
14 }
15
16 //L value expr.
17
18 int main()
19 {
20     Myclass m;
21     I
22     func(m);
23 }
24
```

I value

```
8 void func(Myclass&&)
9 {
10    std::cout << "func(Myclass &&)\n";
11 }
12
13 void func(const Myclass&)
14 {
15    std::cout << "func(const Myclass &)\n";
16 } //L value expr.
17
18 int main()
19 {
20    Myclass m;
21
22    func(static_cast<Myclass&&>(m));
23 }
```

LVALUE'Y R VALUE ÇEVİRDK

```
6 void func(Myclass&&)
7 {
8     std::cout << "func(Myclass &&)\n";
9 }
0
1 void func(const Myclass&)
2 {
3     std::cout << "func(const Myclass &)\n";
4 }
5
6 //L value expr.
7 // misnomer
8
9 int main()
0 {
1     Myclass m;
2
3     func(static_cast<Myclass&&>(m));
4     func(std::move(m));
5 }
```

//programcı bir sınıf için destructor bildirmiș

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

```
class Myclass {  
public:  
    Myclass() = default;  
    ~Myclass();  
    Myclass(const Myclass&) = default; //!  
    Myclass& operator=(const Myclass&) = default; //!  
};
```

//programcı bir sınıf için default ctor bildirmiş

```
class Myclass {  
public:  
    Myclass(const Myclass&);  
};
```

```
class Myclass {  
public:  
    ~Myclass() = default;  
    Myclass(const Myclass&);  
    Myclass& operator=(const Myclass&) = default; // !!  
};
```

//programcı bir sınıf için copy assignment bildirmiş

```
class Myclass {  
public:  
    Myclass& operator=(const Myclass&);  
};
```

```
class Myclass {  
public:  
    Myclass() = default;  
    ~Myclass() = default;  
    Myclass(const Myclass&) = default; //!!  
    Myclass& operator=(const Myclass&);  
};
```

I

//programcı bir sınıf için move ctor bildirmiş

```
class Myclass {  
public:  
    Myclass(Myclass&&);  
};  
  
class Myclass {  
public:  
    ~Myclass() = default;  
    Myclass(const Myclass&) = delete;  
    Myclass& operator=(const Myclass&) = delete;  
    Myclass(Myclass&&);  
};
```

I

//programcı bir sınıf için move assignment

```
class Myclass {  
public:  
    Myclass& operator=(Myclass&&);  
};  
  
class Myclass {  
public:  
    Myclass() = default;  
    ~Myclass() = default;  
    Myclass(const Myclass&) = delete;  
    Myclass& operator=(const Myclass&) = delete;  
  
    Myclass& operator=(Myclass&&);  
};
```

	default constructor	destructor	copy constructor	copy assignment	move constructor	move assignment
hiçbiri	defaulted	defaulted	defaulted	defaulted	defaulted	defaulted
constructor	not declared	defaulted	defaulted	defaulted	defaulted	defaulted
default constructor	user declared	defaulted	defaulted	defaulted	defaulted	defaulted
destructor	defaulted	user declared	defaulted	defaulted	not declared	not declared
copy constructor	not declared	defaulted	user declared	defaulted	not declared	not declared
copy assignment	defaulted	defaulted	defaulted	user declared	not declared	not declared
move constructor	not declared	defaulted	deleted	deleted	user declared	not declared
move assignment	defaulted	defaulted	deleted	deleted	not declared	user declared

```
5  
6 void func(const std::string& r)  
7 {  
8     std::string s = r;
```

Üstteki l istiyor ve kullanacak, alttaki r
value alcak ve çalacak

```
10  
11 }  
12 void func(std::string&& r)  
13 {  
14     std::string s = std::move(r);  
15 }  
16 }
```

Bir referans (const sol tareaf referansı ya da sağ taraf referansı)
Bir geçici nesneye (bir Pr value expression kategorisindeki) bir sınıf nesnesine
bağlandığında geçici nesnenin hayatı referans olan ismi kapsamına endekslenir.
referans olan ismin kapsamının sonuna kadar geçici nesne hayatı tutulur.]

12
13
14
15
16
17
18
19
20
21 user defined conversion

22
23 conversion ctor

24 type-cast operator function

25
26
27 dilimizin örtülü dönüşümler konusunda çok önemli bir kuralı var:

28
29 standard conversion + UDC

30 UDC + standard conversion

31
32 UDC + UDC

33
34 conversion sequence

```
5
6 class Myclass {
7     public:
8         Myclass();
9         Myclass(int);
10    };
11
12
13
14
15 int main()
16 {
17     Myclass m;
18
19     double dval{ 2.3 };
20     //double ==> int ==> Myclas
21     m = dval;
22
23
24 }
```



```
5
6 class Nec {
7 public:
8     Nec();
9     explicit Nec(int);
10 }
11
12 void foo(Nec);
13 void bar(const Nec&);
14 void baz(Nec&&);

15
16 int main()
17 {
18     int ival{ 2456 };

19
20     Nec mynec;

21
22     mynec = static_cast<Nec>(ival);
23     mynec = (Nec)ival;
24     mynec = Nec(ival); I
25 }
```

```
11 public:  
12     explicit Myclass(int)  
13     {  
14         std::cout << "Myclass(int)\n";  
15     }  
16  
17     Myclass(double)  
18     {  
19         std::cout << "Myclass(double)\n";  
20     }  
21  
22 };
```

double girer hata yok

```
23  
24 int main()  
25 {  
26     Myclass x = 5;  
27 }
```



```
class MyClass {};  
void foo(MyClass);  
elision  
int main()  
{  
    foo(MyClass{});  
}  
I
```

```
// named return value optimization
std::string get_str()
{
    std::string str;
    //code
    str = "hasan";
    str += "can";

    return str;
}
```

nrvo retval ile str optimize olur aynı bellekte

```
int main()
{
    std::string retval = get_str();
```

I

59
60
61 NRVO is not mandatory
62 for NRVO a callable copy/move ctor is req
63
64
65
66
67 nRVO
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

is not mandatory
for NRVO a callable copy/move ctor is req
Tipik bir C++ derleyicisi hangi durumlarda NRVO yapabilir?
Tipik bir C++ derleyicisi hangi durumlarda NRVO yapamaz?

3847 asla client kodlar için friend bildirimi yapmayın!!
3848 bunun yerine bu öğeleri public yapın.

3849
3850 Bir sınıf

- 3851
3852 a) bir "free function" (global function) için
3853
3854 b) bir başka sınıfın üye fonksiyonu için
3855
3856 c) Bir başka sınıfın tüm fonksiyonları için
3857

3858 Dikkat!

3859 I
3860 frişend bildiriminin sınıfın
3861 public, protected ya da private bölümünde yapılması
3862 bildirimin anlamını ve etkisini değiştirmez

```
class Nec {
public:
    void foo(int);
};

class Myclass {
private:
    friend void Nec::foo(int);
    void ~~~~~func();
};

void Nec::foo(int)
{
    Myclass m;

    m.func();
}
```

```
4 class Myclass;
5
6 class Nec {
7 public:
8     Nec(const Myclass&);
9     Nec();
10};
11
12
13 class Myclass {
14 public:
15     friend Nec::Nec(const Myclass&);
16     friend Nec::Nec();
17 private:
18     int mx;
19};
20
21
22
23 Nec::Nec(const Myclass& m)
24 {
25     auto val = m.mx;
26 }
27
28 Nec::Nec()
29 {
30     Myclass m;
31     m.mx = 10;
32     // 
33 }
```

I

3926
3927 eğer A sınıfı B sınıfına friend'lik vermiş ise
3928 B sınıfı da A sınıfına friend'lik vermiş kabul edilmez.

3930
3931 arkadaşımın arkadaşı benim arkadaşım değildir

3932
3933
3934 A sınıfı B sınıfına friend'lik vermiş olsun
3935 B sınıfı da C sınıfına friend'lik vermiş olsun

I

3936
3937 bu durumda A sınıfı C sınıfına friend'lik vermiş olmaz
3938 ya ni C sınıfı A sınıfının private bölümüne erişemez

operator overloading mekanizmasının genel kuralları

bu mekanizma ile çağrılan fonksiyonlara "operator function" denir.

Bir "operator function"

ya bir free function

ya da non-static member function olmak zorunda

sınıfların static üye fonksiyonları operator fonksiyon u olamaz

=====

I
en az bir operand

ya bir sınıf türünden

ya enum türlerinden olmali

=====
olmayan operatör overload edilemez
=====

overload edilemeyen operatörler var

```
::          scope resolution
.
sizeof
ternary    ?  :
.*
```

operator=

operator()

operator+

operator->

operator%

operator+=

operator-=

operator*=

operator>>=

aşağıdaki operatörler yalnızca
member operator function

[]

->

()

=

tür dönüştürme operatörleri

I

```
2
3 class Myclass {
4 public:
5     Myclass operator+()const
6     {
7         std::cout << "sign operator\n";
8         return *this;
9     }
10
11    Myclass operator+(const Myclass&)const
12    {
13        std::cout << "addition operator\n";
14        return *this;
15    }
16
17};
```

addition

```
18
19 int main()
20 {
21     Myclass mx;
22
23     auto val1 = mx + mx;
24     auto val2 = mx.operator+(mx);
25 }
```

```
class Myclass {
public:
    Myclass operator+()const
    {
        std::cout << "sign operator\n";
        return *this;
    }

    Myclass operator+(const Myclass&)const
    {
        std::cout << "addition operator\n";
        return *this;
    }
};

int main()
{
    Myclass mx;

    auto val1 = +mx;
    auto val2 = mx.operator+();
}
```

```
Microsoft Visual Studio
sign operat
sign operat
```

```
D:\CPP_J
Press any
```

```
class Date {  
public:  
    Date(int, int, int);  
};  
  
std::ostream& operator<<(std::ostream&, const Date&);  
  
int main()  
{  
    using namespace std;  
  
    Date mydate{ 3, 5, 1998 };  
    ///  
    cout << mydate;           I  
    operator<<(cout, mydate);
```

```
// ADL  
// argument dependant lookup  
  
int main()  
{  
    endl(std::cout);  
}
```

Dikkat!

örtülü (*implicit*) olarak yapılan dönüşümler çok tehlikelidir.

Yanlışlıkla yapılan tür dönüşümlere kodlama hatalarının en tipik kaynaklarından biridir.

hack-reference

```
1286  
1287 bir donksiyon adresi türüne dönüşüm yapan  
1288 t.d.o  
1289 olabilir  
1290  
1291 Dikkat!  
1292  
1293 enun türleri sınıf türleri olmamasına karşın;  
1294  
1295 enum türleri için de global operator fonksiyonları tanımlamabilir
```

```
int main(void)  
{  
    int x;  
  
    while (cin >> x) {  
        cout << x << "\n";  
    }  
}
```

```
2  
3 class Myclass {  
4  
5     static int x;  
6 };
```

```
7  
8  
9 /*  
10  sınıfların statik veri elemanları sınıf nesnesine ait değil.  
11  fiziksl olaraj sınıf nesnesinin içinde değil  
12  sınıfa ait statik ömürlü değişkenler
```

```
13  
14  class scope  
15  access control  
16  lojik açıdan sınıfla ilgili
```

I

*/

```
class Nec;  
  
/// myclass.h  
class Myclass {  
  
public:  
    static Nec mx;  
};
```

legaldir, static olmasa illegal

```
/// myclass.h
class Myclass {
public:
    static int mx;
};

int main()
{
    Myclass m1;
    Myclass m2;

    m1.mx = 5; //gecerli ama dikkat! cunku mx degiseninin n1 m1 ile ne m2 ile bir ilişkisi yok
    m2.mx = 15;
}
```

geçerli

```
3
4 // class MyClass {
5
6     public:
7         int my;
8         static int mx;
9
10    };
11
12
13 int main()
14 {
15     MyClass m1;
16     MyClass m2;
17     MyClass* p = new MyClass;
18
19     m1.mx = 5;
20
21     m2.mx = 5;           I
22
23     MyClass::mx = 5;
24
25     p->mx = 5;
26 }
27
28
```

Modern C++ öncesinde de
sınıfların
const static ve tam sayı turlerinden
veri elemanlarına
sınıf tanımı içinde ilk değer verilebilir

```
1 //myclass.h
2 #include <vector>
3 #include <string>
4
5
6 class Myclass {
7
8     public:      head only library
9
10    private:
11        inline static std::vector<int> msvec{};
12        inline static std::string msname{ "necati ergin" };
13    };
```

I

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

legal çünkü statik datamemberin thisi yok

```
class Myclass {  
public:  
    void foo() const  
    {  
        mx = 10; I  
    }  
  
private:  
    static int mx;  
};
```

```
class Myclass {  
public:          legal  
    static void foo()  
    {  
        mx = 5;    I  
    }  
  
private:  
    static int mx;  
};
```

```
1 #include <string>
2 #include <iostream>
3
4 1 basar
5
6 class Myclass {
7
8 public:
9     void foo() const
10    {
11        mx = 10; I
12    }
13
14 private:
15     static int mx;
16 };
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
```

```
3
4
5 class MyClass {
6
7 public:
8     static int foo()
9     {
10         return 1;
11     }
12     static int x;
13 };
14
15
16
17 int foo()
18 {
19     return 99999;
20 }
21
22
23 int MyClass::x = foo();
24
25 int main()
26 {
27     std::cout << MyClass::x << "\n"
28 }
29
```

```
int foo(int),
```

```
int foo(int) I
```

```
using FPTR = int (*)(int);
```

```
int main(void)      function type
{
    auto fp1 = foo;
    auto fp2 = &foo;
    int (*fp3)(int) = &foo;
    FPTR fp4 = &foo;
    decltype(fp1) fp5 = &foo;
}
```

```
3
4
5 class MyClass {
6
7 public:
8     static int foo()
9     {
10         return 1;
11     }
12     static int x;
13 };
14
15
16
17 int foo()
18 {
19     return 99999;
20 }
21
22
23 int MyClass::x = foo();
24
25 int main()
26 {
27     std::cout << MyClass::x << "\n"
28 }
29
```

```
public:  
    static Complex create_cartesian(double r, double i)  
    {  
        return Complex{ r, i };  
    }  
    static Complex create_polar(double angle, double distance)  
    {  
        return Complex{ angle, distance, 0};  
    }  
private:  
    Complex(double r, double i);  
    Complex(double a, double d, int);  
};
```

named ctor

```
int main()  
{  
    auto c1 = Complex::create_cartesian(1.2, 5.6);  
    auto c2 = Complex::create_polar(.5, .8347);  
}
```

```
#include <vector>
```

sadece dinamik nesne olutmaya izin veriyor

```
class Donly {  
public:  
    static Donly* create()  
    {  
        return new Donly{};  
    }  
    Donly(const Donly&) = delete;  
    Donly& operator=(const Donly&) = delete;  
private:  
    Donly();  
}
```

```
class Myclass {  
  
public:  
    static Myclass& get_instance()  
    {  
        if (!mp) {  
            mp = new Myclass;  
        }  
  
        return *mp;  
    }  
    void foo();  
    void bar();  
    void baz();  
private:  
    inline static Myclass* mp{};  
};  
  
int main()  
{  
    Myclass::get_instance().bar();  
  
    //  
    Myclass::get_instance().baz();  
    auto& sng = Myclass::get_instance();  
  
    sng.bar();  
    sng.foo();  
}
```

singleton(tek nesne
örüntüsü) abi simdi bi
nesne olsun heryerden
erisilebilsin. 1 adet
olsun statik iceriyor bu
yuzden, destructor
olsun mu vb

```
//fonksiyona sol taraf değeri  
  
//a)func içinde bir myclass nesnesi oluşturacaksınız  
//  değerini sizin fonksiyona gönderdiğiniz nesneden alacak  
//  
//b) foo içinde bir Myclass nesnesine atama yapacaksınız  
//  değerini sizin fonksiyona gönderdiğiniz nesneden alacak  
  
void func(const Myclass &m)  
{  
    Myclass x = m;  
}  
  
void func(Myclass&& m)  
{  
    Myclass x = std::move(m);  
}
```

```
int main()
{
    int a[5] = { 2, 4, 6, 8, 10 };

    for (int ival : a) {
        std::cout << ival << " ";
    }

    /*for (int* ptr = a; ptr != a + 5; ++ptr) {
        int ival = *ptr;
        std::cout << ival << " ";
    }*/
}
```

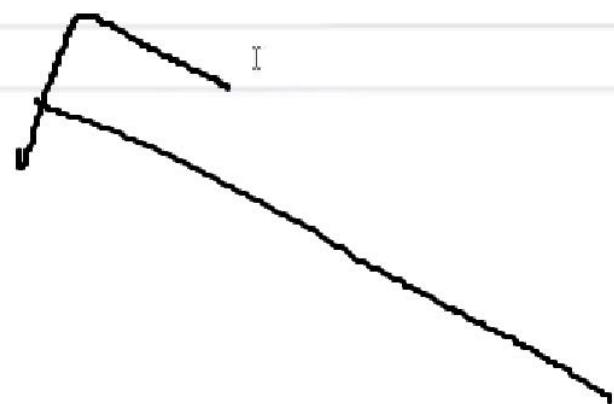
```
#include <iostream>

int a[5] = { 1, 2, 3, 4, 5 };

int(* foo())[5]
{
    return &a;
}

int main()
{
    auto p = foo();

    for (int i = 0; i < 5; ++i) {
        std::cout << (*p)[i] << " ";
    }
}
```



```
int foo();
```

ival10dan büyükse ife girer c++17

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

    if (int ival = foo(); ival > 10) {
        cout << ival << "\n";
    }
    else {
        ival;
    }
}
```

Dikkat!

nested type'lara ilişkin kurallar
C ve C++ dillerinde farklı

Dikkat!

enclosing type member type'in private bolumune erişemez, ancak
member type enclosing type'in private bolumune erişebilir.

```
    class Nested {};
public:
    static Nested foo() { return Nested{}; }
private:
};
```

I

```
// bir isim kaynbak kodda yazilmadigi surece
// erisim kontrolu yapilanaz
```

```
int main()
{
    auto x = Myclass::foo();
```

```
544 //myclass.h  
545 #include "a.h"  
546 #include "b.h"  
547 #include "c.h"
```

pimpl

```
548  
549  
550 class Myclass {  
551  
552 private:  
553     A ax;  
554     B bx;  
555     C ck;  
556 };
```

```
557 ======  
558 //myclass.h
```

```
559  
560 class Myclass {  
561  
562 private:  
563     struct pimpl;  
564     pimpl *mp;  
565 }
```

```
78 private:
79     struct pimpl;
80     pimpl *mp;
81 };
82
83
84 //myclass.cpp
85
86 Myclass::Myclass() : mp (new pimpl)
87 {
88     //
89 }
90
91 struct Myclass::pimpl {
92     A ax;
93     B bx;
94     C cx;
95 };
96
97 void Myclass::foo()
98 {
99     mp->ax.f2();
100    mp->bx.bar();
101    mp->cx.baz();

---


102 }
103
```

pimpl

1

bir namespace içindeki ismi

o namespace dışında;

namespace ismi ile nitelendeden kullanabilir miyim?

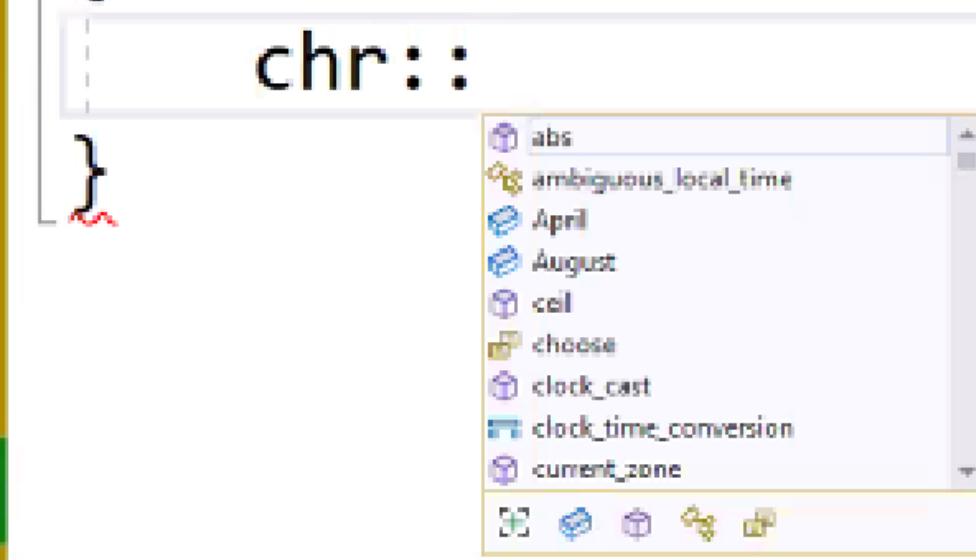
using declaration

using namespace (directive) decl;

ADL (argument dependant lookup)

```
namespace chr = std::chrono;

int main()
{
    chr::
```



The screenshot shows an IDE's code editor with a completion dropdown open. The code above has been typed, and the cursor is at the end of the line. A dropdown menu lists various members of the `std::chrono::duration<Rep, Period>` class, including `abs`, `ambiguous_local_time`, `April`, `August`, `ceil`, `choose`, `clock_cast`, `clock_time_conversion`, and `current_zone`. Below the list are standard C++ completion icons for opening brackets, closing brackets, and so on.

```
//unnamed namespace
```

```
I
```

```
-> namespace {
```

```
    ...
```

```
}
```

```
2 namespace {
3     int a = 5;
4 }
```

```
5
6 namespace {
7     int b = 15;
8 }
```

sadece kod dosyasında kullanmak istiyorsan
global static kullanmak yerine unnamed
namespace kullanabilirsin

```
10
11 int main()
12 {
13     a = b;
14 }
```

```
//#include ...
namespace neco_xyz_proj {
    int ival = 5;
}

namespace nec = neco_xyz_proj;

int main()
{
    neco_xyz_proj::ival++;
    nec::ival++;
}
```

688
689 using bildirimi, bildirime konu ismi, bildirimin yapıldığı scope
690 Yani o isim sadece o kapsamda bildirilmiş gibi ele alınır.

691
692
693 unnamed namespace (isimsiz -anonim isim alanı)

694
695 ADL (argument dependant lookup)

696 argümana bağlı isim arama

697

698

699

700

unnamed namespace (isimsiz -anonim isim alanı)

ADL (argument dependant lookup)

argümana bağlı isim arama

Eğer bir fonksiyon çağrıısı

nitelenmemiş bir isim kullanılarak yapıldığında

söz konusu fonksiyon ismi

eger fonksiyona gönderilen argüman ya da argümanlardan biri

bir namespace içinde tanımlanan bir türe ilişkin ise

söz konusu fonksiyon ismi

o namespace'te de (argümanın iliskin olduğu) aranır

```
3
4
5  namespace neco {
6      enum Color {Blue, Purple, White};
7
8      void foo(Color);
9  }
10
11
12
13  int main()
14  {
15      foo(neco::Blue);
16  }
17
18
```

```
4
5  namespace neco {
6    class Erg {
7    };
8
9
10   void foo(Erg);
11 }
12
13 void foo(neco::Erg);
14
15
16 int main()
17 {
18   neco::Erg x;
19
20   foo(x);
21 }
22
23
24
```

ambiguity

```
- int main()          BOL ADL
{
    std::vector<int> ivec;
    //vector<int>::iterator
    sort(begin(ivec), end(ivec));
}
```

```
namespace x{
    inline namespace Y {
        int ival;
    }
}
```

inline namespace bildirimi ile x'ten y
namespace'ine eriebildik

```
int main()
{
    x::ival
}
```

```
namespace nec {  
  
#ifdef VER1  
    inline  
#endif  
namespace ver1 {  
    class MyClass {  
        ;  
    }  
  
#ifdef VER2  
    inline  
#endif  
namespace ver2 {  
    class MyClass {  
        ;  
    }  
}  
  
int main()  
{  
    nec::MyClass x;  
}
```

VER MAKROSU
TANIMLANDIYSA O
CLASS CAGRILIR

31inci karakter gelene kadar kapasite dolmicak

```
int main()
{
    string str{ "Omer Faruk Akyol" };

    std::cout << "str.size() = " << str.size() << "\n";
    std::cout << "str.capacity() = " << str.capacity() << "\n";
```

Microsoft Visual Studio Debug Console

```
str.size() = 16
str.capacity() = 31      ↵
```

```
C:\Users\necat\source\repos\cpp_january\Release\cpp_january
Press any key to close this window . . .
```

erase 50.karakterden sonrası siliyor. shrinktofit ise yeni kapasete göre büzüyor stringi

```
string str{ "mustafa ersoy" };

str.assign(300'000, 'a');

std::cout << "str.size() = " << str.size() << "\n";
std::cout << "str.capacity() = " << str.capacity() << "\n";

str.erase(50);
std::cout << "str.size() = " << str.size() << "\n";
std::cout << "str.capacity() = " << str.capacity() << "\n";

str.shrink_to_fit();

std::cout << "str.size() = " << str.size() << "\n";
std::cout << "str.capacity() = " << str.capacity() << "\n";
```

cinar

```
string str{ "mustafa ersoy" };

char ar[] = "cinar gursoy";

std::cout << "str = " << str << "\n";

str.assign(ar, 5);
std::cout << "str = " << str << "\n";

}
```

erk

```
int main()
{
    char str[] = "berkin kara";

    string word;

    word.assign(str + 1, str + 4);

    cout << word << "\n";
```

(local variable) char str[12]
Search Online

```
int main()
{
    char str[] = "berkin kara";
    string word;
    word.assign(str + 1, 4);
    cout << word << "\n";
```

erki

```
int main()
{
    char str[] = "berkin kara";

    string word;

    word.assign(str + 1, 4);

    cout << word << "\n";
```

farukisiker

```
int main()
{
    string name{ "farukisiker" };

    string s;

    s.assign(name, 3);           I

    std::cout << "s = " << s << "\n";

}

}
```

isik

```
int main()
{
    string name{ "farukisiker" };

    string s;

    s.assign(name, 5, 4);I

    std::cout << "s = " << s << "\n";

}
```

```
str.xxx(const char *p); //cstring param sonunda null yoksa ub
```

```
str.xxx(const char *p, string::size_type); //data taşma oluyorsa ub
```

```
str.xxx(const char *ps, const char *pe); range
```

```
str.xxx(const string &);
```

```
str.xxx(const string &, string::size_type idx); substr
```

```
str.xxx(const string &, string::size_type idx, string::size_type n);
```

```
str.xxx(string &&);
```

```
str.xxx(string::size_type, char); //fill param size_type kdr char karakterinden
```

```
str.xxx(initializer_list<char>); //fill param
```

substring

initializer-list oldugundan

```
int main()
{
    string s1(52, 'A');
    string s2{ 52, 'A' };

    std::cout << "s1.size() = " << s1.size() << "\n";
    std::cout << "s2.size() = " << s2.size() << "\n";
}
```

```
Microsoft Visual Studio Debug Console
s1.size() = 52
s2.size() = 2
```

```
C:\Users\necat\source\repo
Press any key to close this window.
```

```
^
7  using namespace std;
8
9
10 int main()
11 {
12     //if with initializer      C++17
13
14     string name{ "sezgin ceylan" };
15
16     if (auto idx = name.find('c'); idx != string::npos) {
17         std::cout << "bulundu\n";
18     }
19
20 }
```

alabilecek size't'nin max degeri

```
5  [#include <initializer_list>
6
7  using namespace std;
8
9
10 int main()
11 {
12     string str{ "mustafa erdem" };
13
14     string s(str, 3);
15     string s(str, 3, string::npos);
16
17 }
```

null alır parametre vermezsen, resize kapasiteyi arttırıyor

```
int main()
{
    string str("murat");

    std::cout << "str.size() = " << str.size() << "\n";

    //str.resize(20); I
    str.resize(20, '!');

    ps(str);
}
```

Microsoft Visual Studio Debug Console
str.size() = 5
[murat!!!!!!!!!!!!!!]

kapasiteyi artıtabilen operatorler

```
=  
push_back  
insert  
assign  
+=  
resize
```

I

3.karakterden sonra A eklenir insert

```
// INSERT (ITERATOR INTERFACE)

int main()
{
    string s("012345689");

    ps(s);

    s.insert(s.begin() + 3, 'A');
    ps(s);
}
```

II

5.indexten 10a kadar a ekler

```
// insert (iterator interface)

int main()
{
    string s("012345689");

    ps(s);

    // belirli bir indekse
    //
    s.insert(5, 10, 'a');
```

2yi siler

```
// erase (iterator interface)
//  
  
int main()
{
    string str{ "0123456789" };

    ps(str);
    str.erase(str.begin() + 2);
    ps(str);
```

09 hariç siler

```
// erase (iterator interface)
//
int main()
{
    string str{ "0123456789" };

    ps(str);
    str.erase(str.begin() + 1 ,str.end() - 1);
    ps(str);
```

bu ne biliyon mu string reverse

```
string name{ "cinar gursoy" };
reverse(name.begin(), name.end())
```

