**2023.10.18**

**Template**

template argumanları derleyici hangi yöntemlerle anlar:

* deduction
* CTAD (Class Template Argumant Deduction

std::vector vec{1,2,3,4}; // vector template olmasına rağmen CTAD ile böyle tanımladık.

**Template Argument Deduction**

**template<typename** T**>**

**void** foo**(**T**)**

**{**

**}**

**int** bar**(int);**

**int** main**()**

**{**

***// T için yapılan çıkarım:***

foo**(**19**);** ***// int***

foo**(**1.9**)** ***// double***

**const** **int** ival **=** 4**;**

foo**(**ival**)** ***// int (const'luk düşer)***

**int** y**{};**

**int&** r **=** y**;**

foo**(**r**);** ***// int (ref'lık düşer)***

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

foo**(**5**);** ***// int\****

**const** **int** a**[**5**]{};**

foo**(**a**);** ***// const int\****

foo**(**bar**);** ***//int (\*)(int) // function pointer***

**}**

***// Derleyicinin yaptığı çıkarımı görmek için kullanılabilir***

**template** **<typename** T**>**

**class** TypeTeller**;**

**template** **<typename** T**>**

**void** foo**(**T**)**

**{**

TypeTeller**<**T**>** x**;** ***// hatalı çünkü incomplete type***

**}**

**int** main**()**

**{**

foo**(**"bar"**);** ***// const char\****

**}**

**template<typename** T**>**

**void** foo**(**T**,** T**);**

**int** main**()**

**{**

foo**(**12**,** 5**);** ***// ikiside int çıkarımı yapılır***

foo**(**12**,** 5.8**);** ***// ambiguity T için tek bir çıkarım var olabilir***

foo**(**"naci"**,** "cemal"**);** ***// string***

**}**

**template** **<typename** T**>**

**void** foo**(**T**&)**

**{**

TypeTeller**<**T**>** x**;**

**}**

**int** main**()**

**{**

**const** **int** x **=** 10**;**

foo**(**x**);** ***// const int çıkarımı yapılır ref semantiğinden dolayı***

**int** a**[**4**]{};** ***// int[4]***

foo**(**a**);**

**const** **int** a**[**4**]{};** ***// const int[4]***

foo**(**a**);**

**}**

***// mülakat sorusu:***

**template<typename** T**>**

**void** foo**(**T**&,** T**&);**

**int** main**()**

**{**

foo**(**"can"**,** "eda"**);** ***// const char[4]***

foo**(**"naci"**,** "cemal"**);** ***// const char[5] ve const char[6] ambiguity var***

