**2023.10.16**

***/\****

***noexcept:***

***noexcept specifier***

***noexcept operator***

***noexcept specifier:***

***void foo(int)noexcept;***

***exception kesinlikte göndermicem diyor bu fonksiyon bunun avantajı:***

***kodu yazan buna göre kad yazabilir.***

***derleyici daha uygun bir kod seçebilir***

***eğer bu kod run-time'da exception gönderirse terminate fonksiyonu çağrılır.***

***\*/***

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

**{**

**if** **(**1**)**

**{**

**throw** std**::**exception**{};**

**}**

**}**

**void** foo**()noexcept**

**{**

func**();**

**}**

**void** mytermanite**()**

**{**

std**::**cout **<<** "my terminate called\n"**;**

abort**();**

**}**

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

**{**

set\_terminate**(**mytermanite**);**

***// exception yakalanamaz ve terminate fonkisyonu çağrılır***

**try**

**{**

foo**();**

**}**

**catch** **(const** std**::**exception**&** ex**)**

**{**

std**::**cout **<<** "exception caught: " **<<** ex**.**what**()** **<<** "\n"**;**

**}**

**}**

***// daha çok generic programlama ile alakalı***

**void** foo**()noexcept(true);**

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

**void** bar**()noexcept(false);**

**void** bar**();**

**void** foo**()noexcept(sizeof(int)** **>** 2**);**

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

**void** func**(**T**)noexcept(**std**::**is\_nothrow\_copy\_constructible\_v**<**T**>);**

**noexcept operator**

constexpr bool b = noexcept(expr); compiler time oluşur true ya da false döner

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

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

**{**

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

**constexpr** **bool** b **=** **noexcept(**a**[**3**]);** ***// exception göndermez true***

**int** x **=** 423**;**

**constexpr** **bool** b **=** **noexcept(++**x**);** ***// exception göndermez true***

***// exception gönderir çünkü foo(int) noexcept değil***

**constexpr** **bool** b **=** **noexcept(**foo**(int));**

**}**

**class** Myclass

**{**

**public:**

**void** foo**(int,** **int)noexcept;**

**};**

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

**{**

***// true döner***

**constexpr** **bool** b **=** **noexcept(**Myclass**{}.**foo**(**1**,** 5**));**

**}**

**class** Myclass

**{**

**public:**

**void** foo**(int,** **int)noexcept;**

**};**

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

**{**

***// true döner***

**constexpr** **bool** b **=** **noexcept(**Myclass**{}.**foo**(**1**,** 5**));**

**}**

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

***//specifier // operator***

**void** func**()noexcept(noexcept(**Myclass**{}));**

***/\****

***operator olan Myclass türüden bir nesneyi default olarak init ettiğimizde***

***exception throw etmiyorsa true olacak***

***func fonkisyonun no throw garantisi ise de operator olan noexcept fonksiyonun***

***true olup olmamasına bağlı***

***\*/***

**class** Myclass

**{**

**public:**

Myclass**&** **operator++();**

**};**

**void** func**(**Myclass m**)noexcept(noexcept(++**m**));** ***// no except değil***

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

**void** func**(**T x**)noexcept(noexcept(++**x**)** **&&** **noexcept(--**x**));**

**class** Myclass

**{**

**public:**

Myclass**&** **operator++()noexcept;**

Myclass**&** **operator--()noexcept;**

**}**

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

**{**

**constexpr** **auto** b **=** **noexcept(**func**(**12**));**

***/\****

***true döner func int türüden parametre almış gibi olacak***

***++x ve --x except göndermez.***

***\*/***

Myclass mx**;**

**constexpr** **auto** b **=** **noexcept(**func**(**mx**));** ***// true***

**}**

**class** Base

**{**

**public:**

**virtual** **void** func**()noexcept;**

**};**

**class** Der **:** **public** Base

**{**

**public:**

**void** func**()override;** ***// syntax hatası noexcept garantisi vermemiz lazım***

**};**

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

**void** bar**(int)noexcept;**

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

**{**

***// geri dönüş değeri int olan bir fonksiyonun adresini tutar***

**int** **(\***fp**)(int)noexcept;**

**auto** ival **=** fp**(**43**);** ***// exception vermicek***

fp **=** foo**;** ***// syntax hatası verir çünkü foo noexcept değil***

**int** **(\***br**)(int);**

br **=** bar**;** ***// syntax hatası vermez***

**}**

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

**{**

**int** x **=** 35**;**

**constexpr** **bool** b **=** **noexcept(**x**++);**

std**::**cout **<<** "x = " **<<** x "\n"**;** ***// x = 35 yazar unevaluated context***