▲ 1 of 2 ▼ `constexpr void reverse<_BidIt>(const _BidIt _First, const _BidIt _Last)`

erase_if

```
string str{ "buraya bakin burada bu kara mermerin altinda..." };

//str.erase(remove_if(str.begin(), str.end(), [](char c) {return string{ "aeoui" }.find(c) != string::npos; }))
erase_if(str, [](char c) {return string{ "aeoui" }.find(c) != string::npos; })
ps(str);
```

find

I

rfind son geçtiği yerin indexi

find_first_not_of bunlardan birinin ilk geçtiği yer

find_first_of bunlardan birinin ilk olmadığı yer

find_last_of bu karakterlerinden son geçtiği yer

find_last_not_of bunlardan birinin son olmadığı yer

//C++23

can'i string ypr

The screenshot shows a code editor window with the following C++ code:

```
//uDL
int main()
{
    using namespace std;
    "ahmet" + "can"s
    //operator""s("ahmet", 5)
}
```

A yellow circle highlights the character 's' in the string "can"s. A cursor icon is positioned to the right of the highlighted character.

string_view 2 pointer arasında tutar veriyi böylece
copy yapmayz, okuma amacli

```
string str(10000, 'A');
```

```
string_view sw{ str };
```

! görünce devamini almiyor

```
cout << "bir yazı giriniz: ";
getline(cin, str, '!');
ps(str);
```

```
using namespace std;  
  
int ival{ 3534612 };  
string name{ "zeynep" };  
  
name + to_string(ival)
```

idx 5 tir

```
using namespace std;  
  
string str{ "92835murathan" };  
           ^  
  
size_t idx;  
  
int x = stoi(str, &idx, 16);  
  
std::cout << "idx = " << idx << "\n";
```

küçükten büyüğe

```
string str("yarin ramazan ayinin ilk gunu");
//...
pd(str);
transform(str.begin(), str.end(), str.begin(), &std::toupper);
pd(str);
```

İçeriyor mu

```
int main()

using namespace std;

string str{ "alican345.jpeg" };

if (str.contains("j|")
```

T

defaultu public , class olsaydi defaultu private

```
class Base {  
    ...  
};
```

```
struct Der : Base {};  
struct Der : public Base {};
```

I

BASE'IN KULLANDIgÜİ
HERYERDE DER DE
KULLANILABILRMELİ ,ILKESI
SOLID

```
3  
4  
5  
6 class Base {  
7     |  
8     public:  
9         |    void foo();  
10        |    void bar();  
11        |    void baz();  
12    };  
13    |  
14  
15 class Der : public Base {  
16     |  
17     |  
18     |};
```

```
class Base {  
public:  
};  
  
class Der : public Base {  
};  
  
int main(void)  
{  
    Der myder;  
    Base* baseptr = &myder; //upcasting  
    Base& baseref = myder; //upcasting  
  
    Base x = myder; //!!! (object slicing) (nesne dilimlenmesi)  
}
```

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

erisim kontrolu hatası

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

```
int main(void)  
{  
    Der myder;  
  
    myder.foo(12);  
}
```

```
class Base {
public:
    void foo();
    void bar();
    void baz();
    Base(int);
    Base(int, int);
    Base(double);
};

class Der : public Base {
public:
    using Base::Base;
};

int main()
{
    Der myder(12);           I
    Der myder(1, 2);
```

C++ dilinde, kalıtımında taban sınıfların üye fonksiyonlarını 3 kategoriye ayıriyoruz

1. Türemiş sınıflara hem bir interface veren hem de implementasyon veren fonksiyon(lar)
2. Türemiş sınıflara hem bir interface veren hem de bir default implementasyon veren fonksiyon
override (function overriding)
polymorphic class
3. Türemiş sınıflara bir interface veren ancak bir implementasyon vermeyen fonksiyonlar
abstract class (soyut sınıf)
Soyut sınıflar türünden nesne oluşturulamaz (pointer ya da referans oluşturulabilir)

abstract class (mucerret)

concrete class (müşahhas)

override

```
class Base {  
public:  
    virtual int foo(int, int) const;  
};
```

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

```
class Audi : public Car {  
public:  
    Car* clone()override  
    {  
        return new Audi(*this);  
    }  
  
    void start()override  
    {  
        std::cout << "AUDI has just started\n";  
    }  
  
    void run()override  
    {  
        std::cout << "AUDI is running now\n";  
    }  
}
```

polimorfik Taban sınıfların destructor'ı
ya public virtual olmalı
ya da protected non-virtual olmalı

```
class Base {  
//...  
};
```

```
class Der final : public Base {  
};
```

```
class Nec : public Der {  
};
```

I

/// nec foo fonksyionu override edemez

```
class Base {  
public:  
    virtual void foo();  
};  
  
class Der : public Base {  
public:  
    void foo() override final;  
};  
  
class Nec : public Der {  
public:  
    void foo()override:
```

```
#include <iostream>

///

class Base {
public:
    virtual void foo();
};

class Der : public Base {
public:
    void foo() override final; // Yellow circle here
};

class Nec : public Der {
public:
    void foo()override;
};
```

nec foo fonksiyonu override edemez

15:38 ✓/

```
#include <iostream>
#include <initializer_list>
#include <vector>

int main()
{
    const int a[] = { 2, 4, 5, 7, 9 };
    std::vector<int> vec(a, a + sizeof(a) / sizeof(*a));

    std::cout << "vec.size() = " << vec.size() << "\n";
}
```

17:38 ✓/

```
5
6     std::cout << *carptr << "\n";
7     carptr->start();
8     carptr->run();
9
10
11    auto vp = dynamic_cast<Volvo*>(carptr);
12    if (vp) {
13        vp->open_sunroof();
14        (void)getchar();
15    }
16
17
18 int main()
19 {
20     for (int i = 0; i < 1000; ++i) {
21         Car* cp = create_random_car();
22         car_game(cp);
23         delete cp;
24     }
25 }
```

```
07  
08  
09 typeid op. ile oluşturulmuş bir ifade I  
10 aslında  
11 std::type_info sınıfı türünden bir nesneye cons|
```

```
1 #include "car.h"
2
3
4 void car_game(Car* carptr)
5 {
6     std::cout << *carptr << "\n";
7     carptr->start();
8     carptr->run();
9
10
11     if (typeid(*carptr) == typeid(Volvo)) {
12         Volvo* pv = static_cast<Volvo*>(carptr);
13     }
14 }
15
16
17 int main()
18 {
19     for (int i = 0; i < 1000; ++i) {
20         Car* cp = create_random_car();
21         car_game(cp);
22         delete cp;
23     }
24 }
```

SFINAE

Substitution failure is not an error
enable_if

CRTTP

curiously recurring template pattern

lambda

positive lambda idiom

protected kalıtımı ile private kalıtımı arasında ciddiye alınabilecek tek fark

protected kalıtımından taban sınıfın public bölümü

türemiş sınıfın private bölümüne değil

protected bölümüne eklenir.

Bu durumda multi-level inheri. ile yeni türetilen sınıflar bu öğelere erişebilir.

[[[]]]

```
#include <iostream>
#include <vector>
```

```
int foo();
void bar();
void baz();
void func();
```

```
int main()
{
    int x = foo();

    switch (x) {
        case 1: bar(); [[fallthrough]];
        case 2: baz(); break;
        case 3: func(); break;
    }
}
```

breaksiz yer alt satırı geçiyor derleyici
hata vermiyor ben breaksiz kullanmayı
tercih ediyorum demek istiyoruz

```
10     void speak()
11     {
12         std::cout << "Person::speak() this      : " << this << "\n";
13     }
14 };
15
16
17 class Worker : virtual public Person {
18
19 };
20
21
22 class Teacher : virtual public Person {
23
24 };
25
26 class MathTeacher : public Worker, public Teacher {
27
28 public:
29     void solve_equation()
30     {
31     }
32 };
33
34
35 int main()
36 {
37     MathTeacher mt;
38
39     //Person* p = &mt;           I
40     Person* p1 = static_cast<Teacher*>(&mt);
41     Person* p2 = static_cast<Worker*>(&mt);
42
43     std::cout << "p1 = " << p1 << "\n";
44 }
```

VIRTUAL INHERITANCE CLASS YAPISI AMBIGIOTUNS SORUNU KALKAR

LEGAL

```
2
3
4 class BaseX {
5     public:
6         void foo();
7 }
8
9
10 class BaseY {
11     public:
12         int foo(int, int);
13 }
14
15
16 class Der : public BaseX, public BaseY {
17     public:
18         using BaseX::foo;
19         using BaseY::foo;
20 }
21
22
23 int main()
24 {
25     Der myder;           I
26
27     myder.foo();
28     myder.foo(12, 45);|
29 }
```

catch 3u yazyor long exxeption eder

```
64
65 int main()
66 {
67
68     try {
69         f1();
70     }
71     catch (int x) {
72         std::cout << "exception caught in function main.. catch(int x) x = " << x << "\n";
73     }
74     catch (double) {
75         std::cout << "exception caught in function main.. catch(double)\n";
76     }
77     catch (long) {
78         std::cout << "exception caught in function main.. catch(long)\n";
79     }
80     catch (...) {
81         std::cout << "exception caught in function main.. catch(...)\n";
82     }
83
84
85
86     std::cout << "main devam ediyor\n";
```



void foo(int)noexcept;

bu fonk. exception throw etmiyor garantisi verir

ayni tanimlama

```
void func()noexcept(false);  
void func();
```

I

```
void bar()noexcept;  
void bar()noexcept(true);
```

▷ DBL_TRUE_MIN
▷ FLT_TRUE_MIN

ayni tanimlama

Strong exception safety

Basic idea of writing exception safe code is as follows:

Separate the function into two parts:

- Prepare: can throw, but does not modify the object.
- Commit: doesn't throw, and modifies the object.

A very inefficient, but illustrative way of making previous code strongly exception safe:

```
void insert(int key, string value)
{
    auto newKeys = keys;
    auto newValues = values;
    newKeys.push_back(key);
    newValues.push_back(value);

    std::swap(newKeys, keys);
    std::swap(newValues, values);
}
```



kod tekrari

Consistent exception handling

How can we refactor the error handling code to reduce duplication?

```
void doSomeWork()
{
    try
    {
        someWork();
    }
    catch (const NetworkError& exc)
    {
        loadFromFile();
    }
    catch (const InvalidData& exc)
    {
        dropConnection();
    }
    catch (const TooMuchData& exc)
    {
        reSendSmallerRequest();
    }
}
```

```
void doSomeOtherWork()
{
    try
    {
        otherWork();
        otherStuff();
    }
    catch (const NetworkError& exc)
    {
        loadFromFile();
    }
    catch (const InvalidData& exc)
    {
        dropConnection();
    }
    catch (const TooMuchData& exc)
    {
        reSendSmallerRequest();
    }
}
```

```
void doNothing()
{
    try
    {
        sleep();
        wait();
        sleep();
    }
    catch (const NetworkError& exc)
    {
        loadFromFile();
    }
    catch (const InvalidData& exc)
    {
        dropConnection();
    }
    catch (const TooMuchData& exc)
    {
        reSendSmallerRequest();
    }
}
```



çözüm

Necati Ergin

Consistent exception handling

We can use Lippincott Functions (aka. exception dispatcher).

```
void doSomeWork()
{
    try
    {
        someWork();
    }
    catch (...)
    {
        handleExceptions();
    }
}

void doSomeOtherWork()
{
    try
    {
        otherWork();
        otherStuff();
    }
    catch (...)
    {
        handleExceptions();
    }
}
```

Catch any exception, and let exception handling function take care of it.

Current exception is a thread-local global, so it can be accessed inside handleExceptions().

```
void handleExceptions()
{
    try
    {
        throw;
    }
    catch (const NetworkError& exc)
    {
        loadFromFile();
    }
    catch (const InvalidData& exc)
    {
        dropConnection();
    }
    catch (const TooMuchData& exc)
    {
        reSendSmallerRequest();
    }
}
```

generic programming & templates

function template

class template

alias template (C++11)

variable template

concepts

T degiskeninin tipini hata raporundan öğrenme syntaxı

derleyicinin gerçek kodu yazmak içi
template parametrelerine karşılık gelecek argümanların ne olduğunu bilmesi

1. explicit template argument
2. deduction

II

template argument deduction

Dikkat! C++17 standardına kadar söz konusuydu.

CTAD

```
template <typename>
class TypeTeller;
```

```
template <typename T>
void func(T)
{
    TypeTeller<T> x;
}
```

initilazerlist

```
int main()
```

```
{
    func(12);
}
```

I

1. explicit template argument

2. deduction

template argument deduction

Dikkat! C++17 standardına kadar söz konusuydu.

CTAD

3. default template argument

Dikkat!!!

Bir istisna dışında

auto type deduction ile

template argument deduction

tamamen aynıdır.

```
template <typename T, std::size_t n>
constexpr std::size_t asize(T(&)[n])
{
    return n;
}

int main()
{
    int a[] = { 3, 5, 7, 9, 1, 3, 3, 4 };

    constexpr auto val = asize(a);
}
```



I

```
auto func(int x)->int
{
    /**
     * return 1;
}
```

int func(int x) ile ayn ey

```
template <typename T, typename U> <T> Provide sample template arguments for IntelliSense ✓
auto sum(T x, U y)-> decltype(x + y)
{
}
```

trailing return type

sadece double ile çarlabılır

```
template <typename T>
void func(T) = delete;
```

```
void func(double);
```

```
int main()
{
    func(12.3);
}
```

```
template <typename T>
void func(T)
{
    std::cout << "primary template for type " << typeid(T).name() << "\n";
}

template < >
void func(int)
{
    std::cout << "explicit specialization for func<int>\n";
}

int main()
{
    func(1.2);
    func(1.2f);
    func(1u);
    func(1);
}
```

```
template <typename T, typename U>
class Nec {
public:
    Nec()
    {
        std::cout << "primary template\n";
    }
};

template <>
class Nec<int, char> {
public:
    Nec()
    {
        std::cout << "Nec<int, char> explicit spec.";
    }
};
```

```
//compile-time da factorial hesaplanması

template <int n>
struct Factorial {
    static const int value = n * Factorial<n - 1>::value;
};

template <>
struct Factorial<1> {
    static const int value = 1;
};

template <>
struct Factorial<0> {
    static const int value = 1;
};

int main()
{
    Factorial<12>::value
```

k

I

```
template <typename T>
using Arten = T[10];
```

```
int main()
{
    Arten<int> x;
    Arten<double> y;
```

m2 varsayılan olarak int alır

```
template <typename T = int>
class Myclass {
};

int main()
{
    Myclass<double> m1;
    Myclass<> m2;
```

```
template <typename T>
struct IsLValueReference : std::false_type {
};

template <typename T>
struct IsLValueReference<T&> : std::true_type {
};

template <typename T>
class Refclass {
    static_assert(IsLValueReference<T>::value, "only for LVal & types");
};

int main()
{
    Refclass<int> x;
```

```
template <typename T, typename U = std::enable_if_t<std::is_pointer_v<T>>>
void func(T x);

int main()
{
    int x{};
    func(&x);
}
```

I

```
utility.h      main.cpp  x  (Global Scope)
try
7   |     using type = T;
8   | };
9
10  template <typename T>
11  struct RemoveReference<T&> {
12    |     using type = T;
13  };
14
15  template <typename T>
16  struct RemoveReference<T&&> {
17    |     using type = T;
18  };
19
20
21  template <typename T>
22  using RemoveReference_t = typename RemoveReference<T>::type;
23
24  template <typename T>
25  RemoveReference_t<T>func(T);
26
```

utility.h main.cpp

(Global Scope)

```
1 #include <iostream>
2 #include <type_traits>
3
4 //template parameter pack
5 // pack expansion
6
7 template <typename ...Ts>
8 void func(Ts ...args)
9 {
10
11 }
12
13 template <typename T, typename U, typename W>
14 void func(T t, U u, W w);
15
16
17 int main()
18 {
19     func(12, 34L, 4.5);
20 }
21 }
```

```
int foo(int, int, double)
{
    std::cout << __FUNCSIG__ << "\n";
    return 4;
}
```

```
int main()
{
    foo(1, 3, 3.4);
}
```

Microsoft Visual Studio Debug Console

```
int __cdecl foo(int,int,double)

C:\Users\necat\source\repos\cpp_
Press any key to close this window
```

```
// unary left fold
template <typename ...TS>
auto sum(const TS& ...args)
{
    (... + args)
```

```
// unary left fold
template <typename ...TS> <T> Provide
auto sum(const TS& ...args)
{
    return (... + args); // (((p1 + p2) + p3) + p4) + p5
```

```
// unary right fold
```

```
template <typename ...TS> <T> Provide sample te
└ auto sum(const TS& ...args)
{
    return (args + ...);
    //return p1 + (p2 + (p3 + (p4 + p5)));
}
```

```
template<typename... Ts>
└ int subtract(int num, Ts... args)
{
    return (num - ... - args); //Binary left fold
}

int main()
{
    int result = subtract(100, 50, 20, 7); //result' is (((100 - 50) - 20) - 7) = 23
    std::cout << "result = " << result << "\n";
}
```

```
template<int N>
[-]constexpr int fibo()
{
    if constexpr (N >= 2)
        return fibo<N - 1>() + fibo<N - 2>();
    else
        return N;
}

[-]int main()
{
    constexpr auto y = fibo<8>();
}
```

Standard Template Library

containers (sınıf şablonları) = kap
LIST-DOUBLE, FORWARDLIST-SINGLELINKED,
SETanahtar arar,

sequence containers

C arrays

std::vector;

std::deque

std::list

std::forward_list C++11

std::string

std::array

multisette anahtarsayıslı Artabilir,
map-id verip telno almak,

C++11

sequence containers

C arrays

std::vector;

std::deque

std::list

std::forward_list C++11

std::string

std::array C++11

associative containers

std::set

std::multiset

std::map

std::multimap

unordered associative containers

std::unordered_set

std::unordered_multiset

std::unordered_map

std::unordered_multimap

container

iterator

std::ranges

STL 2.0

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

    int a[] = { 11, 41, 95, 16, 7, 9 ,12 };

    sort(begin(a), end(a));

    for (auto i : a)
        cout << i << "\n";
}
```

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
```

```
vector  
list  
deque  
map  
multimap  
set  
multiset
```

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
```

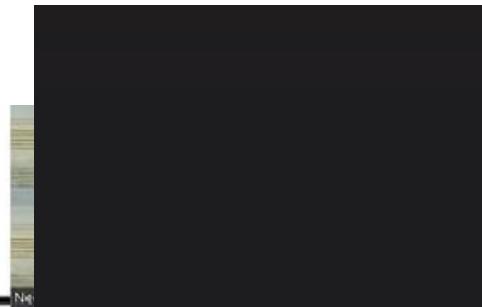
```
string  
char*
```

Expression	Effect
<code>*iter</code>	Provides read access to the actual element
<code>iter ->member</code>	Provides read access to a member of the actual element
<code>++iter</code>	Steps forward (returns new position)
<code>iter++</code>	Steps forward
<code>iter1 == iter2</code>	Returns whether two iterators are equal
<code>iter1 != iter2</code>	Returns whether two iterators are not equal
<code>TYPE(iter)</code>	Copies iterator (copy constructor)

Table 9.3. Operations of Input Iterators

Expression	Effect
$*iter$	Provides access to the actual element
$iter->member$	Provides access to a member of the actual element
$++iter$	Steps forward (returns new position)
$iter++$	Steps forward (returns old position)
$iter1 == iter2$	Returns whether two iterators are equal
$iter1 != iter2$	Returns whether two iterators are not equal
$TYPE()$	Creates iterator (default constructor)
$TYPE(iter)$	Copies iterator (copy constructor)
$iter1 = iter2$	Assigns an iterator

Table 9.4. Operations of Forward Iterators



Expression	Effect
<code>--iter</code>	Steps backward (returns new position)
<code>iter--</code>	Steps backward (returns old position)

Table 9.5. Additional Operations of Bidirectional Iterators

```
template <typename Iter>
void algo(Iter beg, Iter end)
{
    if constexpr (std::is_same_v<typename Iter::iterator_category, std::random_access_iterator_tag>) {
        beg += 5;
        std::cout << "random access iterator\n";
    }
    else if constexpr (std::is_same_v<typename Iter::iterator_category, std::bidirectional_iterator_tag>) {
        for (int i = 0; i < 5; ++i) {
            ++beg;
        }
        std::cout << "bidirectional iterator\n";
    }
}

int main()
{
    using namespace std;

    list<int> ilist(10);

    algo(ilist.begin(), ilist.end());
}
```



```
#include <iterator>
#include <type_traits>
```

```
template <typename InIter, typename OutIter> <T> Provide sample template arg
OutIter Copy(InIter beg, InIter end, OutIter destbeg)
{
    while (beg != end) {
        *destbeg = *beg;
        ++beg;
        ++destbeg;
    }
    return destbeg;
}
```



```
#include <iterator>
#include <type_traits>
```

```
template <typename InIter, typename OutIter> <T> Provide sample template arg
OutIter Copy(InIter beg, InIter end, OutIter destbeg)
{
    while (beg != end) {
        *destbeg++ = *beg++;
    }

    return destbeg;
}
```

I



```
template <typename InIter, typename OutIter>
OutIter Copy(InIter beg, InIter end, OutIter destbeg)
{
    while (beg != end) {
        *destbeg++ = *beg++;
    }

    return destbeg;
}

int main()
{
    using namespace std;

    vector<int> ivec{ 1, 3, 5, 7, 9, 13 };
    list<int> ilist(6);

    Copy(ivec.begin(), ivec.end(), ilist.begin());
}
```

```
std::advance  
std::distance  
std::next  
std::prev  
std::iter_swap
```

advance ref verir, next yeni

tag dispatch

advance



```
next(iter, 5)
```

```
#include <iostream>
#include <vector>
#include <list>
#include <algorithm>

template <typename Iter>
void display_range(Iter beg, Iter end)
{
    while (beg != end) {
        std::cout << *beg++ << " ";
    }
    std::cout << "\n";
}

int main()
{
    using namespace std;

    vectorivec{ 3, 6, 9, 12, 15, 18 };

    auto riter_beg = ivec.rbegin();
    auto riter_end = ivec.rend();
    display_range(riter_beg, riter_end);
    display_range(riter_end.base(), riter_beg.base()); //
```



sondaki 2yi silio

```
#include <iostream>
#include <vector>
#include <list>
#include <algorithm>
#include "nutility.h"

int main()
{
    using namespace std;

    vector<int> ivec{ 1, 2, 3, 2, 4, 2, 7, 8, 2, 6, 2, 9 };
    int val = 2;

    if (auto riter = find(ivec.rbegin(), ivec.rend(), val); riter != ivec.rend()) {
        ivec.erase(prev(riter.base()));
    }
    else {
        std::cout << "bulunamadi\n";
    }

    print(ivec, cout);
}
```



```
| #include "nutility.h"
```

girilen isimden kaç tane olduğunu döndürüyor

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

    list<string> slist;
    rfill(slist, 20000, rname);
    //print(slist, cout);

    string name;

    std::cout << "sayılacak isim: ";
    cin >> name;

    cout << count(slist.begin(), slist.end(), name);
}
```

```
#include <vector>
#include <list>
#include <algorithm>
#include "nutility.h"
```

copyif

```
template <typename InIter, typename OutIter, typename UnPred> <T> Prov
OutIter Copy_if(InIter beg, InIter end, OutIter destbeg, UnPred f)
{
    while (beg != end) {
        if (f(*beg)) {
            *destbeg++ = *beg;
        }
        ++beg;
    }

    return destbeg;
}
```



```
template <typename InIter, typename UnPred>
InIter Find_if(InIter beg, InIter end, UnPred f)
```

```
#include <algorithm>
#include "nutility.h" if

template <typename InIter, typename UnPred> <T> Provide sample template arguments for Inte
InIter Find_if(InIter beg, InIter end, UnPred f)
{
    while (beg != end) {
        if (f(*beg)) {
            return beg;
            ++beg;
        }
    }

    return end;
}
```

```
template <typename InIter, typename UnPred>
int Count_if(InIter beg, InIter end, UnPred f)
{
    int cnt{};

    while (beg != end) {
        if (f(*beg)) {
            ++cnt;
        }
    }
}
```



Necati Ergin

```
uary          (Global Scope)          main()
```

```
4 #include <algorithm>
5 #include "nutility.h"
6
7
8 using namespace std;
9
10 int iseven(int x)
11 {
12     return x % 2 == 0;
13 }
14
15 int main()
16 {
17     vector<int> ivec;
18     rfill(ivec, 100'000, Rand{ 0, 10'000 });
19
20     cout << count_if(ivec.begin(), ivec.end(), iseven) << "\n";
21 }
```

```
7
8     using namespace std;
9
10
11 class DivPred {
12 public:
13     DivPred(int val) : mval{val} {}
14     bool operator()(int x) const
15     {
16         return x % mval == 0;
17     }
18 private:
19     int mval;
20 };
21
22
23 int main()
24 {
25     vector<int> ivec;
26     rfill(ivec, 100'000, Rand{ 0, 10'000 });
27
28     int x;
29     std::cout << "kac tam boluneleri saymak istiyorsunuz: ";
30     cin >> x;
31
32     cout << count_if(ivec.begin(), ivec.end(), DivPred{x}) << "\n";

```

```
1
2 class LenPred {
3     public:
4         LenPred(std::size_t len) : mlen{len} {}
5         bool operator()(const std::string &s) const
6         {
7             return s.length() == mlen;
8         }
9     private:
10        std::size_t mlen;
11    };
12
13
14 int main()
15 {
16     vector<string> svec;
17     rfill(svec, 100'000, rname);
18
19     size_t len;
20     std::cout << "uzunlugu kac olanlari saymak istiyorsunuz: ";
21     cin >> len;
22
23     cout << count_if(svec.begin(), svec.end(), LenPred{len}) << "\n";
24 }
```

```
#include <iostream>
#include <vector>
#include <list>
#include <algorithm>
#include "nutility.h"
#include <string>
```

lambda ifadesi

```
using namespace std;
```

```
int main()
{
    vector<string> svec;
    rfill(svec, 100'000, rname);

    for (size_t i{ 3 }; i < 14; ++i) {
        cout << i << " " << count_if(svec.begin(), svec.end(),
            [i](const string& s) {return s.length() == i; }) << "\n";
    }
}
```



```
1 #include <iostream>
2 #include <vector>
3 #include <list>
4 #include <algorithm>
5 #include "nutility.h"
6 #include <string>
7
8
9 using namespace std;
10
11
12
13 int main()
14 {
15     [](int x) {return x * 5; }
16 }
```

bu ifade bir class pr value . (10)
yazarsak sonuna 50 sonucunu
verir

I

7
8
9 using namespace std;

10

11

12 class ztr_yuq {

13 public:

14 auto operator()(int x) const

15 {

16 return x * 5;

17 }

18 };

19

20 int main()

21 {

22 //[](int x) {return x * 5; }

23 ztr_yuq{}(10)

24 }

25 }

lambda ifadesinin yazdır class

```
utility.h      out.txt      nota.txt
              ↴      (Global Scope)
7
8
9     using namespace std;
10
11
12    int main()
13    {
14        vector<string> svec;
15        rfill(svec, 20, [] {return rname() + ' ' + rfname(); });
16        print(svec, cout, "\n");
17
18
19        if (auto iter = find_if(svec.begin(), svec.end(), [] (const string& s) {
20            return s.find("kan") != string::npos; }); iter != svec.end()) {
21            cout << "bulundu " << *iter << "\n";
22        }
23    }
24
25 }
```

```
using namespace std;

int main()
{
    vector<string> svec;
    rfill(svec, 200'000, [] {return rname() + ' ' + rfname(); });
    //print(svec, cout, "\n");

    string key;
    std::cout << "aranan yaziyi girin : ";
    cin >> key;

    if (auto iter = find_if(svec.begin(), svec.end(), [key](const string& s) {
        return s.find(key) != string::npos; }); iter != svec.end()) {
        cout << "bulundu " << *iter << "\n";
    }
    else {
        std::cout << "bulunamadi\n";
    }
}
```



```
template <typename InIter, typename OutIter, typename Ufunc>
OutIter Transform(InIter beg, InIter end, OutIter destbeg, Ufunc f)
{
    while (beg != end) {
        *destbeg++ = f(*beg++);
    }

    return destbeg;
}

int main()
{
    using namespace std;

    vector<size_t> lenvec;

    Transform(svec.begin(), svec.end(), back_inserter(lenvec), [] (const string& s) {return s.length(); });
    print(lenvec, cout);
}
```

std::transform C++ standard kütüphanesinde bulunan bir algoritmadır. Bu algoritma, bir giriş aralığındaki elemanları başka bir aralığa dönüştürmek için kullanılır.

```
5
6
7 std::string Reverse(std::string s)
8 {
9     std::reverse(s.begin(), s.end());
10    return s;
11 }
12
13 int main()
14 {
15     using namespace std;
16
17     vector<string> svec;
18     rfill(svec, 10, rname);
19     print(svec, cout);
20     transform(svec.begin(), svec.end(), svec.begin(), Reverse);
21     print(svec, cout);
22
23 }
```

I

```
9     int sum_square(int a, int b)
10    {
11        return a * a + b * b;
12    }
13
14 int main()
15 {
16     using namespace std;
17
18     int a[] = { 1, 3, 5, 7, 12, 67 ,20 ,56, 90 };
19     vector<int> ivec{ 11, 23, 85, 17, 112, 637 ,240 ,546, 190 };
20
21     list<int> mylist;
22
23     transform(begin(a), end(a), ivec.begin(), back_inserter(mylist), sum_square);
24     print(mylist, cout);
25 }
```

