

11 Nisan 2022

int&& x = 5; ==> x'in türü (veri tipi, türü) => int&& veya rvalue reference  
x'in oluşturduğu ifadenin değer kategorisi (value category) =>  
lvalue expr.

- bir isim(örneğin r) bir referansa ait olsa da ister sağ taraf referansı olsun, ister  
sol taraf referansı olsun o ismin oluşturduğu ifade lvalue expression'dır.

ifadelerin türü referans olamaz.

1) Constexpr variable (değişkenin sabit olma garantis̄ı var,)  
(oturumu: konsteksp̄r) bunu const şartname

Constexpr : constant expression

\* derleme zamanında değeri tespit edilebilir.

- implicitly const, implicitly inline

\* Constexpr ile tanımlanan dēīsken aynı zamanda implicitly const.

Constexpr int x = Constexpr int const x

\* Constexpr bil̄ir niteliklerini dep̄il ama implicitly olarak dep̄izlenir.  
de const yapar.

## 2) Constexpr function (asıl devrin burası) :

- şartı : bütün parametrelerine constexpr sabit ifadesi ile yapılmıştır, ancak o zaman genellikle deper sabit ifadesi olur.

\* C++ dilinde efficiency: artırmak en önemli etkenlerdir  
Dörtme taneanda elde edilir.

constexpr int sum-square(int a, int b)

{  
 return a\*a + b\*b;

}

int main()

{

constexpr auto x = sum-square(3,4);

}

25 yazmak gibi

int main()

{

int x{};

constexpr int\* p = &x; // sadece hata, x sabit olmadığı için.

}

## ODR (one definition rule): acronym

yazılımlı varlıkların sadece bir yerde tanımları olur.  
 Bir yazılımın tanımı asla baslik dosyasına konulmamalı!  
 istisnaları var!

denendi!

inline fonksiyonlar: ODR'i ihlal etmez, baslik dosyasına kopyulabilir.

```
inline void func() {  
    //  
}
```

- // reduce the function call overhead
- // when the inline function is called
- whole code of the inline function
- gets inserted.
- // at compile time
- // not a command, request to compiler

## inline değişkenler:

inline int x=10; // kaf h. dosyasında bulunursa bulunur  
 ODR'i ihlal etmez. link aşamasında  
 bir tane sayılır.

constexpr int x=10; // constexpr, implicitly const. tur.  
 Yani inline static eklenen de  
 implicitly inline'dır.

## How To COMPILE C++ code in GCC?

```
$ g++ -o hellowsrd constexprdd.cpp main.cpp
```

executable name

Reader's fork you

```
$ ./hellowsrd
```

## Function overloading (funksiyon yüklenmesi)

hatalı çevirisi: fonsiyonun asırı yüklenmesi

aynı isimle birden fazla fonsiyonun bir arada bulunması.

derleyici tarafından fonsiyonun yapanın neyi: fonsiyon  
yapıldığı

a) static binding (early binding): Compile time'da belli  
oluyorsa

b) late binding (dynamic binding):

function overloading, **Static binding** tır. Programın  
Girişme zamanına ilave maliyeti yok.

## Function overloading resolution: aynı isimli fonsiyonlar birarada bulunduğunda fonsiyona yapan çağrıda hangi fonsiyonun başlanacağı kararını verir.

Is it function overloading ?

1) aynı scope (kapsam) 'da olmalı

2) function signature (imzaları) farklı olacak  
int func(int, int, int)

geri dönüş türü dahil edilmez !!!

global'de ve lokal'de  
bibirilen fons.

Const olua imzada  
farklılık oluşturmayı.

