**2023.10.25**

**Template Value Type**

**template** **<typename** T**,** T v**>**

**struct** IntegralConstant

**{**

**static** **constexpr** T value **=** v**;** ***// sabit***

**using** value\_type **=** T**;** ***// bir tür***

**using** type **=** IntegralConstant**;** ***// oluşturalan struct kendisi***

**constexpr** **operator** value\_type**()const** **noexcept** **{return** value**;}** ***// tür dönüşüm operator***

**constexpr** value\_type **operator()()const** **noexcept** **{return** value**;}**

**};**

**using** TrueType **=** IntegralConstant**<bool,** **true>** ***// true***

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

**{**

IntegralConstant**<int,** 5**>::**value**;** ***// sabit***

IntegralConstant**<int,** 5**>::**value\_type**;** ***// bir tür***

IntegralConstant**<int,** 4**>::**type ***// IntegralConstant<int, 4>***

**constexpr** **auto** val **=** IntegralConstant**<int,** 4**>{}** **+** IntegralConstant**<int,** 3**>:{};** ***// 7***

**constexpr** **auto** val **=** IntegralConstant**<int,** 5**>{}();** ***// val = 5***

**}**

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

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

**{**

**static\_assert(sizeof(**T**)** **>** 2**,** "sizeof value must be greater than 2"**);**

**}**

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

**{**

func**(**'A'**);**

**}**

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

**struct** IsPointer **:** FalseType **{};**

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

**struct** IsPointer**<**T**\*>** **:** TrueType **{};**

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

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

**{**

**static\_assert(**IsPointer**<**T**>::**value**,** "only for pointer type"**);**

**}**

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

**{**

**}**

#include <type\_traits>

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

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

**{**

**static\_assert(**std**::**is\_pointer\_v**<**T**>);**

***// pointer olmayan türden arguman gönderirsek syntax hatası verir***

**}**

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

**{**

**}**

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

**struct** RemoveReference

**{**

**using** type **=** T**;**

**};**

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

**struct** RemoveReference**<**T**&>**

**{**

**using** type **=** T**;**

**};**

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

**struct** RemoveReference**<**T**&&>**

**{**

**using** type **=** T**;**

**};**

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

**using** RemoveReference\_t **=** **typename** RemoveReference**<**T**>::**type**;**

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

**{**

RemoveReference**<int>::**type ***// int***

RemoveReference**<int&>::**type ***// int***

RemoveReference**<int&&** **>::**type ***// int***

**}**

**Tag-Dispatch**

***// r value gelince ayrı bir kod l value gelince ayrı kod***

***// tag-dispatch***

#include <type\_traits>

#include <iostream>

**void** bar**(**std**::**true\_type**)**

**{**

std**::**cout **<<** "implementaion for l values\n"**;**

**}**

**void** bar**(**std**::**false\_type**)**

**{**

std**::**cout **<<** "implementaion for r values\n"**;**

**}**

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

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

**{**

bar**(**std**::**is\_lvalue\_reference**<**T**>{});**

**}**

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

**{**

func**(**12**);** ***// implementaion for r values***

**int** x**{** 21 **};**

func**(**x**);** ***// implementaion for l values***

**}**

***// tam sayı türleri için ayrı implementasyon olmayanlar için ayrı***

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

**void** func\_impl**(**T**,** std**::**true\_type**)**

**{**

std**::**cout **<<** "tam sayi türleri için\n"**;**

**}**

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

**void** func\_impl**(**T**,** std**::**false\_type**)**

**{**

std**::**cout **<<** "tam sayi olmayan türler icin\n"**;**

**}**

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

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

**{**

func\_impl**(**x**,** std**::**is\_integeral**<**T**>{});**

**}**

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

**{**

func**(**12**);** ***// tam sayi türleri için***

func**(**12.2**);** ***// tam sayi olmayan türler icin***

**}**

**Static If (cpp 17)**

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

**auto** get\_value**(**T x**)**

**{**

**if** **constexpr(**std**::**is\_pointer\_v**<**T**>)**

**{**

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

**}**

**else**

**{**

**return** x**;**

**}**

**}**

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

**{**

**int** x **=** 5**;**

**double** dval **=** 4.971**;**

**int\*** ip **{&**x**};**

**double\*** dp**{&**dval**};**

std**::**cout **<<** get\_value**(**x**)** **<<** "\n"**;**

std**::**cout **<<** get\_value**(**dval**)** **<<** "\n"**;**

std**::**cout **<<** get\_value**(**ip**)** **<<** "\n"**;**

std**::**cout **<<** get\_value**(**dp**)** **<<** "\n"**;**

**}**

**Standart Template Library**

**Iterators**

**template<typename** Iter**>**

**void** print\_array**(**Iter beg**,** Iter end**)**

**{**

**while(**beg **!=** end**)**

**{**

std**::**cout **<<** **\***beg **<<** ' '**;**

**++**beg**;**

**}**

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

**}**

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

**{**

**int** a**[**5**]** **=** **{**1 **,** 3 **,** 4**,** 5**};**

print\_array**(**a **,**a **+** 5**);**

std**::**vector**<double>** dvec**{**1.2**,** 4.5**,** 7.3**,** 9.3**,** 2.45**};**

print\_array**(**dvec**.**begin**(),** dvec**.**end**());**

std**::**list**<**std**::**string**>** names**{**"melike"**,** "tamer"**,** "serhat"**,** "burak"**};**

print\_array**(**names**.**begin**(),** names**.**end**());**

**}**

***template<typename T, typename A>***

***class Vector***

***{***

***class iterator***

***{***

***public:***

***T& operator\*();***

***bool operator!=(const Iterator&)const;***

***operator++();***

***operator++(int);***

***};***

***iterator begin();***

***iterator end();***

***}***

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

**{**

vector**<int>** ivec**(**1000**);**

**auto** bg **=** ivec**.**begin**();**

**auto** iter **=** begin**(**ivec**);** ***// global***

**}**

**Iterator Category**

iterator category:

* input\_iterator
* output\_iterator
* forward\_iterator
* bidirectional\_iterator
* random\_access\_iterator

STL Container'ların iterator category'si belirlidir.

**template<typename** Iter**>**

**void** func**(**Iter beg**,** Iter end**)**

**{**

***// compile time***

**}**

***//std::output\_iterator\_tag***

***//std::input\_iterator\_tag***

***//std::forward\_iterator\_tag***

***//std::bidirectional\_iterator\_tag***

***//std::random\_access\_iterator\_tag***

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

**{**

vector**<**string**>::**iterator**::**iterator\_category ***// iterator\_category söyler***

**}**

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

**struct** IsSame **:** std**::**false\_type **{};**

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

**struct** IsSame**<**T**,** T**>** **:** std**::**true\_type **{};**

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

**constexpr** **bool** IsSame\_v **=** IsSame**<**T**,** U**>::**value**;**

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

**{**

IsSame\_v**<int,** **double>** ***// false***

IsSame\_v**<int,** **int>** ***// true***

**}**

**template<typename** Iter**>**

**void** algo**(**Iter beg**,** Iter end**)**

**{**

**if** **constexpr(**std**::**is\_same\_v**<**Iter**::**iterator\_category**,** std**::**random\_access\_iterator\_tag**>)**

**{**

*///*

**}**

**}**