**}**

**template<typename** T**>**

**void** foo**(**T**&,** T**);**

**int** main**()**

**{**

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

foo**(**a**,** a**);**

***// illegal çünkü ilki için a[5] ikincisi için \*a çıkarımı yapılacak***

**}**

**template<typename** T**>**

**void** foo**(**T**&);**

**int** bar**(int);**

**int** main**()**

**{**

foo**(**bar**);** ***//int (int)***

**}**

**Forwarding Reference ya da Universal Reference**

**template<typename** T**>**

**void** func**(**T**&&)** **{}**

**int** main**()**

**{**

**int** x **=** 10**;**

**const** **double** dval **=** 4.5**;**

***// bu fonksiyona ne gönderirsek gönderilem legal olacak***

func**(**x**);** ***// L value***

func**(**45**)** ***// R value***

**}**

1. ihtimal

arguman L value

2. ihtimal

arguman R value

**Reference Collapsing:**

T& &

T& &&

T&& &

-Yukarıdakiler için T& çıkarımı yapılır

-T&& && için ise T&& çıkarımı yapılır

**class** Nec**{};**

**using** ref **=** Nec**&;**

**using** refref **=** Nec**&&;**

**int** main**()**

**{**

Nec mynec**;**

ref**&** r **=** Nec**{};** ***// r'in türü Nec& & 'ten Nec&***

ref**&&** r **=** Nec**{};** ***// r'in türü Nec& &&'ten Nec&***

refref**&&** r **=** Nec**{};** ***// r'in türü Nec&& &&'ten Nec&&***

**}**

**template<typename** T**>**

**void** func**(**T**&&)**

**{**

***// T için çıkarım Nec& olarak yapılır***

***// x'in türü Nec& && olur***

***// reference collapsing ile x'in türü Nec&***

**}**

**class** Nec**{};**

**int** main**()**

**{**

Nec mynec**;**

***// T&&***

func**(**mynec**);** ***// T için çıkarım :Nec&***

**}**

**template** **<typename** T**>**

**void** func**(**T**&&** x**)**

**{**

**}**

**int** main**()**

**{**

**const** **int** x**{};**

func**(**x**);** ***// const int&***

**}**

**template** **<typename** T**,** **int** N**>**

**void** func**(**T**(&**r**)[**N**])**

**{**

**}**

**int** main**()**

**{**

**int** a**[**20**]{};**

**double** b**[**10**]{};**

func**(**a**);** ***// func<int, 20>(a)***

func**(**b**);** ***// func<double, 10>(b)***

func**(**"melike"**);** ***// func<const char, 7>("melike")***

**}**

**template** **<typename** T**,** **typename** U**>**

**void** func**(**T **(\*)**U**)** ***// T 'ın çıkarımı int(\*)(double)***

**{**

**}**

**int** foo**(double);**

**int** main**()**

**{**

func**(**foo**);**

**}**

**template** **<typename** T**>**

**void** func**(**T**)**

**{**

**}**

**int** main**()**

**{**

**void** **(\***fp1**)(int)** **=** func**;** ***// func'in int spec***

**void** **(\***fp2**)(double)** **=** func**;** ***// func'in double spec***

**}**

**template** **<typename** T**>**

**void** Swap**(**T**&** t1**,** T**&** t2**)**

**{**

T temp **{**std**::**move**(**t1**)};**

t1 **=** std**::**move**(**t2**);**

t2 **=** std**::**move**(**temp**);**

**}**

***// C++20***

**template** **<typename** T**,** **typename** U**>**

**void** func**(**T x**,** U y**)**

**{**

**}**

***// yukarıdakinin kısaltması***

**void** func**(auto** x**,** **auto** y**)**

**{**

**}**

Template Functions'larda iki farklı geri dönüş değeri:

* trailing return type
* auto return type

**Trailing Return Type**

**auto** main**()** **->** **int**

**{**

**}**

**int** bar**(int);**

**int** **(\***foo**())(int)** ***// return: int(\*)(int)***

**{**

**return** bar**;**

**}**

***// bunun daha anlaşılır kılmak için***

**auto** foo**()** **->** **int(\*)(int)**

**{**

**return** bar**;**

**}**

trailing return type özellikle:

bir template fonksiyonun geri dönüş değerini fonksiyonun parametreleriyle oluşturalacak değerlerle yapılacağı zaman kullanılır.

**template** **<typename** T**,** typaname U**>**

**auto** sum **(**T x**,** U y**)** **->** **decltype(**x **+** y**)**

**{**

**}**

**Auto Return Type**

**auto** foo**(int** x**)**

**{**

***// derleyici return ifadesinin türünü çıkarımı return ifadesine göre çıkarması***

**return** x **\*** 1.3**;**

**}**

***// ambiguity biri double biri int***

**auto** foo**(int** x**)**

**{**

**if** **(** x **>** 10**)**

**{**

**return** 1**;**

**}**

**return** 2.3**;**

**}**

**template** **<**typaname T**>**

**auto** foo**(**T x**)**

**{**

**return** x**.**bar**();** ***// bar fonksiyonun geri dönüş değeri olur***

**}**

**class** Myclass

**{**

**public:**

**using** value\_type **=** **int;**

**}**

**template** **<class** T**>**

**void** func**(**T x**)**

**{**

**typename** T**::**value\_type y**{};** ***// nested type olduğunu belirtmek için***

T**::**value\_type y**{};** ***// bu value\_type static bir türmüş gibi ifade ediyor***

**}**

**template** **<**typaname R**,** typaname T**,** typaname U**>**

R sum**(const** T**&** x**,** **const** U**&** y**)**

**{**

**return** x **+** y**;**

**}**

**int** main**()**

**{**

**int** x **=** 45**;**

**float** f **=** 3.4f**;**

**auto** val **=** sum **<double>(**x**,** f**);** ***// val double***

***// auto val = sum<double, int, float>(x, f)***

**}**

**Overloading in Template Functions**

**template<typename** T**>**

**void** func**(**T**)**

**{**

std**::**cout **<<** "function template : " **<<** **typeid(**T**).**name**()** **<<** "\n"**;**

**}**

**void** func**(int)**

**{**

std**::**cout **<<** " func(int) \n"**;**

**}**

***/\****

***overload olan template değil fonksiyon şablonunundan oluşturulan spec***

***void func(int)***

***void func<double>(double)***

***void func<int>(int)***

***\*/***

**int** main**()**

**{**

func**(**13.6**);** ***// void func<int>(double)***

func**(**12**);** ***// void func(int) seçilir şablondan üretilen değil***

**}**

**template** **<**typaname T**>**

**void** func**(**T**)** **=** **delete;**

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

**int** main**()**

**{**

***// sadece int arguman olan fonksiyon sytanx hatası olmıcak***

func**(**12**);**

**}**