2023 07 26

Const

```
int main(void)
{
    // Cpp'da geçersiz init etmemiz gerek
    const int x; // C'de geçerli
}
```

```
int foo();

int main(void)
{
    const int x = 10;
    int a[x]; // C'de geçersiz C++'da geçerli

    switch (foo())
    {
        case x; // C'de geçersiz Cpp'de geçerli
    }
}
```

```
// C'de
int x = 10; // external linkage
static y = 10; // internal linkage
int main()
{

}

// Cpp'de
const int x = 10; // internal linkage
extern const int x = 10; // external linkage
```

```
// top level const
int main()
{
    int x = 10;
    // const pointer to int
    // top level const
    // right const
    int * const p = &x;  // burda const olan p değişkenin kendisi

    int y = 34;
    /*
    top level const olarak tanımlanmış variable'in
    adresi değişemez
    */
    p = &y; // geçersiz

    *p = 34; // geçerli
}
```

const int * p ile int const *p arasıda fark yok !!! Const'un yıldızdan önce ile sonra olmasu arasında fark var

```cpp
// pointer to const int
// low level const
int main()
{
    int x = 10;
    int y = 34;

    const int* p = &x;
    p = &y; // geçerli
    *p = 12; // geçersiz
}
```

```cpp
// SORU
typedef int* IPTR; // top level const
typedef const int* CIPTR; // low level const
int main()
{
    int x = 5;
    const IPTR p = &x; // int *const p = &x

    int y = 12;
    p = &y; // syntax hatası
    *p = 56; // geçerli

    CIPTR p1 = &y; // geçerli
    CIPTR p1 = 32; // geçersiz
}
```

```cpp
// const pointer to const int
int main()
{
     int x = 10;
     // *ptr ve ptr'ye atama yapmak syntax hatası
     const int* const ptr = &x;
}
```

```cpp
// array decay
int main()
{
    int a[] = {1, 2, 3, 4, 5};
    //int *p = &a; // syntax hatası

    int *p = a; // array decay
    auto *ptr = &a[0];
}
```

```cpp
int main()
{
    // C'de geçerli Cpp'de geçersiz
    char *p = "emre"; // char[5]
    /*
        Cpp'da "emre" const char* decay olur ve
        const char* dan char* dönüşüm yok
    */
    const char *p = "emre"; // Cpp de geçerli
}
```

str karakter dizisi bir dizi olarak bellekte yer kaplar ve üzerinde değişiklik yapılabilirken, p karakter dizisi işaretçisi, "emre" ifadesini gösteren bir adresi tutar ve bu ifade bir sabit karakter dizisi olduğu için üzerinde değişiklik yapmak uygun değildir.

```
int main()
{
    char str[] = "emre";
    char *p = "emre"; // static ömürlü
}
```

```
struct Data{
    // empty struct C'de geçersiz Cpp'de geçerli
};
```

```
struct Data {
    int a, b, c;
};

int main()
{
    struct Data mydata; // C'de struct ile tanımlamam gerek
}
```

```
int nec = 0;
int main()
{
    struct nec {
        char str[64];
    };
    // C'de çıktı 4 çünkü struct ile tanımlamadık int'i aldı
    // Cpp'de çıktı 64 olur
    printf("sizeof(nec) = %zu\n" sizeof(nec));
}
```

Initilization

```
int main()
 {
    int x; // default init
    int y = 10; // copy init

    int z(10); // direct init
    // küme parantezi init'Leri genel adı uniform init
    int k{ 10 }; //direct list init
    int v{}; // value init

    int m(); // fonksiyon bildirimidir

    ++m; //syntax hatası
 }
```

```
// neden uniform initilization
 /*
    1. uniform olması
    2. narrowing conversion
        veri kaybına neden olan dönüşümlere denir.
        örnek: (float - int) (int x = 3.4)

    3. most vexing parse
        değişken tanımlamasıyla fonksiyon bildirimlerinin karışması

 */
```

```
// norrowing conversion
 int main()
 {
     //narrowing conversion
     int x1 = 3.4; // legal
     int x2(3.4);   // legal

     int x3{ 3.4 }; // syntax hatası
 }
```

```
//most vexing parse

class A{

};

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

int main()
{
    // most vexing parse
    B bx(A()); // fonksiyon bildirimi
    B bx{ A{} }; // değişken bildirimi

    /*
        B bx(A()); böyle yazmak ile B bx(A(*)()); böyle yazmak aynı şey
        decay oluyor
    */

}
```

C ve Cpp'deki fonksiyon bildirimlerinin

```
// ikiside aynı
void func(int a[]);
void func(int *a);

// üçüde aynı
void func(int **p);
void func(int *p[20]);
void func(int *p[]);

// ikiside aynı
void func(int (*p)[20]);
void func(int p[][20]);

// ikiside aynı (function pointer)
void func(int(*)(int));
void func(int(int)); // decay oluyor yani fonksiyondan türünden fonksiyon adresine
dönüşüyor
```

```
struct Data{

};
// parametresi olmayan geri dönüş değeri  Data olan fonksiyon
void foo(Data(*)());
void foo(Data());
```

```
// C'de nullptr
int main()
{
    int *p =  NULL; // NULL başka başlık dosyalarından gelir
    int *x = 0; // 0 nullptr'a dönüşüyor
}
```

```
// C++ nullptr --keyword (modern cpp ile geldi)
int main()
{
    //nulptr bir sabit, keyword ve türü nullptr_t
    // pointer olmayan değişkene atamak syntax hatası
    int *p = nullptr;
    int x = nullptr; // bu syntax hatası
    // tür güvenliğini sağlar type-safety
}
```