和C#中的Lambda类似。

lambda语法：

例：程序stl\_test10

// lambda表达式

[] {

*cout* << "hello lambda" << *endl*;

}();

auto lambda = [] (const *string*& str) {

*cout* << str.*c\_str*() << *endl*;

};

lambda("test lambda");

// 带有返回值的Lambda

auto d\_return = []() -> double {

return 42;

};

double d = d\_return();

*cout* << d << *endl*; // 42

输出为：

hello lambda

test lambda

Lambda可以访问外部作用域中未被传递为实参的变量，有两种方式：

1. [=]，按值传递方式传递给lambda表达式，只能访问不能修改
2. [&]，按引用传递方式传递给lambda表达式，可以修改

例：程序stl\_test10

// 使用外部作用域未被传递为实参的数据

int x = 0;

int y = 42;

// x以值传递传递给lambda，是可读数据，但不能改变

// y以引用传递给lambda，可以改变它

auto qqq = [x, &y] {

*cout* << "x: " << x << *endl*;

*cout* << "y: " << y << *endl*;

// ++x; // error，不能改变x的值

++y;

};

x = y = 77;

qqq();

qqq();

*cout* << "final y: " << y << *endl*;

输出为：

x: 0

y: 77

x: 0

y: 78

final y: 79

从结果可以看出，以值传递的x在外部被改变后，不会影响Lambda内部的值，以引用传递的y在外部被改变后，会影响Lambda内部的值，而且在Lambda内部改变y时，也会反过来影响到外部作用域。

可变的Lambda：可以修改按值传递的外部数据

例：程序stl\_test10

// 可变Lambda，可以修改按值传递的外部作用域变量

int id = 0;

auto lambda\_mutable = [id] () mutable {

*cout* << "id: " << id << *endl*;

++id;

};

id = 42;

lambda\_mutable(); // 0

lambda\_mutable(); // 1

*cout* << "extern id: " << id << *endl*; // 42

输出为：

id: 0

id: 1

extern id: 42

从结果可以看出：按值传递的外部数据在mutable修饰的Lambda中可以改变，但不会影响到外部作用域的数据，同样外部作用域的数据的改变也不会影响到Lambda内部。

Lambda作为返回值

// 返回值为Lambda表达式

function<int(int, int)> ReturnLambda()

{

return[](int x, int y)

{

return x \* y;

};

}

auto lf = ReturnLambda();

*cout* << lf(6, 7) << *endl*;

输出结果为：42