(Global Scope)

```
6    []
7
8
9 int main()
10 {
11     using namespace std;
12
13     int a[] = { 1, 3, 5, 7, 12, 67 ,20 ,56, 90 };
14     vector<int> ivec{ 11, 23, 85, 17, 112, 637 ,240 ,546, 190 };
15
16     list<int> mylist;
17
18     transform(begin(a), end(a), ivec.begin(), back_inserter(mylist),
19               [] (int a, int b) {return a * a + b * b; });
20     print(mylist, cout);
21
22 }
```

(Global Scope)

```
1 #include <algorithm>
2 #include <vector>
3 #include <string>
4 #include "nutility.h"
5 #include <list>
6
7
8 template <typename Iter, typename Ufunc> <T> Provide sample template arguments for IntelliSense ▾
9 Ufunc for_each(Iter beg, Iter end, Ufunc f)
10 {
11     while (beg != end) {
12         f(*beg++);
13     }
14
15     return f;
16 }
17
18
19 template <typename InIter1, typename InIter2, typename OutIter, typename BiFunc>
20 OutIter Transform(InIter1 beg1, InIter1 end, InIter2 beg2, OutIter destbeg, BiFunc f)
21 {
```

```
#include "matrix.h"
#include <list>

using namespace std;

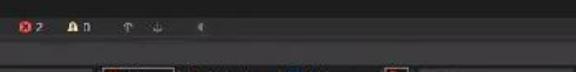
void increment(int& x)
{
    ++x;
}

int main()
{
    int a[] = { 1, 4, 7, 2, 3, 9, 8, 12, 54, 22, 33 };
    for_each(begin(a), end(a), increment);

    for (auto i : a) {
        std::cout << i << " ";
    }
}
```

```
2 #include <vector>
3 #include "nutility.h"
4 #include <string>
5
6
7 int main()
8 {
9     using namespace std;
10
11     std::vector<std::string> svec;
12
13     rfill(svec, 20, rname);
14     print(svec, std::cout);
15
16 //pair<vector<string>::iterator, vector<string>::iterator> ip = minmax_element(svec.begin(), svec.end());
17 auto ip = minmax_element(svec.begin(), svec.end());
18 cout << *ip.first << " " << *ip->second |
19
20
21 }
```

Necati Er



structor binding

```
2 #include <vector>
3 #include "nutility.h"
4 #include <string>
5
6
7 int main()
8 {
9     using namespace std;
10
11     std::vector<std::string> svec;
12
13     rfill(svec, 20, rname);
14     print(svec, std::cout);
15
16     //pair<vector<string>::iterator, vector<string>::iterator> ip = minmax_element(svec.begin(), svec.end());
17     auto [iter_min, iter_max] = minmax_element(svec.begin(), svec.end());I
18     //auto iter_min = ip.first;
19     //auto iter_max = ip.second;
20
21 }
22
```

No issues found 4

```
2 #include <functional>
3 #include <vector>
4 #include <string>
5 #include "nutility.h"
6 #include <algorithm>
7
8
9
10 int main()
11 {
12     using namespace std;
13
14     vector<string> svec;
15     rfill(svec, 20, [] {return rname() + ' ' + rfname(); });
16     sort(svec.begin(), svec.end(), greater<string>{});
17     print(svec, cout, "\n");
18
19 }
20
21
22
23
24
```

functional

```
2 #include <functional>
3 #include <vector>
4 #include <string>
5 #include "nutility.h"
6 #include <algorithm>
7
8
9
10 int main()
11 {
12     using namespace std;
13
14     vector<string> svec;
15     rfill(svec, 20, [] {return rname() + ' ' + rfname(); });
16     sort(svec.begin(), svec.end(), greater<string>{});
17     print(svec, cout, "\n");
18
19 }
20
21
22
23
24
```

```
#include <functional>
#include <vector>
#include <string>
#include "nutility.h"
#include <algorithm>
#include <list>

int main()
{
    using namespace std;

    vector<int> x{ 2, 4, 5, 6, 7, 9, -10 , -7, 1, 90 };
    vector<int> y{ 1, 2, 7, 16, 17, 19, 110 , -97, 31, 490 };

    list<int> mylist;
    //

    transform(x.begin(), x.end(), y.begin(), back_inserter(mylist), plus{});

    print(mylist, cout);
}
```

`std::back_inserter` fonksiyonu, konteynlere eleman eklemeyi kolaylaştırır.
Bu, mevcut bir konteyninin sonuna elemanlar eklemek için dinamik
olarak büyütmen bir konteynir kullanmanız gereken durumlarda özellikle kullanışlıdır.

Microsoft Visual Studio Debug Console

3 6 12 22 24 28 100 -104 32 580

C:\Users\necat\source\repos\cpp_january\Debug\cpp_january.exe (process 12064) exited with
Press any key to close this window . . .

Necati Ergin

```
2 #include <functional>
3 #include <vector>
4 #include <string>
5 #include "nutility.h"
6 #include <algorithm>
7 #include <list>
8
9
10
11
12 int main()
13 {
14     using namespace std;
15
16     vector<int> ivec;
17     rfill(ivec, 20, Rand{ 1, 9 });
18     print(ivec, cout, "");
19     for (auto it = ivec.begin(); it != ivec.end(); ++it) {
20         if (*it % 2 == 0)
21             *it = 0;
22     }
23     print(ivec, cout, "");
24
25
26
27 }
```

```
4 #include <string>
5 #include "nutility.h"
6 #include <algorithm>
7 #include <list>          std::remove_copy işlevi, belirli bir değeri çıkarmak ve yeni bir aralık oluşturmak için kullanışlıdır.
8                                Bu işlev, bir aralığı değiştirmeden veya yeniden düzenlemeden belirli değerleri kaldırırmak için kullanılabilir.
9
10
11
12 int main()
13 {
14     using namespace std;
15
16     vector<int> ivec;
17     rfill(ivec, 100, Rand{ 0, 5 });
18     print(ivec, cout);
19     vector<int> dest;
20
21     remove_copy(ivec.begin(), ivec.end(), back_inserter(dest), 3);
22
23     print(dest, cout);
24 }
```

0 Errors | 0 Warnings | 0 Messages | Build Only

a olan isimleri vektörden çkarr

```
7 #include <list>
8 #include <fstream>
9
10
11
12
13 int main()
14 {
15     using namespace std;
16
17     vector<string> svec;
18     rfill(svec, 1000, [] {return rname() + ' ' + rfname(); });
19     vector<string> dest;
20
21     remove_copy_if(svec.begin(), svec.end(), back_inserter(dest),
22                     [] (const string& s) {return s.contains('a'); });
23
24     std::ofstream ofs{ "out.txt" };
25     if (!ofs) {
26         std::cerr << "out.txt dosyasi olusturulamadi\n";
27         exit(EXIT_FAILURE);
28     }
29
30     print(dest, ofs, "\n");
31
32
33
```

```
2 [ #include <iostream>
3
4
5
6     template <typename Iter>
7     struct IteratorTraits {
8         using value_type = typename Iter::value_type;
9         using pointer = typename Iter::pointer;
10        using reference = typename Iter::reference;
11    };
12
13
14     template <typename T>
15     struct IteratorTraits<T*> {
16         using value_type = T;
17         using value_type = T;
18    };
19
```

std::iterator_traits kullanmanız gereken durumlar genellikle iteratorların jenerik programlamada veya kendi iterator türlerini oluştururken ihtiyaç duyduğunuz iterator özelliklerine erişmek istediğiniz durumlardır.
Önceden tanımlanmış standart iterator türleriyle çalışırken, bu özelliklere genellikle doğrudan erişebilirsiniz.

gibi kullanılır:

```
cpp Copy code
template <typename Iterator>
struct iterator_traits {
    using iterator_category;
    using value_type;
    using difference_type;
    using pointer;
    using reference;
};
```

Bu şablon, bir iterator türü belirtilerek özelleştirilebilir. Örneğin, bir
`std::vector<int>::iterator` türünün özelliklerine erişmek için
`std::iterator_traits<std::vector<int>::iterator>` kullanılabilir.

Özellikle, `std::iterator_traits` kullanarak aşağıdaki bilgilere erişebilirsiniz:

- `iterator_category`: Iteratorün kategori türüdür (input iterator, output iterator, forward iterator, bidirectional iterator, random access iterator gibi).
- `value_type`: Iteratorün işaret ettiği elemanın türüdür.
- `difference_type`: Iteratorler arasındaki mesafe için kullanılan türdür.

```
template <typename Iter> <T> Provide sample template arguments for IntelliSense ▾ ✎  
void func(Iter beg, Iter end)  
{  
    typename std::iterator_traits<Iter>::iterator_category
```

```
using namespace std;  
list<int> mylist{ 1, 5, 7, 9, 2, 3, 8 };  
  
vector<int> ivec{ mylist.begin(), mylist.end() };
```

list'i vector'e çevirdik range vererek

```
int main()      31 baa gelir
{
    using namespace std;

    vector<Date> dvec;

    dvec.emplace_back(1, 1, 1987);
    dvec.emplace_back(5, 5, 1988);
    dvec.emplace_back(9, 9, 1989);
    dvec.emplace(dvec.begin(), 31, 12, 2023);

    print(dvec, cout, "\n");
}
```

I

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

    vector<string> tvec;
    rfill(tvec, 10, rtown);

    while (!tvec.empty()) {
        print(tvec, cout);
        _getch();
        tvec.pop_back();
    }
}
```



en üstteki ismi siliyor

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

    vector<string> name_vec;
    rfill(name_vec, 10, [] {return rname() + ' ' + rfname(); });

    while (!name_vec.empty()) {
        print(name_vec, cout, "\n");
        (void)getchar();
        name_vec.erase(name_vec.begin());
        std::system("cls");
    }
}
```

batan3karakter siliyor

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

    vector<string> name_vec;
    rfill(name_vec, 10, [] {return rname() + ' ' + rfname(); });
    print(name_vec, cout, "\n");
    (void)getchar();
    name_vec.erase(name_vec.begin(), next(name_vec.begin(), 3));
    std::system("cls");
    print(name_vec, cout, "\n");
}
```

silme

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

    vector<string> name_vec;
    rfill(name_vec, 10, [] {return rname() + ' ' + rfname(); });
    cout << "size = " << name_vec.size() << "\n";

    //name_vec.clear();
    //name_vec.erase(name_vec.begin(), name_vec.end());
    //name_vec.resize(0);
    //name_vec = {};
    name_vec = vector<string>{};

    cout << "size = " << name_vec.size() << "\n";
}
```

ivec ilistten küçükse true döndürcek ki
true, 2 farklı containeri kıyaslamak için
o fonk. çağrıldı

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

    vector<int> ivec{ 3, 6, 8, 1 };
    list<int> ilist{ 3, 6, 9, 1 };

    lexicographical_compare(ivec.begin(), ivec.end(), ilist.begin(), ilist.end())
}
```

```
4 class ytz65 {
5     public:
6         void operator()()const
7         {
8             ;
9         }
10    };
11
12 int main()
13 {
14     [](){}();
15     ytz65{}();
```

stateless lambda
capture clause

bir lambda ifadesinde olmazsa
olmazdır.parametre alınca veri
yapısı bildirmis olrz. bossa sınıfın
statik datamemberleri yokdemektr

2.ifade eger fonksiyon parametre almiyorsa
yazilmayabilir. aliyorsa parametrleri ordan bildirilir

```
2 #include <iostream>
3
4 int main()
5 {
6     auto y = [] (int x) {return x * x; }(10);
7 }
```

3.ifade ise fonksiyonun iley yapan
blocludur. return x;

y nin türü int ve degeri 100

144 basar

```
2 #include <iostream>
3
4
5 int main()
6 {
7     using namespace std;
8
9     auto y = [] (int x) {return x * x; };
10
11    cout << y(12) << "\n";
12
13
14
```

DER LAMBALAR, bunlar kullanyorsan parametre

```
[ ]()mutable {code}
[ ]()->int{code}
[ ]()noexcept{code}
[ ]()constexpr{code}
```

```
- class xyz125_ {  
public:  
    xyz125_(int x) : x(x) {}  
    bool operator()(int val)  
    {  
        return val % x == 0;  
    }  
private:  
    int x;  
};
```

```
- int main()  
{  
    using namespace std;  
    int x = 5;  
    [x](int val) {return val % x == 0; }
```

```
int main()
{
    using namespace std;
    int x = 5;
    int y = 15;
    int z = 20;

    auto f = [=](int val) {return val * (x + y + z); };

}
```

capture all

simdi lambda ifadesinde () overloadi consttur.
mutable yazarak const'luk ortadan kalkar ve data
membera yeni deger geçilebilri

```
- int main()
{
    using namespace std;
    int x = 5;

    auto f = [x](int val)mutable {x += val;  };
}
```

x'i ref ile çardmz için

```
int main()
{
    using namespace std;
    int x = 5;

    auto f = [&x](int i) {x *= i; };
    f(20);

    cout << "x = " << x << "\n";
}
```

otomatik deger x'e müdahale olur
100 basar

val'in türü doubledir . ckarm

```
#include <iostream>
#include "nutility.h" yaparken seçtik
#include <random>

int main()
{
    using namespace std;

    auto f = [] (int x) -> double {return x + 5; };

    auto val = f(20);
}
```



L ,

fonksiyonun `constexpr` olmasini
engelleyen bir ifade varsa syntax
hatasi olur

```
int main()
{
    auto f = [](int x) constexpr {
        static int y = 6;
        return x * y;
    }
}
```

```
- class xyzt78345 {
public:
    template <typename T>
    auto operator()(T x)
    {
        return x * x;
    }
};
```

```
- int main()
{
    [](auto x) {
        return x + x;
    }
}
```

```
5 //include <vector>
6
7 void func(int) { putchar('i'); }
8 void func(double) { putchar('d'); }
9 void func(long) { putchar('l'); }
10
11 using namespace std;
12
13 int main()
14 {
15     vector<int> vec(20);
16
17     //for_each(vec.begin(), vec.end(), func);
18     //for_each(vec.begin(), vec.end(), (void (*)(int))func);
19     for_each(vec.begin(), vec.end(), [] (auto x) {func(x); })
20 }
```

15 hata 16 legal ama verbose 17 k

```
8  
9  
10 int main()  
11 {  
12     auto f = [](auto &&... param) {  
13         print(std::forward<decltype(param)>(param)...);  
14     };  
15  
16     f(12, 3.4, 4.5f, "necati ergin");  
17 }  
18  
19 }
```

generalized lambda expression

```
3
4  class com_gen {
5    public:
6      com_gen(int x) : val {x * 6} {}
7    private:
8      int val;
9  };
```

initializer capture

```
10
11
12 int main()
13 {
14     using namespace std;
15
16     int x = 10;
17
18     auto f = [val = x * 6](int a) {
19         return val + a;
20     };
21
22     cout << f(20) << "\n";
23 }
```

```
3
4  class Nec {
5    public:
6      Nec() = default;
7
8      Nec(Nec&&); //move ctor
9      Nec& operator=(Nec&&); //move assignment
10 }
11
12
```

move only class

```
13 int main()
14 {
15   Nec x;
16
17   Nec y = std::move(x);
18   x = std::move(y)
```

```
3
4 class Nec {
5     public:
6         Nec() = default;
7
8         Nec(Nec&&); //move ctor
9         Nec& operator=(Nec&&); //move assignment
10    };
11
12
13 int main()      x'i tasimak için syntax
14 {
15     using namespace std;
16
17     Nec x;
18
19     [x = move(x)] () {}
20
21
22
23
24 }
```

x'i tasimak için syntax



```
4 #include <string>
5
6
7 using namespace std;
8
9
10
11 int main()
12 {
13
14     int (*fp)(int) = [] (int x) {return x * 6;};
15
16     cout << fp(20) << "\n";
17
18
19
20
21 }
22
23
24
```

120 basar örtülü dönüşüm var



240% No issues found

Error List

Entire Solution

0 Errors

0 Warnings

0 Messages

07 Build Only

```
int main()
{
    auto f = +[](int x) {return x * 5; };
}
```

positive lambda idiom, f function pointer türü olur



```
int foo(int val);
```

```
- int main()
{
    int a = 5, b = 7;

    const int x = [=](int val) {
        /**
         * return a * b + 3;
    }

}
```

iife özellikle const değerlere ilk değer atarken

```
//unique
```

```
int main()
```

{

```
    using namespace std;
```

```
    vector<int> ivec;
```

```
    rfill(ivec, 50, Rand{ 0, 5 });
```

```
    std::cout << "ivec.size() = " << ivec.size() << "\n";
```

```
    auto logic_end = remove(ivec.begin(), ivec.end(), 3);
```

```
    std::cout << "ivec.size() = " << ivec.size() << "\n";
```

```
    cout << "silinmemis oge sayisi: " << distance(ivec.begin(), logic_end) << "\n";
```

```
    cout << "silinmis   oge sayisi: " << distance(logic_end, ivec.end()) << "\n";
```



```
//remove
//remove_if
//unique

int main()
{
    using namespace std;

    vector<int> ivec;
    rfill(ivec, 50, Rand{ 0, 5 });

    std::cout << "ivec.size() = " << ivec.size() << "\n";

    auto logic_end = remove(ivec.begin(), ivec.end(), 3);
    std::cout << "ivec.size() = " << ivec.size() << "\n";
    cout << "silinmemis oge sayisi: " << distance(ivec.begin(), logic_end) << "\n";
    cout << "silinmis   oge sayisi: " << distance(logic_end, ivec.end()) << "\n";

    for_each(ivec.begin(), logic_end, [](int x) {cout << x; });
    std::cout << "\n";

    ivec.erase(logic_end, ivec.end());

    std::cout << "ivec.size() = " << ivec.size() << "\n";
}
```

```
using namespace std;
```

```
//remove  
//remove_if  
//unique
```

erase remove idiom

```
int main()  
{  
    using namespace std;  
  
    vector<int> ivec;  
    rfill(ivec, 50, Rand{ 0, 5 });  
  
    std::cout << "ivec.size() = " << ivec.size() << "\n";  
  
    ivec.erase(remove(ivec.begin(), ivec.end(), 3), ivec.end());  
}
```

```
using namespace std;

int main()
{
    using namespace std;

    vector<int> ivec;
    rfill(ivec, 50, Rand{ 0, 5 });

    std::cout << "ivec.size() = " << ivec.size() << "\n";

    //ivec.erase(remove(ivec.begin(), ivec.end(), 3), ivec.end());
    //since C++20

    std::erase(ivec, 3)

    std::cout << "ivec.size() = " << ivec.size() << "\n";
    print(ivec, cout, "");
}
```

```
using namespace std;
```

kaç değer sildiyse onu gösterir c++20 ile eklendi

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

    vector<int> ivec;
    rfill(ivec, 50, Rand{ 0, 5 });

    std::cout << "ivec.size() = " << ivec.size() << "\n";

    //ivec.erase(remove(ivec.begin(), ivec.end(), 3), ivec.end());
    //since C++20

    auto n = std::erase(ivec, 3);

    std::cout << "ivec.size() = " << ivec.size() << "\n";
    print(ivec, cout, "");
}
```

```
include <random>
include "date.h"
include <fstream>

using namespace std;

int main()

using namespace std;

vector<Date> dvec;
mt19937 eng{ 39845u };
uniform_int_distribution dist_day{ 1, 28 };
uniform_int_distribution dist_mon{ 1, 12 };
uniform_int_distribution dist_year{ 1940, 2023 };

generate_n(back_inserter(dvec), 10000, [&] {
    return Date{ dist_day(eng), dist_mon(eng), dist_year(eng) };
});

std::ofstream ofs{ "out.txt" };
if (!ofs) {
    std::cerr << "out.txt dosyasi olusturulamadi\n";
    exit(EXIT_FAILURE);
}

dvec.erase(remove_if(dvec.begin(), dvec.end(), [](const Date& dt) {
    return dt.month() > 3;
}), dvec.end());

print(dvec.begin(), dvec.end(), ofs, "\n");
```

```
gl_04.txt gl_03.txt gl_02.txt gl_01.txt notlar.txt out.txt main.cpp* □ X (Global Scope)
1 int main()
2 {
3     using namespace std;
4
5     vector<Date> dvec;
6     mt19937 eng{ 39845u };
7     uniform_int_distribution dist_day{ 1, 28 };
8     uniform_int_distribution dist_mon{ 1, 12 };
9     uniform_int_distribution dist_year{ 1940, 2023 };
10
11    generate_n(back_inserter(dvec), 10000, [&] {
12        return Date{ dist_day(eng), dist_mon(eng), dist_year(eng) };
13    });
14
15    std::ofstream ofs{ "out.txt" };
16    if (!ofs) {
17        std::cerr << "out.txt dosyasi olusturulamadi\n";
18        exit(EXIT_FAILURE);
19    }
20
21    /*dvec.erase(remove_if(dvec.begin(), dvec.end(), [](const Date& dt) {
22        return dt.month() > 3;
23    }), dvec.end());*/
24
25    auto n = erase_if(dvec, [](const Date& dt) {
26        return dt.month() != 6;
27    });
28
29    std::cout << n << " tane oge silindi\n"
30
31    print(dvec.begin(), dvec.end(), ofs, "\n");
32
33 }
```

```
9 #include <fstream>
10
11 using namespace std;
12
13 int main()
14 {
15     using namespace std;
16
17     string str;
18     std::cout << "bir cümle girin: ";
19     getline(cin, str);
20     cout << "[" << str << "]\\n";
21
22     str.erase(remove_if(str.begin(), str.end(), [](char c) {return
23         string{ "aeoui" }.find(c) != string::npos; }), str.end());
24
25     cout << "[" << str << "]\\n";
26
27
28 }
```

```
gl_03.txt gl_02.txt gl_01.txt notlar.txt out.txt main.cpp* X (Global Scope) main()

#include <fstream>

using namespace std;

int main()
{
    using namespace std;

    string str;
    std::cout << "bir cümle girin: ";
    getline(cin, str);
    cout << "[" << str << "]\n";

    //str.erase(remove_if(str.begin(), str.end(), [] (char c) {return      string{ "aeoui" }
    erase_if(str, [] (char c) {return      string{ "aeoui" }.find(c) != string::npos; });

    cout << "[" << str << "]\n";
}
```

```
std::cout << n << " tane oge silindi\n";  
print(dvec.begin(), dvec.end(), ofs, "\n");
```

}

uniq ardisiklar silindi

```
36666555511122233366699913577772221900001  
36512369135721901
```



deque ve vector, C++'ın STL (Standard Template Library) kütüphanesinin parçası olan farklı veri yapılarıdır. İşlevsel olarak benzer olsalar da bazı farklılıklar vardır:

Bellek Yapısı: vector dinamik bir diziye benzerken, deque (Double Ended Queue) çift uçlu bir kuyruk yapısını temsil eder. vector, bellekte ardışık bir blokta depolanan elemanları içerirken, deque ise bir dizi küçük blok olarak depolanan parçalara ayrılır. Bu, deque'nun daha dinamik bir bellek yapılarına sahip olduğu anlamına gelir ve eleman ekleme/çıkarma işlemlerinde daha esneklik sağlar.

İçerik Erişimi: vector, bellek bloğuna doğrudan erişim sağlayan bir dizi üzerinden çalışır. Bu, elemanlara doğrudan erişim ve rastgele erişim sağlar. Öte yandan, deque'de bellek blokları parçalara ayrıldığı için elemanlara doğrudan erişim maliyetli olabilir. Bunun yerine, deque öğelerine erişmek için aralık tabanlı erişim sağlar.

Kapasite Büyütme Stratejisi: vector'ün kapasitesi, ekleme işlemleri sırasında otomatik olarak genişletilir. Yeni bir eleman eklendiğinde, kapasite yetersizse, vector mevcut kapasitesini genişletir ve mevcut elemanları yeni bellek bloğuna kopyalar. Bu işlem genellikle bir kopyalama maliyeti gerektirir. deque ise iki uca birden eleman ekleme/çıkarma yapabilme özelliğine sahip olduğu için kapasite genişletmesi daha verimlidir. Ön taraftan veya arkadan eklenen elemanlar için kapasite büyütme işlemi yapmaz.

Performans: vector ve deque arasında performans farkı olabilir. vector, bellekte ardışık bir blokta depolandığından, rastgele erişimde daha hızlıdır. Ancak, deque daha esnek bir yapıya sahip olduğundan, baş ve sondan eleman ekleme/çıkarma işlemleri daha hızlı olabilir. İşlem türüne ve kullanım senaryosuna bağlı olarak performans farklılıklar gözlemlenebilir.

İteratör Geçerliliği: vector'ün bellek bloğu genişletildiğinde geçerli olan iteratorlerini korur. Ancak, deque'de bellek blokları parçalara ayrıldığından, yeni bloklar eklenip çıkarıldığında iteratorler geçerliliğini yitirebilir.

Bu farklılıklar göz önüne alındığı

11:02 ✓

Bu farklılıklar göz önüne alındığında, vector hızlı rastgele erişim gerektiren durumlar için daha uygunken, deque baş ve sondan eklemelerin daha yaygın olduğu senaryolarda daha verimli olabilir. Seçim, kullanım senaryosuna ve ihtiyaçlara bağlı olarak yapılmalıdır.

11:03 ✓

vector'u bir döngü ile dolaşarak
eger vectordeki ismin uzunluğu

- a) 5 ise vectorden silinecek
- b) 6 ise bir tane daha eklenecek

metin hasan mert necati remzi turhan ali veli selaMİ
mert NECATİ necati turhan turhan ali veli selaşelaMİ

partial sort 12:43 ✓✓

stable sort 12:43 ✓✓

Siz

partial sort

en yüksek olan 20 taneyi sıralamak için vb 12:44 ✓✓

```
#include <vector>
#include <string>
#include "nutility.h"
#include <fstream>

//NRVO

int main()
{
    using namespace std;

    auto vec = get_name_vec(10000);

    size_t n;

    std::cout << "ilk kac oge: ";
    cin >> n;

    partial_sort(vec.begin(), next(vec.begin(), n), vec.end());

    std::ofstream ofs{ "out.txt" };
    if (!ofs) {
        std::cerr << "out.txt dosyasi olusturulamadi\n";
        exit(EXIT_FAILURE);
    }
}
```

```
#include <vector>
#include <string>
#include "nutility.h"
#include <fstream>

//NRVO

int main()
{
    using namespace std;
    auto vec = get_name_vec(10000);

    vector<string> destvec(10);

    partial_sort_copy(vec.begin(), vec.end(), destvec.begin(), destvec.end());

    print(destvec, cout, "\n");
}
```

en küçük 10u destvece attı

```
#include <vector>
#include <string>
#include "nutility.h"
#include <fstream>
```

5000 eleman medyani gösterir
digerleri sirali degildir

```
//NRVO

int main()
{
    using namespace std;

    auto vec = get_name_vec(10000);

    nth_element(vec.begin(), vec.begin() + 5000, vec.end());

    std::ofstream ofs{ "out.txt" };
    if (!ofs) {
        std::cerr << "out.txt dosyasi olusturulamadi\n";
        exit(EXIT_FAILURE);
    }

    print(vec, ofs, "\n");
}
```



`'partition'`, C++'ın STL (Standard Template Library) kütüphanesinde bulunan bir algoritmadır. `'partition'` algoritması, bir aralıktaki elemanları belirli bir koşula göre böler ve bu koşulu sağlayan elemanları bir tarafına yerleştirirken, koşulu sağlamayanları diğer tarafına yerleştirir.

`'partition'` algoritması aşağıdaki gibi kullanılır:

cpp

Copy code

```
template <class ForwardIterator, class UnaryPredicate>
ForwardIterator partition(ForwardIterator first, ForwardIterator last, Unary
```

- `'first'` ve `'last'`, `'partition'` işleminin uygulanacağı elemanların aralığını belirtir. `'first'` aralığın başlangıç iterator'ünü, `'last'` ise aralığın sonunu bir sonraki iterator'ü gösterir.
- `'pred'` ise bir koşulu ifade eden bir işlev veya işlev nesnesi olan bir `'UnaryPredicate'` türüdür. Bu koşul, elemanların nasıl bölüneceğini belirler. `'pred'` işlevi, elemanları tek tek alır ve `'true'` veya `'false'` değeri döndürür. `'true'` döndürdüğünde elemanın koşulu sağladığı, `'false'` döndürdüğünde ise koşulu sağlamadığı kabul edilir.