- Scopes:
- a) namespace
  - b) class
  - c) block
  - d) function
  - e) function prototype
- 

Function redeclaration: bir syntax hatasıdır.

```
int foo(int);  
double foo(int); // hata  
double foo(int, int); // function overloading  
int foo(const int); // function redeclaration  
// parametrem const olususlu  
// içinde farklılık olusturur.
```

---

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

```
void foo(int&); // function overloading var, int& baska bir tür
```

```
int func(int);
```

```
int func(int32_t); // function overloading olup olumlu  
derleyiciye boyl.
```

```
void func(int );
void func(int, int);
void func(int, double);
void func(char* );
void func(int, color);

int main(){

// func(??) // 1. aşamada derleyici fonksiyon argumanlarına hiç bakmadan bu
noktada visible olan fonksiyonların listesini çıkartır. aday fonksiyonlar belirlenir, şu
an 4 aday fonksiyon görülür. (candidate function)
```

// 2. aşamada derleyici fonksiyona gönderilen elemanlardan hangi  
fonksiyon çağrılarının legal olduğuna bakıyor ve çağrırlması uygun olmayanlar  
aday fonksiyonlardan çıkarılır. (viable functions (uygun fonksiyonlar)) burada  
argüman sayısı ile bildirilen fonksiyonun parametre sayısı eşit olacak , her bir  
argümandan ilgili parametreye geçerli bir tür dönüşümü olacak

// 3. aşamada birkaç seçenek kalırsa derleyici seçemezse "ambiguity  
(belirsizlik)" olur.

```
func(12, 56); // 2-3 olabilir
               // int türünden enum türüne otomatik tür dönüşümü yok
}
```

### **function overloading resolution methods:**

<https://www.dcs.bbk.ac.uk/~roger/cpp/>

- 1) variadic conversion (variadic olmayan kazanır)
- 2) user-defined conversion
- 3) standart conversion // 1-2'nin dışındakiler
  - a) exact match (argüman türü ile parametrenin türü aynı olması )
  - b) promotion (terfi- yükseltme)
  - c) (normal) conversion ==>> ambiguity

exact match > promotion > conversion

user defined-conversion variadic conversion'dan daha iyi ama onun dışındaki  
bütün dönüşümlere kaybeder.

1) variadic function ==> void func(int, ...);  
...'den önceki parametrelere argüman göndermek zorunlu !!!!  
variadic uyum olabilecek en düşük seviye !!! (iki viable fonksiyondan variadic olmayan kazanır !!!)

2) user-defined conversion: normalde yok, o dönüşümde kullanılabilecek fonksiyon bildirirsek

```
struct Data{  
    Data(int); => bu initialization'dan dolayı user-defined conversion olur  
}  
  
void func(Data)  
  
int main(){  
    func(12);  
}
```

3) variadic ve user-defined conversion yoksa standart conversion durumunda ne olacak ???

- a) exact match (argüman türü ile parametrenin türü aynı olması )
- b) promotion (terfi- yükseltme)
- c) (normal) conversion ==> ambiguity

exact match > promotion > conversion

**a) exact match:**

**a1) array decay : array to pointer conversion**

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

```
int main(){  
    int a[10];  
    func(a); // dizinin ismi dizinin ilk elemanın adresine dönüştürüyor. Buna  
    array decay denir!! exact match'dir.  
}
```

## a2) const conversion:

```
void func(const int*);  
  
int main(){  
    int x = 10;  
    func(&x); // &x'ün türü int*, func ise const int*, exact match  
}
```

✖ &x : adres ise türü => int\*  
&y : referans ise türü => int&

## a3) function pointer (function to pointer conversion)

```
void func(int (*)(int ));  
int foo(int);  
  
int main(){  
    func(foo); // exact match kabul edilir  
}
```

12. => double  
12.34 => double  
12.34F => float  
(float)12.4 => yanlışşşşş  
12.4L => long double

10 > 4 => karşılaştırma ifadesi bool türden değer üretir  
'A' => karakter literalı, char'dır.  
12e3 => scientific notation olduğunda türü double'dır.  
0x1ac4 => int tür (hex sayı sisteminde yazılması türünü değiştirmez, boyutuna bakılır)  
0b11011 => int tür (ikili sayı sistemi)

