**2023.10.23**

Explicit Template

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

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

**{**

std**::**cout **<<** 1**;**

**}**

***//explicit template***

**template<>**

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

**{**

std**::**cout **<<** 2**;**

**}**

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

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

**{**

std**::**cout **<<** 3**;**

**}**

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

**{**

**int\*** p**=** **nullptr;**

func**(**p**);** ***// 3 yazar***

**}**

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

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

**{**

std**::**cout **<<** 1**;**

**}**

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

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

**{**

std**::**cout **<<** 2**;**

**}**

***//explicit template***

**template<>**

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

**{**

std**::**cout **<<** 3**;**

**}**

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

**{**

**int\*** p**=** **nullptr;**

func**(**p**);** ***// 3 yazar ( explicit template)***

**}**

***/\****

***önce daha sspesifik olan template seçilir sonra***

***explicit template seçilir***

***\*/***

**Partial Specialization**

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

**struct** Myclass

**{**

**public:**

Myclass**()**

**{**

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

**}**

**};**

***// partial specialization***

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

**struct** **<**Myclass**<**T**\*>>**

**{**

**public:**

Myclass**()**

**{**

std**::**cout **<<** "partial spec. T \*\n"**;**

**}**

**}**

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

**{**

Myclass**<int>** m1**;** ***// primary spec***

Myclass**<int\*>** m2**;** ***// partial spec***

**}**

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

**struct** Mert

**{**

Mert**()**

**{**

std**::**cout **<<** "primary template\n"

**}**

**};**

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

**struct** Mert**<int,** T**>**

**{**

Mert**()**

**{**

std**::**cout **<<** "partial specialization\n"

**}**

**};**

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

**{**

Mert**<double,** **int>** m1**;** ***// primary template***

Mert**<int,** **int>** m2**;** ***// partial template***

**}**

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

**struct** Mert

**{**

Mert**()**

**{**

std**::**cout **<<** "primary template\n"

**}**

**};**

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

**struct** Mert**<**std**::**pair**<**T**,** U**>>**

**{**

Mert**()**

**{**

std**::**cout **<<** "partial specialization\n"

**}**

**};**

**template** **<int** BASE**,** **int** EXP**>**

**struct** Power

**{**

**static** **const** **int** value **=** BASE **\*** Power**<**BASE**,** EXP **-** 1**>::**value**;**

**};**

**template<int** BASE**>**

**struct** Power**<**BASE**,** 0**>**

**{**

**static** **const** **int** value **=** 1**;**

**}**

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

**{**

**constexpr** **int** val **=** Power**<**2**,** 7**>::**value**;**

**}**

***/\****

***alias template (tür eş işim şablonu)***

***variable template***

***variadic templates***

***perfect forwarding***

***\*/***

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

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

**{**

***/\****

***foo'ya gönderilen arg L value ya da R value olabilir***

***arg const ya da non const da olabilir. Eğer func'ta bu durumları***

***korursa buna perfect forwarding denir.***

***\*/***

func**(**x**);**

**}**

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

**{**

foo**(**arg**);**

**}**

**class** Myclass **{};**

**void** foo**(**Myclass**&)**

**{**

std**::**cout **<<** "Myclass&\n"**;**

**}**

**void** foo**(const** Myclass**&)**

**{**

std**::**cout **<<** "const Myclass&\n"**;**

**}**

**void** foo**(**Myclass**&&)**

**{**

std**::**cout **<<** "Myclass&&\n"**;**

**}**

***// her foo için farklı call\_foo yazabiliriz ama parametre sayısı artıkça işler zorlaşcak***

***// bunun yerine***

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

**void** call\_foo**(**T**&&** x**)**

**{**

***// std::forward dönüştürüyor***

foo**(**std**::**forward**<**T**>(**x**));**

**}**

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

**{**

Myclass m**;**

**const** Myclass cm**;**

foo**(**m**);** ***// Myclass&***

foo**(**cm**);** ***// const Myclass&***

foo**(**Myclass**{});** ***// Myclass&&***

**}**

**Alias Template**

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

**using** gset **=** std**::**set**<**T**,** std**::**greater**<**T**>>;**

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

**{**

**using** **namespace** std**;**

set**<int,** greater**<int>>** myset**;**

gset**<int>** myset1**;**

**}**

**Variadic Template**

**template** **<typename** **...**TS**>**

**class** Myclass **{};**

**template** **<typename** **...**TS**>**

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

**template** **<int** **...**VALS**>**

**class** Myclass

**{**

**public:**

**static** **constexpr** **auto** x **=** **sizeof...(**VALS**);**

**};**

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

**{**

**constexpr** **auto** val **=** Myclass**<**1**,** 3**,** 7 **,**6 **,**8**>::**x**;** ***// 5 olur, parametre sayısını verir***