**}**

**class** Myclass

**{**

**public:**

**~**Myclass**();**

***// dtor exception throw edemez eğer ederse terminate olur***

**};**

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

**{**

**constexpr** **bool** b **=** **noexcept(**Myclass**{}.~**Myclass**());**

***/\****

***true döner***

***dilin kurallarına göre dtor'lar noexcept olmalı***

***\*/***

**}**

Bu fonksiyonlar noexcept garantisi vermeli

-move ctor

-swap

-memory deallocation

**class** Nec

**{**

**public:**

Nec**()noexcept;**

**};**

**class** Myclass

**{**

Nec nec**;**

***// derleyici default ettiği special fonkisyonların kendi noexcept yapma kararını verir***

**};**

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

**{**

***/\****

***Eğer Nec'in ctor noexcept ise true değilse false***

***\*/***

**constexpr** **bool** b **=** **noexcept(**Myclass**{});**

**}**

**GENERİC PROGRAMLAMA IN CPP (TEMPLATES)**

C'de türden bağımsız kod

**void** gswap**(void** **\***vp1**,** **void\*** vp2**,** **size\_t** sz**)**

**{**

**char\*** p1 **=** **(char\*)**vp1**;**

**char\*** p2 **=** **(char\*)**vp2**;**

**while** **(**sz**--)**

**{**

**char** temp **=** **\***p1**;**

**\***p1**++** **=** **\***p2**;**

**\***p2**++** **=** temp**;**

**}**

**}**

**void** greverse**(void** **\***vpa**,** **size\_t** size**,** **size\_t** sz**)**

**{**

**char\*** ps **=** **(char\*)**vpa**;**

**char\*** pe **=** ps **+** **(**size **-** 1**)** **\*** sz**;**

**while** **(**ps **<** pe**)**

**{**

gswap**(**ps**,** pe**,** sz**);**

ps**+=** sz**;**

pe **-=** sz**;**

**}**

**}**

#define swap\_fn(t) swap\_##t(t\* p1, t \*p2) ?

**{**

t temp **=** **\***p1**;**

**\***p1 **=** **\***p2**;**

**\***p2 **=** temp**;**

**}**

swap\_fn**(int)**

**Template Kategorileri**

* function template
* class template
* variable template
* alias template

***// Genel Syntax***

***/\****

***template paramters <> açısal parantez içinde template parametre'leri olur.***

***template parametres:***

***type parameters***

***non-type parametre***

***template parametre***

***// ikiside aynı anlama gelir T yerine herhangi bir tür gelebilir (int, string vb)***

***template <typename T>***

***template <class T>***

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

***template <typename>***

***\*/***

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

**class** Myclass

**{**

**public:**

T foo**(**T **\*);**

**}**

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

**class** Myclass

**{**

**public:**

T foo**(**U**);**

***// U ve T farklı ya da aynı tür olabilir***

**};**

**template** **<int** N**>**

**class** Myclass

**{**

**int** a**[**N**];**

**};**

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

**void** func**(**T**(&)[**N**])** ***// array, string***

**{**

**}**

**template<typename** T**,** std**::size\_t** N**>**

**struct** Array

**{**

T a**[**N**];**

**};**

***// aslında***

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

**{**

std**::**array**<double,** 20**>** x**;**

**}**

***//////////////***

***// T type parametre x type parametre türünden non type parametre***

**template** **<typename** T**,** T x**>**

**class** Myclass

**{**

**};**

***// Myclass<long, 20L>***

***/\****

***Derleyici türü nasıl anlar***

***1. deduction***

***fonksiyon / sınıf (Cpp17)***

***2. explicit template argument***

***3. varsayılan arguman***

***\*/***