## b) promotion

int altı argümanların int'e yükseltilmesi dönüşümü integral promotion'dır

unsigned short, signed short => int

3 char türünden => int

bool => int

float => double // promotion

c) **Conversion** → bütün bunlar sağlanmazsa ambiguity'dir  
glüsünde de conversion varsa ambiguity

```
int main(){  
    func(10); // int = double or int = unsigned int , normal  
conversion'da ambiguity  
    func(10u); // unsigned int kazanındı. exact match  
    func(10.); // double kazanındı, exact match  
    func(10.f); //double çağrırlıır promotion , unsigned olan  
conversion,  
}
```

**bazı özel durumlar var, bunları özel kural olarak bilmeniz  
gerekıyor !!!**

1) kullanmayın !!

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

```
int main(){
```

func(0); // kullanma, başa bela açar, çünkü int overload gelse bu  
fonksiyon kaybeder, veya double gelse ambiguity olur.

```
    func(nullptr)  
}
```

C++ nullptr yerine 0 kullanmayın !!!

```
void func(int * );  
void func(double); //  
int main(){  
    foo(0); // ambiguity, çünkü 0'dan nullptr dönüşüm conversion, 0'dan  
double'a dönüşüm conversion  
}
```

# Const overloading

```
void func(int * );
void func(const int*); => overloading'tir. const overloading
```

## const overloading kuralı:

T\* ile const T\* overloading'i varsa, T ile çağrılmınca T\* olan, const T ile çağrılmınca const T\* olan fonksiyon çağrılmış olur !!!

T& ile const T& overloading'i varsa, T ile çağrılmınca T& olan, const T ile çağrılmınca const T& olan fonksiyon çağrılmış olur !!!

```
void func(int );
void func(const int& );
int main(){
    func(5); // ambiguity
}
```

```
void func(void *);
int main(){
    int ival {};
    func(&ival); // geçerli, void * türüne
    dönüşür
}
```

```
void func(bool);
int main(){
    int ival = 10;
    func(&ival); // geçerli, nesne adresleri bool türüne örtülü
    dönüşümde true'ya dönüşür, nullptr false'a dönüşür
}
```

```
void func(bool);
void func(void *);
int main(){
    int ival = 10;
    func(&ival); // sadece bu void func(void *) olsa, geçerli idi. ikisi olunca void *in
    üstünlüğü var. (özel durum)
}
```

en karışık yerlerden biri  
fonksiyonların birden fazla değişkeni olursa

**kural : bir fonksiyonun seçilebilmesi için en az bir argümanda diğerleri üstünlük sağlayacak, üstünlük sağladığı argümanların dışındaki diğer argümanlarda da diğerlerinden kötü olmayacak**

```
void func(int, double, int);
void func(double, long, int);
void func(char, float, double);
```

```
int main(){
    func(10, 20, 5u); // 1. exact match üstünlük sağladı, 2.'de üçü eşit conversion, 3. 'de eşit
conversion
    func(10, 20, 5.f); // ambiguity, 3. üstünlük sağladı.
}
```

**r value reference 'ı sentax'a eklenme gereklisi:**

- 1) Taşıma Semantiği (verimle ilgili)
- 2 ) Perfect forwarding (generic programlama ile ilgili)

```
void func(int&); // lvalue reference parametreli fonksiyon
void func(int&&); // rvalue reference parametreli fonksiyon
```

```
int main (){
    int x {2};
    func(x); // üstteki çağrılır, lvalue to lvalue reference
    func(10); // alttaki çağrılır, rvalue to rvalue reference
}
```

**kural : fonksiyonu lvalue ifade ile çağrırsanız lvalue ref, rvalue ile çağrırsanız rvalue ref çağrırlır.**

```
void func(const T&);
void func(T&&);

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

int main (){
    int x {2};
    func(x); // üstteki çağrılır
    func(10); // ikisi de viable, ambiguity yok, rvalue ref'in reference'a üstünlüğü var !!!
}
```