**}**

**template** **<typename** **...**Ts**>** ***// template parametre pack***

**void** func**(**Ts **...**args**)** ***// function parametre pack***

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

**{**

***// void func<int, double, long>***

func**(**1**,** 2.3**,** 45L**);**

**}**

**template** **<typename** **...**Ts**>**

**void** func**(**Ts**&&** **...**args**);**

**template** **<typename** **...**Ts**>**

**void** foo**(**Ts **&&** **...**args**)**

**{**

std**::**forward**<**Ts**>(**args**)...**

***//func(std::forward<int>(p1), std::forward<double>(p2), std::forward<int>(p3));***

**}**

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

**{**

**int** ival**{};**

**double** dval**{};**

foo**(**12**,** dval**,** ival**);**

**}**

**template** **<typename** **...**TS**>**

**void** func**(**TS **...**params**)**

**{**

**int** a**[]** **=** **{**params**...};**

***// int a[] = {p1, p2, p3, p4};***

**}**

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

**{**

func**(**1**,** 5**,** 7**,** 9**);**

**}**

**class** A

**{**

**public:**

**void** fc**();**

**void** foo**();**

**};**

**class** B

**{**

**public:**

**void** fc**();**

**void** foo**();**

**};**

**class** C

**{**

**public:**

**void** fc**();**

**void** foo**();**

**};**

**template<typename** **...**TS**>**

**class** Myclass **:** **public** TS**...**

**{**

**using** Ts**::**foo**...;**

**};**

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

**{**

Myclass**<**A**,** B**,** C**>** m1**;**

m1**.**fa**();**

m1**.**fb**();**

m1**.**fc**();**

**}**

**Recursive Instantiation**

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

**void** print**(const** T**&** t**)**

**{**

std**::**cout **<<** t **<<** " "**;**

**}**

**template** **<typename** T**,** **typename** **...**Ts**>**

**void** print**(const** T**&**t**,** **const** Ts**&** **...**args**)**

**{**

print**(**args**...)**

**}**

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

**{**

print**(**1**,** 2.3**,** "alican"**,** 4.5f**);**

**}**

**template** **<typename** **...**TS**>**

**void** print**(const** Ts**&** **...**args**)**

**{**

**using** Ar **=** **int[];**

Ar**{** **((**std**::**cout **<<** args **<<** '\n'**),** 0**)...};**

**}**

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

**{**

print**(**2**,** 6 **,**1.2**,** "emre"**,** std**::**string**{**"bahtiyar"**},** 84234**);**

**}**

**Fold Expressions ( katlama ifadeleri)**

**template** **<typename** **...**Ts**>**

**auto** sum**(**Ts **...**args**)**

**{**

**(**args **+** **...)** ***// unary rigth fold***

***// derleyici böyle yapar p1 + (p2 + (p3 + p4))***

**}**

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

**{**

**using** **namespace** std**;**

std**::**cout **<<** sum**(**1**,** 3**,** 6**,** 5**)** **<<** "\n"**;**

**}**

Variadic Templates

* recursive function instantiation
* static if
* init list

Fold Expression

* unary right fold
* unary left fold
* binary right fold
* binary left fold

**class** Myclass **{};**

**template** **<typename** T**,** **typename** **...**Ts**>**

std**::**unique\_ptr**<**T**>** MakeUnique**(**Ts**&&** **...**args**)**

**{**

**return** std**::**unqiue\_ptr**<**T**>(new** T**(**std**::**forward**<**Ts**>(**args**)...));**

**}**

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

**{**

MakeUnique**<**Myclass**>(**2**,** 3**,** 16**);**

**}**

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

**constexpr** T pi **=** T**(**3.143214434212L**);**

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

**{**

**auto** x **=** pi**<int>;** ***// x = 3***

**}**

**template** **<size\_t** n**>**

**constexpr** **size\_t** fact **=** n **\*** fact**<**n **-** 1**>;**

**template<>**

**constexpr** **size\_t** factorial**<**0**>** **=** 1**;**

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

**{**

**auto** x **=** fact**<**7**>;**

**}**