`'partition'` işlemi gerçek

içinde i olanlar x te digerleri y de

```
#include <deque>
#include <list>

int main()
{
    using namespace std;

    auto vec = get_name_vec(100);

    deque<string> x;
    list<string> y;           I

    partition_copy(vec.begin(), vec.end(), back_inserter(x), back_inserter(y),
        [](&const string& s) {
            return s.contains('i');
        });

    print(y, cout, "\n");
```

```
#include <deque>
#include <list>

int main()
{
    using namespace std;

    auto vec = get_name_vec(100);

    deque<string> x(100);
    list<string> y(100);

    auto [iter_x, iter_y] = partition_copy(vec.begin(), vec.end(), x.begin(), y.begin(),
        [] (const string& s) {
            return s.contains('i');
        });

    print(x.begin(), iter_x, cout, "\n");
}
```

eger sıralıysa true döndürür alttaki büyükten küçüğe

```
#include <deque>
#include <list>

int main()
{
    using namespace std;

    int a[]{ 45, 21, 12, 6, 5, 1, -4, -6};

    cout << boolalpha << is_sorted(begin(a), end(a)) << "\n";
    cout << boolalpha << is_sorted(begin(a), end(a), greater{}) << "\n";
}
```

```
#include <deque>
#include <list>
```

küçükten büyüğe sıralamadaki sıralamayı
bozan 5'i döndürür

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

    int a[] { 3, 6, 9, 12, 5, 6, 89 };

    auto iter = is_sorted_until(begin(a), end(a));

    cout << *iter << "\n";
}
```

o

```
using namespace std;

vector<int> ivec;
rfill(ivec, 16, Rand{ 0, 1000 });
print(ivec, cout);
make_heap(ivec.begin(), ivec.end());
print(ivec, cout);

std::cout << "en buyuk deger : " << ivec.front() << "\n";
pop_heap(ivec.begin(), ivec.end());

cout << "en son oge : " << ivec.back() << "\n";

ivec.pop_back();
print(ivec, cout);
```

```
896 786 870 755 667 723 793 398 161 663 152 367 423 223 567 363
```

```
-----
```

```
en buyuk deger : 896
```

```
en son oge : 896
```

```
870 786 793 755 667 723 567 398 161 663 152 367 423 223 363
```

```
{ using namespace std;

vector<int> ivec;
rfill(ivec, 16, Rand{ 0, 1000 });
print(ivec, cout);
make_heap(ivec.begin(), ivec.end());
print(ivec, cout);

ivec.push_back(2345);
push_heap(ivec.begin(), ivec.end());
print(ivec, cout);
```

```
int main()
```

```
{ Microsoft Visual Studio Debug Console
```

```
595 438 898 146 599 579 508 450 803 261 123 3 855 365 364 978
```

```
-----
```

```
978 803 898 595 599 855 508 450 438 261 123 3 579 365 364 146
```

```
-----
```

```
2345 978 898 803 599 855 508 595 438 261 123 3 579 365 364 146 450
```

```
-----
```

```
C:\Users\necat\source\repos\cpp_january\Release\cpp_january.exe (process 21564) ex
Press any key to close this window . . .
```

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

    vector<int> ivec;
    rfill(ivec, 16, Rand{ 0, 1000 });

    print(ivec, cout);

    make_heap(ivec.begin(), ivec.end());
```

make_heap(ivec.begin(), ivec.end());

```
    while (!ivec.empty()) {
        cout << ivec.front() << " kuyruktan cikiyor\n";
        pop_heap(ivec.begin(), ivec.end());
        ivec.pop_back();
        (void)getchar();
    }
}
```

kuyruktan çıkarıyor önce
küçük olani

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

    vector<int> ivec;
    rfill(ivec, 16, Rand{ 0, 1000 });

    print(ivec, cout);

    make_heap(ivec.begin(), ivec.end(), greater{});
}

while (!ivec.empty()) {
    cout << ivec.front() << " kuyruktan cikiyor\n";
    pop_heap(ivec.begin(), ivec.end(), greater{});
    ivec.pop_back();
    (void)getchar();
}
```

```
using namespace std;

vector<string> svec;

rfill(svec, 40, rname);
print(svec, cout);

auto iter = svec.begin();

while (iter != svec.end()) {
    if (iter->length() == 5) {
        iter = svec.erase(iter);
    }
    else if (iter->length() == 6) {
        iter = svec.insert(iter, *iter);
        advance(iter, 2);
    }
    else {
        ++iter;
    }
}

print(svec, cout);
```



```
vector<string> svec;

rfill(svec, 40, rname);
print(svec, cout);

auto iter = svec.begin();

while (iter != svec.end()) {
    if (iter->length() == 5) {
        iter = svec.erase(iter);
    }
    else if (iter->length() == 6) {
        iter = svec.insert(iter, *iter);
        advance(iter, 2);
    }
    else {
        ++iter;
    }
}

print(svec, cout);
```

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

    forward_list<int> x{ 2, 5, 7, 9, 3, 1, 9 };

    auto iter = next(x.begin());
    std::cout << "*iter = " << *iter << "\n";

    x.insert_after(iter, -1);           -1'i 7nin bulunduğu
    for (auto i : x)                 konuma ekler
        std::cout << i << " ";
}
```

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

    forward_list<int> x{ 2, 5, 7, 9, 3, 1, 9 };

    auto iter = next(x.begin());
    std::cout << "*iter = " << *iter << "\n";

    x.erase_after(iter);

    for (auto i : x)
        std::cout << i << " ";
}
```



normal begin'i çağırırsan 5in konumuna verileri ekler, before_begin ile en basa ekleme yapabiliriz

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

    forward_list<int> x{ 2, 5, 7, 9, 3, 1, 9 };

    x.insert_after(x.before_begin(), {444, 555, 666});

    for (auto i : x)
        std::cout << i << " ";
}
```

```
4 #include <map>
5 #include "nutility.h"
6
```

```
7 using namespace std;
```

HER isimden bir TANE
OLCAK

```
8 int main()
9 {
10     auto fcmp = spred();
11
12     set<string, decltype(fcmp)> myset;
13
14     for (int i = 0; i < 1000; ++i) {
15         myset.insert(rname() + ' ' + rfname());
16         print(myset, cout, "\n");
17         (void)getchar();
18         system("cls");
19 }
```

No issues found

```
set<string> myset;

rfill(myset, 8, rname);
print(myset, cout);

std::cout << "myset.size() = " << myset.size() << "\n";

cout << "eklenecek ogeyi girin: ";
string name;
cin >> name;

//pair<set<string>::iterator, bool> retval = myset.insert(name);
auto retval = myset.insert(name);

if (retval.second) {
    cout << "ekleme yapildi\n";
    std::cout << "myset.size() = " << myset.size() << "\n";
    cout << "eklenmis isim :" << *retval.first << "\n";
    cout << "eklenmis ogenin indeksi : " << distance(myset.begin(), retval.first) << "\n";
    print(myset, cout);
}
else {
    cout << "ekleme yapılmadi\n";
    cout << *retval.first << "\n";
    cout << "var olan anahtarın indeksi : " << distance(myset.begin(), retval.first) << "\n";
}
```

Necati Ergin

```
#include <string>
#include <set>
#include <map>
#include "nutility.h"
#include "date.h"

using namespace std;

int main()
{
    set<int> myset{ 3, 6, 9, 12, 78, 90 ,234 };
    print(myset, cout);

    myset.insert(myset.begin(), 45); //hint insert
    myset.emplace_hint(myset.begin(), 45); //hint insert
}
```

olmasi gerektigi yere 45i eklerler

Necati Ergin

```
#include <string>
#include <set>
#include <map>
#include "nutility.h"
#include "date.h"

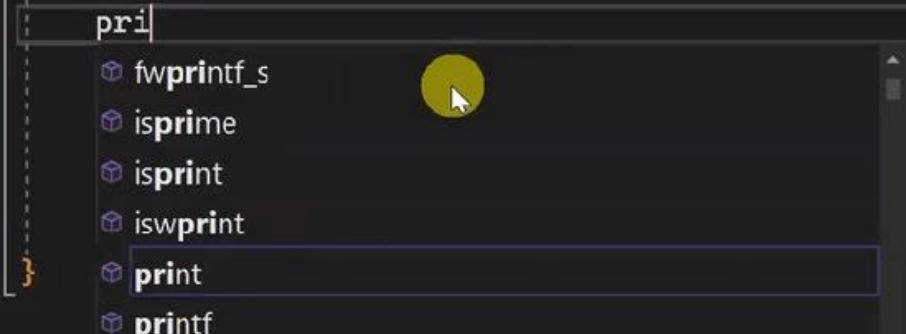
using namespace std;

int main()
{
    set<int> myset{ 3, 6, 9, 12, 78, 90 ,234 };

    auto beg = myset.find(9);
    auto end = myset.find(90);

    myset.erase(beg, end);

    pri|
```



A screenshot of a C++ development environment showing code completion. The cursor is at the end of the word 'pri'. A dropdown menu lists several functions: fwprintf_s, isprime, isprint, iswprint, print, and printf. The 'print' option is highlighted with a blue rectangle, and a green circle with a white arrow points to it, indicating it is the selected suggestion.

9 12 78 silinir

Necati Ergin

kaç öge sildigini döndürür

```
#include <string>
#include <set>
#include <map>
#include "nutility.h"
#include "date.h"

using namespace std;

int main()
{
    multiset<int> myset;
    rfill(myset, 100, Rand{ 0, 120 });
    print(myset, cout);

    int val;
    std::cout << "silinecek deger : ";
    cin >> val;

    auto n = myset.erase(val);
    cout << n << " oge silindi\n";
}
```

Necati Ergin

15.SATIR 2 döndürür, 17. sair 7nin gelebilecegi en uzak yer olan 6 döndürür, 18.satr ise kaç adet 7 oldugunu döndürür

```
7
8     using namespace std;
9
10    int main()
11    {
12        multiset<int> myset{ 2, 5, 7, 7, 7, 7, 9, 9, 13, 13, 45, 89 };
13
14        auto iter_lower = myset.lower_bound(7);
15        cout << "distance for lower bound : " << distance(myset.begin(), iter_lower) << "\n";
16        auto iter_upper = myset.upper_bound(7);
17        cout << "distance for upper bound : " << distance(myset.begin(), iter_upper) << "\n";
18        cout << "distance for equal range : " << distance(iter_lower, iter_upper) << "\n";
19    }
```

yeni eklenecek olan isimi lower_bound kullanilarak eklenmesi gereken yere insert ile ekleniyor, ekleyecegi konumuda iter_lower tutuyor

```
4 #include <algorithm>
5
6
7 int main()
8 {
9     using namespace std;
10
11     vector<string> svec;
12
13     for (int i = 0; i < 10; ++i) {
14         auto name = get_random_name();
15         cout << name << " eklenecek " << "\n\n";
16         auto iter_lower = lower_bound(svec.begin(), svec.end(), name);
17         svec.insert(iter_lower, name);
18         print(svec, cout, "\n");
19
20     }
21 }
```

I

```
{  
    using namespace std;  
  
    set <string> myset { "esra", "haydar", "fulya", "galip", "murat", "necati", "tarik"};  
    print(myset, cout);  
  
    string old_key, new_key;  
  
    cout << "eski ve yeni anahtarları girin: ";  
    cin >> old_key >> new_key;  
  
    if (auto iter = myset.find(old_key); iter != myset.end()) {  
        auto handle = myset.extract(iter);  
        //std::cout << "myset.size() = " << myset.size() << "\n";  
        //getchar();  
        handle.value() = new_key;  
        myset.insert(std::move(handle));  
        print(myset, cout);  
    }  
}
```



extract, eski değeri siliyor ama destructoru
çağırıyor o nesne için. yerine yenikey ekleniyor

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

    map<string, int> m{
        {"murat", 1345},
        {"ayse", 456},
        {"deniz", 333},
        {"aydin", 678},
        {"burak", 222},
        {"sezai", 456},
        {"can", 1345},
    };

    m.insert(pair<string, int>{"nazan", 4912});
    m.insert(pair{ "remzi", 218 });
    m.insert(make_pair("ulvi", 555));
    m.emplace("osman", 962);

    for (const auto&[name, no] : m) {
        cout << name << " " << no << "\n";
    }
}
```

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

    map<string, int> m;

    for (int i = 0; i < 5; ++i) {
        m.emplace(rname(), Irand{ 1000, 9000 }());
    }

    std::cout << "m.size() = " << m.size() << "\n";

    for (const auto& p : m)
        cout << p << "\n";

    cout << "isim ve numara girin: ";
    string name;
    int no;

    cin >> name >> no;
    if (auto [iter, inserted] = m.insert({ name, no }); inserted) {
        std::cout << "ekleme yapildi " << iter->first << "\n";
    }
    else {
        std::cout << "var olan kisinin numarasi " << iter->second << "\n";
    }
}
```

```
{  
using namespace std;  
  
map<string, int> m;  
  
for (int i = 0; i < 5; ++i) {  
    m.emplace(rname(), Irand{ 1000, 9000 }())  
}  
  
std::cout << "m.size() = " << m.size() << "\n";  
print(m, "\n");  
  
string name;  
int no;  
std::cout << "isim ve numara giriniz: ";  
cin >> name >> no;  
  
m[name] = no;  
  
std::cout << "m.size() = " << m.size() << "\n";  
print(m, "\n");
```

```
Microsoft Visual Studio Debug Console  
m.size() = 4  
[hakan, 3944]  
[keriman, 3344]  
[tayyar, 3963]  
[zekai, 1549]
```

```
-----  
isim ve numara giriniz: hakan 888  
m.size() = 4  
[hakan, 888]  
[keriman, 3344]  
[tayyar, 3963]  
[zekai, 1549]
```

```
-----  
C:\Users\necat\source\repos\cpp_janua  
Press any key to close this window .
```

necati eklendi

```
{  
using namespace std;  
  
map<string, int> m;  
  
for (int i = 0; i < 5; ++i) {  
    m.emplace(rname(), Irand{ 1000, 9000 }());  
}  
  
std::cout << "m.size() = " << m.size() << "\n";  
  
print(m, "\n");  
  
string name;  
int no;  
std::cout << "isim ve numara giriniz: ";  
cin >> name >> no;  
  
m[name] = no;  
  
std::cout << "m.size() = " << m.size() << "\n";  
  
print(m, "\n");
```

```
Microsoft Visual Studio Debug Console  
m.size() = 5  
[ata, 8442]  
[celik, 1467]  
[doran, 3409]  
[esin, 3434]  
[sidre, 7169]  
-----  
isim ve numara giriniz: necati 123  
m.size() = 6  
[ata, 8442]  
[celik, 1467]  
[doran, 3409]  
[esin, 3434]  
[necati, 123]  
[sidre, 7169]  
-----
```

cok hos

```
5 #include "nutility.h"
6 #include <vector>
7
8
9
10 //her bir isimden kaç tane var?
11
12 int main()
13 {
14     using namespace std;
15
16     vector<string> svec;
17     rfill(svec, 10000, rname);
18     std::cout << "svec.size() = " << svec.size() << "\n";
19
20     map<string, int> cmap;
21
22     for (const auto& name : svec) {
23         ++cmap[name];
24     }
25 }
26
27
28 }
```



A tooltip is displayed over the line of code `++cmap[name];`. The tooltip contains the following information:

- (local variable) std::map<std::string, int> cmap
- Search Online

```
map<string, int> cmap;

for (const auto& name : svec) {
    cmap[name]++;
}

vector<pair<string, int>> vec{ cmap.begin(), cmap.end() };
sort(vec.begin(), vec.end(), [](const auto& p1, const auto& p2) {
    return p1.second > p2.second;
});

std::ofstream ofs{ "out.txt" };
if (!ofs) {
    std::cerr << "out.txt dosyasi olusturulamadi\n";
    exit(EXIT_FAILURE);
}

ofs << left;

for (const auto& [name, cnt] : vec) {
    ofs << setw(24) << name << "    " << cnt << "\n";
}
```

kac tane isimi büyükten
küçüğe sıralanır

```
//her bir isimden kaç tane var?  
  
int main()  
{  
    using namespace std;  
  
    map<int, string> mymap;  
  
    for (int i = 0; i < 10; ++i) {  
        mymap.emplace(Irand{ 0, 20 }(), rname());  
    }  
  
    print(mymap, "\n");  
  
    try {  
        mymap.at(12) += "can";  
        cout << "oge degistirldi";  
        print(mymap, "\n");  
    }  
  
    catch (const std::exception& ex) {  
        std::cout << "12 anahtari mevcut degil\n";  
        std::cout << "exception caught: " << ex.what  
    }  
}
```

Microsoft Visual Studio Debug Console

```
[8, emirhan]  
[9, kamil]  
[12, canan]  
[14, emre]  
[18, atakan]  
[19, ediz]  
-----  
oge degistirldi[8, emirhan]  
[9, kamil]  
[12, canan] can  
[14, emre]  
[18, atakan]  
[19, ediz]  
-----  
C:\Users\necat\source\repos\cpp_...  
Press any key to close this window
```

```
//her bir isimden kaç tane var?
```

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

    map<string, string> mymap;

    for (int i = 0; i < 10; ++i) {
        mymap.emplace(rname(), rtown());
    }

    print(mymap, "\n");

    cout << "eski ve yeni isimleri girin: ";
    string old_name, new_name;
    cin >> old_name >> new_name;

    if (auto iter = mymap.find(old_name); iter != mymap.end()) {
        auto handle = mymap.extract(iter);
        handle.key() = new_name;
        handle.mapped() += " ilcesi";
        mymap.insert(move(handle));
        print(mymap, "\n");
    }
    else {
        std::cout << "bulunamadi\n";
    }
}
```

extract kullanım



```
using namespace std;

//    sehir  isim
multimap<string, string> mymap;

for (int i = 0; i < 1000; ++i) {
    mymap.emplace(rtown(), rname());
}

print(mymap, "\n");

string town;

std::cout << "sehri girin: ";
cin >> town;

auto [iter_lower, iter_upper] = mymap.equal_range(town);

print(iter_lower, iter_upper, "\n");
```

```
OutIter Copy(Iter beg, Iter end, OutIter destbeg)
{
    while (beg != end)
        *destbeg++ = *beg++;

    return destbeg;
}
```

mucizevi

```
template <typename T>
class OstreamIterator {
public:
    OstreamIterator(std::ostream& os, const char *psep = "") :m_os(os), mp_sep(psep) {}
    OstreamIterator& operator++() { return *this; }
    OstreamIterator& operator++(int) { return *this; }
    OstreamIterator& operator*() { return *this; }
    OstreamIterator& operator=(const T& val)
    {
        m_os << val << mp_sep;
        return *this;
    }
private:
    const char* mp_sep;
    std::ostream& m_os;
};
```

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

    vector<string> svec{ "ali", "tan", "eda" , "ece", "ege" , "naz" };
    Copy(svec.begin(), svec.end(), OstreamIterator<string>(cout, "\n"));
}
```

```
vector<Date> dvec;
rfill(dvec, 10000, Date::random);

std::ofstream ofs{ "out.txt" };
if (!ofs) {
    std::cerr << "out.txt dosyasi olusturulamadi\n";
    exit(EXIT_FAILURE);
}

//yukaridaki dosyaya sadece ayin n.gunu olan tarihleri yazdiriniz
int mday;
std::cout << "ayin kacinci gunu: ";
cin >> mday;

copy_if(dvec.begin(), dvec.end(), ostream_iterator<Date>(ofs, "\n"),
        [mday](const Date& date) {return date.month_day() == mday; });
```

```
template <>
struct std::hash<Date> {
    std::size_t operator()(const Date& d) const
    {
        return std::hash<int>{}(d.month_day()) +
               std::hash<int>{}(d.month()) +
               std::hash<int>{}(d.year());
    }
};

int main()
{
    using namespace std;

    unordered_set<Date> myset;
    //unordered_set<Date, hash<Date>> myset;

    for (int i = 0; i < 10; ++i) {
        myset.insert(Date::random());
    }

    for (const auto& d : myset) {
        std::cout << d << "\n";
    }
}
```

```
struct DateHasher {    }

    std::size_t operator()(const Date& d) const
    {
        return std::hash<int>{}(d.month_day()) +
               std::hash<int>{}(d.month()) +
               std::hash<int>{}(d.year());
    }
};

int main()
{
    using namespace std;

    unordered_set<Date, DateHasher> myset;
    //unordered_set<Date, hash<Date>> myset;

    for (int i = 0; i < 10; ++i) {
        myset.insert(Date::random());
    }

    for (const auto& d : myset) {
        std::cout << d << "\n";
    }
}
```

```
1
2
3     std::size_t hash_date(const Date & d)
4     {
5
6         return std::hash<int>{}(d.month_day()) +
7             std::hash<int>{}(d.month()) +
8             std::hash<int>{}(d.year());
9     }
0
1
2
3 int main()
4 {
5     using namespace std;
6
7     unordered_set<Date, size_t(*)(const Date&)> myset(hash_date);
8     //unordered_set<Date, hash<Date>> myset;
9
0     for (int i = 0; i < 10; ++i) {
1         myset.insert(Date::random());
2     }
3
4     for (const auto& d : myset) {
5         std::cout << d << "\n";
6     }
7
8 }
```

```
61             std::hash<int>{}(p.getz());
62         }
63     };
64
65
66 int main()
67 {
68     using namespace std;
69
70     unordered_set<Point, PointHasher> myset;
71
72
73     for (int i = 0; i < 1000; ++i) {
74         myset.insert(Point::random_point());
75     }
76
77     std::cout << "myset.size() = " << myset.size() << "\n";
78
79     std::cout << "myset.bucket_count() = " << myset.bucket_count() << "\n";
80     cout << "load factor = " << myset.load_factor() << "\n";
81     cout << "load factor = " << static_cast<float>(myset.size()) / myset.bucket_count() << "\n";
82
83
84 }
```

80 81 ayn eye dönürür

```
1 #include <iostream>
2 #include <array>
3
4
5
6 int main()
7 {
8     using namespace std;
9
10    array ar{ 1, 3, 5, 7, 9 };
11
12    //int* ptr = ar;//gecersiz
13    auto p = ar.data();
14 }
```



```
1 #include <iostream>
2 #include <array>
3
4
5
6 int main()
7 {
8     using namespace std;
9
10    array ar{ 1, 3, 5, 7, 9 };
11
12    //int* ptr = ar;//gecersiz
13    auto p = ar.data();
14    auto p = &ar[0]
15 }
```

```
1 #include <iostream>
2 #include <array>
3 #include <iostream>
4
5
6
7     template <typename T, std::size_t size>
8     std::ostream& operator<<(std::ostream& os, const std::array<T, size>& ar)
9     {
10         os << '[';
11
12         for (size_t i{}; i < size - 1; ++i) {
13             os << ar[i] << ", ";
14         }
15
16         return os << ar.back() << ']';
17     }
18
19
20
21     int main()
22     {
23         using namespace std;
24
25         array ar1{ 5, 6, 7, 1, 9, 3, 2 };
26         array ar2{ 3.4, 1.1, 7.8, 9.3 };
27
28         cout << ar1 << "\n" << ar2 << "\n";
29     }
```

fontlar

```
1 #include <iostream>
2 #include <array>
3 #include <iostream>
4 #include <string>
5 #include "nutility.h"
6
7
8     template <typename T, std::size_t size>
9 std::ostream& operator<<(std::ostream& os, const std::array<T, size>& ar)
10 {
11     os << '[';
12
13     for (size_t i{}; i < size - 1; ++i) {
14         os << ar[i] << ", ";
15     }
16
17     return os << ar.back() << ']';
18 }
19
20
21
22 int main()
23 {
24     using namespace std;
25
26     array<int, 5> ar{2, 4, 6, 7, 9};
27
28     for (auto iter = ar.cbegin(); iter != ar.crend(); ++iter) {
29         std::cout << *iter << "\n";
30     }
31
32 }
```



ar13 ile dolar

```
25
26
27     int main()
28     {
29         using namespace std;
30
31         array<int, 5> ar;
32
33         ar.fill(13);
34
35         copy(ar.begin(), ar.end(), ostream_iterator<int>(cout, " "));
36
37 }
```

simdi get0 3 döndürür get7 syntax hatası

```
int main()
{
    array ar{ 3, 5, 8, 2, 4, 6, 9 };
    cout << get<0>(ar) << "\n";
    cout << get<7>(ar) << "\n";
}
```



```
int main()
{
    array ar{ 3, 5, 8};

    auto &[x, y, z] = ar;

    ++x;
    ++y;
    ++z;

    for (auto i : ar) {
        std::cout << i;
    }
}
```

perfect forwarding

```
template <typename ...Args> <T> Provide sample template arguments for In  
class Stack {  
  
public:  
    void push(const T& x)  
    {  
        c.push_back(x);  
    }  
  
    void push(T&& x)  
    {  
        c.push_back(std::move(x));  
    }  
  
    template <typename ...Args> <T> Provide sample template arguments for In  
    void emplace(Args && ...args)  
    {  
        c.emplace_back(std::forward<Args>(args)...);  
    }  
  
protected:  
    std::vector<T> c;  
};
```

```
int main()
{
    const auto f = [] (const Date& d1, const Date& d2) {
        return d1.month_day() < d2.month_day();
    };

    priority_queue<Date, vector<Date>, decltype(f)> x;

    for (int i = 0; i < 10; ++i) {
        x.push(Date::random());
    }

    while (!x.empty()) {
        std::cout << x.top();
        x.pop();
        (void)getchar();
    }
}
```

```
cpp_january
4     template <typename T>
5     class ReferenceWrapper {
6
7     public:
8         ReferenceWrapper(T &t) : mp{&t} {}
9         operator T& () {
10             return *mp;
11         }
12
13         T& get() {
14             return *mp;
15         }
16
17         ReferenceWrapper operator=(T& t) {
18             mp = &t;
19         }
20
21     private:
22         T* mp;
23     };
24
25
26     int main()
27     {
28         int x{};
29
30         ReferenceWrapper<int> r{ x };
31
32         int ival = r.operator int& ();
33         int y = 10;
34
35
36
37 }
```

```
7
8     int main()
9     {
10        using namespace std;
11
12        int x = 10;
13
14        reference_wrapper<int> r{ x };
15
16        cout << r << "\n";
17        cout << r.operator int& () << "\n";
18
19        int ival = 560;
20
21        r = ival;
22
23        cout << "r = " << r << '\n';
24
25
26
27
28 }
```

r = 560

```
4
5     template <typename T>
6     std::reference_wrapper<T> Ref(T& t)
7     {
8         return reference_wrapper<T>{t};
9     }
10
11    int main()
12    {
13        using namespace std;
14
15        int x = 10;
16
17        auto y = Ref(x)
18
19
20
21 }
```

y'nin türü reference-wrapper
template int açılımı

```
using namespace std;
```

```
int x = 10;      r1 wrapper int açlm r2
```

```
auto r1 = ref(x);
```

```
auto r2 = cref(x);
```



```
using namespace std;
```

```
int x = 10;
```

```
auto r1 = ref(x);
```

```
auto r2 = cref(x);
```

```
++r1;
```

```
cout << "x = " << x << '\n';
```

||

```
++r2;
```

ref(m) alternatif

```
[ } ;  
-int main()  
{  
    MyClass m;  
    func<MyClass &>(m);  
}
```

```
[#include <algorithm>
```

```
template <typename ForIt, typename F> <T> Provide s
void generate(ForIt beg, ForIt end, F fn)
{
    while (beg != end) {
        *beg++ = fn();
    }
}

int main()
```

```
template <typename ForIt, typename F>
void Generate(ForIt beg, ForIt end, F fn)
{
    while (beg != end) {
        *beg++ = fn();
    }
}

int main()
{
    using namespace std;

    mt19937 eng;

    vector<unsigned int> ivec(1000);

    Generate(ivec.begin(), ivec.end(), ref(eng));
}
```

```
void foo(int x, int y, int z)
{
    std::cout << "x = " << x << '\n';
    std::cout << "y = " << y << '\n';
    std::cout << "z = " << z << '\n';

    std::cout << "-----\n";
}

int main()
{
    using namespace std;

    auto fn = std::bind(foo, 10, 20, 30);

    fn();
}
```



```
void foo(int x, int y, int z)
{
    std::cout << "x = " << x << '\n';
    std::cout << "y = " << y << '\n';
    std::cout << "z = " << z << '\n';

    std::cout << "-----\n";
}

int main()
{
    using namespace std;
    using namespace placeholders;

    auto fn = bind(foo, 10, 20, _1);

    fn(333);
```

10 20 333



```
void foo(int x, int y, int z)
{
    std::cout << "x = " << x << '\n';
    std::cout << "y = " << y << '\n';
    std::cout << "z = " << z << '\n';

    std::cout << "-----\n";
}

int main()
{
    using namespace std;
    using namespace placeholders;

    auto fn = bind(foo, _1, 10, 20);
    fn(333);
}
```

333 10 20

I

```
void foo(int x, int y, int z)
{
    std::cout << "x = " << x << '\n';
    std::cout << "y = " << y << '\n';
    std::cout << "z = " << z << '\n';

    std::cout << "-----\n";
}

int main()
{
    using namespace std;
    using namespace placeholders;

    auto fn = bind(foo, _1, _2, 20);

    fn(999, 777);
```



999 777 20

```
void foo(int x, int y, int z)
{
    std::cout << "x = " << x << '\n';
    std::cout << "y = " << y << '\n';
    std::cout << "z = " << z << '\n';

    std::cout << "-----\n";
}

int main()
{
    using namespace std;
    using namespace placeholders;

    auto fn = bind(foo, _1, _1, _1);

    fn(3);
```

3 3 3

```
void foo(int x, int y, int z)
{
    std::cout << "x = " << x << '\n';
    std::cout << "y = " << y << '\n';
    std::cout << "z = " << z << '\n';

    std::cout << "-----\n";
}

int main()
{
    using namespace std;

    vector<int> ivec;
    rfill(ivec, 100, Irand{ 0, 100 });
    print(ivec);
    int x;

    std::cout << "kactan buyuk olanlar yazilsin: ";
    cin >> x;

    auto fn = bind(greater<int>{}, placeholders::_1, x);
    copy_if(ivec.begin(), ivec.end(), ostream_iterator<int>(cout, " "), fn);
}
```

```
    std::cout << "x = " << x << '\n';
    std::cout << "y = " << y << '\n';
    std::cout << "z = " << z << '\n';

    std::cout << "-----\n";
}

int main()
{
    using namespace std;

    vector<int> ivec;
    rfill(ivec, 100, Irand{ 0, 100 });
    print(ivec);
    int x;

    std::cout << "kactan buyuk olanlar yazilsin: ";
    cin >> x;

    copy_if(ivec.begin(), ivec.end(), ostream_iterator<int>(cout, " "),
            bind(greater<int>{}, placeholders::_1, x));
}
```

```
#include <functional>
#include "nutility.h"
```

99 60 90

```
void foo(int x, int y, int z)
{
    std::cout << "x = " << x << '\n';
    std::cout << "y = " << y << '\n';
    std::cout << "z = " << z << '\n';

    std::cout << "-----\n";
}

int main()
{
    using namespace std;
    using namespace placeholders;

    int a = 77;
    int b = 33;

    bind(foo, 99, _2, _1)(90, 60);
```



`std::bind` işlevi, bir işlev veya işlev nesnesini argümanlarla birleştirerek yeni bir işlev nesnesi oluşturur. İlk argüman olarak işlevi veya işlev nesnesini alır ve ardından işlevin parametreleri yerine gelecek değerleri alır. `_1` ve `_2` gibi yer tutucu parametreler, sonradan belirli bir değerle doldurulacak olan parametre pozisyonlarını temsil eder.

