**2023.11.01**

**Lambda Expression**

***/\****

***lambda expression***

***lambda ifade kullandığım yerde bir sınıf nesnesi kullanmış oluyorum.***

***Closure Object oluşturur (PR Value)***

***lambda introducer***

***[lambda introducer](parametre değişkeni){fonksiyon ana bloğu(code)}***

***[](){//code}***

***[]()constexpr{//code}***

***[]()mutable{//code}***

***[]()-> type{//code}***

***[]()noexpect{//code}***

***eğer parametresi olmıcaksa parametre parantezini kullanmayabiliriz:***

***[]{}***

***Ama Cpp 23 ile gelen bir özellikle niteleyicilerden herhangi birini***

***kullanıyorsak, yine de parametre değişkeni kullanmayabiliriz. Daha***

***eski Cpp standartlarda kullanmak zorundayız.***

***[]mutable{} cpp23***

***[]()mutable{} cpp23'ten eskiler***

***\*/***

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

**{**

**[]()** **{}();** ***// boş lambda ifadesi***

**}**

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

**{**

**auto** f **=** **[](int** x**)** **{return** x **\*** x**;** **};**

**}**

**template<typename** F**>**

**void** func**(**F f**)**

**{**

**auto** val **=** f**(**12**);**

**}**

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

**{**

func**([](int** a**)** **{return** a **\*** a 34**;})**

**}**

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

**{**

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

vector**<**string**>** svec**;**

rfill**(**svec**,** 100**,** rname**);**

***// lambda ifadelerin en tipik kullanıldığı yer***

count\_if**(**svec**.**begin**(),** svec**.**end**(),** **[](const** string**&** s**)** **{return** s**.**length**()** **==** 7**});**

**}**

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

**{**

**auto** val **=** **[](int** x**)** **{** **return** x **\*** x **+** 5**;}(**20**);** ***// x = 20 verdil***

**}**

***// lambda function içinde statik ömürlü nesneleri doğrudan isimleriyle kullanabiliriz***

**int** g **=** 5**;**

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

**{**

***// g'yi burda tanımlarsak syntax hatası***

**auto** f **=** **[](int** a**)** **{return** a **\*** g**;};**

**}**

**Member Mutable Parametre vs Lambda Mutable Expression**

**class** Myclass

**{**

**mutable** **int** x**;** ***// const fonksiyonlar bile x'i değiştirebilir.***

**};**

***// yukarıdaki ve aşağıdaki mutable kavramları birbirinden tamamen farklı***

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

**{**

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

***// illegal çünkü derleyeci const fonksiyon yazar ++x yapamaz***

**auto** f **=** **[**x**]()** **{++**x**;};**

***// const fonksiyon olarak yazmaz eğer mutable kullanırsak***

f **=** **[**x**]()mutable{++**x**;};**

string str **{**"emre"**};**

f **=** **[**str**]()** **mutable{**str**[**0**]** **=** 'a'**;};**

**int** k **=** 3**,** l **=** 4**,** z **=** 9**;**

f **=** **[**x**,** y**,** z**](int** a**)**

**{**

**return** a **\*** **(**x **+** y **+** z**);**

**};**

***// capture all by copy***

f **=** **[=](int** a**)**

**{**

**return** a **\*** **(**x **+** y **+** z**);**

**};**

**}**

**Capture All by Reference**

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

**{**

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

**int** a **=** 67**;**

**[&**a**]()**

**{**

**++**a**;** ***// call\_by\_reference***

**};**

**int** b **=** 21**;**

**int** c **=** 1**;**

***// capture all by reference***

**auto** f **=** **[&]()** **{};**

***//trailing return type***

f **=** **[](int** x**)->double**

**{**

**return** x **\*** x**;** ***// return double***

**};**

**}**

**class** xyz\_12\_nct

**{**

**public:**

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

**auto** **operator()(**T x**)**

**{**

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

**}**

**}**

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

**{**

***// template lambda***

**auto** f **=** **[](auto** x**)** ***// cpp 14***

**{**

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

**};**

**}**

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

**{**

***// ikisi farklı türler***

**auto** f1 **=** **[]()** **{};**

**auto** f2 **=** **[]()** **{};**

std**::**is\_same\_v**<decltype(**f1**),** **decltype(**f2**)>;** ***// false***

**auto** f3 **=** f1**;** ***// bu kod geçerli ( copy ctor'ları var )***

***/\****

***Cpp 20 ve sonrasıda bunun geçerli olması için lambda fonksiyon***

***stateless olmalı***

***int x = 5;***

***auto f1 = [x]() {}; // statefull***

***\*/***

**decltype(**f1**)** f4**;** ***// cpp20 ve sonrası geçerli***

**}**

**Positive Lambda**

***// positive lambda***

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

**{**

**auto** x **=** **[](int** a**)** **{return** a **\*** a**};** ***// closure type***

std**::**cout **<<** **typeid(**x**).**name**()** **<<** '\n'**;**

x **=** **+[](int** a**)** **{return** a **\*** a**};** ***// function pointer***

std**::**cout **<<** **typeid(**x**).**name**()** **<<** '\n'**;**

**}**

**Noexpect Lambda**

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

**{**

**auto** f **=** **[](int** x**)** **{return** x **\*** x**};**

**noexcept((**f**(**12**));** ***// false***

f **=** **[](int** x**)noexcept{return** x **\*** x**};**

**noexcept((**f**(**12**));** ***// true***

**}**

**Constexpr Lambda**

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

**{**

***// constexpr olmasını engellecek bir durum olursa syntax hatası verecek***

**auto** f **=** **[](int** x**)** **constexpr**

**{**

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

**return** x **\*** 2**;**

**}** ***// syntax hatası***

**}**

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

**class** Myclass

**{**

**};**

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

**{**

Myclass**<decltype>([]{})** m1**;** ***// cpp 20***

**}**

**template<auto** x **=** **[]{}>**

**struct** Myclass

**{**

**inline** **static** **int** ival **=** 5**;**

**};**

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

**{**

Myclass**<>** m1**;**

m1**.**ival**++;**

m1**.**ival**++;**

m1**.**ival**++;**

Myclass**<>** m2**;**

***// m1 ve m2 farklı objeler***

std**::**cout **<<** m1**.**ival **<<** "\n"**;** ***// 8***

std**::**cout **<<** m2**.**ival **<<** "\n"**;** ***// 5***

**}**

**Lambda İfadeleriyle:**

1) isimlendirilmiş değişken halen getirip kullanabiliriz

2) bir fonksiyon şablonuna arguman olarak gönderebiliriz

***vector<int> ivec {1, 2, 3, 4, 5, 6}***

***sort(ivec.begin(), ivec.end(), [](int a, int b)***

***{***

***return abs(a) < abs(b);***

***});***