در عبارت lambda بدون اسم هم می توان تابع recursive نوشت

Syntax تابع lambda به این شکل است :

[ ](int x) { return x % 2 ; }

تابع لامبدا می تواند مقادیر بیرون از خودش را هم به همراه داشته باشد ( به جز پارامتر های ورودی)

این کار به صورت امکان پذیر است : 1- با مقدار 2- با reference

برای با مقدار پاس دادن متغیر ها به این گونه عمل میکنیم

[=](int x) { sum += x ; }

و همچنین برای by reference پاس دادن مقادیر

[&](int x) { sum += x ; }

در واقع = برابر مقدار و & برابر reference است

می توان به صورت جدا برای هر متغیر هم مشخص کرد که کدام متغیر ها با مقدار و کدام ها با reference دریافت شوند

[&sum, factor] (int a ) { sum += factor \* a ;}

در خط بالا sum با reference و factor با مقدار دریافت شدند (نداشتن علامت قبل اسم متغیر به معنی مقدار است)

نکته مهم در توابعی مانند for\_each این است که تابعی که انتظار دارد به آن پاس شود باید فقط یک ورودی int بگیرد برای همین نمی توان sum یا factor را به عنوان ورودی تابع مشخص کرد

Type این lambda expression به این شکل مشخص می شود:

function<bool (int) >

برای تعریف template lambda میتوان به این شکل عمل کرد

Template<typename T>

Using iset = function <bool (T)> ;

نکته دیگری که در مورد توابع لامبدا می توان متذکر شد این است که توانایی تبدیل داده را به آرگومان استاندارد خود دارد

یعنی تابعی که قرار است یک point دریافت کند (نقطه مشخص شده با x و y ) وقتی شما همان x و y را در قالب یک لیست به عنوان ورودی تابع به آن بدهی خودش عمل تبدیل را انجام می دهد(چراکه تایپ ورودی خودرا می داند)

K(point(0,-2)) ;

K({0 , -2} ) ;

این دو باهم برابرند

تغییرات C++ 11 :

اضافه شدن move semantics در C++ 11

اضافه شدن nullptr

اضافه شدن unique\_ptr که احتیاج به delete کردن ندارد

اضافه شدن shared\_ptr که reference count دارد

Type conversion :

در سه شکل constructor و operator= و cast operator

Cast operator :

Operator A()

که همان تبدیل نوع است

وقتی ورودی یک تابع نوع A است که copy constructor برای تبدیل از نوع B به خود را دارد و به آن تابع متغیر نوع B پاس میشود خودش copy constructor A را صدا میزند و این تبدیل انجام می شود

اگر همچین چیزی مدنظر نباشد باید قبل تعریف تابع کلمه explicit مشخص شود

Cast یک پوینتر به کلاس base خود امن است اما بلعکس آن امن نیست

اگر bp پوینتری به کلاس base باشد و dp پونتری به کلاس derive شده از آن , dp را به دو شکل میتوان به bp cast کرد :

Dp = (derived\*) bp ;

Dp = dynamic\_cast<derived\*> (bp) ;

خط دوم اگر این dynamic\_cast سازگار نباشد در پونتر ما null میگذارد

میتوان برای چک کردن یکی بودن تایپ ها این کار را کرد :

#include <typeinfo>

If(typeid(a) == typeid(b)) { ….}

که یک متد به اسم name هم دارد که اسم تایپ رو هم می نویسد :

Cout << typeid(a).name() ;