Örneğin, aşağıdaki gibi bir C++ kod parçasında bu ifadeyi kullanabilirsiniz:

```
cpp Copy code

#include <functional>
#include <iostream>

void foo(int x, int y) {
    std::cout << "x: " << x << ", y: " << y << std::endl;
}

int main() {
    auto boundFunc = std::bind(foo, 99, std::placeholders::_1, std::placeholders::_2);
    boundFunc(42, 55); // foo(99, 42, 55) çağrısını yapar

    return 0;
}
```

Yukarıdaki örnekte, `std::bind` işlevi kullanılarak `foo` işlevi birleştiriliyor. `_1` ve `_2` yer tutucu parametreleri, gerçek değerlerle değiştirilmesi gereken parametre pozisyonlarını temsil eder. `boundFunc`, `foo` işlevinin 99 ile birleştirilmiş bir sürümünü temsil eder ve ardından gerçek değerlerle çağrılabılır.



Verdiğiniz kod parçasında, `'auto fn = bind(foo, _3, _1, _2);'` ifadesiyle bir `'fn'` adında bir işlev nesnesi oluşturuluyor. Ardından `'fn(10, 20, 30);'` ifadesiyle bu işlev nesnesi çağrılmıyor.



Burada `'std::bind'` işlevi kullanılarak `'foo'` işlevine bir bağlantı yapılıyor. Yer tutucu nesneleri `'_3'`, `'_1'` ve `'_2'` kullanılıyor.

`'fn(10, 20, 30);'` ifadesi, `'fn'` işlev nesnesini çağrıırken gerçek değerlerle yer tutucuların yerini doldurur. `'_3'` yer tutucusu, `'fn'` 'nin ilk argümanı olan `'10'` ile eşleşir. `'_1'` yer tutucusu, `'fn'` 'nin ikinci argümanı olan `'20'` ile eşleşir ve `'_2'` yer tutucusu da `'fn'` 'nin üçüncü argümanı olan `'30'` ile eşleşir. Sonuç olarak, `'foo'` işlevi çağrılr ve parametreler `'30'`, `'10'` ve `'20'` olarak aktarılır.

Bu durumda, `'foo'` işlevinin parametre sırası değiştirilmiş oluyor. İlk argümanı `'_3'`, ikinci argümanı `'_1'` ve üçüncü argümanı `'_2'` ile belirtiliyor. Yani, `'fn(10, 20, 30);'` çağrısi aslında `'foo(30, 10, 20);'` çağrısını gerçekleştiriyor.

Sonuç olarak, verdiğiniz kod parçası, `'foo'` işlevini çağırmak için `'std::bind'` işlevini kullanarak parametrelerin sırasını değiştiren bir işlev nesnesi oluşturur ve bu işlev nesnesini `'fn'` olarak adlandırır. Ardından `'fn'` 'yi çağrıırken gerçek değerlerle birlikte işlevi çağrıır.

```
- void foo(int& r)
{
    ++r;
}
```

30 olur kopyalar

```
- int main()
{
    using namespace std;
    using namespace placeholders;

    int x{ 30 };

    auto fn = bind(foo, x);

    fn();

    cout << "x = " << x << '\n';
}
```

A yellow circular icon containing a white cursor symbol, positioned to the right of the code editor window.

```
- void foo(int& r)
{
    ++r;
}
```

31 olr

```
- int main()
{
    using namespace std;
    using namespace placeholders;

    int x{ 30 };

    auto fn = bind(foo, ref(x));

    fn();

    cout << "x = " << x << '\n';
}
```

containerlarda reference parametere almak doğrudan mümkün degil ama bu sekilde açılım yaparak ref yapabildik
++myvecler calisir 1deger arttirir

```
vector<reference_wrapper<int>> myvec{ a, b, c };
```

```
++myvec[0];  
++myvec[1];  
++myvec[2];
```

```
cout << "a = " << a << '\n';  
cout << "b = " << b << '\n';  
cout << "c = " << c << '\n';
```



```
int operator()(int a, int b) const
{
    return a * b;
}

int main()
{
    using namespace std;
    using namespace placeholders;

    auto f1 = [](int x, int y) {return x * x + y * y; };
    auto f2 = std::bind(foo, _1, _2, 400);

    //std::function<int(int, int)> fn{ sum };
    //std::function<int(int, int)> fn{ Functor{} };
    //std::function<int(int, int)> fn{ f1 };
    //std::function<int(int, int)> fn{ [](int a, int b) {return a - b; } };
    std::function<int(int, int)> fn{ f2};

    cout << fn(100, 20) << "\n";
}
```

```
class Functor {
public:
    int operator()(int a, int b) const
    {
        return a * b;
    }
};

using myfn = std::function<int(int, int)>;

int main()
{
    using namespace std;
    using namespace placeholders;

    vector < function<int(int, int)>> myvec;

    myvec.push_back(sum);
    myvec.push_back(Functor{});
    myvec.push_back([](int a, int b) {return a * b - 1})
}
```

```
#include <random>
#include <algorithm>
#include <vector>
#include <fstream>
#include <iostream>
#include <functional>
#include "nutility.h"
#include <list>

int foo(int, int);

int main()
{
    using namespace std;
    function<int(int, int)> f;
}
```

```
[-] class MyClass {
public:
    int bar(int);
    int foo(int);
    int baz(int);
};

using mftype = int(MyClass::*)(int);

[-] int main()
{
    //int(MyClass:: * fa[3])(int) = {
    mftype fa[3] = {
        &MyClass::bar,
        &MyClass::foo,
        &MyClass::baz,
    };
}
```

```
    return x;
}

};

int main()
{
    using namespace std;

    Myclass m;

    cout << "&m = " << &m << '\n';

    auto fp = &Myclass::foo;

    (m.*fp)(34);
}
```

noktayldz bir operatördür ve
overload edilemez

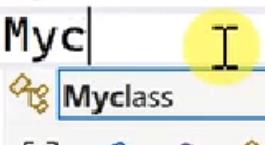
```
7      using namespace std;
8
9      Myclass m;
10
11     cout << "&m = " << &m << '\n';
12
13     auto fp = &Myclasous::foo;
14
15     (m.*fp)(34);
16
17     Myclass* p{ &m };
18
19     //((*p).*fp)(45);
20     (p->*fp)(56);
21 }
```

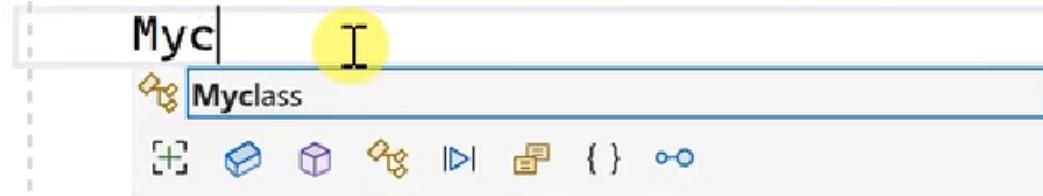
I

verbose saylr invoke geliy

```
int foo(int x)
{
    std::puts("int Myclass::foo(int x)");
    std::printf("x = %d\n", x);
    std::cout << "this = " << this << "\n";
    return x;
}
```

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

    Myclass m;
    Myc| I
    
    { } o-
```



```
class MyClass {
public:
    int foo(int x)
    {
        std::puts("int MyClass::foo(int x)");
        std::printf("x = %d\n", x);
        std::cout << "this = " << this << "\n";
        return x;
    }
};

int main()
{
    using namespace std;

    MyClass m;

    cout << "&m = " << &m << '\n';
    invoke(&MyClass::foo, m, 12);
}
```

```
int foo(int x);  
int bar(int x);  
int baz(int x);  
int func(int x);  
};
```

temaya bak

```
void g(Myclass m, int val, int (Myclass::*fp)(int))  
{  
    //  
    (m.*fp)(val);  
}
```

```
int main()  
{  
    Myclass mx;  
  
    g(mx, 20, &Myclass::foo);  
    g(mx, 20, &Myclass::bar);  
    g(mx, 20, &Myclass::baz);  
}
```

```
int main()
{
    Myclass mx;

    using ftype = int(Myclass::*)(int);

    ftype ar[] = {
        &Myclass::foo,
        &Myclass::bar,
        &Myclass::baz,
    };
    for (auto fp : ar) {
        (mx.*fp)(12);
        std::invoke(fp, mx, 13);
    }
}
```

```
#include <functional>

struct Point {
    int mx, my, mz;
};

int main()
{
    using namespace std;

    Point p = { 35 ,56, 90 };

    //&p.mx

    //auto dptr = &Point::mx;
    int Point::* dptr = &Point::mx;

    cout << p.*dptr << "\n";
}
```

```
#include <functional>

struct Point {
    int mx, my, mz;
};

int main()
{
    using namespace std;
    Point p = { 35 ,56, 90 };
    //&p.mx

    //auto dptr = &Point::mx;
    int Point::* dptr = &Point::mx;

    cout << p.*dptr << "\n";
    p.*dptr = 90;
    cout << "p.mx = " << p.mx << '\n';
    invoke(dptr, p) = 888;
```

I

```
#include <functional>
```

```
struct Point {  
    int mx, my, mz;  
};
```

```
void func(Point& p, int Point::*mp)  
{  
    p.*mp = 555;  
    //  
}
```

```
int main()  
{  
    Point mypoint{ 4, 7, 9 };  
  
    func(mypoint, &Point::mx);  
    func(mypoint, &Point::my);  
    func(mypoint, &Point::mz);  
}
```

I

lojikdegili

```
bool is_even(int x)
{
    return x % 2 == 0;
}

int main()
{
    using namespace std;

    int ival{ 34546 };

    cout << boolalpha;

    cout << is_even(ival) << " " << is_even(ival + 1) << "\n";

    auto fn = not_fn(is_even);

    cout << fn(ival) << " " << fn(ival + 1) << "\n";
```

move ctor çarır

```
using namespace std;  
  
vector<Myclass> myvec(100);  
move_iterator<vector<Myclass>::iterator> iter{ myvec.begin() };  
  
Myclass x = *iter;
```



bunun da kolay hali var

```
template <typename Iter>
std::move_iterator<Iter> MakeMoveIterator(Iter it)
{
    return std::move_iterator<Iter>{it};
}
```



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

    vector<Myclass> myvec(100);

    auto iter = MakeMoveIterator(myvec.begin());
}
```

✓ No issues found

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

    vector<Myclass> myvec(100);

    auto s = *make_move_iterator(myvec.begin());
}

I
```

No issues found

0 Errors 0 Warnings 0 Messages 0.7 Build Only

move assignment cagrilir

```
};

int main()
{
    using namespace std;

    vector<Myclass> x(10);
    vector<Myclass> y(10);

    copy(make_move_iterator(x.begin()), make_move_iterator(x.end()), y.begin());
}
```

I

No issues found

bu std::move gelen move

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

    vector<Myclass> x(10);
    vector<Myclass> y(10);

    std::move(x.begin(), x.end(), y.begin());
}
```

I

om: Build



Functions (3.3%) were compiled, the rest were copied from previous compilation.
ions were new in current compilation
ions had inline decision re-evaluated but remain unchanged
generating code

12 13

```
using namespace std;

int main()
{
    tuple tx{ 12, 4.5, "mustafa"s };

    cout << get<0>(tx) << "\n";
    get<0>(tx)++;
    cout << get<0>(tx) << "\n";
}
```

I

No issues found

0 Errors 0 Warnings 0 Messages 0/7 Build Only

```
template <typename ...Args>
auto MakeTuple(Args&& ...args)
{
    return std::tuple(std::forward<Args>(args)...);
}

int main()
{
    auto tp = MakeTuple(12, 6.76, Date{ 5, 6, 2023 });
}
```

I

✖ 1 ⚠ 0 ↑ ↓



```
#include <tuple>

using namespace std;

auto foo()
{
    //
    return tuple{ 12, Date{2, 4, 1987}, string{"yilmaz savaskan" } };
}

int main()
{
    auto [age, bdate, name] = foo();
}
```

I

using Person= tuple int

```
int main()
{
    vector<Person> pvec;

    for (int i = 0; i < 10000; ++i) {
        pvec.emplace_back(Irand{ 5, 70 }(), rname(), Date::random());
    }

    std::ofstream ofs{ "out.txt" };
    if (!ofs) {
        std::cerr << "out.txt dosyasi olusturulamadi\n";
        exit(EXIT_FAILURE);
    }

    ofs << left;

    for (const auto& tp : pvec) {
        ofs << setw(8) << get<0>(tp) << "    " << setw(16) << get<1>(tp) << "    " << get<2>(tp) << "\n";
    }
}
```

I

```
int main()
{
    vector<Person> pvec;

    for (int i = 0; i < 100'000; ++i) {
        pvec.emplace_back(Irand{ 5, 70 }(), rname(), Date::random());
    }

    std::ofstream ofs("out.txt");
    if (!ofs) {
        std::cerr << "out.txt dosyasi olusturulamadi\n";
        exit(EXIT_FAILURE);
    }

    ofs << left;

    sort(pvec.begin(), pvec.end(), greater{});
    for (const auto& [age, name, bdate] : pvec) {
        ofs << setw(8) << age << "    " << setw(16) <<
            name << "    " << bdate << "\n";
    }
}
```

```
tuple<int, string, Date> foo()
{
    return { 23, "metin", Date{12, 5, 1987} };
}

int main()
{
    int x = 23;
    string str{ "tekin" };
    Date bdate{ 3, 5, 1999 };

    tie(x, str, bdate) = foo();

    //tuple<int&, string&, Date&>(x, str, bdate) = foo();
}
```

I

```
bool operator<(const Date& d1, const Date& d2)
{
    /*return d1.y_ != d2.y_ ? d1.y_ < d2.y_ :
     *      d1.m_ != d2.m_ ? d1.m_ < d2.m_ :
     *      d1.d_ < d2.d_ ;*/
    return tuple(d1.y_, d1.m_, d1.d_) < tuple(d2.y_, d2.m_, d2.d_);
}
```

I

```
void print(int x, int y, int z)
{
    std::cout << x << " " << y << " " << z << "\n"
}
```

I

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

    tuple tx{ 4, 6, 9 };

    apply(print, tx);
    print(get<0>(tx), get<1>(tx), get<2>(tx));
}
```

```
void make_double(int& x, int& y, int& z)
{
    x *= 2;
    y *= 2;
    z *= 2;
}

int main()
{
    using namespace std;

    tuple tx{ 4, 6, 9 };

    apply(make_double, tx);
    cout << get<0>(tx) << "\n";
    cout << get<1>(tx) << "\n";
    cout << get<2>(tx) << "\n";
}
```

```
std::ostream& operator<<(std::ostream& os, const std::tuple<Ts...> & tx)
{
    std::apply
    (
        [&os](Ts const& ...tupleArgs)
        {
            os << '[';
            std::size_t n{ 0 };
            ((os << tupleArgs << (++n != sizeof...(Ts) ? ", " : "")), ...);
            os << ']';
        }, tx
    );
    return os;
}

int main()
{
    using namespace std;

    cout << tuple{ 43, 5.6, "necati", 'A', Date{1, 4, 1987} } << '\n';
}
```

```
void* operator new(std::size_t sz)
{
    std::cout << "operator new called sz = " << sz << "\n";

    void* vp = std::malloc(sz);
    if (!vp) {
        std::bad_alloc{};
    }

    std::cout << "the address of allocated block is : " << vp << "\n";
    return vp;
}

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

    std::free(vp);
}
```

```
int main()
{
    auto p = new Myclass;

    std::cout << "nesne delete edilecek\n";
    delete p;
    std::cout << "main devam ediyor\n";
```

```
1
2
3
4
5
6 int main()
7 {
8     using namespace std;
9
10    std::cout << "sizeof(Myclass) = " << sizeof(Myclass) << "\n";
11    size_t n;
12
13    std::cout << "kac nesne: ";
14    cin >> n;
15
16    Myclass* p = new Myclass[n];
17
18    delete[] p;
19
20 }
```



```
int main()
{
    //const Myclass* p = new const Myclass;
    auto p = new const Myclass;
    p (local variable) const Myclass *p
    const Myclass* p = new const Myclass;
    Search Online
```



C++ dilinde "placement new" özelliği, bellek yönetimi ve nesne oluşturma işlemlerinde kullanılan bir tekniktir. Normalde C++ dilinde "new" operatörü kullanılarak bellekten bir nesne oluşturulur ve bu nesnenin ömrü sona erdiğinde "delete" operatörü kullanılarak bellekten geri alınır. Ancak, "placement new" kullanıldığında, önceden ayrılmış bir bellek bloğunda bir nesne oluşturulur ve bu nesnenin ömrü programcı tarafından yönetilir. "placement new" kullanmanın temel amacı, özel bellek alanlarına nesneleri yerleştirebilmektir.

Yani, "placement new" kullanılarak, belirli bir bellek bölgesine, örneğin bir bellek havuzuna veya belirli bir donanım kaynağına sahip bellek alanlarına nesneler oluşturulabilir. Bu özellik, özelleştirilmiş bellek yönetimi senaryolarında veya düşük seviyeli donanım etkileşimlerinde faydalı olabilir.

"placement new" sözdizimi şu şekildedir:

cpp

Copy code

```
#include <new>

// ...

void* operator new(size_t size, void* ptr);
void operator delete(void* ptr, void* mem);
```

```
class MyClass {
public:
    MyClass(int val) : value(val) {}

    void print() const { std::cout << "Value: " << value << std::endl; }

private:
    int value;
};

int main() {
    // Bellekten 100 byte alan ayrılıyor
    void* memory = ::operator new(100);

    // placement new kullanarak MyClass nesnesini oluşturuyoruz
    MyClass* obj = new (memory) MyClass(42);

    // Nesneyi kullanabiliriz
    obj->print();

    // Nesneyi bellekten elle geri alıyoruz (destructor çağrılmaz, çünkü bel
    obj->-MyClass();

    // Belleği geri veriyoruz
    ::operator delete(memory);

    return 0;
}
```

"placement new" ile oluşturulan nesneler, "delete" operatörü kullanılarak değil, nesnenin destructor'u çağrılarak ve bellekten elle geri alınarak yok edilmelidir. Bu nedenle "placement new" kullanırken nesnelerin yaşam döngüsüne dikkat etmek önemlidir. Ayrıca, "placement new" ile oluşturulan nesnelerin belleği manuel olarak geri verilmelidir.

```
#include <iostream>
```

```
class MyClass {
```

```
    char mbuf[1024 * 1024];
```

```
}
```

ne operatorü throw edecek olursa nullptr
ediyor ve if'e girer

```
int main()
```

```
{
```

```
    using namespace std;
```

```
// newI MyClass
```

```
MyClass* p = new(nothrow) MyClass;
```

```
if (!p) {
```

```
    //...
```

```
}
```

```
class UniquePtr {  
  
public:  
    UniquePtr() = default;  
  
    UniquePtr(T *p) : mp{p} {}  
  
    ~UniquePtr()  
    {  
        if (mp)  
            D{}(mp);  
    }  
  
    T& operator*()  
    {  
        return *mp;  
    }  
  
    T* operator->()  
    {  
        return mp;  
    }  
  
private:  
    T* mp{};
```

uniqueptr asayı yukarı böyle bisi
bir pointer wrapper

```
template <typename T> struct DefaultDelete {
    void operator()(T* p)
    {
        delete p;
    }
};
```

I

```
template <typename T, typename D = DefaultDelete<T>>
class UniquePtr {

public:
    UniquePtr() = default;

    UniquePtr(T *p) : mp{p} {}

    ~UniquePtr()
    {
        if (mp)
            D{}(mp);
    }

    T& operator*()
```

```
#include <memory>
#include <string>
#include "date.h"
#include <iostream>
#include <vector>

int main()
{
    using namespace std;

    unique_ptr<Date> uptr{ new Date{7, 7, 1977} };

    cout << "uptr : " << (uptr ? "dolu" : "bos") << '\n';
    unique_ptr<Date> p = move(uptr);
    cout << "uptr : " << (uptr ? "dolu" : "bos") << '\n';
    cout << "p    : " << (uptr ? "dolu" : "bos") << '\n';

    (void)getchar();
}
```

C:\Users\necat\source\repos\cpp_january\Release

```
uptr : dolu
uptr : bos
p    : bos
```

```

#include <string>
#include "date.h"
#include <iostream>
#include <vector>

int main()
{
    using namespace std;

    unique_ptr<Date> up1{ new Date{7, 7, 1977} };
    unique_ptr<Date> up2{ new Date{8, 8, 1988} };

    cout << "up1 : " << (up1 ? "dolu" : "bos") << '\n';
    cout << "up2 : " << (up2 ? "dolu" : "bos") << '\n';

    cout << *up1 << "\n" << *up2 << "\n";

    up1 = std::move(up2);

    cout << "up1 : " << (up1 ? "dolu" : "bos") << '\n';
    cout << "up2 : " << (up2 ? "dolu" : "bos") << '\n';

    (void)getchar();
}

```

up1 destruct oluyor ve yeniden up2 ile move assignment çağrıiyor up2 nullptr
up1 eski up2'nin verilerini tutuyor

I

```
1 #include <memory>
2 #include <string>
3 #include "date.h"
4 #include <iostream>
5 #include <vector>
6
7
8 int main()
9 {
10     using namespace std;
11
12     unique_ptr<Date> up1{ new Date{7, 7, 1977} };
13     unique_ptr<Date> up2{ new Date{8, 8, 1988} };
14
15
16     vector<unique_ptr<Date>> uvec;
17
18     uvec.reserve(20);
19
20     uvec.push_back(move(up1));
21
22
23
24 }
```

```
1 #include <memory>
2 #include <string>
3 #include "date.h"
4 #include <iostream>
5 #include <vector>
6
7
8
9
10
11 int main()
12 {
13     using namespace std;
14
15     //unique_ptr<Date> up1 = new Date{ 7, 7, 1977 };
16     unique_ptr<Date> up2(new Date{ 7, 7, 1977 });
17
18 }
19
```

ctor explicit oldugu icin 15.
satir syntax hatasi dir

```
1 #include <memory>
2 #include <string>
3 #include "date.h"
4 #include <iostream>
5 #include <vector>
6
7
8 template <typename T, typename ...Args>
9 std::unique_ptr<T> MakeUnique(Args&& ...args)
10 {
11     return std::unique_ptr<T>{new T(std::forward<Args>(args)...)};
12 }
13
14
15
16 int main()
17 {
18     auto uptr = MakeUnique<Date>(3, 5, 1985);
19 }
```

I

sag taraf ref yolluyor efsane

```
4 #include <iostream>
5 #include <vector>
6
7
8     template <typename T, typename ...Args>
9     std::unique_ptr<T> MakeUnique(Args&& ...args)
10    {
11        return std::unique_ptr<T>{new T(std::forward<Args>(args)...)};
12    }
13
14 void func(std::unique_ptr<Date>);
15 std::unique_ptr<Date> foo()
16 {
17
18     return MakeUnique<Date>(3, 5, 1985);
19 }
20
21
22 int main()
23 {
24     func(MakeUnique<Date>(3, 5, 1985));
25
26
27 }
28
```

```
4 | #include <iostream>
5 | #include <vector>
6 |
7 |
8 |
9 | using namespace std;
10|
11|
12 int main()
13 {
14     vector<unique_ptr<Date>> uvec;
15     |
16     uvec.reserve(100);
17
18     uvec.push_back(unique_ptr<Date>{});
19     uvec.push_back(make_unique<Date>(5, 7, 2023));
20     uvec.emplace_back(new Date{ 1 ,5, 1987 });
21
22 }
```

```
'  
8  
9     using namespace std;  
10  
11  
12 int main()  
13 {  
14     auto upx = make_unique<Date>(4, 7, 2012);  
15  
16     cout << "upx = " << (upx ? "dolu" : "bos") << "\n";  
17  
18 //upx.reset();  
19 //upx.reset(nullptr);  
20 //upx = nullptr;  
21 upx = {};  
22  
23 (void)getchar();  
24 cout << "upx = " << (upx ? "dolu" : "bos") << "\n";  
25 }
```

reset ald nesnenin içi bosa dolduruyor.
doluysa önce

```
8
9     using namespace std;
10
11    int main()
12    {
13        auto upx = make_unique<Date>(4, 7, 2012);
14
15        cout << "upx = " << (upx ? "dolu" : "bos") << "\n";
16
17
18        upx.reset(new Date{ 1, 1, 2011 });
19
20
21        (void)getchar();
22        cout << "upx = " << (upx ? "dolu" : "bos") << "\n";
23        cout << "upx = " << (upx ? "dolu" : "bos") << "\n";
24    }
25
```

release degeri

```
8
9     using namespace std;
10    p ye verir dtor cagrilmez
11
12 int main()
13 {
14     auto upx = make_unique<Date>(4, 7, 2012);
15
16     cout << "upx = " << (upx ? "dolu" : "bos") << "\n";
17
18     auto p = upx.release();
19     cout << "upx = " << (upx ? "dolu" : "bos") << "\n";
20
21     delete p;
22 }
```

I

Bir sınıfın veya kütüphanenin `release` yöntemi, genellikle bir kaynağı (bellek bloğu, dosya tanıtıcısı, veritabanı bağlantısı vb.) serbest bırakmak veya ilişkilendirmek için kullanılır. Bu yöntem, kaynağı ilgili sahibinden ayırırken, kaynağı serbest bırakmaz veya yok etmez. Bu, kaynağın başka bir yerde (örneğin, başka bir nesne veya yöntemde) kullanılabilmesini sağlar.

Örneğin, `std::unique_ptr` sınıfı `release` yönteme sahiptir. Bu yöntem, `std::unique_ptr` nesnesinin sahip olduğu kaynağı serbest bırakmadan, bu kaynağı başka bir `std::unique_ptr` nesnesine veya doğrudan bir işaretçiye aktarır. Bu, kaynağın mülkiyetini değiştirirken, bellek sizıntısı veya çift serbest hataları olmadan kaynağın güvenli bir şekilde aktarılmasını sağlar.

İşte `std::unique_ptr` sınıfının `release` yöntemini kullanmanın basit bir örneği:

```
cpp Copy code

#include <memory>

int main() {
    std::unique_ptr<int> ptr(new int(42));

    int* rawPtr = ptr.release(); // ptr'dan kaynağı al, ptr'yi boş hale geti

    // rawPtr Üzerinden kaynağı kullanabilirsin, ancak artık ptr tarafından
    // Bellek sizıntısı önlemek için rawPtr'yi uygun bir şekilde silmelisin

    delete rawPtr;

    return 0;
}
```

Yukarıdaki örnekte, `ptr` adında bir `std::unique_ptr<int>` nesnesi oluşturulur ve bir `int` değerine sahip bir bellek bloğunu (`new int(42)`) işaret eder. `ptr.release()` çağrıları, `ptr`'nin sahip olduğu kaynağı `rawPtr` adlı bir işaretçiye aktarırken, `ptr`'yi boş hale getirir. Bu sayede, `rawPtr` üzerinden kaynağı kullanabilirsiniz, ancak artık `ptr` tarafından serbest bırakılmaz. Bu örnekte, `rawPtr`'yi `delete` anahtar kelimesiyle doğru bir şekilde silmek önemlidir. Aksi takdirde, bellek sizıntısı oluşabilir.

