lessons 37 - 39

DERS 37

compile time da template kullanılarak geçilmiş typname T nin türü ne ? nasıl buluruz?

ilk **META function** örneği: (template class yani).

Metafonksiyonlar, şablon tür parametreleri ve sabit ifadeler aracılığıyla derleme zamanında yapılan işlemleri ifade eder. tür sistemini manipüle etme yeteneği sağlar.

```
template <typename T , T v>
struct integral_constant{
   static constexpr T value = v; // T nin değeri
   using value_type = T; // T nin typı
   using type = integral_constant; // ilgili sınıfın tipi
   constexpr operator value_type()const noexcept{return value;}
   constexpr value_type operator()()const noexcept{return value
};

using true_type = integral_constant<bool , true>;
using false_type = integral_constant<bool , false>;

template <typename T>
struct is_pointer: public false_type{};

template <typename T>
struct is_pointer<T*>: public true_type{}; // pointer T ler buna

template <typename T>
constexpr bool is_pointer_v = is_pointer<T>::value;

// example func:
```

```
template <typename T>
void foo(T x)
{
   static_assert(is_pointer_v<T> , "compile time da uyarı: T sa
}
```

BU YUKARDAKİNİN HAZIRI VAR!!: **#include <type_traits>** Yukardakiler ve çok daha fazlası (tüm tür vs hersey kontrolü içinde)

```
#include <type_traits>

template <typename T>
void foo(T x)
{
   static_assert(std::is_pointer_v<T> , "compile time da uyarı:
}
```

SORU: girilen türün içinde & (referans) veya && var, bunu nasıl kaldırırız? Compile time da.

```
template <typename T >
struct remove_referance{
   using type = T;
}

template <typename T >
struct remove_referance<T&>{  //  partial specialization
   using type = T; // bak burda T, & den arınmış halde burda ty
}

template <typename T >
struct remove_referance<T&&>{  //  partial specialization
   using type = T;  // bak burda T, && den arınmış halde burda t
}
```

```
template <typename T >
using remove_referance_t = typename remove_referance<T>::type;

// ex func
void foo()
{
   remove_referance_t<int> x;
   remove_referance_t<int&> x; // int oldu :)
}
```

incele hazır neler var: Yukardakilerin hepsi hazır içinde :

#include <type_traits>

https://en.cppreference.com/w/cpp/header/type_traits

```
#include <type_traits>

template <typename T>
void bar(std::false_type)  // R valuelerde bu işlem
{
}

template <typename T>
void bar(std::true_type) // L valuelerde bu işlem
{
}

template <typename T>
void foo(T&& x)  // X value (R + L)
{
   bar(std::is_lvalue_referance<T>{}); // bu eğer T l value ise

// mesela -> std::is_integral<T>{}  int mi değil mi siteyi incel
}
```

```
//main
foo(12);
int x;
foo(x);    // yani compile timeda function seçtirdik
```

TÜM YUKARDAKİLERİN DAHA KOLAYI **C++ 17** da geldi **if constexpr** compile time da template tür kontrolüne bağlı kontrol yaptırılıo

```
template <typename T>
auto get_value(T x)
{
   if constexpr (std::is_pointer_v<T>)  // compile time ta T poi
   {
      return *x;
   }
   else{
     return x;
   }
}

// main
int x = 5;
get_value(x);
```

STD: Standart **Tamplate** Library : BAK bu kütüphanenin olayı zaten tamplate :)

itarators: anlatıldı, STD içindeki string vector vs.. hepsinin içinde var olay şu: **pointer benzeri sınıf**, konum tutuyor

```
std::string x {"t78", "dd", "test123"};
auto rt= x.begin(); // t78
x.end();  // test123
// bu begin end olayları iterator içinde,
```

```
// bu neyi sağlıo bu yukardaki ifadeler STD nin yine başka kütü
++rt;// bu direk iter oldu mesela
```

DERS 38

std algoritmaların parametre değerleri **itarator'**dur

predicate : bool dönderen fonksiyonlara denir : bool *is_init_done()*

- cpp de algoritmalar için loop lardan kaçının, hersey var!!
- algoritmalar C deki mantık fonksiyon pointer ister, (**LAMBDA** burda çok kullanışlı)
- algoritmaların girdileri adres değil iterator genelde(yani pointer takliti yapan classlar), C dekinden girdi çıktı sıralamalar ters oluo

SORU: vectorun içindeki 2 ye tam bölünenleri yazdır:

```
bool is_even(int x){
  return x%2 == 0;
}

int main(){

  using namespace std;

  vector<int> ivec{5,4,8,9,14};
  list<int> ilist(5);

// begin/end ler poninter gibi davranan iteratorler , is_even fu
 CopyIf(ivec.begin(),ivec.end() , ilist.begin() , is_even);
// listeye 2 ye bölünenler yazıldı
}
```

YUKARDAKİNİN DAHA İYİSİ: vectorun içindeki X istenen değere tam bölünenleri yazdır:

```
class is_even{
public:
   is_even(int _x): x(_x){}
   bool operator()(int y)const
   {
       return y%x == 0;
   }
private:
  int x;
};

int main(){

  using namespace std;

  vector<int> ivec{5,4,8,9,14};
  list<int> ilist(5);

  CopyIf(ivec.begin(),ivec.end() , ilist.begin() , is_even{2});

  CopyIf(ivec.begin(),ivec.end() , ilist.begin() , is_even{5});

}
```

ÇOK ÇOK DAHA İYİSİ **LAMBDA:** derleyiciye sınıf türü yazdırır

```
int main(){

  using namespace std;

  vector<int> ivec{5,4,8,9,14};
  list<int> ilist(5);

  int n = 3;  // kaça bölünebilenleri istiyorsun
```

```
  CopyIf(ivec.begin(),ivec.end() , ilist.begin() ,
     [n](int x){return x%n == 0;}              // bu lamda expre
   );

}
```

DERS 39

- iteratorler ve algoritmalar devam , push_back , next , insert vs..
- iterleri manipule ediyoruz → **transform**