```
int main()
{
    auto upx = make_unique<Date>(4, 7, 2012);

    //if (upx)
    //if (upx != nullptr)
    //if (upx.get())
}

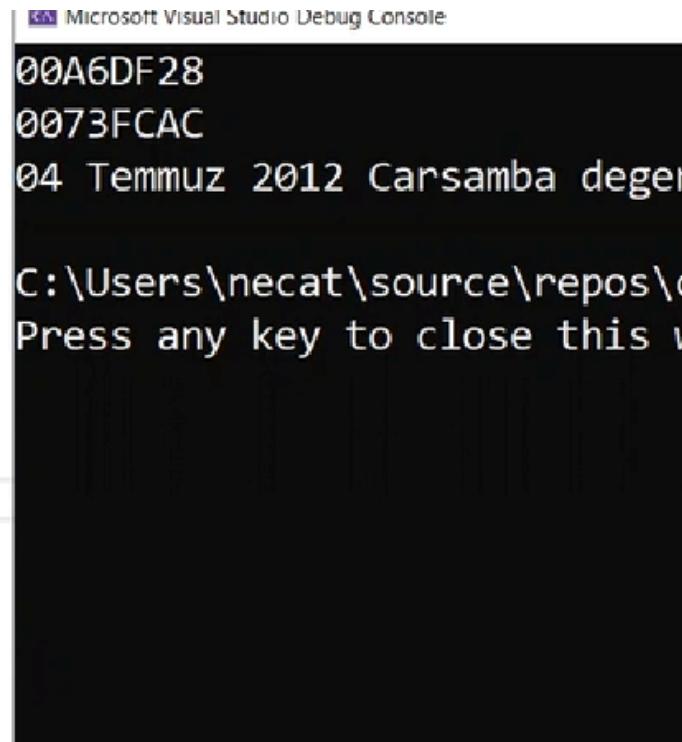
I
```

get olan date'in adresini yazar. ref olan unique

```
using namespace std;

int main()
{
    auto upx = make_unique<Date>(4, 7, 2012);

    cout << upx.get() << "\n";
    cout << &upx << "\n";
}
```



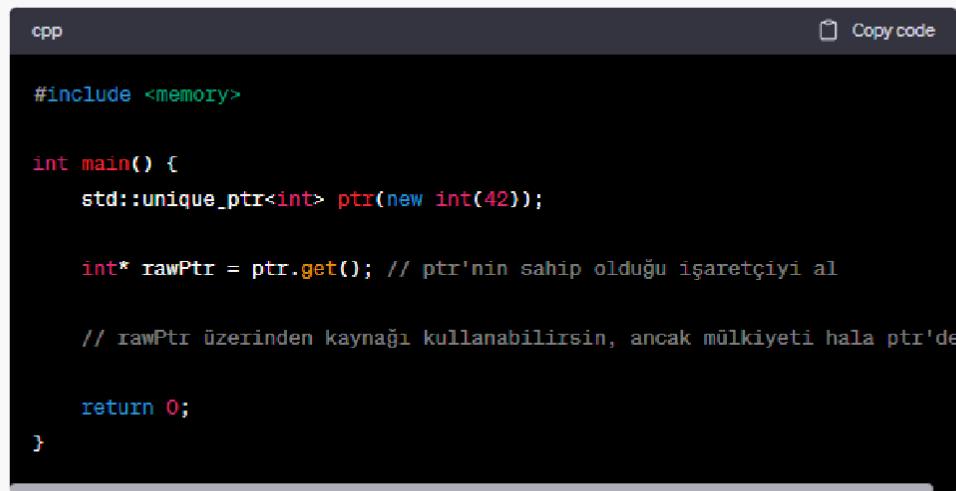
Microsoft Visual Studio Debug Console

00A6DF28
0073FCAC
04 Temmuz 2012 Carsamba deger
C:\Users\necat\source\repos\
Press any key to close this window

C++’ta `std::unique_ptr` sınıfının `get()` fonksiyonu, sahip olduğu kaynagı temsil eden bir işaretçi döndürür. `std::unique_ptr` sınıfı, bellek yönetimi veya kaynak yönetimi gibi durumlarda kullanılan bir akıllı işaretcidir. `get()` fonksiyonu, `std::unique_ptr` nesnesinin sahip olduğu kaynağa doğrudan erişim sağlar.

`get()` fonksiyonu, `std::unique_ptr` nesnesi içinde saklanan işaretçi döndürür. Bu işaretçi, `std::unique_ptr` nesnesinin sahip olduğu kaynağı temsil eden bellek bloğunu veya başka bir kaynağı gösterir. `get()` fonksiyonu, işaretçiyi geri döndürürken, kaynağın mülkiyetini veya yönetimini değiştirmez. Yani, kaynak hala `std::unique_ptr` nesnesi tarafından kontrol edilir.

İşte `get()` fonksiyonunun basit bir örneği:



```
cpp
#include <memory>

int main() {
    std::unique_ptr<int> ptr(new int(42));

    int* rawPtr = ptr.get(); // ptr'nin sahip olduğu işaretçiyi al

    // rawPtr üzerinden kaynağı kullanabilirsin, ancak mülkiyeti hala ptr'de

    return 0;
}
```

Yukarıdaki örnekte, `ptr` adında bir `std::unique_ptr<int>` nesnesi oluşturulur ve bir `int` değerine sahip bir bellek bloğunu (`new int(42)`) işaret eder. `ptr.get()` çağrıSİ, `ptr`’nin sahip olduğu işaretçiyi `rawPtr` adlı bir işaretçiye atar. Bu sayede, `rawPtr` üzerinden kaynağı kullanabilirsiniz, ancak kaynak hala `ptr` tarafından kontrol edilir.

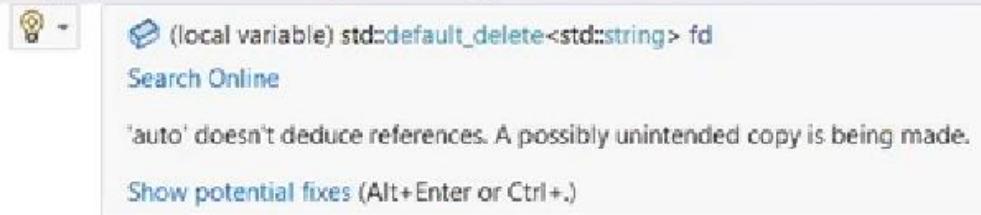
`get()` fonksiyonu genellikle, `std::unique_ptr` nesnesinin işlevsellliğini sınırlamadan işaretçi tabanlı işlemler yapmak veya C tarzı bir API ile etkileşimde bulunmak için kullanılır. Ancak, `get()` fonksiyonunun döndürdüğü işaretçi, kaynağın yaşam döngüsünü kontrol etmez. `std::unique_ptr` nesnesinin ömrünün sona ermesi durumunda, bu işaretçi geçersiz hale gelebilir ve kullanılmamalıdır.

```
[#include <string>

int main()
{
    using namespace std;

    unique_ptr<string> ups;

    auto fd = ups.get_deleter();
```



```
12
13 int main()
14 {
15     using namespace std;
16
17     unique_ptr<string,
18     decltype([](string* p) {
19         cout << "deleter\n";
20         delete p;
21     })>
22
23
24
25 }
```

I

```
#define _CRT_SECURE_NO_WARNINGS
```

```
└#include <memory>
#include <string>
#include <iostream>
#include <cstdio>
```

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

    using file_closer = decltype([](FILE* fp) {fclose(fp); });

    //unique_ptr<FILE, decltype([](FILE* fp) {fclose(fp); }>
    unique_ptr<FILE, file_closer> |
```

```
#define _CRT_SECURE_NO_WARNINGS

#include <memory>
#include <string>
#include <iostream>
#include <cstdio>

struct List {
};

typedef struct List* ListHandle;

ListHandle create_list(void);
void destroy_list(ListHandle);

int main()
{
    using namespace std;

    unique_ptr< List, decltype([](ListHandle p) { destroy_list(p); })> uptr{ create_list() };
}
```

```
1 #define _CRT_SECURE_NO_WARNINGS
2
3
4 #include <memory>
5 #include <string>
6 #include <iostream>
7 #include <cstdio>
8
9
10
11 int main()
12 {
13     using namespace std;
14
15     unique_ptr<string[]> uptr(new string[100]);
16
17     for (int i{}; i < 100; ++i) {
18         uptr[i].I
19     }
20 }
```

```
1 #define _CRT_SECURE_NO_WARNINGS
2
3
4 #include <memory>
5 #include <string>
6 #include <iostream>
7 #include <vector>
8 #include <algorithm>
9
10 int main()
11 {
12     using namespace std;
13
14     vector<unique_ptr<string>> vec;
15
16     vec.emplace_back(new string{ "mustafa aksoy" });
17     vec.emplace_back(new string{ "necati ergin" });
18     vec.emplace_back(new string{ "berkin kara" });
19     vec.emplace_back(new string{ "rukİYE mazlum" });
20     vec.emplace_back(new string{ "abdi korkmaz" });
21
22     sort(vec.begin(), vec.end(),
23          [] (const auto& p1, const auto& p2)
24          {
25              return *p1 < *p2;
26          });
27
28     for (const auto& up : vec)
29         cout << *up << "\n";
30
31 }
```

lambdayi yazmazsan pointerleri siralar

```
1 #define _CRT_SECURE_NO_WARNINGS
2
3
4 #include <memory>
5 #include <string>
6 #include <iostream>
7 #include <vector>
8 #include <algorithm>
9
10
11 using namespace std;
12
13 class Fighter {
14 public:
15     Fighter(string name, int age);
16 }
17
18
19
20 std::unique_ptr<Fighter> make_fighter(string name, int ag
21 {
22     return std::make_unique<Fighter>(name, age);
23 }
24
25
26 int main()
27 {
28     auto up = make_fighter("alican", 56);
```

I

```
1 //include "main.h"
2
3 #include <iostream>
4
5 #include <string>
6 #include <iostream>
7 #include <vector>
8 #include <algorithm>
9 #include "car.h"
10
11
12 //sink
13
14 void sink(std::unique_ptr<std::string> up)
15 {
16
17 }
```

I

pass_through

```
7 #include <vector>
8 #include <algorithm>
9 #include "car.h"
10
11
12 //pass_through
13
14 std::unique_ptr<std::string> foo(std::unique_ptr<std::string> up)
15 {
16     std::cout << "uzunluk: " << up->length() << '\n';
17     std::cout << "yazi : " << *up << "\n";
18     return up;
19 }
20
21
22 int main()
23 {
24     using namespace std;
25
26     {
27         auto up = foo(make_unique<string>("necati ergin"));
28         cout << *up << "\n";
29     }
30
31     std::cout << "main devam ediyor\n";
```

```
13
14     using namespace std;
15
16     shared_ptr<string> sp1{ new string {"necati ergin"} };
17
18     cout << *sp1 << "\n";
19
20     cout << "sp1.use_count() = " << sp1.use_count();
21
22     auto sp2 = sp1;
23     cout << "sp1.use_count() = " << sp1.use_count();
24     cout << "sp2.use_count() = " << sp2.use_count();
25
26     {
27         auto sp3 = sp1;
28         cout << "sp1.use_count() = " << sp1.use_count();
29         cout << "sp2.use_count() = " << sp2.use_count();
30         cout << "sp3.use_count() = " << sp3.use_count();
31
32     (void)getchar();
33     cout << "sp1.use_count() = " << sp1.use_count();
34     cout << "sp2.use_count() = " << sp2.use_count();
35
36
37
```

```
Microsoft Visual Studio Debug Console
necati ergin
sp1.use_count() = 1
sp1.use_count() = 2
sp2.use_count() = 2
sp1.use_count() = 3
sp2.use_count() = 3
sp3.use_count() = 3
sp1.use_count() = 2
sp2.use_count() = 2
C:\Users\necat\source\repos\Shared\Debug\Shared.exe - 1 error(s), 0 warning(s)
Press any key to close this window.
```

```
4 #include <memory>
5 #include <string>
6 #include <iostream>
7 #include <vector>
8 #include <algorithm>
9 #include "date.h"
10
11
12 int main()
13 {
14     using namespace std;
15
16     shared_ptr<string> sp1{ new string {"necati ergin"} };
17
18     cout << *sp1 << "\n";
19
20     cout << "sp1.use_count() = " << sp1.use_count() <<
21
22     auto sp2 = sp1;
23     cout << "sp1.use_count() = " << sp1.use_count() <<
24     cout << "sp2.use_count() = " << sp2.use_count() <<
25
26     sp1.reset();
27     (void)getchar();
28     cout << "sp1.use_count() = " << sp1.use_count() <<
29     cout << "sp2.use_count() = " << sp2.use_count() <<
30
31
32 }
33
```

Microsoft Visual Studio Debug Console

```
necati ergin
sp1.use_count() = 1
sp1.use_count() = 2
sp2.use_count() = 2
sp1.use_count() = 0
sp2.use_count() = 1
```

C:\Users\necat\source\repos\cpp_january\|D
Press any key to close this window . . .

```
#define _CRT_SECURE_NO_WARNINGS

#include <memory>
#include <string>
#include <iostream>
#include <vector>
#include <algorithm>
#include "date.h"

int main()
{
    using namespace std;

    {
        shared_ptr<string> sp1(new string{ "alican" }, [](string* p) {cout << "custom deleter\n"; delete p; });
    }

    cout << "main devam ediyor\n";
}
```

```
4 #include <vector>
5 #include <iostream>
6 #include <algorithm>
7
8 using namespace std;
9
10 int main()
11 {
12     auto sp = make_shared<Date>(14, 6, 2023);
13
14     cout << *sp << "\n";
15
16     cout << "sp.use_count() = " << sp.use_count() << '\n';
17
18     weak_ptr<Date> wp(sp);
19
20     cout << "sp.use_count() = " << sp.use_count() << '\n';
21
22     if (wp.expired())
23         std::cout << "kaynak hayatta degil\n";
24     else
25         std::cout << "kaynak hayatta\n";
26
```

```
7
8     using namespace std;
9
10    int main()
11    {
12        auto sp = make_shared<Date>(14, 6, 2023);
13
14        cout << *sp << "\n";
15
16        cout << "sp.use_count() = " << sp.use_count() << '\n';
17
18        weak_ptr<Date> wp(sp);
19
20        cout << "sp.use_count() = " << sp.use_count() << '\n';
21
22        wp.reset();
23        cout << "sp.use_count() = " << sp.use_count() << '\n';
24
25        if (wp.expired())
26            std::cout << "kaynak hayatta degil\n";
27        else
28            std::cout << "kaynak hayatta\n";
29
```

```
4 #include <vector>
5 #include <iostream>
6 #include <algorithm>
7
8 using namespace std;
9
10 int main()
11 {
12     auto sp = make_shared<Date>(14, 6, 2023);
13
14     cout << *sp << "\n";           I
15
16     cout << "sp.use_count() = " << sp.use_count() << "\n";
17
18     weak_ptr<Date> wp(sp);
19
20     cout << "sp.use_count() = " << sp.use_count() << "\n";
21
22     if (auto spx = wp.lock()) {
23         std::cout << "kaynak halen hayatta\n";
24         cout << "sp.use_count() = " << sp.use_count();
25         cout << *spx << "\n";
26         cout << *sp << "\n";
27     }
28     else {
29         std::cout << "kaynak hayatta degil!\n";
30     }
31
32
33
34
```

No issues found

```
C:\Users\necati\source\repos\cpp_january\Release\cpp_january.exe
14 Haziran 2023 Carsamba
sp.use_count() = 1
sp.use_count() = 1
kaynak halen hayatta
sp.use_count() = 2
14 Haziran 2023 Carsamba
14 Haziran 2023 Carsamba
```

```
4 #include <vector>
5 #include <iostream>
6 #include <algorithm>
7
8 using namespace std;
9
10 int main()
11 {
12     auto sp = make_shared<Date>(14, 6, 2023);
13
14     cout << *sp << "\n";           I
15
16     cout << "sp.use_count() = " << sp.use_count() << "\n";
17
18     weak_ptr<Date> wp(sp);
19
20     cout << "sp.use_count() = " << sp.use_count() << "\n";
21
22     if (auto spx = wp.lock()) {
23         std::cout << "kaynak halen hayatta\n";
24         cout << "sp.use_count() = " << sp.use_count();
25         cout << *spx << "\n";
26         cout << *sp << "\n";
27     }
28     else {
29         std::cout << "kaynak hayatta degil!\n";
30     }
31
32
33
34
```

No issues found

```
C:\Users\necati\source\repos\cpp_january\Release\cpp_january.exe
14 Haziran 2023 Carsamba
sp.use_count() = 1
sp.use_count() = 1
kaynak halen hayatta
sp.use_count() = 2
14 Haziran 2023 Carsamba
14 Haziran 2023 Carsamba
```

[weakptr nin ok operatörü vb yok expired ve look var](#)

`'std::weak_ptr'` C++'taki bir akıllı işaretçi sınıfıdır ve C++11 standarıyla birlikte tanıtılmıştır. `'std::weak_ptr'` zayıf bir sahiplik ilişkisi sağlar ve bir nesnenin paylaştırılan sahipliğini takip etmek için kullanılır.

Bir `'std::weak_ptr'`, bir `'std::shared_ptr'`'a (paylaştırılan işaretçi) bağlı olabilir, ancak kaynak nesneyi (hedef nesneyi) sahiplenmez. Bir `'std::weak_ptr'`'ın ömrü, hedef nesne ömrünü uzatmaz. Bunun yerine, hedef nesnenin ömrü boyunca mevcut olup olmadığını kontrol etmek için kullanılır.

Bir `'std::weak_ptr'`'ın kullanımı, döngüsel referansları önlemek veya hedef nesnenin ömrünün sona erip ermediğini kontrol etmek gibi senaryolarda yaygındır.

`'std::weak_ptr'`'lar, örneğin `'std::shared_ptr'`'ların ömrü bittiğinde hedef nesnenin hala geçerli olup olmadığını kontrol etmek için kullanılabilir.

`'std::weak_ptr'`'lar, `'std::shared_ptr'`'dan oluşturulabilir veya bir `'std::shared_ptr'`'a dönüştürülebilir. Kullanılmak istendiğinde `'std::weak_ptr'`'lar, `'std::shared_ptr'`'a dönüştürülerek hedef nesneye erişim sağlayabilir.

Özetle, `'std::weak_ptr'` C++'taki bir akıllı işaretçi sınıfıdır ve paylaştırılan bir kaynak nesnenin sahipliğini takip etmek veya döngüsel referansları önlemek için kullanılır.



```
class Dog {
    std::weak_ptr<Dog> mp_friend;
    std::string m_name;
public:
    Dog(std::string name) : m_name(std::move(name))
    {
        std::cout << "kopek " << m_name << " olusturuldu" << std::endl;
    }

    void print()const
    {
        std::cout << "benim adim " << m_name << "\n";
        if (!mp_friend.expired())
        {
            std::cout << "benim bir arkadasim var ismi " << mp_friend.lock()->m_name << "\n";
        }
        else {
            std::cout << "benim bir arkadasim yok\n";
        }
    }

    bool has_friend()const
    {
        return !mp_friend.expired();
    }

    ~Dog()
    {
        std::cout << "kopek " << m_name << " oyundan cikiyor" << std::endl;
    }

    void bark()
    {
        cout << m_name << " havliyor" << endl;
    }

    void make_friend(shared_ptr<Dog> x)
    {
        mp_friend = x;
        cout << m_name << " " << x->m_name << " ile arkadas oldu" << endl;
    }
}
```

```
template <typename D>
class Base {
public:
    void foo()
    {
        static_cast<D*>(this)->print();
    }
};

class Der : public Base<Der> {

public:
    void print()const
    {
        std::cout << "Der\n";
    }
};

class Mer : public Base<Mer> {

public:
    void print()const
    {
        std::cout << "Mer\n";
    }
};
```

```
#include <iostream>
```

crtp

```
template <typename D>
class Base {
public:
    bool is_greater(const Base& other) const
    {
        return !static_cast<const D &>(*this).operator<(other)
    }
};

class Der : Base<Der> {
```

// CRTP

```
template<typename T>
class Animal {
public:
    void cry()
    {
        static_cast<T*>(this)->make_sound();
    }
};

class Dog : public Animal<Dog> {
public:
    void make_sound()
    {
        std::cout << "hav hav hav!!!\n";
    }
};

class Cat : public Animal<Cat> {
public:
    void make_sound()
    {
        std::cout << "miyav miyav miyav!!!\n";
    }
};

class Lamb : public Animal<Lamb> {
public:
    void make_sound()
    {
        std::cout << "mooooooooooooo!!!\n";
    }
};
```

```
    std::cout << "hav hav hav!!!\n";
}

};

class Cat : public Animal<Cat> {
public:
    void make_sound()
    {
        std::cout << "miyav miyav miyav!!!\n";
    }
};

class Lamb : public Animal<Lamb> {
public:
    void make_sound()
    {
        std::cout << "meeeeeeeeeee!!!\n";
    }
};

int main()
{
    Dog karabas;
    Cat minnos;
    Lamb kuzucuk;

    karabas.cry();
    minnos.cry();
    kuzucuk.cry();
}
```

```
// Eğer bir sınıfın üye fonksiyonu içinde shared_ptr ile hayatı kontrol edilen * this nesnesini gösteren
// shared_ptr'nin kopyasını çikartmak isterseniz sınıfınızı CRTP örüntüsü ile kalıtım yoluyla std::enable_shared_from_this
// sınıfından elde etmelisiniz

#include <memory>
#include <iostream>

using namespace std;

class Neco : public std::enable_shared_from_this<Neco> { //CRTP
public:
    Neco()
    {
        std::cout << "Neco ctor this : " << this << "\n";
    }

    void func()
    {

        //std::cout << "Neco::func() islevi : " << this << "\n";
        ///ben func islevinin bir shared_ptr ile kontrol edilen dinamik Neco nesnesi icin cagrildigina eminim
        //auto sptr = shared_from_this();
        //std::cout << "sptr.use_count() = " << sptr.use_count() << "\n";
    }

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

int main()
{
    auto sp = make_shared<Neco>();
    sp->func();

    //Neco *p = new Neco();
}
```

```
6
7 int main()
8 {
9     using namespace std;
10
11    bitset<32> bx{87345u};
12    bitset<32> by{803459345u};
13
14    cout << bx << "\n";
15    cout << by << "\n";
16    cout << (bx | by) << "\n";
17
18
19
20
21
22
```

I

```
5
6 int main()
7 {
8     using namespace std;
9
10
11     bitset<32> bs{ "11010101001"};
12
13     auto ux = bs.to_ulong();
14
15     cout << ux << "\n";
16
17
18
19 }
```

I

```
using namespace std;

auto fcmp = [] (bitset<16> x, bitset<16> y) {
    return x.to_ulong() < y.to_ulong();
};

set<bitset<16>, decltype(fcmp)> myset;

myset.insert(8712);
myset.insert(3456);
myset.insert(112);
myset.insert(341);
myset.insert(82);
myset.insert(90);

for (auto bs : myset) {
    cout << bs << "\n";
}
```

bitset için kendi sıralama
algoritmamizi yazdik

I

```
1 #include <iostream>
2 #include <bitset>
3 #include <algorithm>
4 #include <set>
5 #include <vector>
6 #include <string>
7 #include "nutility.h"
8 #include <list>
9
10
11 int main()
12 {
13     using namespace std;
14
15     vector<int> ivec;
16     list<int> ilist;
17
18     rfill(ivec, 20, Irand{ 0, 50 });
19     rfill(ilist, 20, Irand{ 0, 50 });
20
21     sort(ivec.begin(), ivec.end());
22     ilist.sort();
23
24     print(ivec);
25     print(ilist);
26
27     set_intersection(ivec.begin(), ivec.end(), ilist.begin(), ilist.end(), ostream_iterator<int>(cout, " "));
28
29
30
31 }
```

2 containerdeda olan verileri 1er boslukla bastirir

I

```
1 #include <iostream>
2 #include <bitset>
3 #include <algorithm>
4 #include <set>
5 #include <vector>
6 #include <string>
7 #include "nutility.h"
8 #include <list>
9
10
11 int main()
12 {
13     using namespace std;
14
15     vector<int> ivec;
16     list<int> ilist;
17
18     rfill(ivec, 20, Irand{ 0, 50 });
19     rfill(ilist, 20, Irand{ 0, 50 });
20
21     sort(ivec.begin(), ivec.end());
22     ilist.sort();
23
24     print(ivec);
25     print(ilist);
26
27 //set_intersection(ivec.begin(), ivec.end(), ilist.begin(), ilist.end(), ostream_iterator<int>(cout, " "));
28 set_union(ivec.begin(), ivec.end(), ilist.begin(), ilist.end(), ostream_iterator<int>(cout, " "));
29
30
31 }
32 }
```

```
int main()
{
    vector<Person> pvec;

    for (int i = 0; i < 100'000; ++i) {
        pvec.emplace_back(Irand{ 5, 70 }(), rname(), Date::random());
    }

    std::ofstream ofs("out.txt");
    if (!ofs) {
        std::cerr << "out.txt dosyasi olusturulamadi\n";
        exit(EXIT_FAILURE);
    }

    ofs << left;

    sort(pvec.begin(), pvec.end(), greater{});
    for (const auto& [age, name, bdate] : pvec) {
        ofs << setw(8) << age << "    " << setw(16) <<
            name << "    " << bdate << "\n";
    }
}
```

```
template <typename T, typename U> // Provide sample template arguments for Intellisense ✓
std::ostream& operator<<(std::ostream& os, const std::pair<T, U>& p)
{
    return os << '[' << p.first << ", " << p.second << ']';
}
```

bastırılıyor. union ise birleşim kümesi
set_symmetricmatis var ondada a'nın ve b'nin birbirleriyle
kesişimlerinin birleşimi alınıyor

```
int a[3] = { 1, 2, 3 };

auto foo() -> int (*)[3]
{
    return &a;
}
```

böylece Person
per{sinan,kocatürk};
auto x [name,surname]
=per; kullanımda

```
template <typename T, typename U> // Provide sample template arguments for Intellisense - /
std::ostream& operator<<(std::ostream& os, const std::pair<T, U>& p)
{
    return os << '[' << p.first << ", " << p.second << ']';
}
```

kendi
structure
bindingimizi
yazdik Person
sinifi için

```
template <>
struct std::tuple_element<0, Person> {
    using type = std::string;
};

template <>
struct std::tuple_element<1, Person> {
    using type = std::string;
};

template <>
struct std::tuple_size<Person> {
    constexpr static std::size_t value = 2;
};

template <std::size_t n>
auto get(const Person &p)
{
    if constexpr (n == 0)
        return p.get_name();
    else if constexpr (n == 1)
        return p.get_surname();
}

int main()
```

No issues found

```
5 |#include <type_traits>
6 |
7 |
8 |
9 |int a[3] = { 1, 2, 3 };
10 |
11 |
12 |
13 |auto foo() -> int (*)[3]
14 |
15 |{
16 |    return &a;
17 |}
```

```
4 int main()
5 {
6     int x = 5;
7     int* ptr{ &x };
8
9     // prvalue    ==> T
10    // lvalue    ==> T&
11    // xvalue    ==> T&&
12
13    decltype(*ptr)
14 }
15 }
```

```
3
4 int main()
5 {
6     int x = 5;
7     int* ptr{ &x };

8
9     // prvalue    ==> T
10    // lvalue     ==> T&
11    // xvalue     ==> T&&

12
13 decltype(*ptr)
14
15 }
```

decltypein operandı olan ifade pr value olduğunda t
türünü l value olduğunda tref türünü x value
olduğunda trefref değerini ediniyorz

09:58 ✓

```
3  
4 int main() I  
5 {  
6     int y = 10;  
7  
8     decltype(auto) x = (y);  
9  
10    x  
11 }  
12 }
```

x in türü int & y parantez içinde olmasa int



`'decltype(auto)'` C++14'te tanıtılan bir dil özelliğidir. Bu özellik, bir değişkenin türünü bir ifadeyi değerlendirderek otomatik olarak çıkarmak için kullanılır. `'auto'` anahtar kelimesiyle benzer bir şekilde çalışır, ancak `'auto'`'dan farklı olarak referans ve `'const'` niteliklerini de korur.



`'decltype(auto)'` ifadesi, genellikle bir ifadeyi geri döndüren ve dönüş türünü otomatik olarak belirlemek istediğiniz durumlarda kullanılır. Örneğin, aşağıdaki gibi bir fonksiyon düşünelim:

cpp

Copy code

```
decltype(auto) myFunction() {
    // ...
}
```

Bu durumda, `'decltype(auto)'` ifadesi, fonksiyonun içindeki ifadeyi değerlendirderek geri dönüş türünü otomatik olarak belirleyecektir. Eğer ifade bir lvalue ise, geri dönüş türü de bir lvalue referansı olacaktır. Eğer ifade bir rvalue ise, geri dönüş türü de bir rvalue referansı olacaktır.

Örneğin:

cpp

Copy code

```
int x = 5;
decltype(auto) y = x; // y, int& türünde olur

int foo();
decltype(auto) z = foo(); // z, int&& türünde olur
```

`'decltype(auto)'` ifadesi, özellikle template programlamada, kodun daha genel ve esnek olmasını sağlamak için kullanılabilir. Çünkü `'decltype(auto)'` ifadesi, ifadeyi değerlendirderek türü çıkardığı için, ifade ne olursa olsun otomatik olarak uygun bir tür elde edilebilir.



C++ dilinde bir fonksiyonun `'auto'` ve `'decltype(auto)'` ile dönüş değerini belirlemesi arasında bazı farklar vardır. İşte bu iki yaklaşım arasındaki temel farkları:



1. **Tür Çıkarımı:** `'auto'` ile bir fonksiyon dönüş değeri belirlendiğinde, derleyici fonksiyonun geri dönüş ifadesinden türü çıkarır. Yani, geri dönüş ifadesi ne ise, o türü otomatik olarak belirler. Öte yandan, `'decltype(auto)'` kullanıldığında, fonksiyonun geri dönüş türü, dönüş ifadesinin türünü yansıtır. `'decltype(auto)'` ifadesi, ifadeyi değerlendirerek geri dönüş türünü belirler.
2. **Referans ve Const Nitelikleri:** `'auto'` ile dönüş değeri belirlendiğinde, referans ve const nitelikleri otomatik olarak çıkarılır. Yani, eğer geri dönüş ifadesi bir lvalue ise, `'auto'` tarafından belirlenen tür lvalue referansı olmaz. Benzer şekilde, eğer geri dönüş ifadesi const bir nesne ise, `'auto'` tarafından belirlenen tür const niteliğini yitirir. Bununla birlikte, `'decltype(auto)'` ifadesi, dönüş ifadesinin referans ve const niteliklerini korur.

Örneklerle açıklamak gerekirse:

```
cpp Copy code

int foo();
auto bar() {
    return foo(); // Geri dönüş türü int
}

int foo();
decltype(auto) baz() {
    return foo(); // Geri dönüş türü int&
}

const int x = 42;
auto qux() {
    return x; // Geri dönüş türü int
}

const int x = 42;
decltype(auto) quux() {
    return x; // Geri dönüş türü const int&
}
```

tarafından belirlenir.

- Referans ve Const Nitelikleri: `auto` ile dönüş değerini belirlendiğinde, referans ve const nitelikleri otomatik olarak çıkarılır. Yani, eğer geri dönüş ifadesi bir lvalue ise, `auto` tarafından belirlenen tür lvalue referansı olmaz. Benzer şekilde, eğer geri dönüş ifadesi const bir nesne ise, `auto` tarafından belirlenen tür const niteliğini yitirir. Bununla birlikte, `decltype(auto)` ifadesi, dönüş ifadesinin referans ve const niteliklerini korur.

Örneklerle açıklamak gereklidir:

```
cpp Copy code

int foo();
auto bar() {
    return foo(); // Geri dönüş türü int
}

int foo();
decltype(auto) baz() {
    return foo(); // Geri dönüş türü int&
}

const int x = 42;
auto qux() {
    return x; // Geri dönüş türü int
}

const int x = 42;
decltype(auto) quux() {
    return x; // Geri dönüş türü const int&
}
```

Yukarıdaki örnekte, `bar()` fonksiyonunun geri dönüş türü `int` olurken, `baz()` fonksiyonunun geri dönüş türü `int&` olacaktır. Aynı şekilde, `qux()` fonksiyonunun geri dönüş türü `int` olurken, `quux()` fonksiyonunun geri dönüş türü `const int&` olacaktır.

Bu farklar, fonksiyonların geri dönüş değerini belirlerken referans ve const niteliklerinin korunması veya çıkarılması gerekiği durumlarda `decltype(auto)`'nın kullanımını gerektirebilir.

```
#include <type_traits>
```

primitive return type

```
int a[3] = { 1, 2, 3 };
```

l

```
auto foo() -> int (*)[3]
```

```
{
```

```
    return &a;
```

```
}
```

```
3  
4 int main()  
5 {
```

```
6     int x = 5;  
7     int* ptr{ &x };
```

```
8  
9     // prvalue    ==> T  
10    // lvalue     ==> T&  
11    // xvalue     ==> T&&
```

```
12  
13     decltype(*ptr)
```

```
14  
15 }
```

decltypein operandi olan ifade pr
value oldugunda t türünü l value
oldugunda tref türünü x value
oldugunda Trefref degerini
ediniyoruz

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

    vector<string> svec;
    rfill(svec, 20, rname);

    for (const auto& s : svec) {
        cout << s << ' ';
    }

    std::ofstream ofs{ "out.txt" };
    if (!ofs) {
        std::cerr << "out.txt dosyasi olusturulamadi\n";
        exit(EXIT_FAILURE);
    }

    for (const auto& s : svec) {
        ofs << s << ' ';
    }

    ostringstream os;

    for (const auto& s : svec) {
        os << s << ' ';
    }

    std::cout << "\n";

    string str = os.str();
    reverse(str.begin(), str.end());
    //ranges::reverse(str); (C++20 ranges)
    cout << str << '\n';
}
```

```
class Ostream {  
  
public:  
    Ostream& operator<<(int);  
    Ostream& operator<<(double);  
    Ostream& operator<<(long);  
    Ostream& operator<<(void *);  
    Ostream& operator<<(Ostream&(*fp)(Ostream&))  
    {  
        return fp(*this);  
    }  
}  
  
Ostream& Endl(Ostream &os)  
{  
    os.put('\n');  
    os.flush()  
    return os;  
}  
  
cout.operator<<(&func);  
cout << func;  
cout << Endl  
    I  
cout.operator<<(endl);  
cout << x
```

```
8
9
10 std::ostream& dl(std::ostream& os)
11 {
12     return os << "\n-----\n";
13 }
14
15
16
17 int main()
18 {
19     using namespace std;
20
21     int x{ 83745 };
22     double d{ 47.834 };
23     string name{ "uygar pinar" };
24
25     cout << x << dl << d << dl << name << dl;
26
27 }
```

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

    int ival = 47892;
    cout << ival << '\n';

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

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

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

    int ival = 47892;
    cout << ival << '\n';

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

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

```
7 #include <algorithm>
8 #include <random>
9 #include <chrono>
```

sayiyi isareti ile yaziyor

```
10
11
12
13 int main()
14 {
15     using namespace std;
16
17     cout << 1234 << "\n"; [yellow dot]
18     cout.setf(ios::showpos);
19     cout << 1234 << "\n";
20
21
22
23 }
```

```
0 //include <iostream>
1 #include <chrono>
2
3 void func(std::ostream& os)
4 {
5     std::cout << "boolalpha : " << (os.flags() & std::ios::boolalpha ? "set" : "unset") << "\n";
6     std::cout << "uppercase : " << (os.flags() & std::ios::uppercase ? "set" : "unset") << "\n";
7     std::cout << "showpos   : " << (os.flags() & std::ios::showpos? "set" : "unset") << "\n";
8     std::cout << "showpoint : " << (os.flags() & std::ios::showpoint? "set" : "unset") << "\n";
9 }
10
11 int main()
12 {
13     func(std::cout);
14 }
```

```
1
2 class fmguard {
3 public:
4     fmguard(std::ostream& os) :m_os(os), m_flags{os.flags()}
5     {
6
7     }
8
9     ~fmguard()
10    {
11         m_os.flags(m_flags);
12    }
13
14 private:
15     std::ostream& m_os;
16     std::ios::fmtflags m_flags;
17 };
18
19
20 void foo(std::ostream& os)
21 {
22     fmguard myguard{ os };           I
23     // ...
24 }
```

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

    cout << bitset<32>(ios::basefield) << "\n";
    cout << bitset<32>(ios::dec) << "\n";
    cout << bitset<32>(ios::oct) << "\n";
    cout << bitset<32>(ios::hex) << "\n";
}
```

hex	oct	dec		basefield
left	right	internal		adjustfield
fixed	I	scientific		floatfield

hem fixed hemde scientific olursa hex olarak ayarlanır

```
void print_state(std::ostream& os)
{
    std::cout << "fixed      : " << (os.flags() & std::ios::fixed ? "set" : "unset") << "\n";
    std::cout << "scientific : " << (os.flags() & std::ios::scientific? "set" : "unset") << "\n\n";
}

int main()
{
    using namespace std;

    cout << 2.3 << "\n" << 32947862345.7345 << "\n";
    cout.setf(ios::fixed, ios::floatfield);
    cout << 2.3 << "\n" << 32947862345.7345 << "\n";
    cout.setf(ios::scientific, ios::floatfield);
    cout << 2.3 << "\n" << 32947862345.7345 << "\n";
    /**
     cout.setf(ios::fixed | ios::scientific);

    cout << 45.89435 << "\n";
}
```

I

```
4410  
4411 scientific  
4412  
4413  
4414 int ival = 435;  
4415  
4416 .width  
4417 .fill  
4418  
4419  
4420 fill character  
4421  
4422  
4423 ??????435      right  
4424 435??????    left  
4425  
4426 +      981  
4427 -      35678  
4428 +      726  
4429  
4430  
4431  
4432 left|right|internal      adjustfield
```

```
7 class Triple {  
8  
9     public:  
10  
11         Triple(int x, int y, int z) : mx{ x }, my{ y }, mz{ z } { }  
12  
13     Triple() { }  
14  
15     ~Triple() { }  
16  
17     void set(int a, int b, int c)  
18     {  
19         mx = a;  
20         my = b;  
21         mz = c;  
22     }  
23  
24  
25     friend std::ostream& operator<<(std::ostream& os, const Triple& t)  
26     {  
27         std::ostringstream oss;  
28         oss << "(" << t.mx << ", " << t.my << ", " << t.mz << ")";  
29  
30         return os << oss.str();  
31     }  
32  
33     private:  
34         int mx{}, my{}, mz{};  
35     };  
36
```

reis oss ile return etmesek
maindeki herhangi bir
formatlama isleminde ilk
gördüğü deger olan
(formatlanır digerleri
standart olur

```
std::ostream& bhex(std::ostream& os)
{
    os.setf(std::ios::hex, std::ios::basefield);
    os.setf(std::ios::uppercase | std::ios::showbase);
    return os;
}

int main()
{
    using namespace std;

    cout << 47802 << bhex << " " << 47802 << '\n';
}
```

```
#include <string>

class ns {
public:
    ns() = default;
    explicit ns(int val) : mval{val} {}
    friend std::ostream& operator<<(std::ostream& os, const ns& x)
    {
        auto n = x.mval;
        while (n--)
            os.put(' ');
        return os;
    }

private:
    int mval{};
};

int main()
{
    using namespace std;

    int x{ 23 };
    string name{ "berkin" };
    int y{ 5765 };

    cout << x << ns(4) << name << ns(8) << y << ns(12) << "ali" << ns();
}
```

▲ 2 of 4 ▼ ns()

47802 1 4 : 0x0000000000000000
0XBABA true 4.00000
dateexp dateint notarbit
cpp_january (Global Scope)
C:\Users\necat\source\repos\cpp_january\Debug\cpp_january.exe (process 15592) exited with code 0
Press any key to close this window . . .

```
7 #include <iostream>
8 #include <ctime>
9 #include <string>
10 #include "nutility.h"
11 #include <fstream>
12 #include <iostream>
13 #include <algorithm>
14 #include "date.h"
15 #include <bitset>
16 #include <set>
17
18
19 int main()
20 {
21     using namespace std;
22
23     ostream os{ cout.rdbuf() };
24     os << hex << boolalpha << uppercase << showbase << showpoint;
25
26     cout << 47802 << " " << (10 > 5) << " " << 4. << "\n";
27     os << 47802 << " " << (10 > 5) << " " << 4. << "\n";
28 }
29 }
```

date.cpp date.h main.cpp triple.h notlar.txt

(Global Scope)

```
5 #include "triple.h"
6 #include <iomanip>
7 #include <sstream>
8 #include <ctime>
9 #include <string>
10 #include "nutility.h"
11 #include <fstream>
12 #include <sstream>
13 #include <algorithm>
14 #include "date.h"
15 #include <bitset>
16 #include <set>
17
18
19 int main()
20 {
21     using namespace std;
22
23     int x;
24
25     cout << "bir tam sayi girin: ";
26     cin >> x;
27
28     if (cin.good())
29         std::cout << "hata yok stream iyi durumda\n";
30     else
31         std::cout << "hata var stream iyi durumda degil\n";
32
33
34
35
36
```

<code>ios_base::iostate</code> flags			<code>basic_ios</code> accessors						
<code>eofbit</code>	<code>failbit</code>	<code>badbit</code>	<code>good()</code>	<code>fail()</code>	<code>bad()</code>	<code>eof()</code>	<code>operator bool</code>	<code>operator!</code>	
false	false	false	true	false	false	false	true	false	
false	false	true	false	true	true	false	false	true	
false	true	false	false	true	false	false	false	true	
false	true	true	false	true	true	false	false	true	
true	false	false	false	false	false	true	true	false	
true	false	true	false	true	true	true	false	true	
true	true	false	false	true	false	true	false	true	
true	true	true	false	true	true	true	false	true	

l.cpp.january (Global Scope)

```
4 {  
5     if (is.rdstate() == 0) {  
6         std::cout << "stream is in good state\n";  
7     }  
8  
9     if (is.rdstate() & std::ios::eofbit)  
10        std::cout << "eof bit set\n";  
11  
12    if (is.rdstate() & std::ios::failbit)  
13        std::cout << "fail bit set\n";  
14  
15    if (is.rdstate() & std::ios::badbit)  
16        std::cout << "badbit set\n";  
17 }  
18 I  
19 int main()  
20 {  
21     using namespace std;  
22  
23     int x;  
24  
25     cout << "bir sayi girin: ";  
26     cin >> x;  
27  
28     display_stream_state(cin);  
29 }
```

0 % - No issues found

For List: Entire Solution 0 Errors 0 Warnings 0 Messages Build Only

```
1 #include <iostream>
2
3 int main()
4 {
5     using namespace std;
6
7     cin.clear(ios::failbit | ios::badbit);
8
9     cout << boolalpha;
10
11    cout << "good  : " << cin.good() << "\n";
12    cout << "fail  : " << cin.fail() << "\n";
13    cout << "bad   : " << cin.bad() << "\n";
14    cout << "eof   : " << cin.eof() << "\n";
15    cout << "o bool: " << cin.operator bool() << "\n";
16    cout << "op !  : " << cin.operator !() << "\n";
17
18 }
```

I

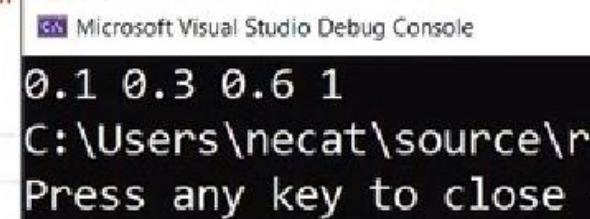
```
1 #include <iostream>
2 #include <iomanip>
3 #include <string>
4
5 int main()
6 {
7     using namespace std;
8
9     int x{};
10
11    while (std::cout << "enter an integer: " && !(std::cin >> x)) {
12        if (cin.eof()) {
13            std::cout << "you have not entered anything!\n";
14            cin.clear();
15        }
16        else {
17            cin.clear();
18            string line;
19            getline(cin, line);
20            cout << "[" << line << "] is not a valid integer!\n";
21        }
22    }
23
24    cout << "your number is : " << x << "\n";
25 }
```

```
1 #include <iostream>
2 #include <iterator>
3 #include <vector>
4 #include <fstream>
5 #include <algorithm>
6
7 int main()
8 {
9     using namespace std;
10    ifstream ifs{ "out.txt" }; I
11    if (!ifs) {
12        cerr << "dosya acilamadi\n";
13        return 1;
14    }
15    vector<string> svec{ istream_iterator<string>{ifs}, {} };
16
17    cout << "svec.size() = " << svec.size() << '\n';
18    copy(svec.begin(), svec.end(), ostream_iterator<string>(cout, "\n"));
19
20 }
```

```
1 #include <iostream>
2 #include <iterator>
3 #include <vector>
4 #include <fstream>
5 #include <algorithm>
6 #include <sstream>
7 #include <algorithm>
8 #include <numeric>
9 #include "nutility.h"
10
11
12 int main()
13 {
14     using namespace std;
15
16     istreamiss iss{ "12 34 56 23 45 98 12 67" };
17
18     cout << accumulate(istream_iterator<int>{iss}, {}, 0) << "\n"
19
20
21 }
```

```
7 #include <algorithm>
8 #include <numeric>
9 #include "nutility.h"
10
11
12 int main()
13 {
14     using namespace std;
15
16     cout << "sayilari girin: ";
17
18     cout << *max_element(istream_iterator<int>{cin}, {} ) << "\n";
19
20
21 }
```

```
1 // from cppreference.com
2
3 #include <algorithm>
4 #include <iostream>
5 #include <iterator>
6 #include <numeric>
7 #include <sstream>
8
9 int main()
10 {
11     std::istringstream str("0.1 0.2 0.3 0.4");
12     std::partial_sum(std::istream_iterator<double>(str), {},
13                      std::ostream_iterator<double>(std::cout, " "));
14 }
15
16 }
```



// ofstream
// ifstream
// fstream

ostringstream
istringstream
stringstream

ostream
istream
iostream

I

```
//ios::in  
//ios::out  
//ios::app  
//ios::trunc  
//ios::binary  
//ios::ate  
  
ifstream ifs{ "netati.txt"};
```

```
#include <algorithm>
#include <iostream>
#include <iterator>
#include <numeric>
#include <sstream>
#include <fstream>
#include "nutility.h"

int main()
{
    using namespace std;

    ofstream ofs{ "necati.txt" };

    vector<string> svec;

    rfill(svec, 10000, rname);
    cout << "svec.size() = " << svec.size() << '\n';

    copy(svec.begin(), svec.end(), ostream_iterator<string>{ofs, "\n"});
}
```

```
#include <algorithm>
#include <iostream>
#include <iterator>
#include <numeric>
#include <sstream>
#include <fstream>
#include "nutility.h"
#include "file.h"

int main()
{
    using namespace std;

    auto ofs = create_text_file("necati.txt");

    ofs << hex << uppercase;

    for (int i = 0; i < 1000; ++i) {
        ofs << Irand{0, 1'000'000}() << '\n';
    }

}
```

```
#include <algorithm>
#include <iostream>
#include <iterator>
#include <numeric>
#include <sstream>
#include <fstream>
#include "nutility.h"
#include "file.h"

int main()
{
    using namespace std;

    auto ifs = open_text_file("necati.txt");

    int val;
    cout << "kaca tam bolunenler yazilsin : ";
    cin >> val;

    copy_if(istream_iterator<int>{ifs}, {}, ostream_iterator<int> {cout, "\n"},
            [val](int x) {
                return(x % val == 0);
            });
}
```

I

```
{  
    using namespace std;  
  
    constexpr int n = 1'000'000;  
  
    ofstream ofs{ "primes.txt" };  
    if (!ofs) {  
        cerr << "dosya olusturulamadi\n";  
        return 1;  
    }  
  
    ofs << left;  
  
    int prime_count{};  
    int x{ 2 };  
  
  
    while (prime_count < n) {  
        if (isprime(x)) {  
            if (prime_count && prime_count % 10 == 0)  
                ofs << '\n';  
            ofs << setw(10) << x << " ";  
            ++prime_count;  
        }  
        ++x;  
    }  
}
```

1milyona kadar asallar
formatlayarak yazıyor

```
#include <conio.h>
#include <iomanip>
```

1m

```
//ilk 10000 asal sayıyı formatlı olarak primes.txt dosyasına yazınız
```

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

    auto ifs = open_text_file("primes.txt");

    vector<int> ivec{ istream_iterator<int>{ifs}, {} };

}
```

I

```
#include <iostream>
#include <fstream>
#include "nutility.h"
#include "file.h"
#include <vector>
#include <set>
#include <conio.h>
#include <iomanip>

//ilk 10000 asal sayiyi formatli olarak primes.txt dosyasina yaziniz

int main()
{
    using namespace std;

    auto ifs = open_text_file("primes.txt");

    int len{};

    cout << "kac basamakli : ";
    cin >> len;

    copy_if(istream_iterator<string>{ifs}, {}, ostream_iterator<string>{cout, "\n"}, 
        [len](const string& s) {
            return s.length() == len && s.front() == s.back();
    });
}
```

ostr kisiselistirilmis bir cout

```
int main()
{
    using namespace std;
    ostream ostr{ cout.rdbuf() };

    ostr.setf(ios::boolalpha | ios::uppercase | ios::showbase);
    ostr.setf(ios::hex, ios::basefield);
    ostr.setf(ios::scientific, ios::floatfield);
    ostr.precision(2);

    int x = 47'802, y = 57'054;
    double dval = 4187.7233;

    cout << x << " " << y << " " << dval << " " << (x > y) << '\n';
    ostr << x << " " << y << " " << dval << " " << (x > y) << '\n';
}
```

```
//main.cpp dosyasini std output'a yazdirin
```

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

    ifstream ifs{ open_text_file("nutility.h") };

    cout << ifs.rdbuf() << "\n";

}
```

```
int main(int argc, char **argv)
{
    using namespace std;

    if (argc != 3) {
        cerr << "kullanim: <kopyala> <kaynak dosya ismi> <edef dosya ismi>\n";
        return 1;
    }

    ifstream ifs{ argv[1], ios::binary };
    if (!ifs) {
        cerr << argv[1] << " acilamadi\n";
        return 2;
    }

    ofstream ofs{ argv[2], ios::binary };
    if (!ofs) {
        cerr << argv[2] << " olusturlamadi\n";
        return 3;
    }

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

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

    ostringstream oss;

    auto cbuf = cout.rdbuf();| I
    cout.rdbuf(oss.rdbuf());| I

    cout << "omerhan cakmak";
    cout.rdbuf(cbuf);
    cout << "mehmet eskol\n";

    auto name = oss.str();

    cout << name << "\n";
```

bir stream ifadesinin exception throw etmesi
için failbit burda bu sekil bildirildi. farkli
hatalarda bildirilebilirdi tabi

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

    cin.exceptions(ios::failbit);

    try {
        int x{};
        cout << "bir tam sayi girin: ";
        cin >> x;
    }
    catch (const std::exception& ex) {
        cout << "exception caught: " << ex.what() << '\n';
    }
}
```

```
using namespace std;

cin.exceptions(ios::failbit | ios::eofbit | ios::badbit);

auto ist = cin.exceptions();

if (ist == 0) {
    std::cout << "exception throw etmez\n";
    return 0;
}

if (ist & ios::failbit) {
    std::cout << "failbit\n";
}

if (ist & ios::eofbit) {
    std::cout << "eofbit\n";
}

if (ist & ios::badbit) {
    std::cout << "badbit\n";
}

std::cout << " set edildiginde exception throw eder\n";
```

I

```
cout << typeid(ratio_add<one_third, two_fifth>::type).name() << "\n";
cout << typeid(ratio_subtract<one_third, two_fifth>::type).name() << "\n";
cout << typeid(ratio_divide<one_third, two_fifth>::type).name() << "\n";
cout << typeid(ratio_multiply<one_third, two_fifth>::type).name() << "\n";
ratio_less_v<ratio....>
```

ADL (Argument-Dependent Lookup), C++ dilinde bir dil özellikleidir. ADL, fonksiyon çağrıları için geçerlidir ve hangi fonksiyonların çağrılacağını belirlemek için argüman türlerine dayalı olarak arama yapar.

ADL, aynı isme sahip olan fonksiyonların farklı ad alanlarında bulunmasına izin verir. Eğer bir fonksiyon çağrısı, o fonksiyonun bildiriminin yapıldığı ad alanında yoksa, C++ dilindeki normal arama kuralları sırasında aranacak yerlerin yanı sıra fonksiyonun argümanlarına bakarak ilgili ad alanlarında da arama yapılır.

Bir örnek vermek gereklidir:

```
cpp Copy code

namespace example {
    struct MyStruct {};

    void myFunction(MyStruct obj) {
        // Fonksiyon içeriği
    }
}

int main() {
    example::MyStruct obj;
    myFunction(obj); // ADL kullanılarak example ad alanında myFunction aranır
    return 0;
}
```

Yukarıdaki örnekte, `myFunction` çağrılarında `MyStruct` türü `example` ad alanında tanımlıdır. Bu durumda ADL, `myFunction`'ı `example` ad alanında arayacak ve uygun fonksiyonu bulacaktır.

ADL, STL (Standard Template Library) ile sıkça karşılaşılan bir durumdur ve C++ dilinde özellikle şablonlarla çalışırken fonksiyon arama sürecini etkiler.

Sonuç olarak, ADL, C++ dilinde fonksiyon çağrıları için argüman türlerine dayalı olarak ek arama imkanı sağlayan önemli bir dil özelliğidir.

```
constexpr double operator""_cm(long double x)
{
    return static_cast<double>(x / 100.);
}

constexpr double operator""_m(long double x)
{
    return static_cast<double>(x);
}

constexpr double operator""_km(long double x)
{
    return static_cast<double>(x * 1000);
}

int main()
{
    constexpr auto distance = 234.34_m + 0.123_km + 345.91_cm;
    operator""_km(0.123)
```

```
constexpr int operator""_b(const char* p) I
{
    int ret{};

    while (*p) {
        if (!(*p == '0' || *p == '1'))
            throw std::runtime_error{ "bad binary constant" };
        ret = ret * 2 + *p - '0';
        ++p;
    }

    return ret;
}
```

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

    try {
        auto val = 11131111_b;
        cout << "val = " << val << '\n';

    }
    catch (const std::exception& ex) {
        std::cout << "exception caught: " << ex.what() << '\n';
    }
}
```

```
class Meters
{
    double mts;
public:
    class PreventUsage {};
    explicit constexpr Meters(PreventUsage, double meter) : mts{ meter } {}

    explicit constexpr operator double()const { return mts; }
};

constexpr Meters operator""_m(long double m)
{
    return Meters(Meters::PreventUsage{}, static_cast<double>(m));
}

std::ostream& operator<<(std::ostream& os, const Meters& m)
{
    return os << static_cast<double>(m) << " meters";
}

#include <iostream>

int main()
{
    constexpr auto m1 = 34.56_m;

    double dval = static_cast<double>(m1);
```

```
4 template <typename T> <T> Provide sample template arguments for IntelliSense
5 void func(T&& x)
6 {
7
8     auto&& r = foo(std::forward<T>(x));
9     /**
10
11     std::forward<decltype(r)>(r);|
12
13
14
15
16 }
17
18
19
20 int main()
21 {
22 }
```

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

    mt19937 eng;
    std::ofstream ofs{ "out.txt" };
    if (!ofs) {
        std::cerr << "out.txt dosyasi olusturulamadi\n";
        exit(EXIT_FAILURE);
    }

    for (int i = 0; i < 20000; ++i) {
        eng();
    }

    stringstream ss;

    ss << eng;

    for (int i = 0; i < 10; ++i) {
        ofs << eng() << "\n";
    }

    ss >> eng;
```

cout eng() txt ile
cout aynı veriye sahip olr

```
#include <fstream>
#include <sstream>

/*
    1487154456
3985498208
614186726
944150852
3608006424
*/
int main()
{
    using namespace std;

    mt19937 eng{ random_device{}() };

    for (int i = 0; i < 10; ++i) {
        cout << eng() << '\n';
    }
}
```

Microsoft Visual Studio Debug Console

```
253002748
2634438246
3077217478
3133711134
598600716
159447522
1609073423
115565442
4011755012
987146619

C:\nec\ a.exe (process 2172) ex
Press any key to close this wi
```

```
1 int main()
2 {
3     using namespace std;
4     using namespace std::chrono;
5
6
7     mt19937 eng(system_clock::now().time_since_epoch().count());
8
9     for (int i = 0; i < 5; ++i) {
10         cout << eng() << '\n';
11     }
12 }
```

I

```
9
10 int main()
11 {
12     using namespace std;
13
14     mt19937 eng;
15     minstd_rand mrand;
16
17     uniform_int_distribution dist{1, 6};
18
19     for (int i = 0; i < 100; ++i) {
20         cout << dist(eng);
21         _getch();
22     }
23 }
```

I

```
L1 int main()
L2 {
L3     using namespace std;
L4
L5     mt19937 eng;
L6
L7     map<int, int> cmap;
L8
L9     uniform_real_distribution dist{10.0, 45.0};
L10
L11    cout << fixed << setprecision(2) << '\n';
L12
L13    for (int i = 0; i < 100; ++i) {
L14        cout << dist(eng) << '\n';
L15    }
L16
L17 main()
```



```
using namespace std;

mt19937 eng;

normal_distribution<> dist{ 50., 5. };
std::ofstream ofs{ "out.txt" };
if (!ofs) {
    std::cerr << "out.txt dosyasi olusturulamadi\n";
    exit(EXIT_FAILURE);
}

ofs << fixed << setprecision(4);           I
for (int i = 0; i < 1000; ++i) {
    ofs << dist(eng) << '\n';
}
```

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

    mt19937 eng;

    normal_distribution<> dist{ 50., 5. };

    map<int, int> cmap;

    std::ofstream ofs{ "out.txt" };
    if (!ofs) {
        std::cerr << "out.txt dosyasi olusturulamadi\n";
        exit(EXIT_FAILURE);
    }

    constexpr int n = 1'000'000;

    ofs << fixed << setprecision(4);
    for (int i = 0; i < n; ++i) {
        ++cmap[static_cast<int>(round(dist(eng)))];
    }

    ofs << setfill('0');

    for (const auto& [key, count] : cmap) {
        ofs << setw(2) << key << " " << string(count / 1000, '*') << '\n';
    }
}
```

Normal dağılım

```
int main()
{
    using namespace std;
    mt19937 eng;
    uniform_int_distribution<> dist(3, 9);

    decltype(dist) x(dist.param());
}

I
```

hem dist hem x
aynı parametrelerle
ctor edilmiş oldu

```
{  
    using namespace std;  
|           I  
mt19937 eng;  
  
double a[12] = { 10, 10, 11, 10, 10, 11, 10, 10, 11, 10, 10, 11, };  
  
discrete_distribution<> dist{ begin(a), end(a) };  
  
map<int, int> cmap;  
  
for (int i = 0; i < 1'000'000; ++i) {  
    ++cmap[dist(eng)];  
}  
  
cout << setfill('0');  
  
for (const auto& [val, count] : cmap) {  
    cout << setw(2) << val + 1 << " " << count << "\n";  
}
```

```
02 80587  
03 88500  
04 80600  
05 81122  
06 88524  
07 80534  
08 80811  
09 88404  
10 81118  
11 80578  
12 88599
```

Press E

```
C:\nec\a.exe (process 13260) exited with code 0.  
Press any key to close this window . . .
```

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

    bernoulli_distribution dist{ 0.51 };
    mt19937 eng;

    int count{};

    for (int i = 0; i < 10'000'000; ++i) {
        count += dist(eng) ? 1 : 0;
    }

    cout << count << "\n";
}

int main()
{
    using namespace std;

    const vector<string> mons = { "Ocak", "Subat", "Mart", "Nisan", "Mayis", "Haziran",
        "Temmuz", "Agustos", "Eylul", "Ekim", "Kasim", "Aralik" };

    int n;

    std::cout << "kac tane ay istiyorsunuz: ";
    cin >> n;

    sample(mons.begin(), mons.end(), ostream_iterator<string>{cout, " "}, n, mt19937{ random_device{}() });
}
```

optional'e 1den fazla parametreli bisiler ctor etmek için 2 farklı yöntem

i

```
using namespace std;  
  
//optional<complex<double>> x(in_place, |2.3, 5.6)  
make_optional<complex<double>>(5.6, 6.7)
```

```
operator*  
operator->  
  
o.value()  
o.operator*  
o.operator->  
o.value_or  
o.has_value  
o.operator bool|
```

```
#include <vector>

class MyClass {
public:

    ~MyClass()
    {
        std::cout << "destructor = " << this << "\n";
    }

    unsigned char buf[1024]{};
};

int main()
{
    using namespace std;

    auto x = make_optional<MyClass>();

    //x.reset();
    //x = nullopt;
    //x = optional<MyClass>{};    I
    x = {};
```



```
(void)getchar();
std::cout << "main devam ediyor\n";
}
```

runtimedeki dtor çağrımanın
4 yolu

```
#include <complex>
#include <vector>

struct Person {
public:
    bool has_email()const;
    std::string e_mail()const;
};

std::optional<std::string> get_email_address(const Person&p)
{
    /**
     * if (p.has_email()) {
     *     return "necatiergin@gmail.com";
     */
    return std::nullopt;
    return {};
}
```

```
#include <string>
#include <optional>
#include <iostream>
#include <complex>
#include <vector>

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

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

int main()
{
    using namespace std;

    auto opx = make_optional<Myclass>(45);
    opx.emplace(56);

    (void)getchar();
}
```

emplace kullandığm için 45 değerini tutan nesne dtor edilecek. hemen ardından diğer 56nesnesi için ctor ve dtor çağrılacak normal move assignment opx = {56} denseydi dtor çağrılmadan tanımlama yap

```
#include <optional>
#include <string>
#include <iostream>

// aldigı string'i int degere donusturecek - inte donusturemez ise deger dondurmeyecek

std::optional<int> to_int(const std::string& s)
{
    try {
        return std::stoi(s);
    }
    catch (...) {
        //return {};
        return std::nullopt;
    }
}

std::optional<int> to_int2(const std::string& s)
{
    std::optional<int> ret; // deger tutmuyor
    try {
        ret = std::stoi(s);
    }
    catch (...) {}

    return ret;
}

int main()
{
    for (auto s : { "42", " 077", "necati", "0x33" }) {
```

2 farklı int çeviren fonksiyon

I

```
#include <string>

class UserRecord
{
public:
    UserRecord(const std::string& name, std::optional<std::string> nick, std::optional<int> age)
        : m_name{ name }, m_nick{ nick }, m_age{ age } I
    {
    }
    friend std::ostream& operator << (std::ostream& stream, const UserRecord& user);
private:
    std::string m_name;
    std::optional<std::string> m_nick;
    std::optional<int> m_age;
}; string_view

std::ostream& operator << (std::ostream& os, const UserRecord& user)
{
    os << user.m_name << ' ';
    if (user.m_nick) {
        os << *user.m_nick << ' ';
    }
    if (user.m_age)
        os << *user.m_age << " yasinda.';

    return os;
}
```

No issues found

```
int main()
{
    char s[] { 'n', 'e', 'c', 'o' };

    std::string_view sv{ s, 4 };

    std::cout << sv << '\n'; //no problem
    std::cout << sv.data() << '\n'; //ub
}
```

```
MAIN.CPP - 1000 LİNE  
  
std::string func(std::string_view sv)  
{  
    return sv; //gecersiz  
    //return std::string{ sv }; //gecerli  
    //return sv.data(); //gecerli  
}  
  
I  
void foo(std::string);  
  
int main()  
{  
    std::string_view sw("necati ergin");  
  
    //std::string str = sw; //gecersiz (explicit ctor)  
    std::string str{ sw }; //gecerli  
    std::string s = sw.data(); //gecerli  
    //foo(sw); //gecersiz  
    foo(std::string{ sw }); //gecerli  
    foo(sw.data()); //gecerli  
}
```

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

    string_view sv("omerhan cakmak");

    cout << "sv.length() = " << sv.length() << '\n';
    cout << "|" << sv << "|"\n";

    sv.remove_prefix(4);

    cout << "sv.length() = " << sv.length() << '\n';
    cout << "|" << sv << "|"\n";
}
```

Microsoft Visual Studio Debug Console

```
sv.length() = 14
|omerhan cakmak|
sv.length() = 10
|han cakmak|
C:\nec\1a.exe (process 1164)
Press any key to continue . . .
```

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

    string_view sv("omerhan cakmak");

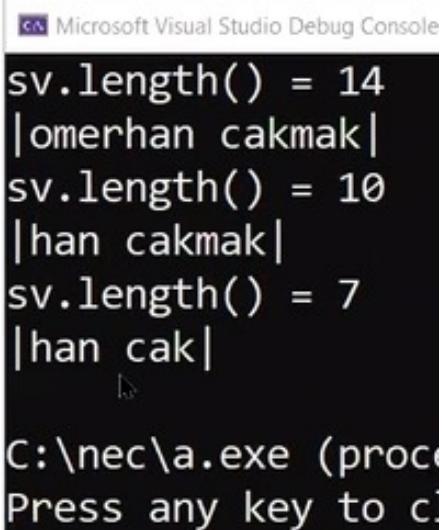
    cout << "sv.length() = " << sv.length() << '\n';
    cout << "|" << sv << "|\\n";

    sv.remove_prefix(4);

    cout << "sv.length() = " << sv.length() << '\n';
    cout << "|" << sv << "|\\n";

    sv.remove_suffix(3);
    cout << "sv.length() = " << sv.length() << '\n';
    cout << "|" << sv << "|\\n";

}
```



Microsoft Visual Studio Debug Console

```
sv.length() = 14
|omerhan cakmak|
sv.length() = 10
|han cakmak|
sv.length() = 7
|han cak|
C:\nec\a.exe (proc)
Press any key to c
```

```
int main()
{
    std::string str{ "    basimda bir bosluk hissi var" };
    std::cout << "(" << str << ")\\n";
    std::string_view sv{ str };

    sv.remove_prefix(std::min(sv.find_first_not_of(" "), sv.size()));
    std::cout << "(" << sv << ")\\n";

    //std::cout << "(" << str << ")\\n";
}

{
    std::string_view sw{ "The good the bad and the ugly" };

    std::cout << std::boolalpha;

    std::cout << sw.length() << "\\n"; //29
    auto swx = sw.substr(0, sw.find(' '));
    std::cout << (swx == "The") << "\\n"; //true

    std::cout << (sw == "The poor") << "\\n"; //false

    std::cout << sw.starts_with("The") << "\\n"; //true
    std::cout << sw.ends_with("pretty") << "\\n"; //false

    //std::cout << "[" << sw << "]";
}
```

I

```
#include "date.h"

int main()                                201
{
    using namespace std;

    variant<int, double, string> vx("mustafa");

    cout << "vx.index() = " << vx.index() << '\n';

    vx = 12;

    cout << "vx.index() = " << vx.index() << '\n';

    vx = 4.5;

    cout << "vx.index() = " << vx.index() << '\n';

}
```

```
variant<int, const char*, string> vx(in_place_index<2>, "murat");

cout << "vx.index() = " << vx.index() << '\n';
```

2 numaralı indexi kullan demek istedik normalde char yıldız ile setlenirdi

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

    variant<int, double, string> vx{ "mustafa savas" };

    auto p = get_if<double>(&vx); I
    if (p) {
        cout << "double : " << *p << "\n";
    }
    else {
        cout << "simdi double tutmuyor\n";
```

```
variant<int, double, string> vx{ 19.8};

if (vx.index() == 0) {
    cout << "int " << get<0>(vx) << '\n';
}
else if (vx.index() == 1) {
    cout << "double " << get<1>(vx) << '\n';
}
else if (vx.index() == 2) {
    cout << "string " << get<2>(vx) << '\n';
}
```

```
if (auto pi = get_if<int>(&vx)) {
    cout << "int : " << *pi << "\n";
}
else if (auto pd = get_if<double>(&vx)) {
    cout << "double : " << *pd << "\n";
}
else if (auto ps = get_if<string>(&vx)) {
    cout << "string : " << *ps << "\n";
}
```

```
using namespace std;

variant<int, double, string> vx{ "mustafa"};

if (holds_alternative<int>(vx)) {
    cout << "int " << get<int>(vx) << '\n';
}
else if (holds_alternative<double>(vx)) {
    cout << "double " << get<double>(vx) << '\n';
}
else if (holds_alternative<string>(vx)) {
    cout << "string" << get<string>(vx) << '\n';
}

if (vx.index() == 0) {
    cout << "int " << get<0>(vx) << '\n';
}
else if (vx.index() == 1) {
    cout << "double " << get<1>(vx) << '\n';
}
else if (vx.index() == 2) {
    cout << "string " << get<2>(vx) << '\n';
}
```

```
struct Visitor {

    void operator()(int x)const
    {
        std::cout << "int : (" << x << ")\\n";
    }

    void operator()(double x)const
    {
        std::cout << "double : [" << x << "]\\n";
    }

    void operator()(const std::string &x)const
    {
        std::cout << "string : \"\" << x << "\"\\n";
    }
};

int main()
{
    using namespace std;

    variant<int, double, string> vx("altan karaman");

    visit(Visitor{}, vx);           I

    vx = 4.57;

    visit(Visitor{}, vx);
```

```
struct Visitor {  
    template <typename T>  
    void operator()(T x) const  
    {  
        std::cout << x << "\n";  
    }  
};  
  
int main()  
{  
    using namespace std;  
  
    variant<int, double, string> vx("altan karaman");  
  
    visit(Visitor{}, vx);  
}
```

```
#include <vector>
#include "date.h"

int main()
{
    using namespace std;

    variant<int, double, string> vx;

    auto print_visitor = [] (const auto& x) {
        std::cout << "[" << x << "] \n";
    };

    auto triple_visitor = [] (auto& x) {
        x = x + x + x;
    };

    vx = 30;

    visit(print_visitor, vx);
    visit(triple_visitor, vx);
    visit(print_visitor, vx);
```

```
#include <vector>
#include "date.h"

int main()
{
    using namespace std;

    variant<int, double, string> vx;

    auto print_visitor = [] (const auto& x) {
        std::cout << "[" << x << "]\\n";
    };

    auto triple_visitor = [] (auto& x) {
        x = x + x + x;
    };

    vx = 30;

    visit(print_visitor, vx);
    visit(triple_visitor, vx);
    visit(print_visitor, vx);
```

```
variant<monostate, Data, int, double> v2; //valid

cout << "index = " << v2.index() << '\n'; // 0

if (holds_alternative<std::monostate>(v2))
    cout << "empty (monostate)\n";
else
    cout << "not empty\n";

if (get_if<std::monostate>(&v2))
    cout << "empty (monostate)\n";
else
    std::cout << "not empty\n";

v2 = Data{ 13 };
cout << "index = " << v2.index() << '\n';
v2 = 23;
cout << "index = " << v2.index() << '\n';
v2 = 4.5;
cout << "index = " << v2.index() << '\n';
v2 = std::monostate{};
//cout << "index = " << v2.index() << '\n';
//v2 = {};
//cout << "index = " << v2.index() << '\n';
}
```

```
cout << boolalpha;

auto f = [](const auto& val) {
    cout << "deger: " << val << '\n';
};

variant<bool, string> var;
var = "necati";

cout << "index: " << var.index() << '\n';
visit(f, var);

var.emplace<1>("ekrem");
cout << "index: " << var.index() << '\n';
visit(f, var);

var.emplace<string>("elif");
visit(f, var);

var = "sinan"s;
visit(f, var);
}
```

```
#include <bitset>
#include <type_traits>

class PrintVisitor {
public:

    template <typename T>
    void operator()(const T& x)
    {
        if constexpr (std::is_same_v<double, T>) {
            std::cout << "implementation for double\n";
        }
        else {
            std::cout << "implementation for other alternatives\n";
        }
    }
};

int main() I
{
    using namespace std;

    variant<int, double, string, bitset<32>> vx{ 23 };

    visit(PrintVisitor{}, vx);
    vx = 3.567;
    visit(PrintVisitor{}, vx);
    vx = "muharrem";
    visit(PrintVisitor{}, vx);
    vx = bitset<32>{ 8732452 };
    visit(PrintVisitor{}, vx);
```

```
#include <type_traits>
#include <iomanip>

int main()
{
    using namespace std;

    using nick = std::string;
    using name = std::string;
    using id = int;

    variant<nick, id, name> vx1(std::in_place_index<2>, "mustafa");
    variant<nick, id, name> vx1(std::in_place_index<0>, "musti");

    vx1.emplace<0>("neco");

}
```

vx1 eşittir neco desen syntax
hatası emplace kullan

```
#include <type_traits>
#include <iomanip>

enum {nick_index, id_index, name_index};

int main()
{
    using namespace std;

    using nick = std::string;
    using name = std::string;
    using id = int;

    variant<nick, id, name> vx1(std::in_place_index<name_index>, "mustafa");
    variant<nick, id, name> vx1(std::in_place_index<0>, "musti");

    vx1.emplace<0>("neco");

}
```

```
struct A {
    void foo(int)
    {
        std::cout << "A:foo(int)\n";
    }
};

struct B {
    void foo(double)
    {
        std::cout << "B:foo(double)\n";
    }
};

struct C {
    void foo(long)
    {
        std::cout << "C:foo(long)\n";
    }
};

struct Der : A, B, C {
    using A::foo;
    using B::foo;
    using C::foo;
};

int main()
{
    Der myder;

    myder.foo(12);
    myder.foo(4.5);
    myder.foo(24L);
}
```

```
5
6 template <typename ...Args>
7 struct Der : Args ... {
8     using Args::foo...;
9 }
0
10 struct A {
11     void foo(int);
12 };
13
14 struct B {
15     void foo(double);
16 };
17
18 struct C {
19     void foo(long);
20 };
21
22 int main()
23 {
24
25     Der<A, B, C> myder;
26
27     myder.foo(12);
28     myder.foo(1.2);
29     myder.foo(34L);
30 }
```

```
auto fn1 = [](int x) {return x * x; };
auto fn2 = [](int x) {return x + x; };
auto fn3 = [](int x) {return x - x; };

struct Der : decltype(fn1), decltype(fn2), decltype(fn3) {

};

int main()
{
```

```
template<typename ... Args>
struct Overload : Args ...
{
    using Args::operator() ...;
};

struct Nec {};
struct Erg {};

int main()
{
    using var_type = std::variant<int, char, unsigned, float, double, long long, Nec, Erg>;
    auto stypes = Overload{
        [](int) { return "int"; },
        [](char) { return "char"; },
        [](unsigned) { return "unsigned"; },
        [](float) { return "float"; },
        [](double) { return "double"; },
        [](long long) { return "long long"; },
        [](Nec) { return "Nec"; },
        [](auto) { return "unknown type"; },
    };
    //std::cout << typeid(stypes).name() << "\n";
    std::vector<var_type> varvec{ 23, 'A', 45U, 5.f, 6.4, 34LL, Nec{}, Erg{} };
    for (auto v : varvec) {
        std::cout << std::visit(stypes, v) << '\n';
    }
}
```

```
struct NecVisitor {

    template<typename T, typename U>
    void operator()(const T& x, const U& y)
    {
        if constexpr (std::is_same_v<T, std::string> && std::is_same_v<U, std::bitset<16>>) {
            std::cout << "string - bitset\n";
        }
        else if constexpr (std::is_same_v<T, int> && std::is_same_v < U, double>) {

            std::cout << "int - double\n";
        }
        else {
            std::cout << "other type combinations\n";
        }
    }
};

int main()
{
    using namespace std;

    variant<int, string, long> vx{ "muhittin" };
    variant<char, double, bitset<16>> vy{ bitset<16>{12765u} };

    vx = 56;  I
    vy = 4.5;

    visit(NecVisitor{}, vx, vy);
}
```

crtp runtime polymorphism

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

    any x = "mustafa"s;

    if (x.type() == typeid(int)) {
        std::cout << "int tutuyor\n";
    }
    else if (x.type() == typeid(double)) {
        std::cout << "double tutuyor\n";
    }
    else if (x.type() == typeid(string)) {
        std::cout << "string tutuyor\n";
    }
}
```

```
any x = 5.6;

try {
    auto val = any_cast<int>(x);
    cout << "val = " << val << '\n';
}

catch (const std::exception& ex) {
    std::cout << "exception caught: " << ex.what() << '\n';
}
```

```
using namespace std;

boolalpha(cout);
any x;

cout << "has value : " << x.has_value() << '\n';
x = "nurettin"s;
cout << "has value : " << x.has_value() << '\n';
x.reset();
cout << "has value : " << x.has_value() << '\n';
x = 6.6;
cout << "has value : " << x.has_value() << '\n';
x = any{};
cout << "has value : " << x.has_value() << '\n';
x = 9L;
cout << "has value : " << x.has_value() << '\n';
x = {};
cout << "has value : " << x.has_value() << '\n';
```

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

    any a{ "Kaya"s };

    auto& ra = std::any_cast<string>(a);

    std::cout << any_cast<string const>(a) << '\n';
    ra[0] = 'M';

    std::cout << any_cast<string const>(a) << '\n';

    auto str = any_cast<string>(&move(a));

    //static_assert(is_same_v<decltype(str), string>);

    std::cout << "a.size(): " << any_cast<string>(&a)->size() << '\n';

    cout << "str: " << str << '\n';
}
```

```
using namespace std;

vector<any> avec;

avec.push_back(12);
avec.push_back(3.4);
avec.push_back("dilek"s);
avec.push_back(Date{ 3, 6, 1987 });
avec.push_back(34.57);
avec.push_back(999);
avec.push_back(Date{ 31, 12, 2023});

for (const auto& a : avec) {
    if (auto p = any_cast<int>(&a)) {
        std::cout << "int : " << *p << '\n';
    }
    else if (auto p = any_cast<double>(&a)) {
        std::cout << "double : " << *p << '\n';
    }
    else if (auto p = any_cast<Date>(&a)) {
        std::cout << "Date : " << *p << '\n';
    }
}
```

```
using vpair = std::pair<std::string, std::any>;  
  
int main()  
{  
    using namespace std;  
  
    vector<vpair> avec;  
  
    avec.push_back({ "int", 56 });  
    avec.push_back({ "double", 2.3 });  
    avec.push_back({ "string", "necati"s});  
    avec.push_back({ "char", 'A'});  
    //...  
}  
I
```

Böyle bir kullanımda var

```
using type = ratio_add<ratio<3, 5>, ratio<4, 5>>::type;  
  
constexpr auto num = type::num;  
constexpr auto den = type::cden;
```

```
cout << typeid(ratio_add<one_third, two_fifth>::type).name() << "\n";  
cout << typeid(ratio_subtract<one_third, two_fifth>::type).name() << "\n";  
cout << typeid(ratio_divide<one_third, two_fifth>::type).name() << "\n";  
cout << typeid(ratio_multiply<one_third, two_fifth>::type).name() << "\n";  
ratio_less_v<ratio....>
```

deca	struct std::ratio<10,1>
hecto	struct std::ratio<100,1>
kilo	struct std::ratio<1000,1>
deci	struct std::ratio<1,10>
centi	struct std::ratio<1,100>
milli	struct std::ratio<1,1000>
micro	struct std::ratio<1,1000000>
nano	struct std::ratio<1,1000000000>
pico	struct std::ratio<1,1000000000000000>

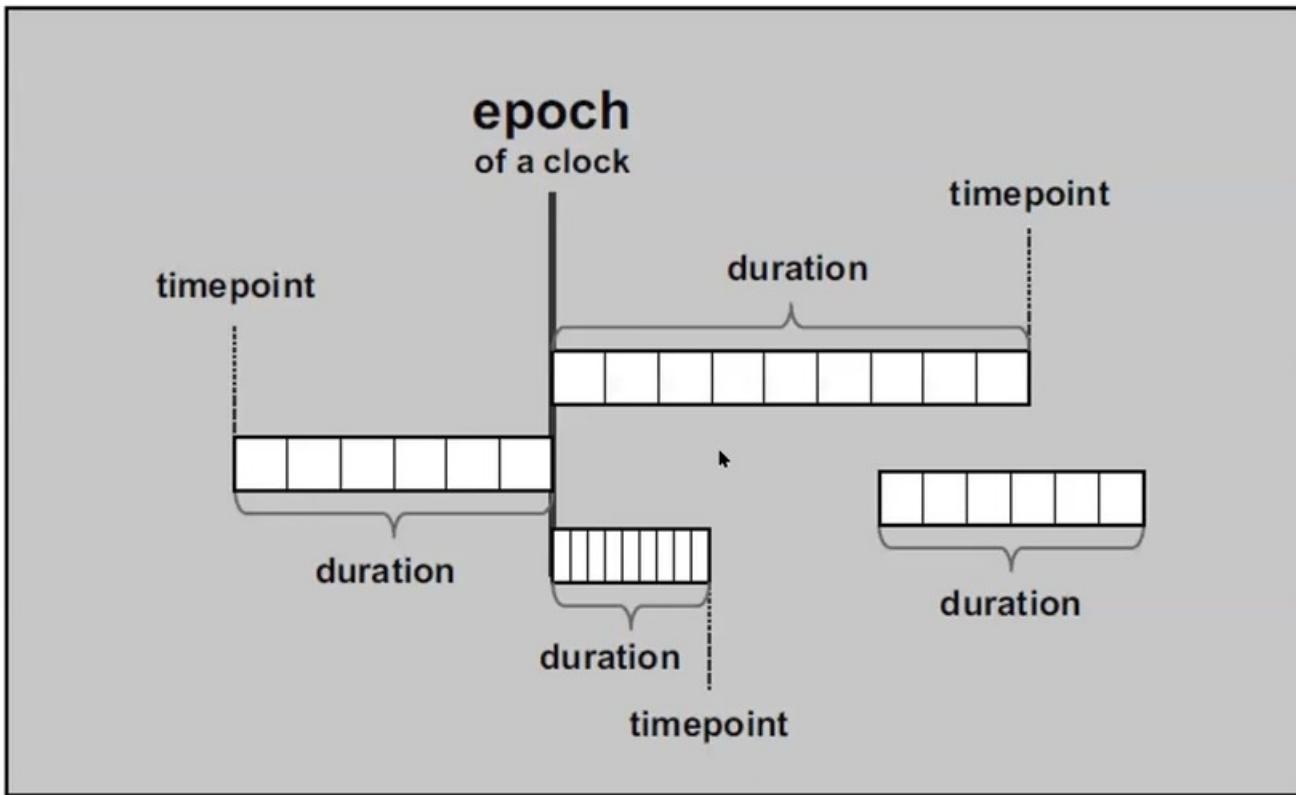


Figure 5.4. Epoch, Durations, and Timepoints

```
cout << typeid(minutes).name() << "\n";
cout << typeid(hours).name() << "\n";
cout << typeid(seconds).name() << "\n";
cout << typeid(milliseconds).name() << "\n";
cout << typeid(microseconds).name() << "\n";
cout << typeid(nanoseconds).name() << "\n";
```

Microsoft Visual Studio Debug Console

```
class std::chrono::duration<int,struct std::ratio<60,1> >
class std::chrono::duration<int,struct std::ratio<3600,1> >
class std::chrono::duration<__int64,struct std::ratio<1,1> >
class std::chrono::duration<__int64,struct std::ratio<1,1000> >
class std::chrono::duration<__int64,struct std::ratio<1,1000000> >
class std::chrono::duration<__int64,struct std::ratio<1,1000000000> >
```

C:\Users\necat\source\repos\chrono\Release\chrono.exe (process 1164) - [File: C:\Users\necat\source\repos\chrono\chrono.h Line: 11]

```
constexpr auto val = 12h + 123min + 239s + 3127ms;
```

Ms cinsinden olur en küçük o diye

Operation	Effect
$d1 + d2$	Process sum of durations $d1$ and $d2$
$d1 - d2$	Process difference of durations $d1$ and $d2$
$d * val$	Return result of val times duration d
$val * d$	Return result of val times duration d
d / val	Return of the duration d divided by value val
$d1 / d2$	Compute factor between durations $d1$ and $d2$
$d \% val$	Result of duration d modulo value val
$d \% d2$	Result of duration d modulo the value of $d2$
$d1 == d2$	Return whether duration $d1$ is equal to duration $d2$
$d1 != d2$	Return whether duration $d1$ differs from duration $d2$
$d1 < d2$	Return whether duration $d1$ is shorter than duration $d2$
$d1 <= d2$	Return whether duration $d1$ is not longer than duration $d2$
$d1 > d2$	Return whether duration $d1$ is longer than duration $d2$
$d1 <= d2$	Return whether duration $d1$ is not shorter than duration $d2$
$++d$	Increment duration d by 1 tick
$d++$	Increment duration d by 1 tick
$--d$	Decrement duration d by 1 tick
$d--$	Decrement duration d by 1 tick
$d += d1$	Extend the duration d by the duration $d1$
$d -= d1$	Shorten the duration d by the duration $d1$
$d *= val$	Multiply the duration d by val
$d /= val$	Divide the duration d by val
$d \%= val$	Process duration d modulo val
$d \%= d2$	Process duration d modulo the value of $d2$

Table 5.21. Arithmetic Operations of durations

```
auto x = 8798723423us;  
auto y= ceil<seconds>(x)  
auto y= floor<seconds>(x)    round var bide  
auto y = duration_cast<seconds>(x);
```

```
cout << "y = " << y << '\n';
```

milliseconds ms {23123}

const char [5]"y = "

Search Online

int ival= ms.count()

küçük veriyi büyük veriye atamak
bu şekilde mümkün ama veri
kayıbı oluyor 8798 saniye basar

```
int n;

std::cout << "milisaniye degerini girin: ";
cin >> n;
milliseconds ms{ n };

Day d = duration_cast<Day>(ms);

cout << d << " ";

hours h = duration_cast<hours>(ms % Day{ 1 });

minutes m = duration_cast<minutes>(ms % hours{ 1 });
cout << m << " ";

seconds s = duration_cast<seconds>(ms % minutes{ 1 });
cout << s << " "; I
milliseconds msx = duration_cast<milliseconds>(ms % seconds{ 1 });
cout << msx << " ";
```

```
template <typename R, typename P>
std::ostream& operator<<(std::ostream& os, const std::chrono::duration<R, P>& dur)
{
    return os << dur.count() << " * " << "[" << P::num << " / " << P::den << "]";
}
```

577*1/10

```
int main()
{
    using namespace std;
    using namespace std::chrono;

    duration<int, ratio<1, 10>> x{ 577 };

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



```
template <typename R, typename P>
std::ostream& operator<<(std::ostream& os, const std::chrono::duration<R, P>& dur)
{
    return os << dur.count() << " * " << "[" << P::num << " / " << P::den << "]";
}
```

577*1/10

```
int main()
{
    using namespace std;
    using namespace std::chrono;

    duration<int, ratio<1, 10>> x{ 577 };

    cout << x << "\n";
```

T

```
1 #include <ratio>
2 #include <chrono>
3 #include <iostream>
4
5
6 int main()
v 7 {
8     using namespace std;
9     using namespace std::chrono;
10
11    //std::cout << "dakika degerini giriniz: ";
12    int n;
13
14    cin >> n;
15    minutes mins{ n };
16
17    auto tp = system_clock::now();
18
19    tp += mins;
20
21    auto t = system_clock::to_time_t(tp);
22
23    cout << ctime(&t) << "\n";
24
25 }
```



```
$ clang++ prog.cc -Wall -Wextra -O2 -march=native -std=gnu++2b -pedantic-errors
```

Run

Share

Mon Feb 17 21:56:38 2025

```
mt19937 eng;
uniform_int_distribution dist{ 0, 100'000 };

std::cout << "islemler basladi\n";
vector<int> ivec;

auto tp_start = steady_clock::now();
generate_n(back_inserter(ivec), 10'000'000, [&] {return dist(eng); });
sort(ivec.begin(), ivec.end()); I

auto tp_end = steady_clock::now();
std::cout << "siralama tamamlandi\n";
cout << duration<double>(tp_end - tp_start).count() << "\n";

(void)getchar();
print(ivec);
```

C++ Standard Library

